AWS Database Migration Service

User Guide

API Version API Version 2016-01-01

AWS Database Migration Service: User Guide

Table of Contents

What Is AWS Database Migration Service?

AWS Database Migration Service (AWS DMS) can migrate your data to and from most widely used commercial and open-source databases such as Oracle, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Amazon Aurora, Amazon DynamoDB, Amazon S3, MariaDB, and MySQL. The service supports homogeneous migrations such as Oracle to Oracle, and also heterogeneous migrations between different database platforms, such as Oracle to MySQL or MySQL to Amazon Aurora. The source or target database must be on an AWS service.

To perform a database migration, AWS DMS 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, though large transactions might require some buffering to disk. Cached transactions and log files are also written to disk.

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

In most cases, when performing a migration, you will also want to migrate most or all of the source schema. If you are performing a homogeneous migration (between two databases of the same engine type), you migrate the schema by using your engine’s native tools to export and import the schema itself, without any data. If your migration is heterogeneous (between two databases that use different engine types), you can use the AWS Schema Conversion Tool to generate a complete target schema for you. If you use the tool, any dependencies between tables such as foreign key constraints need to be disabled during the migration's “full load” and “cached change apply” phases. If performance is an issue, removing or disabling secondary indexes during the migration process will help. For more information on the AWS Schema Conversion Tool, see AWS Schema Conversion Tool.

For information on the cost of database migration, go to the AWS Database Migration Service pricing page.

AWS DMS is currently available in the following regions:

Region	Name
Asia Pacific (Tokyo) Region	ap-northeast-1

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

Migration Planning for AWS Database Migration Service

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

- You will need to configure a network that connects your source and target databases to a AWS DMS replication instance. This can be as simple as connecting two AWS resources in the same VPC as the replication instance to more complex configurations such as connecting an on-premises database to an Amazon RDS DB instance over VPN. For more information, see Network Configurations for Database Migration (p. 9)
- **Source and Target Endpoints** – You will need to know what information and tables in the source database need to be migrated to the target database. AWS DMS supports basic schema migration, including the creation of tables and primary keys. However, AWS DMS doesn't automatically create secondary indexes, foreign keys, user accounts, and so on in the target database. Note that, depending on your source and target database engine, you may need to set up supplemental logging or modify other settings for a source or target database. See the Sources for Data Migration (p. 68) and Targets for Data Migration (p. 95) sections for more information.
- **Schema/Code Migration** – AWS DMS doesn't perform schema or code conversion. You can use tools such as Oracle SQL Developer, MySQL Workbench, or pgAdmin III to convert your schema. If you want to convert an existing schema to a different database engine, you can use the AWS Schema Conversion Tool. It can create a target schema and also can generate and create an entire schema: tables, indexes, views, and so on. You can also use the tool to convert PL/SQL or TSQL to PgSQL and other formats. For more information on the AWS Schema Conversion Tool, see AWS Schema Conversion Tool .
- **Unsupported Data Types** – Some source data types need to be converted into the parallel data types for the target database. For tables listing conversions between database data types, see Reference for AWS Database Migration Service Including Data Conversion Reference and Additional Topics (p. 177).

Introduction to AWS DMS

AWS Database Migration Service (AWS DMS) is a web service that you can use to migrate data from a source database to a target database. To work with AWS DMS, one of your databases must be on an AWS service. You can't migrate from an on-premises database to another on-premises database.

Migration: A High-Level View

To perform a database migration, AWS DMS 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, 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:

- Provision a replication server
- Define source and target endpoints (databases)
- Create one or more tasks to migrate data between the source and target databases.

A typical task consists of three major phases:

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

During the full load, AWS DMS loads data from tables on the source database to tables on the target database, eight tables at a time. 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 change capture for a given table doesn't begin until the full load for that table is started. In other words, the point when change capture starts will be different for each individual table.

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

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

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

In most cases, when performing a migration, you will also want to migrate most or all of the source schema. If you are performing a homogeneous migration (between two databases of the same engine type), you migrate the schema by using your engine’s native tools to export and import the schema itself, without any data. If your migration is heterogeneous (between two databases that use different engine types), you can use the AWS Schema Conversion Tool to generate a complete target schema for you. If you use the tool, any dependencies between tables such as foreign key constraints need to be disabled during the migration's “full load” and “cached change apply” phases. If performance is an issue, removing or disabling secondary indexes during the migration process will help. For more information on the AWS Schema Conversion Tool, see AWS Schema Conversion Tool.

AWS DMS Components

The components you work with when using AWS DMS include the following:

Replication instance

The AWS DMS replication instance runs on an Amazon Elastic Compute Cloud (Amazon EC2) instance. The replication instance provides high-availability and failover support using a Multi-AZ deployment. 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, eliminate I/O freezes, and minimize latency spikes during system backups.

AWS DMS uses a replication server that 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. When creating your replication server, you should consider the following:

- EC2 instance class — Some of the smaller EC2 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 consumes a fair amount of memory and CPU.
- Storage — Depending on the EC2 instance class you select, 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 collected during the load. If your source system is busy or takes large transactions, or if you’re running multiple tasks on the replication server, you might need to increase this amount of storage. Usually the default amount is sufficient.

Source endpoint

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. 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 the change data consumable in a meaningful way, without data loss, for the capture process. For example, Oracle requires the addition of supplemental logging, and MySQL requires row-level bin logging. When using Amazon RDS as a source, we recommend ensuring that backups are enabled and that the source database is configured to retain change logs for a sufficient time (24 hours is usually enough).

Target endpoint

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

Task

You can create one of three possible types of migration tasks:

- 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.
- Migrate existing data and replicate ongoing changes — This option performs a full data load while capturing changes on the source. Once the full load is complete, captured changes are applied to the target. Eventually the application of changes reaches a steady state. At this point you can shut down your applications, let the remaining changes flow through to the target, and then restart your applications pointing at the target.
- Replicate data changes only — In some situations it might be more efficient to copy existing data using a method other than AWS DMS. For example, in a homogeneous migration, using native export/import tools might be more efficient at loading the bulk data. In this situation, you can use AWS DMS to replicate changes starting when you start your bulk load to bring and keep your source and target databases in sync

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 Command Line Interface (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.

Schema and code migration

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

Sources for AWS Database Migration Service

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

On-premises and EC2 instance databases

- Oracle versions 10.2 and later, 11g, and up to 12.1, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data source)
- PostgreSQL version 9.4 and later
- MongoDB
- SAP Adaptive Server Enterprise (ASE) 15.7 and later

Amazon RDS instance databases

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

Targets for AWS Database Migration Service

You can use the following databases as a target for data replication using AWS Database Migration Service.

On-premises and Amazon EC2 instance databases

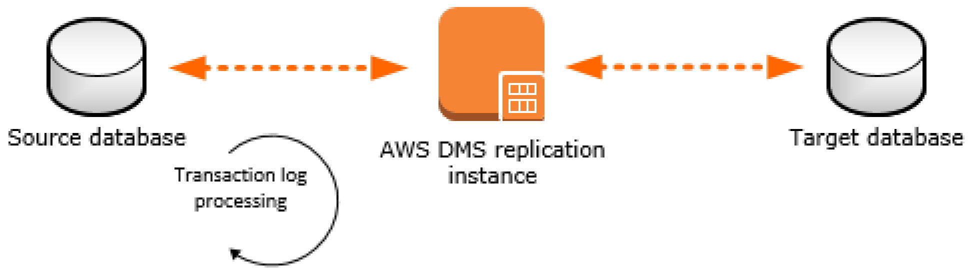
- Oracle versions 10g, 11g, 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL, versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data target)
- PostgreSQL, versions 9.3 and later
- SAP Adaptive Server Enterprise (ASE) 15.7 and later

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

- Oracle versions 11g (versions 11.2.0.3.v1 and later) and 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL, versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data target)
- PostgreSQL, versions 9.3 and later
- Amazon Aurora (supported as a MySQL-compatible data target)
- Amazon Redshift
- Amazon S3
- Amazon DynamoDB

Replication Instances for AWS Database Migration Service

AWS DMS creates a replication instance that runs on an Amazon Elastic Compute Cloud (Amazon EC2) instance in a VPC based on the Amazon Virtual Private Cloud (Amazon VPC) service. You use this replication instance to perform the database migration. The replication instance provides high-availability and failover support using a Multi-AZ deployment when you select the **Multi-AZ** option. In a Multi-AZ deployment, AWS DMS automatically provisions and maintains a synchronous standby replica of the replication instance in a different Availability Zone. The primary replication instance is synchronously replicated across Availability Zones to a standby replica to provide data redundancy, eliminate I/O freezes, and minimize latency spikes.



AWS DMS currently supports the T2 and C4 instance classes for replication instances. The T2 instance classes are low-cost standard instances designed to provide a baseline level of CPU performance with the ability to burst above the baseline. They are suitable for developing, configuring, and testing your database migration process, and for periodic data migration tasks that can benefit from the CPU burst capability. The C4 instance classes are designed to deliver the highest level of processor performance and achieve significantly higher packet per second (PPS) performance, lower network jitter, and lower network latency. You should use C4 instance classes if you are migrating large databases and want to minimize the migration time.

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

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

Public and Private Replication Instances

You can specify whether a replication instance has a public or private IP address that the instance uses to connect to the source and target databases. A replication instance should have a public IP address if the source or target database is located in a network that is not connected to the replication instance's VPC by using a virtual private network (VPN), AWS Direct Connect, or VPC peering.

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

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

AWS DMS Maintenance Window

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

You can manually apply maintenance items at your convenience, or wait for the automatic maintenance process initiated by AWS DMS during your weekly maintenance window. You can find out whether a maintenance update is available for your replication instance by using the AWS CLI or AWS DMS API.

Maintenance items require that AWS DMS take your replication instance offline for a short time. Maintenance that requires a resource to be offline includes required operating system or instance patching. Required patching is automatically scheduled only for patches that are related to security and instance reliability. Such patching occurs infrequently (typically once or twice a year) and seldom requires more than a fraction of your maintenance window.

Use the `DescribeReplicationInstances` API action to determine if there are pending modifications to your replication instance and to see the current time period for your maintenance window. The `PendingModifiedValues` parameter indicates what parameter changes are pending. The `PreferredMaintenanceWindow` parameter shows the current time period for your maintenance window.

To change the time of the maintenance window, use the `ModifyReplicationInstance` API action and set the `PreferredMaintenanceWindow` parameter to your preferred time period.

The default maintenance window time period is determined by the AWS Region. The following table lists the maintenance window for each AWS Region that supports AWS DMS.

Region	Time Block
Asia Pacific (Sydney) Region	12:00–20:00 UTC
Asia Pacific (Tokyo) Region	13:00–21:00 UTC
Asia Pacific (Mumbai) Region	17:30–01:30 UTC
Asia Pacific (Seoul) Region	13:00–21:00 UTC
Asia Pacific (Singapore) Region	14:00–22:00 UTC
Canada (Central) Region	06:29–14:29 UTC
EU (Frankfurt) Region	20:00–04:00 UTC
EU (Ireland) Region	22:00–06:00 UTC
EU (London) Region	06:00–14:00 UTC

Region	Time Block
South America (São Paulo) Region	23:00–07:00 UTC
US East (N. Virginia) Region	03:00–11:00 UTC
US East (Ohio) Region	03:00–11:00 UTC
US West (N. California) Region	06:00–14:00 UTC
US West (Oregon) Region	06:00–14:00 UTC

Setting Up a Network for Database Migration

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

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

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

Network Configurations for Database Migration

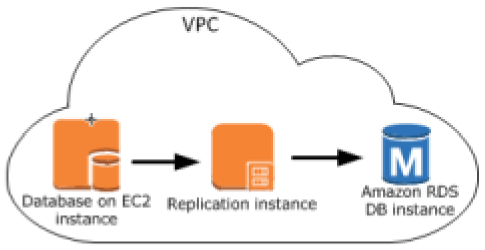
You can use several different network configurations with AWS Database Migration Service. The following are common configurations for a network used for database migration.

Topics

Configuration with All Database Migration Components in One VPC

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 Relational Database Service (Amazon RDS) DB instance or an Amazon Elastic Compute Cloud (Amazon EC2) instance.

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



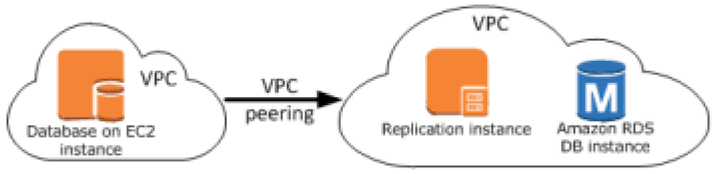
The VPC security group used in this configuration must allow ingress on the database port from the replication instance. You can do this by either ensuring that the security group used by the replication instance has ingress to the endpoints, or by explicitly allowing the private IP address of the replication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

Configuration with Two VPCs

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

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

The following illustration shows a configuration where the source database on an Amazon EC2 instance in a VPC is connected by using VPC peering to a VPC containing the replication instance and the target database on an Amazon RDS DB instance.

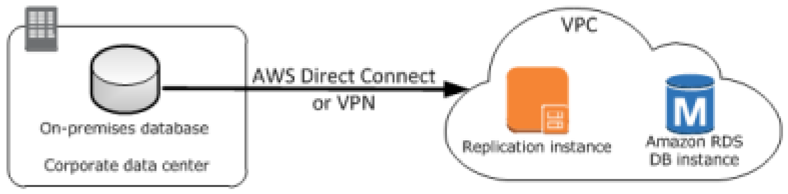


The VPC security groups used in this configuration must allow ingress on the database port from the replication instance.

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

Remote networks can connect to a VPC using several options such as AWS Direct Connect or a software or hardware VPN connection. These options are often used to integrate existing on-site services, such as monitoring, authentication, security, data, or other systems, by extending an internal network into the AWS cloud. By using this type of network extension, you can seamlessly connect to AWS-hosted resources such as a VPC.

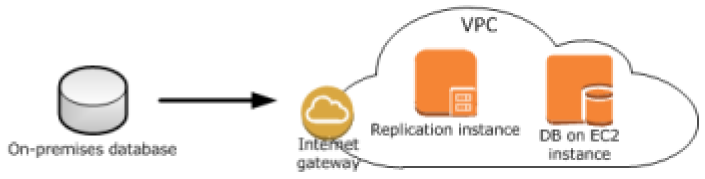
The following illustration shows a configuration where the source endpoint is an on-premises database in a corporate data center. It is connected by using AWS Direct Connect or a VPN to a VPC that contains the replication instance and a target database on an Amazon RDS DB instance.



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

Configuration for a Network to a VPC Using the Internet

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

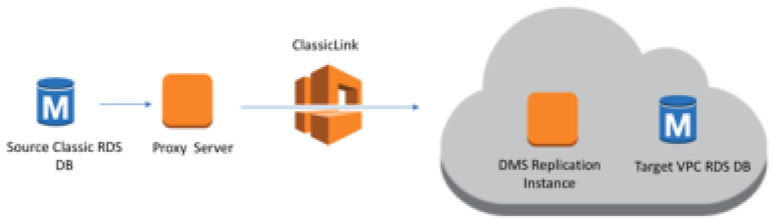


To add an Internet gateway to your VPC, see Attaching an Internet Gateway in the *Amazon VPC User Guide*.

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

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

You can use ClassicLink, in conjunction with a proxy server, to connect an Amazon RDS DB instance that is not in a VPC to a AWS DMS replication server and DB instance that reside in a VPC. ClassicLink allows you to link an EC2-Classic DB instance to a VPC in your account, within the same 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. Since the replication instance in the VPC cannot directly access the source DB instance on the EC2-Classic platform using ClassicLink, you must use a proxy server to connect 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, and port forwarding on the proxy server allows communication between the source DB instance and the target DB instance in the VPC.



For step-by-step instructions on creating a ClassicLink for use with AWS DMS, see Using ClassicLink with AWS Database Migration Service (p. 206).

Creating a Replication Subnet Group

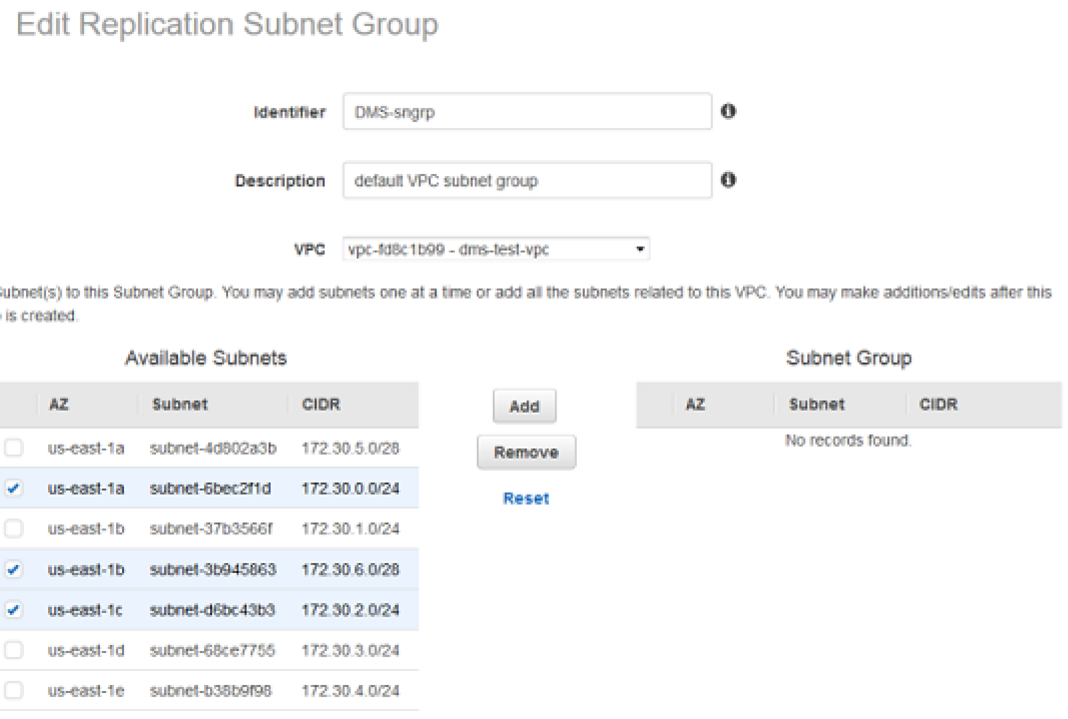
As part of the network to use for database migration, you need to specify what subnets in your Amazon Virtual Private Cloud (Amazon VPC) you plan to use. A *subnet* is a range of IP addresses in your VPC in a given Availability Zone. These subnets can be distributed among the Availability Zones for the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

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

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

To create a replication subnet group

1. Sign in to the AWS Management Console and choose AWS Database Migration Service. Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required for database migration, see IAM Permissions Needed to Use AWS DMS (p. 43).
2. In the navigation pane, choose **Subnet Groups**.
3. Choose **Create Subnet Group**.
4. On the **Edit Replication Subnet Group** page, shown following, specify your replication subnet group information. The following table describes the settings.


Edit Replication Subnet Group

Identifier: DMS-sngrp

Description: default VPC subnet group

VPC: vpc-fd8c1b99 - dms-test-vpc

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

Available Subnets

	AZ	Subnet	CIDR
☐	us-east-1a	subnet-4d802a3b	172.30.5.0/28
✓	us-east-1a	subnet-6bec2f1d	172.30.0.0/24
☐	us-east-1b	subnet-37b3566f	172.30.1.0/24
✓	us-east-1b	subnet-3b945863	172.30.6.0/28
✓	us-east-1c	subnet-d6bc43b3	172.30.2.0/24
☐	us-east-1d	subnet-68ce7755	172.30.3.0/24
☐	us-east-1e	subnet-b38b9f98	172.30.4.0/24

Subnet Group

AZ	Subnet	CIDR
No records found.		

For This Option	Do This
Identifier	Type a name for the replication subnet group that contains from 8 to 16 printable ASCII characters (excluding /,", and @). The name should be unique for your account for the region you selected. You can choose to add some intelligence to the name such as including

For This Option	Do This
	the region and task you are performing, for example `DMS-default-VPC`.
Description	Type a brief description of the replication subnet group.
VPC	Choose the VPC you want to use for database migration. Keep in mind that the VPC must have at least one subnet in at least two Availability Zones.
Available Subnets	Choose the subnets you want to include in the replication subnet group. You must select subnets in at least two Availability Zones.

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

Setting an Encryption Key for AWS Database Migration Service

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

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

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

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

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

Tagging AWS Database Migration Service Resources

You can use AWS Database Migration Service tags to add metadata to your AWS DMS resources. In addition, these tags can be used with IAM policies to manage access to AWS DMS resources and to control what actions can be applied to the AWS DMS resources. Finally, these tags can be used to track costs by grouping expenses for similarly tagged resources.

All AWS DMS resources can be tagged:

- Replication instances
- Endpoints
- Replication tasks
- Certificates

An AWS DMS tag is a name-value pair that you define and associate with an AWS DMS resource. The name is referred to as the key. Supplying a value for the key is optional. You can use tags to assign arbitrary information to an AWS DMS resource. A tag key could be used, for example, to define a category, and the tag value could be a item in that category. For example, you could define a tag key of “project” and a tag value of “Salix,” indicating that the AWS DMS resource is assigned to the Salix project. You could also use tags to designate AWS DMS resources as being used for test or production by using a key such as environment=test or environment =production. We recommend that you use a consistent set of tag keys to make it easier to track metadata associated with AWS DMS resources.

Use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see Cost Allocation and Tagging in *About AWS Billing and Cost Management*.

Each AWS DMS resource has a tag set, which contains all the tags that are assigned to that AWS DMS resource. A tag set can contain as many as ten tags, or it can be empty. If you add a tag to an AWS DMS resource that has the same key as an existing tag on resource, the new value overwrites the old value.

AWS does not apply any semantic meaning to your tags; tags are interpreted strictly as character strings. AWS DMS might set tags on an AWS DMS resource, depending on the settings that you use when you create the resource.

The following list describes the characteristics of an AWS DMS tag.

- The tag key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "aws:" or "dms:". The string might contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\ \-]*)$").

- The tag value is an optional string value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "aws:" or "dms:". The string might contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\ \-]*)$").

  Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of project/Trinity and cost-center/Trinity.

You can use the AWS CLI or the AWS DMS API to add, list, and delete tags on AWS DMS resources. When using the AWS CLI or the AWS DMS API, you must provide the Amazon Resource Name (ARN) for the AWS DMS resource you want to work with. For more information about constructing an ARN, see Constructing an Amazon Resource Name (ARN) for Use with AWS Database Migration Service (p. 16).

Note that tags are cached for authorization purposes. Because of this, additions and updates to tags on AWS DMS resources might take several minutes before they are available.

API

You can add, list, or remove tags for a AWS DMS resource using the AWS DMS API.

- To add a tag to an AWS DMS resource, use the `AddTagsToResource` operation.
- To list tags that are assigned to an AWS DMS resource, use the `ListTagsForResource` operation.
- To remove tags from an AWS DMS resource, use the `RemoveTagsFromResource` operation.

To learn more about how to construct the required ARN, see Constructing an Amazon Resource Name (ARN) for Use with AWS Database Migration Service (p. 16).

When working with XML using the AWS DMS API, tags use the following schema:

```
<Tagging>
  <TagSet>
   <Tag>
     <Key>Project</Key>
     <Value>Trinity</Value>
   </Tag>
   <Tag>
     <Key>User</Key>
     <Value>Jones</Value>
   </Tag>
  </TagSet>
</Tagging>
```

The following table provides a list of the allowed XML tags and their characteristics. Note that values for Key and Value are case dependent. For example, project=Trinity and PROJECT=Trinity are two distinct tags.

Tagging element	Description
TagSet	A tag set is a container for all tags assigned to an Amazon RDS resource. There can be only one tag set per resource. You work with a TagSet only through the AWS DMS API.
Tag	A tag is a user-defined key-value pair. There can be from 1 to 10 tags in a tag set.
Key	A key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "dms:" or "aws:". The string might only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"). Keys must be unique to a tag set. For example, you cannot have a key-pair in a tag set with the key the same but with different values, such as project/Trinity and project/Xanadu.
Value	A value is the optional value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "dms:" or "aws:". The string might only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"). 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.

Constructing an Amazon Resource Name (ARN) for Use with AWS Database Migration Service

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

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

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

In this syntax:

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

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

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

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

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

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

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

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

Resource Type	Sample ARN
Replication instance	`arn:aws:dms:us-east-1:123456789012:rep:QLXQZ64MH7CXF4QCQMGRVYVXAI`
Endpoint	`arn:aws:dms:us-east-1:123456789012:endpoint:D3HMZ2IGUCGFF3NTAXUXGF6S5A`
Migration task	`arn:aws:dms:us-east-1:123456789012:task:2PVREMWNPGYJCVU2IBPTOYTIV4`

DDL Statements Supported by AWS Database Migration Service

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

Supported DDL statements include the following:

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

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

LOB Support for Source Databases

For This Option	Do This
Include LOB columns in replication	Large objects, (LOBs) can sometimes be difficult to migrate between systems. AWS DMS offers a number of options to help with the tuning of LOB columns. To see which and when datatypes are considered LOBS by AWS DMS, see the AWS DMS documentation. **Don't include LOB columns** - When you migrate data from one database to another, you might take the opportunity to rethink how your LOBs are stored, especially for heterogeneous migrations. If you want to do so, there's no need to migrate the LOB data. **Full LOB mode** - In **full LOB mode** AWS DMS migrates all LOBs from source to target regardless of size. In this configuration, AWS DMS has no information about the maximum size of LOBs to expect. Thus, LOBs are migrated one at a time, piece by piece. Full LOB mode can be quite slow. **Limited LOB mode** - In **limited LOB mode**, you set a maximum size LOB that AWS DMS should accept. Doing so allows AWS DMS to pre-allocate memory and load the LOB data in bulk. LOBs that exceed the maximum LOB size are truncated and a warning is issued to the log file. In **limited LOB mode** you get significant performance gains over **full LOB mode**. We recommend that you use **limited LOB mode** whenever possible. **Note** With Oracle, LOBs are treated as VARCHAR data types whenever possible. This approach means AWS DMS fetches them from the database in bulk, which is significantly faster than other methods. The maximum size of a VARCHAR in Oracle is 64K, therefore a limited LOB size of less than 64K is optimal when Oracle is your source database.
Max LOB size (K)	When a task is configured to run in **limited LOB mode**, this option determines the maximum size LOB that AWS DMS accepts. Any LOBs that are larger than this value will be truncated to this value.
LOB chunk size (K)	When a task is configured to use **full LOB mode**, AWS DMS retrieves LOBs in pieces. This option determines the size of each piece. When setting this option, pay particular attention to the maximum packet size allowed by your network configuration. If the LOB chunk size exceeds your maximum allowed packet size, you might see disconnect errors.

AWS DMS Support for AWS CloudFormation

You can provision AWS Database Migration Service resources using AWS CloudFormation. AWS CloudFormation is a service that helps you model and set up your Amazon Web Services 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.

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

- AWS::DMS::Certificate
- AWS::DMS::Endpoint
- AWS::DMS::EventSubscription
- AWS::DMS::ReplicationInstance
- AWS::DMS::ReplicationSubnetGroup
- AWS::DMS::ReplicationTask

Setting Up

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

1. Sign Up for AWS (p. 20)
2. Create an IAM User (p. 20)
3. Determine Requirements (p. 22)

Sign Up for AWS

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

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

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions.

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

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

Create an IAM User

Services in AWS, such as AWS DMS, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The console requires your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or and grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

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

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

1. 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**, and then choose **Add user**.
3. For **User name**, type a user name, such as `Administrator`. The name can consist of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). The name is not case sensitive and can be a maximum of 64 characters in length.
4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to select a new password the next time the user signs in.
5. Choose **Next: Permissions**.
6. On the **Set permissions for user** page, choose **Add user to group**.
7. Choose **Create group**.
8. In the **Create group** dialog box, type the name for the new group. The name can consist of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). The name is not case sensitive and can be a maximum of 128 characters in length.
9. For **Filter**, choose **Job function**.
10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
12. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

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

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

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

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

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

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

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

Determine Requirements

A database migration requires thorough testing and a sound backup strategy. To successfully migrate a database using AWS DMS, you must have expertise, time, and knowledge about your source database, your network, the AWS network, your requirements, and your target database schema, as described following.

Available Regions

Ensure that AWS DMS is available in the region that you need. AWS DMS is currently available in the following regions:

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

Expertise

A database migration requires expertise in several areas.

- You should have a thorough knowledge of the database engine you are migrating from and the database engine you are migrating to.
- You should understand both the network you are migrating from and how that network connects to AWS.
- You should have a thorough knowledge of the AWS service you are migrating to and the AWS Identity and Access Management (IAM) service.
- In most cases, it helps if you have an understanding of software architecture.

Time

Migration projects can take from two weeks to several months to complete.

- A successful migration can require several iterations.

- The migration process can take more time than you anticipate.
- Do you have a hard date on when your database migration must be completed?
- Migration planning can often take longer than the migration itself.

Knowledge of your source database

The size of your database and the data types it contains can have a dramatic impact on your migration.

- How many schemas and tables does your database contain?
- Does your database have any very large tables (more than 5 GB in size)?
- Do you know what the transaction boundaries look like?
- Does your database have data types that AWS DMS does not support?
- Do you have LOBs in your tables? If so, how large are the LOBs?
- Do your tables with LOBs have primary keys?
- How busy is your source database?
- What kind of users, roles, and permissions do you have on the source database?
- When was the last time you vacuumed or compacted your source database?

Knowledge of your network and the AWS network

You must connect the network that the source database uses to AWS.

- How will your database access the AWS network?
- Which Amazon Virtual Private Cloud (Amazon VPC) will you use?
- Which Amazon Elastic Compute Cloud (Amazon EC2) security group will you use?
- How much bandwidth will you need to move all your data?

An understanding of your requirements

The following questions make up much of your migration planning:

- How much downtime can you afford?
- Do you need the source database to be available after migration?
- Do you know why you preferred one target database engine over another database engine?
- What are your high availability requirements?
- Does all the data needs to be migrated?
- Does all the data need to be migrated to the same database?
- Do you understand the benefits of using Amazon RDS (automated backups, high availability, and so on)?
- Do you understand the limits of using Amazon RDS (storage size, admin user, and so on)?
- What happens to your application during the migration process?
- What is your contingency plan if the migration is unsuccessful?

Knowledge of your target database schema

AWS DMS creates only tables and primary keys in the target database. You must recreate any other database requirements.

- You can use the AWS Schema Conversion Tool (AWS SCT) to migrate a database schema. It works best when migrating from one database engine to a different database engine. For more information on the AWS Schema Conversion Tool, see AWS Schema Conversion Tool .
- The AWS SCT does not support schema conversions from and to the same database engine type. If you need to convert a schema when going to the same database engine, use the database engine's native tools for the conversion.
- The AWS SCT does not currently support orchestration.
- Postpone any schema changes until after the migration.

Getting Started

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

For information on the cost of database migration using AWS Database Migration Service, see the AWS Database Migration Service pricing page.

Topics

Start a Database Migration with AWS Database Migration Service

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

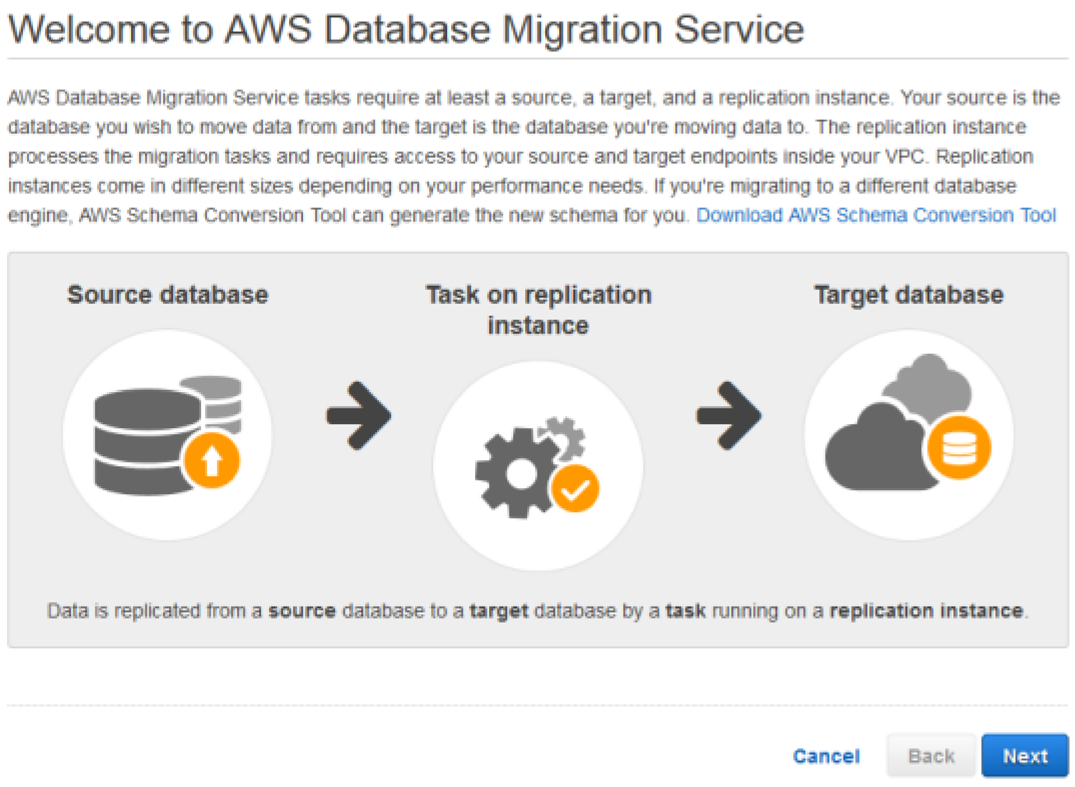
To use the wizard, select **Getting started** for from the navigation pane on the AWS DMS console. You can use the wizard to help create your first data migration. Following the wizard process, you allocate a replication instance that performs all the processes for the migration, specify a source and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

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

- Complete the tasks outlined in Setting Up (p. 20)
- Allocate a replication instance that performs all the processes for the migration
- Specify a source and a target database endpoint
- Create a task or set of tasks to define what tables and replication processes you want to use

Step 1: Welcome

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


Welcome to AWS Database Migration Service

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

Data is replicated from a **source** database to a **target** database by a **task** running on a **replication instance**.

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

- Choose **Next**.

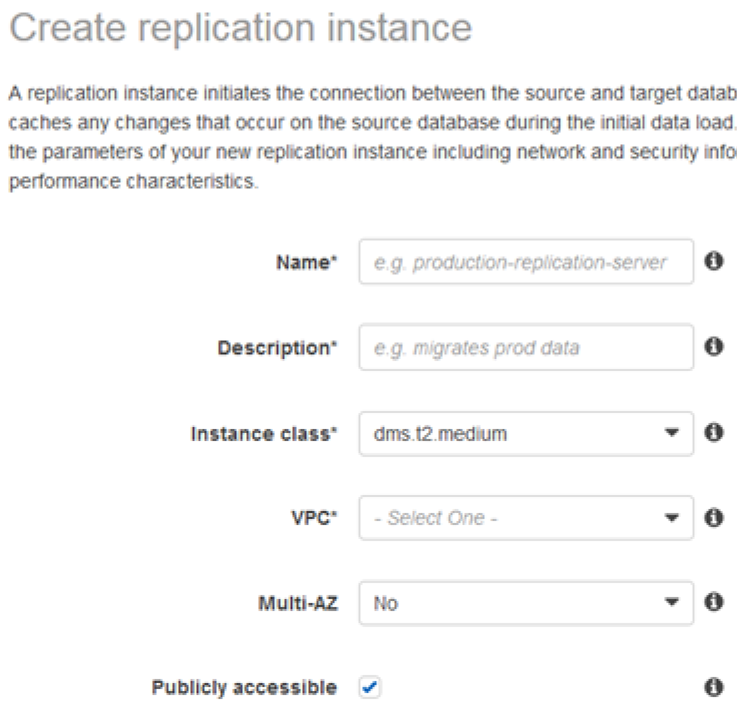
Step 2: Create a Replication Instance

Your first task in migrating a database is to create a replication instance that has sufficient storage and processing power to perform the tasks you assign and migrate data from your source database to the target database. The required size of this instance varies depending on the amount of data you need to migrate and the tasks that you need the instance to perform. For more information about replication instances, see Replication Instances for AWS Database Migration Service (p. 6).

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

To create a replication instance by using the AWS console

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



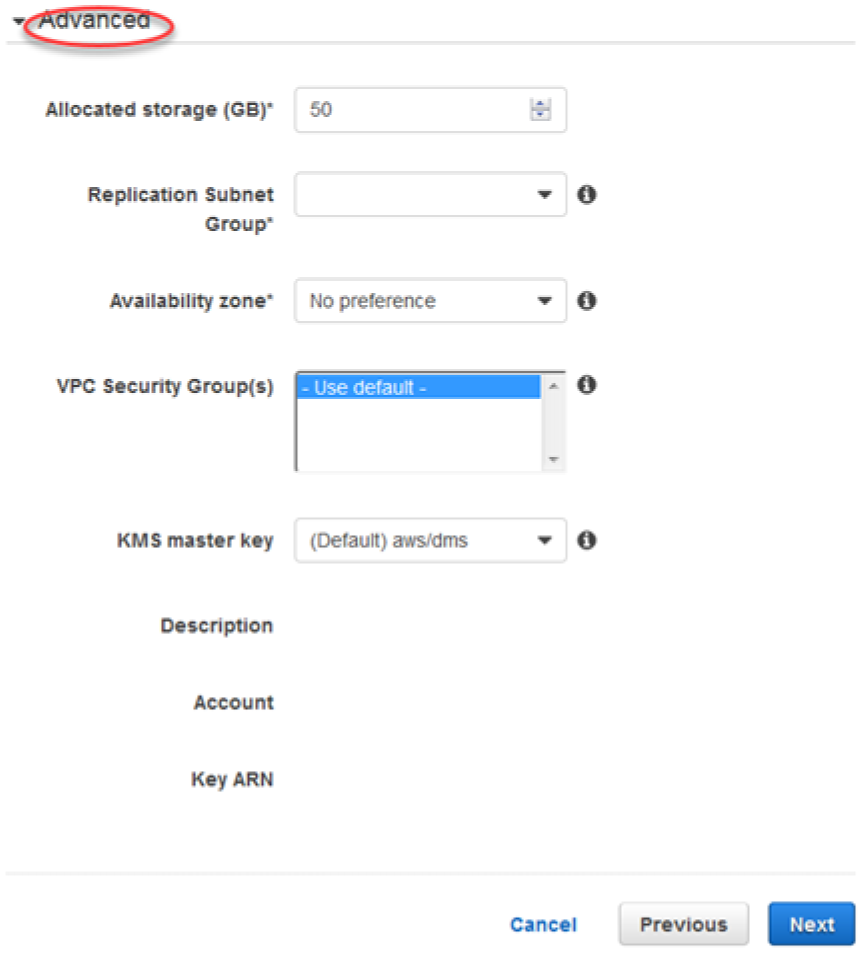
A replication instance initiates the connection between the source and target databases, transfers the data, and caches any changes that occur on the source database during the initial data load. Use the fields below to configure the parameters of your new replication instance including network and security information, encryption details, and performance characteristics.

Name*	e.g. production-replication-server
Description*	e.g. migrates prod data
Instance class*	dms.t2.medium
VPC*	- Select One -
Multi-AZ	No
Publicly accessible	✓

For This Option	Do This
Name	Type a name for the replication instance that contains from 8 to 16 printable ASCII characters (excluding /,", and @). The name should be unique for your account for the region you selected. You can choose to add some intelligence to the name, such as including the region and task you are performing, for example **`west2-mysql2mysql-instance1`**.
Description	Type a brief description of the replication instance.
Instance class	Choose an instance class with the configuration you need for your migration. Keep in mind that the instance must have enough storage, network, and processing power to successfully complete your migration. For more information on how to determine which instance class is best for your migration, see Replication Instances for AWS Database Migration Service (p. 6).

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

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



Advanced

Allocated storage (GB)*	50
Replication Subnet Group*	
Availability zone*	No preference
VPC Security Group(s)	- Use default -
KMS master key	(Default) aws/dms
Description	
Account	
Key ARN	

3. Specify the additional settings. The following table describes the settings.

For This Option	Do This
Allocated storage (GB)	Storage is primarily consumed by log files and cached transactions. For caches transactions, storage is used only when the cached transactions need to be written to disk. Therefore, AWS DMS doesn't use a significant amount of storage.Some exceptions include the following: • Very large tables that incur a significant transaction load. Loading a large table can take some time, so cached transactions are more likely to be written to disk during a large table load. • Tasks that are configured to pause prior to loading cached transactions. In this case, all transactions are cached until the full load completes for all tables. With this configuration, a fair amount of storage might be consumed by cached transactions. • Tasks configured with tables being loaded into Amazon Redshift. However, this configuration isn't an issue when Aurora 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.
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 (p. 12).
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, select the VPC security group that provides access to the DB instance where the database resides.
KMS master key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose **(Default) aws/dms**, the default AWS Key Management Service (AWS KMS) key associated with your account and region is used. For more information on using the encryption key, see Setting an Encryption Key and Specifying KMS Permissions (p. 55).

4. Choose **Next**.

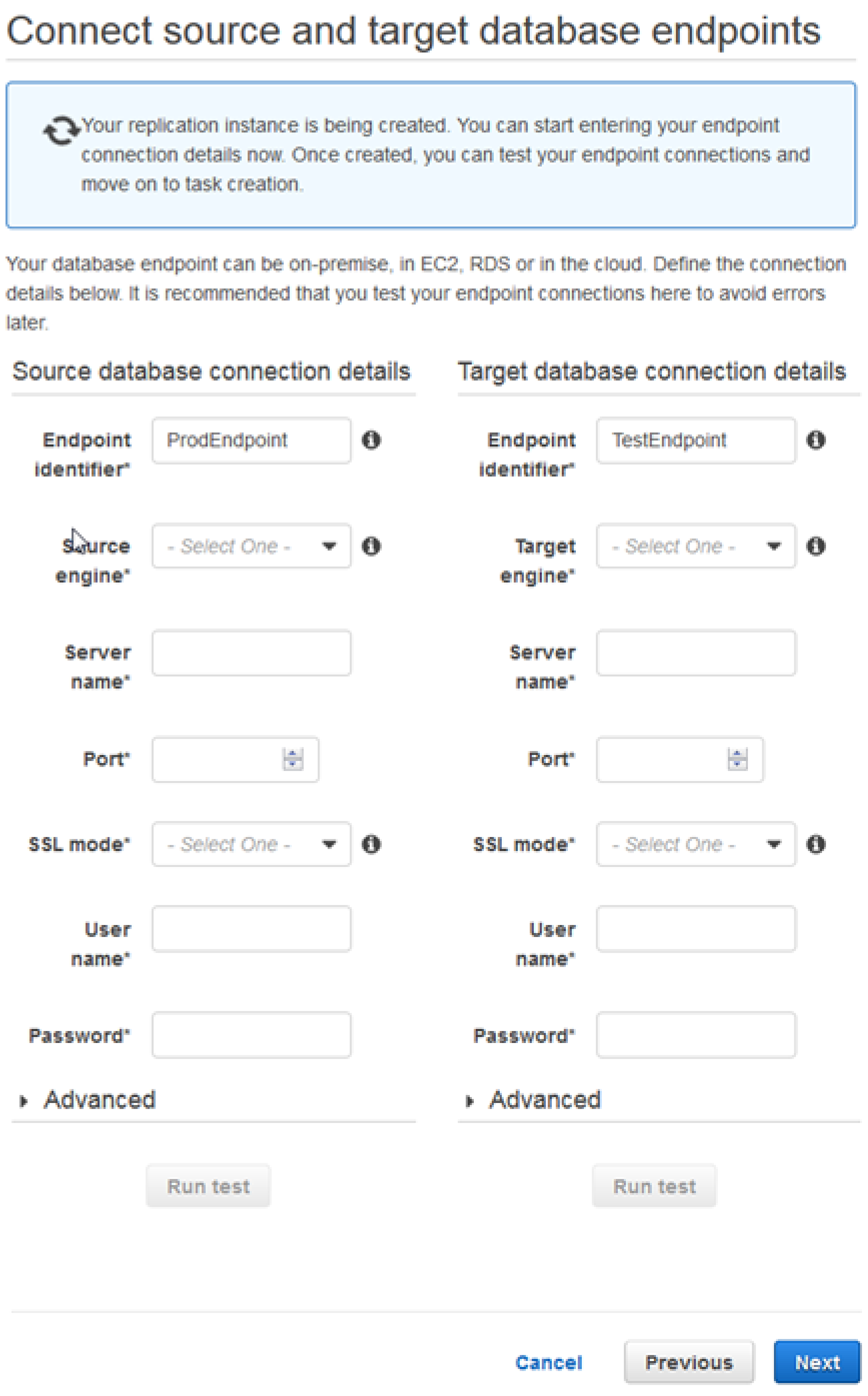
Step 3: Specify Database Endpoints

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

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

To specify source or target database endpoints using the AWS console

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



Connect source and target database endpoints

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

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

Source database connection details

Endpoint identifier*	ProdEndpoint
Source engine*	- Select One -
Server name*	
Port*	
SSL mode*	- Select One -
User name*	
Password*	

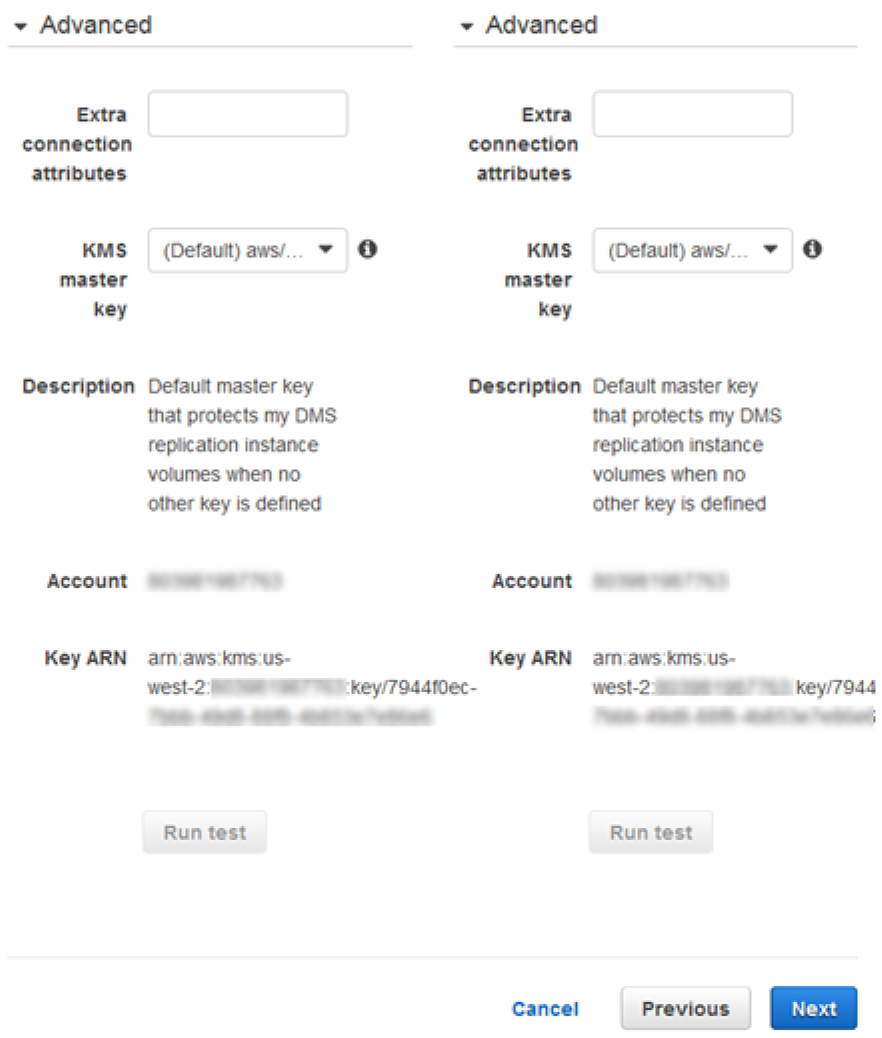
Target database connection details

Endpoint identifier*	TestEndpoint
Target engine*	- Select One -
Server name*	
Port*	
SSL mode*	- Select One -
User name*	
Password*	

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

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

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



Advanced

Extra connection attributes	
KMS master key	(Default) aws/...
Description	Default master key that protects my DMS replication instance volumes when no other key is defined
Account	[redacted]
Key ARN	arn:aws:kms:us-west-2:[redacted]:key/7944f0ec-[redacted]

Advanced

Extra connection attributes	
KMS master key	(Default) aws/...
Description	Default master key that protects my DMS replication instance volumes when no other key is defined
Account	[redacted]
Key ARN	arn:aws:kms:us-west-2:[redacted]:key/7944[redacted]

For This Option	Do This
Extra connection attributes	Type any additional connection parameters here. For more information about extra connection attributes, see Using Extra Connection Attributes with AWS Database Migration Service (p. 199).
KMS master key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose **(Default) aws/dms**, the default AWS Key Management Service (AWS KMS) key associated with your account and region is used. For more information on using the encryption key, see Setting an Encryption Key and Specifying KMS Permissions (p. 55).

Step 4: Create a Task

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

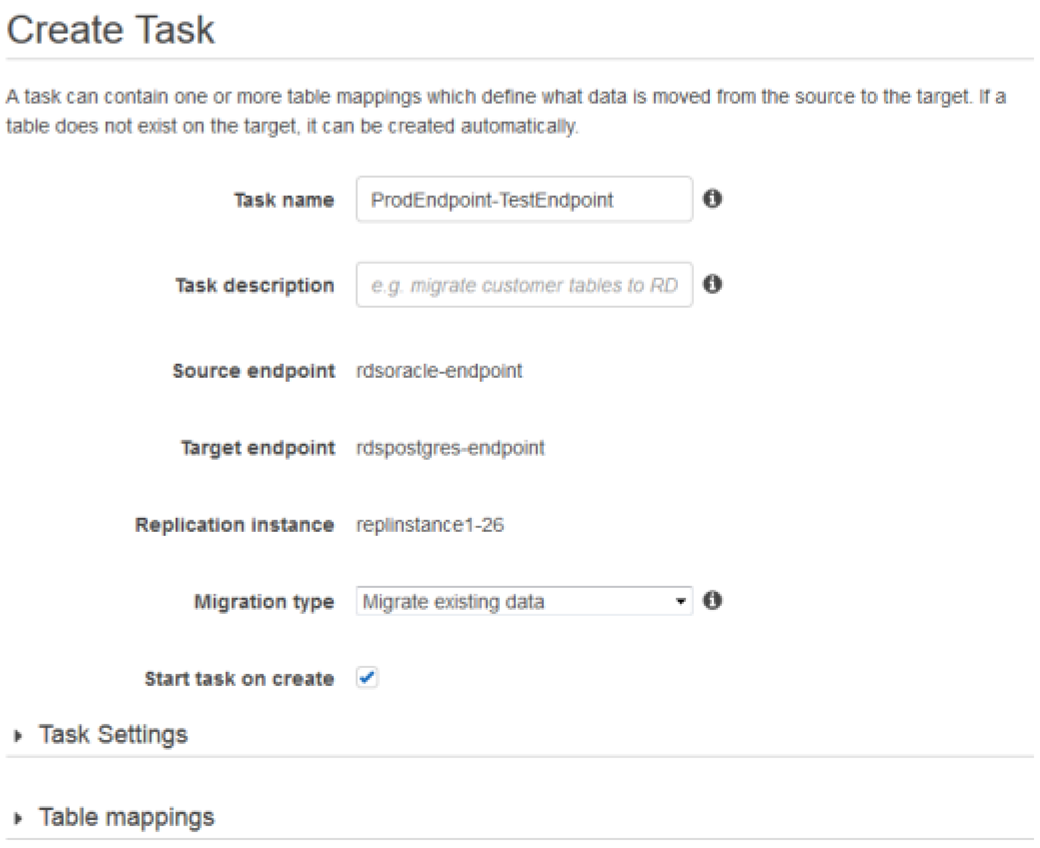
Using AWS DMS, you can specify precise mapping of your data between the source and the target database. Before you specify your mapping, make sure you review the documentation section on data type mapping for your source and your target database.

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

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

To create a migration task

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



Create Task

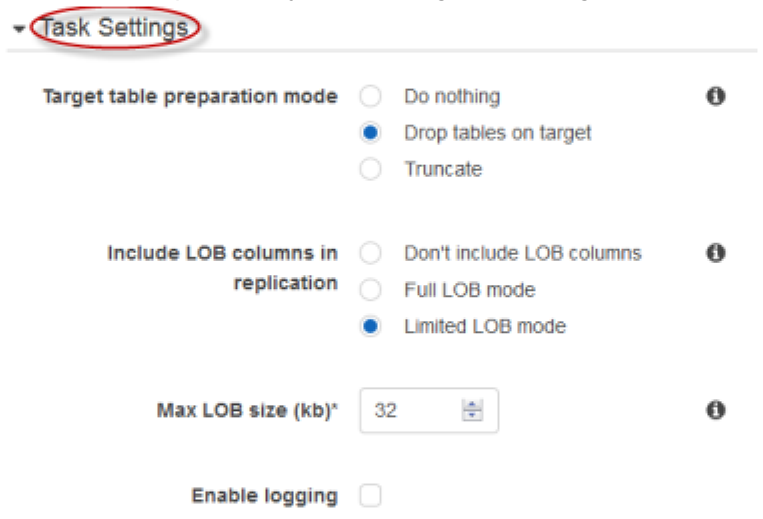
A task can contain one or more table mappings which define what data is moved from the source to the target. If a table does not exist on the target, it can be created automatically.

Task name	ProdEndpoint-TestEndpoint
Task description	e.g. migrate customer tables to RD
Source endpoint	rdsoracle-endpoint
Target endpoint	rdspostgres-endpoint
Replication instance	replinstance1-26
Migration type	Migrate existing data
Start task on create	✓

For This Option	Do This
Task name	Type a name for the task.

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

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



Target table preparation mode: Do nothing [✓ Drop tables on target] Truncate

Include LOB columns in replication: Don't include LOB columns, Full LOB mode [✓ Limited LOB mode]

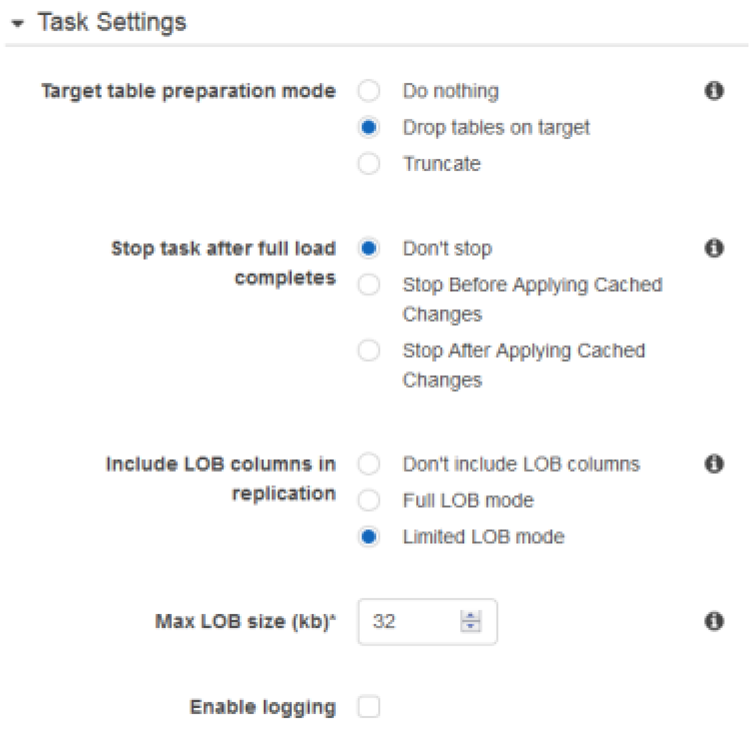
Max LOB size (kb)*: 32

Enable logging: ☐

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

For This Option	Do This
	Limited LOB mode - Truncate LOBs to 'Max LOB Size' This method is faster than using Full LOB Mode. For more information about LOB support in AWS DMS, see LOB Support for Source Databases (p. 17)
Max LOB size (kb)	In **Limited LOB Mode**, LOB columns which exceed the setting of **Max LOB Size** will be truncated to the specified Max LOB Size.
Enable logging	Enables logging by Amazon CloudWatch.

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


Task Settings

Target table preparation mode: Do nothing [✓ Drop tables on target] Truncate

Stop task after full load completes: [✓ Don't stop] Stop Before Applying Cached Changes Stop After Applying Cached Changes

Include LOB columns in replication: Don't include LOB columns Full LOB mode [✓ Limited LOB mode]

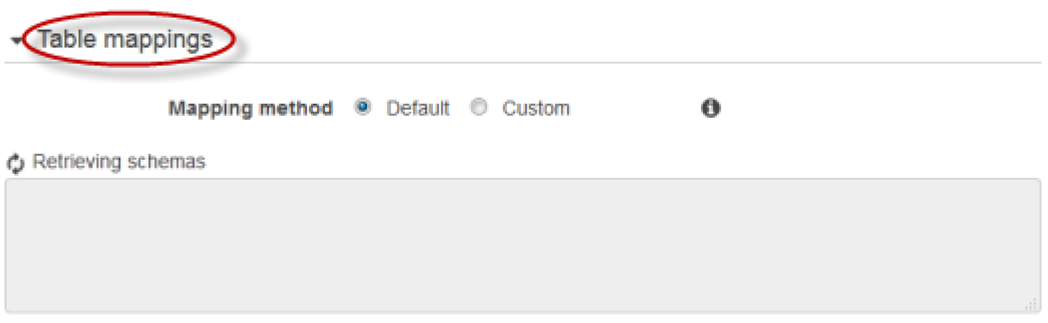
Max LOB size (kb)*: 32

Enable logging: ☐

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

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

3. Choose the **Table mappings** tab, shown following, to set values for schema mapping and the mapping method. If you choose **Custom**, you can specify the target schema and table values. For more information about table mapping, see Using Table Mapping with an Task to Select and Filter Data (p. 134).



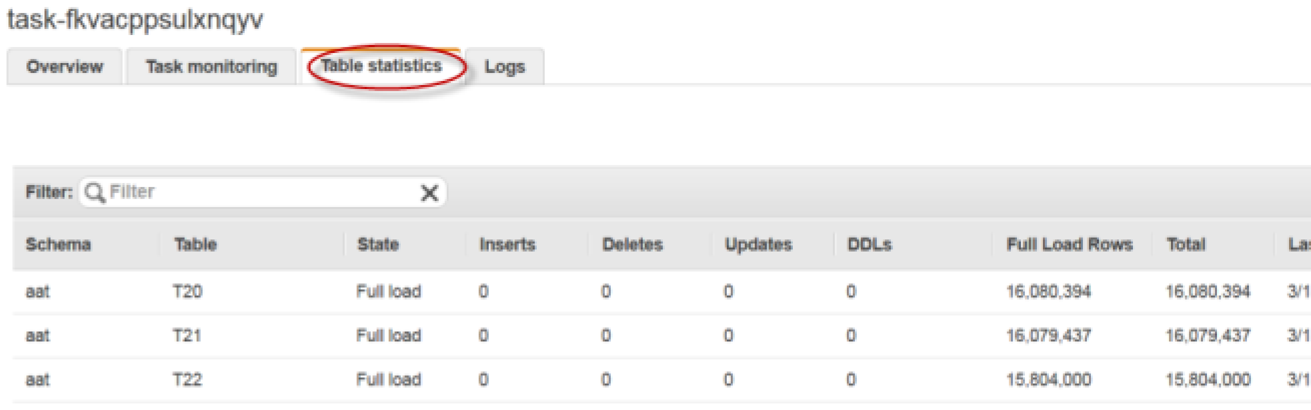
Mapping method [✓ Default] Custom

Retrieving schemas

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

Monitor Your Task

If you select **Start task on create** when you create a task, your task begins immediately to migrate your data when you choose **Create task**. You can view statistics and monitoring information for your task by choosing the running task from the AWS Management Console. The following screenshot shows the table statistics of a database migration. For more information about monitoring, see Monitoring AWS Database Migration Service Tasks (p. 149)



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

Best Practices

To use AWS Database Migration Service (AWS DMS) most effectively, see this section's recommendations on the most efficient way to migrate your data.

Topics

Improving the Performance of an AWS Database Migration Service Migration

A number of factors affect the performance of your AWS DMS migration:

- Resource availability on the source
- The available network throughput
- The resource capacity of the replication server
- The ability of the target to ingest changes
- The type and distribution of source data
- The number of objects to be migrated

In our tests, we've migrated a terabyte of data in approximately 12 to 13 hours under ideal conditions. These ideal conditions included using source databases running on Amazon Elastic Compute Cloud (Amazon EC2) and in Amazon Relational Database Service (Amazon RDS) with target databases in Amazon RDS. Our source databases contained a representative amount of relatively evenly distributed data with a few large tables containing up to 250 GB of data.

Your migration's performance can be limited by one or more bottlenecks long the way. The following list shows a few things you can do to increase performance:

Load 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, you'll want to reduce this number.

Remove bottlenecks on the target

During the migration, try to remove any processes that might compete with each other for write resources on your target database. As part of this process, disable unnecessary triggers, validation, and secondary indexes. When migrating to an Amazon RDS database, it’s a good idea to disable backups and Multi-AZ on the target until you’re ready to cut-over. Similarly, when migrating to non-Amazon RDS systems, disabling any logging on the target until cut over is usually a good idea.

Use multiple tasks

Sometimes using multiple tasks for a single migration can improve performance. If you have sets of tables that don’t participate in common transactions, you might be able to divide your migration into multiple tasks. Transactional consistency is maintained within a task, so it’s important that tables in separate tasks don't participate in common transactions. Additionally, each task independently reads the transaction stream, so be careful not to put too much stress on the source system.

Improving LOB performance

For information about improving LOB migration, see Migrating Large Binary Objects (LOBs) (p. 41).

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. Note that using the *batch optimized apply* option almost certainly violates any referential integrity constraints, so you should disable these during the migration process and enable them again as part of the cut-over process.

Determining the Optimum Size for a Replication Instance

Determining the correct size of your replication instance depends on several factors. The following information can help you understand the migration process and how memory and storage are used.

Tables are loaded individually; by default, eight tables are loaded at a time. While each table is loaded, the transactions for that table are cached in memory. After the available memory is used, transactions are cached to disk. When the table for those transactions is loaded, the transactions and any further transactions on that table are immediately applied to the table.

When all tables have been loaded and all outstanding cached transactions for the individual tables have been applied by AWS DMS, the source and target tables will be in sync. At this point, AWS DMS will apply transactions in a way that maintains transactional consistency. (As you can see, tables will be out of sync during the full load and while cached transactions for individual tables are being applied.)

From the preceding explanation, you can see that relatively little disk space is required to hold cached transactions. The amount of disk space used for a given migration will depend on the following:

- Table size – Large tables take longer to load and so transactions on those tables must be cached until the table is loaded. Once a table is loaded, these cached transactions are applied and are no longer held on disk.
- Data manipulation language (DML) activity – A busy database generates more transactions. These transactions must be cached until the table is loaded. Remember, though, that transactions to an individual table are applied as soon as possible after the table is loaded, until all tables are loaded. At that point, AWS DMS applies all the transactions.
- Transaction size – Data generated by large transactions must be cached. If a table accumulates 10 GB of transactions during the full load process, those transactions will need to be cached until the full load is complete.
- Total size of the migration – Large migrations take longer and the log files that are generated are large.
- Number of tasks – The more tasks, the more caching will likely be required, and the more log files will be generated.

Anecdotal evidence shows that log files consume the majority of space required by AWS DMS. The default storage configurations are usually sufficient. Replication instances that run several tasks might require more disk space. Additionally, if your database includes large and active tables, you may need to account for transactions that are cached to disk during the full load. For example, if your load will take 24 hours and you produce 2GB of transactions per hour, you may want to ensure you have 48GB of space to accommodate cached transactions.

Reducing Load on Your Source Database

During a migration, AWS DMS performs a full table scan of the source table for each table processed in parallel. Additionally, each task will periodically query the source for change information. To perform change processing, you might be required to increase the amount of data written to your databases change log. If you find you are overburdening your source database you can reduce the number of tasks and/or tables per task for your migration. If you prefer not to add load to your source, you might consider performing the migration from a read copy of your source system. However, using a read copy does increase the replication lag.

Using the Task Log to Troubleshoot Migration Issues

At times DMS may encounter issues (warnings or errors) which are only currently visible when viewing the task log. In particular, data truncation issues or row rejections due to foreign key violations are currently only visible via the task log. Therefore, it is important to review the task log when migrating a database.

Schema Conversion

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

Migrating Large Binary Objects (LOBs)

Migration of LOB data is done in two phases. First, the row in the LOB column is created in the target table without the LOB data. Next, the row in the target table is updated with the LOB data. This means that during the migration, the LOB columns of a table must be NULLABLE on the target database. If AWS DMS creates the target tables, it sets LOB columns to NULLABLE, even if they are NOT NULLABLE on the source table. If you create the target tables using some other mechanism, such as Import/Export, the LOB columns must be NULLABLE.

Replication tasks, by default, are set to run in **Full LOB** support mode. While this setting moves all of the LOBS in your tables, the process is also slow. To increase the speed with which your migration task runs, you should create a new task and set the task to use “Limited Size LOB” mode. When you choose this mode you need to ensure that the setting of the MAX LOB parameter is correct. This parameter should be set to the largest LOBS size for all of your tables.

Whenever possible, use the **limited LOB mode** parameter for best performance. If you have a table that contains a few large LOBs and mostly smaller LOBs, consider breaking up the table before migration and consolidating the table fragments as part of migration.

AWS Database Migration Service 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 endpoints have limited LOB support. You cannot use an unlimited LOB size for these target endpoints.

- Amazon Redshift

For endpoints that have full LOB support, you can also set a size limit for LOB data types.

Ongoing Replication

AWS DMS provides comprehensive ongoing replication of data, although it replicates only a limited amount of data definition language (DDL). AWS DMS doesn't propagate items such as indexes, users, privileges, stored procedures, and other database changes not directly related to table data.

If you want to use ongoing replication, you must enable the **Multi-AZ** option on your replication instance. The **Multi-AZ** option provides high availability and failover support for the replication instance.

Changing the User/Schema for an Oracle Target

When using Oracle as a target, we assume the data should be migrated into the schema/user which is used for the target connection. If you want to migrate data to a different schema, you'll need to use a schema transformation to do so. For example, if my target endpoint connects to the user RDSMASTER and you wish to migrate from the user PERFDATA to PERFDATA, you'll need to create a transformation as follows:

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

For more information about transformations, see Selection and Transformation Table Mapping using JSON (p. 139).

Security

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

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

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

Topics

IAM Permissions Needed to Use AWS DMS

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

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dms:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:Get*",
                "cloudwatch:List*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:FilterLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:Describe*",
                "redshift:ModifyClusterIamRoles"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

The AWS DMS console creates several roles that are automatically attached to your AWS account when you use the AWS DMS console. If you use the AWS Command Line Interface (AWS CLI) or the AWS DMS API for your migration, you need to add these roles to your account. For more information on adding these roles, see Creating the IAM Roles to Use With the AWS CLI and AWS DMS API (p. 45).

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

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

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

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

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

The information returned from the command is as follows.

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

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

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

The information returned from the command is as follows.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fine-grained Access Control Using Resource Names and Tags

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

Using Resource Names to Control Access

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

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

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

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

```
$ aws dms delete-replication-instance
   --replication-instance-arn "arn:aws:dms:us-east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV"

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

$ aws dms modify-replication-instance
   --replication-instance-arn "arn:aws:dms:us-east-1:152683116:rep:DOH67ZTOXGLIXMIHKITV"

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

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

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

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

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

```
$ aws dms delete-endpoint
   --endpoint-arn "arn:aws:dms:us-east-1:152683116:endpoint:D6E37YBXTNHOA6XRQSZCUGX"

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

```
$ aws dms modify-endpoint
   --endpoint-arn "arn:aws:dms:us-east-1:152683116:endpoint:D6E37YBXTNHOA6XRQSZCUGX"

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

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dms:*"
            ],
            "Effect": "Deny",
            "Resource": "arn:aws:dms:us-east-1:152683116:task:UO3YR4N47DXH3ATT4YMWOIT"
        }
    ]
}
```

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

```
$ aws dms delete-replication-task
   --replication-task-arn "arn:aws:dms:us-east-1:152683116:task:UO3YR4N47DXH3ATT4YMWOIT"

A client error (AccessDeniedException) occurred when calling the DeleteReplicationTask
 operation:
User: arn:aws:iam::152683116:user/dmstestusr is not authorized to perform:
 dms:DeleteReplicationTask
on resource: arn:aws:dms:us-east-1:152683116:task:UO3YR4N47DXH3ATT4YMWOIT
```

Using Tags to Control Access

AWS DMS defines a set of common key/value pairs that are available for use in customer defined policies without any additional tagging requirements. For more information about tagging AWS DMS resources, see Tagging AWS Database Migration Service Resources (p. 13).

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

- aws:CurrentTime – Represents the request date and time, allowing the restriction of access based on temporal criteria.
- aws:EpochTime – This tag is similar to the aws:CurrentTime tag above, except that the current time is represented as the number of seconds elapsed since the Unix Epoch.
- aws:MultiFactorAuthPresent – This is a boolean tag that indicates whether or not the request was signed via multi-factor authentication.
- aws:MultiFactorAuthAge – Provides access to the age of the multi-factor authentication token (in seconds).
- aws:principaltype - Provides access to the type of principal (user, account, federated user, etc.) for the current request.
- aws:SourceIp - Represents the source ip address for the user issuing the request.

- aws:UserAgent – Provides information about the client application requesting a resource.
- aws:userid – Provides access to the ID of the user issuing the request.
- aws:username – Provides access to the name of the user issuing the request.
- dms:InstanceClass – Provides access to the compute size of the replication instance host(s).
- dms:StorageSize - Provides access to the storage volume size (in GB).

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

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

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

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

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

```
stage production
```

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

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

```

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

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

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$ aws dms delete-replication-task
   --replication-task-arn "arn:aws:dms:us-east-1:152683116:task:RB7N24J2XBUPS3RFABZTG3"

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

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

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

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

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

Setting an Encryption Key and Specifying KMS Permissions

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

The default KMS key (`aws/dms`) is created when you first launch a replication instance and you have not selected a custom KMS master key from the **Advanced** section of the **Create Replication Instance** page. If you use the default KMS key, the only permissions you need to grant to the IAM user account you are using for migration are `kms:ListAliases` and `kms:DescribeKey`. For more information about using the default KMS key, see IAM Permissions Needed to Use AWS DMS (p. 43).

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

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

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

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

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

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

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

Network Security for AWS Database Migration Service

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

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

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

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

- Configuration with All Database Migration Components in One VPC (p. 9) — The security group used by the endpoints must allow ingress on the database port from the replication instance. Ensure that the security group used by the replication instance has ingress to the endpoints, or you can create a rule in the security group used by the endpoints that allows the private IP address of the replication instance access.
- Configuration with Two VPCs (p. 10) — The security group used by the replication instance must have a rule for the VPC range and the DB port on the database.

- Configuration for a Network to a VPC Using AWS Direct Connect or a VPN (p. 10) — a VPN tunnel allowing traffic to tunnel from the VPC into an on- premises VPN. In this configuration, the VPC includes a routing rule that sends traffic destined for a specific IP address or range to a host that can bridge traffic from the VPC into the on-premises VPN. If this case, the NAT host includes its own Security Group settings that must allow traffic from the Replication Instance's private IP address or security group into the NAT instance.
- Configuration for a Network to a VPC Using the Internet (p. 11) — The VPC security group must include routing rules that send traffic not destined for the VPC to the Internet gateway. In this configuration, the connection to the endpoint appears to come from the public IP address on the replication instance.
- Configuration with an Amazon RDS DB instance not in a VPC to a DB instance in a VPC Using ClassicLink (p. 11) — When the source or target Amazon RDS DB instance is not in a VPC and does not share a security group with the VPC where the replication instance is located, you can setup a proxy server and use ClassicLink to connect the source and target databases.
- **Source endpoint is outside the VPC used by the replication instance and uses a NAT gateway** — You can configure a network address translation (NAT) gateway using a single Elastic IP Address bound to a single Elastic Network Interface, which then receives a NAT identifier (nat-#####). If the VPC includes a default route to that NAT Gateway instead of the Internet Gateway, the replication instance will instead appear to contact the Database Endpoint using the public IP address of the Internet Gateway. In this case, the ingress to the Database Endpoint outside the VPC needs to allow ingress from the NAT address instead of the Replication Instance's public IP Address.

Using SSL With AWS Database Migration Service

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

Not all databases use SSL in the same way. Amazon Aurora uses the server name, the endpoint of the primary instance in the cluster, as the endpoint for SSL. An Amazon Redshift endpoint already uses an SSL connection and does not require an SSL connection set up by AWS DMS. An Oracle endpoint requires additional steps; for more information, see SSL Support for an Oracle Endpoint (p. 60).

Topics

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

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

- **none** – The connection is not encrypted. This option is not secure, but requires less overhead.
- **require** – The connection is encrypted using SSL (TLS) but no CA verification is made. This option is more secure, and requires more overhead.

- **verify-ca** – The connection is encrypted. This option is more secure, and requires more overhead. This option verifies the server certificate.
- **verify-full** – The connection is encrypted. This option is more secure, and requires more overhead. This option verifies the server certificate and verifies that the server hostname matches the hostname attribute for the certificate.

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

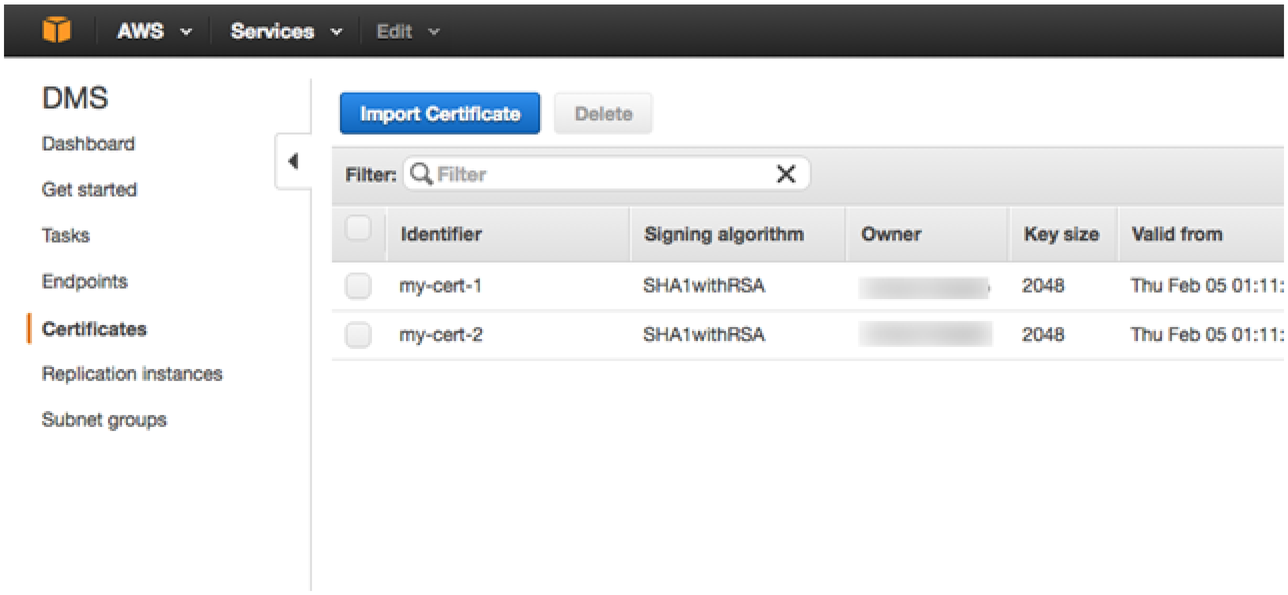
DB Engine	none	require	verify-ca	verify-full
MySQL/MariaDB/ Amazon Aurora	Default	Not supported	Supported	Supported
Microsoft SQL Server	Default	Supported	Not Supported	Supported
PostgreSQL	Default	Supported	Supported	Supported
Amazon Redshift	Default	SSL not enabled	SSL not enabled	SSL not enabled
Oracle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

Limitations on Using SSL with AWS Database Migration Service

- SSL connections to Amazon Redshift target endpoints are not supported. AWS DMS uses an S3 bucket to transfer data to the Redshift database. This transmission is encrypted by Amazon Redshift by default.
- SQL timeouts can occur when performing CDC tasks with SSL-enabled Oracle endpoints. If you have this issue, where CDC counters don't reflect the expected numbers, set the `MinimumTransactionSize` parameter from the `ChangeProcessingTuning` section of task settings to a lower value, starting with a value as low as 100. For more information about the `MinimumTransactionSize` parameter, see Change Processing Tuning Settings (p. 130).
- Certificates can only be imported in the .PEM and .SSO (Oracle wallet) formats.
- If your server SSL certificate is signed by an intermediate CA, make sure the entire certificate chain leading from the intermediate CA up to the root CA is imported as a single .PEM file.
- If you are using self-signed certificates on your server, choose **require** as your SSL mode. The **require** SSL mode implicitly trusts the server's SSL certificate and will not try to validate that the certificate was signed by a CA.

Managing Certificates

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



	Identifier	Signing algorithm	Owner	Key size	Valid from
	my-cert-1	SHA1withRSA	[redacted]	2048	Thu Feb 05 01:11:
	my-cert-2	SHA1withRSA	[redacted]	2048	Thu Feb 05 01:11:

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

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

To create an AWS DMS endpoint with SSL

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

   Note
   If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required for database migration, see IAM Permissions Needed to Use AWS DMS (p. 43).
2. In the navigation pane, choose **Certificates**.
3. Choose **Import Certificate**.
4. Upload the certificate you want to use for encrypting the connection to an endpoint.

   Note
   You can also upload a certificate using the AWS DMS console when you create or modify an endpoint by selecting **Add new CA certificate** on the **Create database endpoint** page.
5. Create an endpoint as described in Step 3: Specify Database Endpoints (p. 29)

To modify an existing AWS DMS endpoint to use SSL:

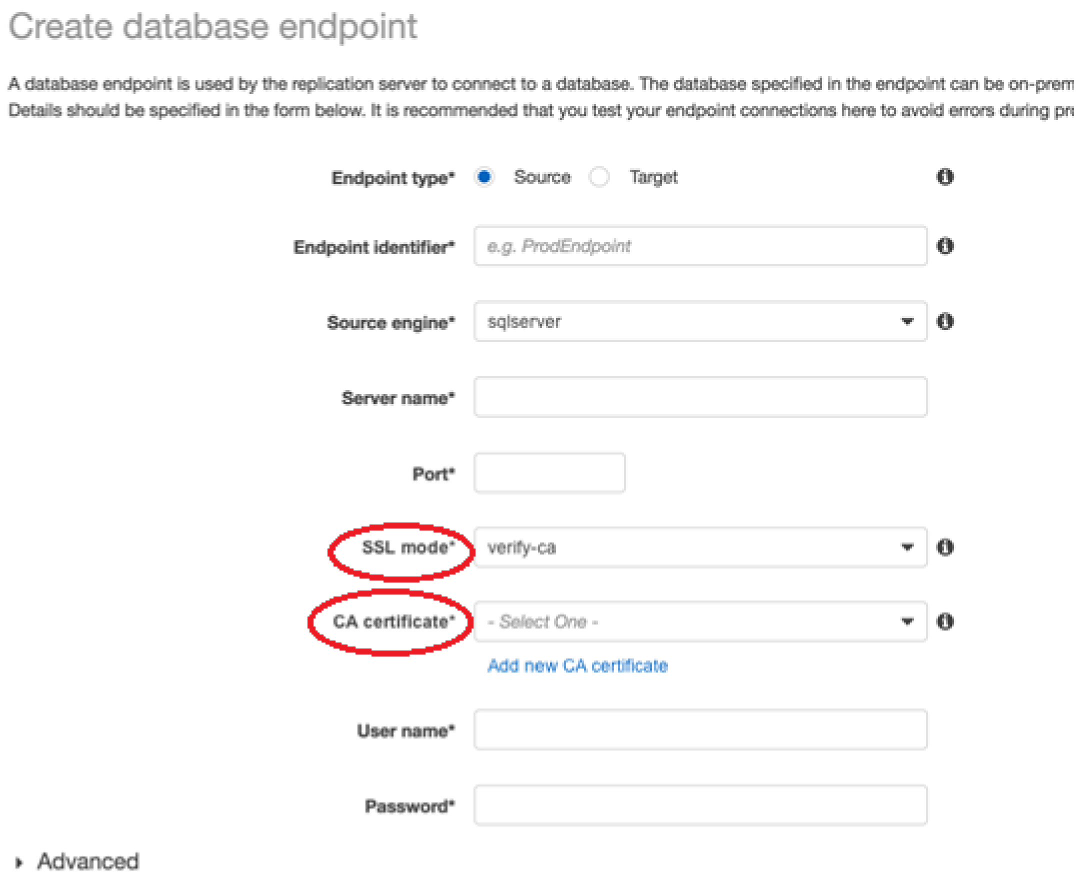
1. Sign in to the AWS Management Console and choose AWS Database Migration Service.

   Note
   If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required for database migration, see IAM Permissions Needed to Use AWS DMS (p. 43).
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 the **Create database endpoint** page.

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

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



Create database endpoint

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

Endpoint type*	[✓ Source] Target
Endpoint identifier*	e.g. ProdEndpoint
Source engine*	sqlserver
Server name*	
Port*	
SSL mode*	verify-ca
CA certificate*	- Select One -
	Add new CA certificate
User name*	
Password*	

Advanced

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

After you create your source and target endpoints, create a task that uses these endpoints. For more information on creating a task, see Step 4: Create a Task (p. 33).

SSL Support for an Oracle Endpoint

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

Topics

- Using an Existing Certificate for Oracle SSL (p. 61)

- Using a Self-Signed Certificate for Oracle SSL (p. 61)

Using an Existing Certificate for Oracle SSL

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

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

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

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

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

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

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

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

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

   ```
   prompt>orapki wallet create -wallet $ORACLE_HOME/ssl_wallet -auto_login_only

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

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

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

Using a Self-Signed Certificate for Oracle SSL

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

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

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

   ```
   mkdir <SELF_SIGNED_CERT_DIRECTORY>
   ```

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

   ```
   cd <SELF_SIGNED_CERT_DIRECTORY>
   ```

3. Create a root key.

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

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

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

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

   ```
   mkdir $ORACLE_HOME/self_signed_ssl_wallet
   ```

6. Create a new Oracle wallet.

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

7. Add the root certificate to the Oracle wallet.

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

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

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

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

   ```
   orapki wallet add -wallet $ORACLE_HOME/self_signed_ssl_wallet
        -dn "CN=`hostname`, OU=Sample Department, O=Sample Company,
        L=NYC, ST=NY, C=US" -keysize 1024 -pwd <password>
   ```

10. List the contents of the Oracle wallet. The list should include the CSR.

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

11. Export the CSR from the Oracle wallet.

   ```
   orapki wallet export -wallet $ORACLE_HOME/self_signed_ssl_wallet
        -dn "CN=`hostname`, OU=Sample Department, O=Sample Company,
        L=NYC, ST=NY, C=US" -request self-signed-oracle.csr -pwd <password>
   ```

12. Sign the CSR using the root certificate.

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

13. Add the Client certificate to the server wallet.

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

14. List the content of the Oracle wallet.

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

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

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

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

16. Stop the Oracle listener.

```
lsnrctl stop
```

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

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

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

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

18. 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>)
        )
)
```

19. Restart the Oracle listener.

```
lsnrctl start
```

20. Show the Oracle listener status.

```
lsnrctl status
```

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

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

22. Verify that you successfully connected using SSL.

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

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

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

```
cd <SELF_SIGNED_CERT_DIRECTORY>
```

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

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

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

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

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

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

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

Changing the Database Password

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

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

1. Sign in to the AWS Management Console and choose AWS DMS. Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see IAM Permissions Needed to Use AWS DMS (p. 43).
2. In the navigation pane, choose **Tasks**.
3. Choose the task that uses the endpoint you want to change the database password for, and then choose **Stop**.
4. While the task is stopped, you can change the password of the database for the endpoint using the native tools you use to work with the database.
5. Return to the DMS Management Console and choose **Endpoints** from the navigation pane.
6. Choose the endpoint for the database you changed the password for, and then choose **Modify**.
7. Type the new password in the **Password** box, and then choose **Modify**.
8. Choose **Tasks** from the navigation pane.
9. Choose the task that you stopped previously, and choose **Start/Resume**.
10. Choose either **Start** or **Resume**, depending on how you want to continue the task, and then choose **Start task**.

Limits for AWS Database Migration Service

This topic describes the resource limits and naming constraints for AWS Database Migration Service (AWS DMS).

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

Limits for AWS Database Migration Service

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

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

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

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

Resource	Default Limit
Tasks	200
Endpoints per instance	20

Sources for Data Migration

AWS Database Migration Service (AWS DMS) can use many of the most popular databases as a source for data replication. The source can be on an Amazon Elastic Compute Cloud (Amazon EC2) instance, an Amazon Relational Database Service (Amazon RDS) instance, or an on-premises database. The source databases include the following.

On-premises and Amazon EC2 instance databases

- Oracle versions 10.2 and later, 11g, and up to 12.1, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data source)
- PostgreSQL 9.3 and later
- SAP Adaptive Server Enterprise (ASE) 15.7 and later
- MongoDB versions 2.6.x and 3.x and later

Amazon RDS instance databases

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

Topics

Using an Oracle Database as a Source for AWS Database Migration Service

You can migrate data from one or many Oracle databases using AWS Database Migration Service (AWS DMS). With an Oracle database as a source, you can migrate data to either another Oracle database or one of the other supported databases.

AWS DMS supports as a migration source all Oracle database editions for versions 10.2 and later, 11g, and up to 12.1 on an on-premises or EC2 instance, and all Oracle database editions for versions 11g (versions 11.2.0.3.v1 and later) and up to 12.1 for an Amazon RDS DB instance. An Oracle account with the specific access privileges is required.

You can use SSL to encrypt connections between your Oracle endpoint and the replication instance. For more information on using SSL with an Oracle endpoint, see Using SSL With AWS Database Migration Service (p. 57).

If you plan to use change data capture (CDC) from your Oracle database, you need to set up supplemental logging. For information on setting up supplemental logging, see Configuring Oracle as a Source (p. 74).

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

Topics

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

Oracle offers two methods for reading the logs when doing change processing: Oracle LogMiner and Oracle Binary Reader.

By default, AWS DMS uses Oracle LogMiner for change data capture (CDC). Alternatively, you can choose to use the Oracle Binary Reader. The Oracle Binary Reader bypasses LogMiner and reads the logs directly. To enable the Binary Reader, you need to modify your source connection to include the following extra connection parameters:

```
useLogminerReader=N; useBfile=Y
```

To enable Logminer, you need to modify your source connection to include the following extra connection parameter or leave the Extra Connection Attribute blank (Logminer is the default):

```
useLogminerReader=Y
```

If the Oracle source database is using Oracle ASM (Automatic Storage Management), the extra connection parameter needs to include the asm username and asm server address. The password field will also need to have both passwords, the source user password, as well as the ASM password.

```
useLogminerReader=N;asm_user=<asm_username>;asm_server=<first_RAC_server_ip_address>/+ASM
```

If the Oracle source database is using Oracle ASM (Automatic Storage Management), the endpoint password field needs to have both the Oracle user password and the ASM password, separated by a comma.

Example: <oracle_user_password>,<asm_user_password>

To use LogMiner or Binary Reader, you must set the correct permissions. For information on setting these permissions, see the following section, Access Privileges Required for Change Data Capture (CDC) on an Oracle Source Database (p. 70)

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

- LogMiner supports most Oracle options, such as encryption options and compression options. Binary Reader doesn't support all Oracle options, in particular options for encryption and compression.
- LogMiner offers a simpler configuration, especially compared to Oracle Binary Reader's direct access setup or if the redo logs are on Automatic Storage Management (ASM).
- LogMiner can be used with Oracle sources that use Oracle transparent data encryption (TDE).

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

- For migrations with a high volume of changes, LogMiner might have some I/O or CPU impact on the computer hosting the Oracle source database. Binary Reader has less chance of having I/O or CPU impact.
- 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 large.
- You need to propagate changes to LOBs in Oracle 12c.

Access Privileges Required for Change Data Capture (CDC) on an Oracle Source Database

The access privileges required depend on whether you use Oracle LogMiner or Oracle Binary Reader for CDC.

The following privileges must be granted to the user account used for data migration when using Oracle LogMiner for change data capture (CDC) with an Oracle source database:

For Oracle versions prior to version 12c, grant the following:

- CREATE SESSION

- EXECUTE on DBMS_LOGMNR
- SELECT ANY TRANSACTION
- SELECT on V_$LOGMNR_LOGS
- SELECT on V_$LOGMNR_CONTENTS

For Oracle versions 12c and higher, grant the following:

- LOGMINING (for example, GRANT LOGMINING TO *<user account>*)
- CREATE SESSION
- EXECUTE on DBMS_LOGMNR
- SELECT on V_$LOGMNR_LOGS
- SELECT on V_$LOGMNR_CONTENTS

The following privileges must be granted to the user account used for data migration when using Oracle Binary Reader for change data capture (CDC) with an Oracle source database:

- SELECT on v_$transportable_platform

  Grant the SELECT on v_$transportable_platform privilege if the Redo logs are stored in Automatic Storage Management (ASM) and accessed by AWS DMS from ASM.
- BFILE read - Used when AWS DMS does not have file-level access to the Redo logs, and the Redo logs are not accessed from ASM.
- DBMS_FILE_TRANSFER package - Used to copy the Redo log files to a temporary folder (in which case the EXECUTE ON DBMS_FILE_TRANSFER privilege needs to be granted as well)
- DBMS_FILE_GROUP package - Used to delete the Redo log files from a temporary/alternate folder (in which case the EXECUTE ON DBMS_FILE_ GROUP privilege needs to be granted as well).
- CREATE ANY DIRECTORY

Oracle file features work together with Oracle directories. Each Oracle directory object includes the name of the folder containing the files which need to be processed.

If you want AWS DMS to create and manage the Oracle directories, you need to grant the CREATE ANY DIRECTORY privilege specified above. Note that the directory names will be prefixed with amazon_. If you do not grant this privilege, you need to create the corresponding directories manually. If you create the directories manually and the Oracle user specified in the Oracle Source endpoint is not the user that created the Oracle Directories, grant the READ on DIRECTORY privilege as well.

If the Oracle source endpoint is configured to copy the Redo log files to a temporary folder, and the Oracle user specified in the Oracle source endpoint is not the user that created the Oracle directories, the following additional privileges are required:

- READ on the Oracle directory object specified as the source directory
- WRITE on the directory object specified as the destination directory in the copy process.

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

The following limitations apply when using an Oracle database as a source for AWS Database Migration Service (AWS DMS) change data capture.

- Oracle LogMiner, which AWS DMS uses for change data capture (CDC), doesn't support updating large binary objects (LOBs) when the UPDATE statement updates only LOB columns.

- For Oracle 11, Oracle LogMiner doesn't support the UPDATE statement for XMLTYPE and LOB columns.
- On Oracle 12c, LogMiner does not support LOB columns.
- AWS DMS doesn't capture changes made by the Oracle DBMS_REDEFINITION package, such as changes to table metadata and the OBJECT_ID value.
- AWS DMS doesn't support index-organized tables with an overflow segment in change data capture (CDC) mode when using BFILE. An example is when you access the redo logs without using LogMiner.
- AWS DMS doesn't support table clusters when you use Oracle Binary Reader.
- AWS DMS doesn't support virtual columns.

Limitations on Using Oracle as a Source for AWS Database Migration Service

The following limitations apply when using an Oracle database as a source for AWS Database Migration Service (AWS DMS). If you are using Oracle LogMiner or Oracle Binary Reader for change data capture (CDC), see Limitations for Change Data Capture (CDC) on an Oracle Source Database (p. 71) for additional limitations.

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

Supported Compression Methods

AWS Database Migration Service supports all compression methods supported by LogMiner.

User Account Privileges Required for Using Oracle as a Source for AWS Database Migration Service

To use an Oracle database as a source in an AWS DMS task, the user specified in the AWS DMS Oracle database definitions must be granted the following privileges in the Oracle database. To grant privileges on Oracle databases on Amazon RDS, use the stored procedure `rdsadmin.rdsadmin_util.grant_sys_object`. For more information, see Granting SELECT or EXECUTE privileges to SYS Objects.

Note
When granting privileges, use the actual name of objects (for example, `V_$OBJECT` including the underscore), not the synonym for the object (for example, `V$OBJECT` without the underscore).

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

For the requirements specified following, grant the additional privileges named:

- If views are exposed, grant SELECT on ALL_VIEWS.
- When you use a specific table list, for each replicated table grant SELECT.
- When you use a pattern for a specific table list, grant SELECT ANY TABLE.
- When you add supplemental logging, grant ALTER ANY TABLE.

- When you add supplemental logging and you use a specific table list, grant ALTER for each replicated table.
- When migrating from Oracle RAC, grant permission to materialized views with the prefixes g_$ and v_$.

Configuring Oracle as a Source

Before using an Oracle database as a data migration source, you need to perform several tasks. For an Oracle database on Amazon RDS, see the following section. You can also use extra connection attributes to configure the Oracle source. For more information about extra connection attributes, see Using Extra Connection Attributes with AWS Database Migration Service (p. 199).

For an Oracle database on premises or on an Amazon EC2 instance, you should do the following:

- **Provide Oracle Account Access** – You must provide an Oracle user account for AWS Database Migration Service. The user account must have read/write privileges on the Oracle database, as specified in User Account Privileges Required for Using Oracle as a Source for AWS Database Migration Service (p. 73).
- **Ensure that ARCHIVELOG Mode Is On** – Oracle can run in two different modes, the ARCHIVELOG mode and the NOARCHIVELOG mode. To use Oracle with AWS Database Migration Service, the database in question must be in ARCHIVELOG mode.
- **Set Up Supplemental Logging** – The following steps, required only when using change data capture (CDC), show how to set up supplemental logging for an Oracle database. For information on setting up supplemental logging on a database on an Amazon RDS DB instance, see Configuring Oracle on an Amazon RDS DB Instance as a Source (p. 75).

  To set up supplemental logging for an Oracle database

  1. Determine if supplemental logging is enabled for the database:
     - Run the following query:

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

       The return result should be from `GE to 9.0.0`.
     - Run the following query:

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

       The returned result should be `YES` or `IMPLICIT`.
     - Enable supplemental logging by running the following query:

       ```
       ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
       ```
  2. Make sure that the required supplemental logging is added for each table:
     - If a primary key exists, supplemental logging must be added for the primary key, either by using the format to add supplemental logging on the primary key or by adding supplemental logging on the primary key columns.

- If no primary key exists and the table has a single unique index, then all of the unique index's columns must be added to the supplemental log. Using `SUPPLEMENTAL LOG DATA (UNIQUE INDEX) COLUMNS` doesn't add the unique index columns to the log.
- If no primary key exists and the table has multiple unique indexes, AWS DMS selects the first unique index. AWS DMS uses the first index in an alphabetically ordered ascending list in this case. Supplemental logging must be added on the selected index's columns. Using SUPPLEMENTAL LOG DATA (UNIQUE INDEX) COLUMNS doesn't add the unique index columns to the log.
- If there is no primary key and no unique index, supplemental logging must be added on all columns.

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

3. Perform additional logging if necessary, for example if a filter is defined for a table.

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

```
ALTER TABLE EXAMPLE.TABLE ADD SUPPLEMENTAL LOG GROUP example_log_group (ID,NAME)
ALWAYS;
```

Note
You can also turn on supplemental logging using a connection attribute. If you use this option, you still need to enable supplemental logging at the database level using the following statement:

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

For more information on setting a connection attribute, see Oracle (p. 201)

Configuring Oracle on an Amazon RDS DB Instance as a Source

Using an Oracle database on an Amazon RDS DB instance as a data migration source requires several settings on the DB instance, including the following:

- **Set Up Supplemental Logging** – AWS DMS requires database-level supplemental logging to be enabled. To enable database-level supplemental logging, run the following command:

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

Note

You can also turn on supplemental logging using a connection attribute. If you use this option, you still need to enable supplemental logging at the database level using the following statement:

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

For more information on setting a connection attribute, see Oracle (p. 201)

In addition to running this command, we recommend that you turn on PRIMARY KEY logging at the database level to enable change capture for tables that have primary keys. To turn on PRIMARY KEY logging, run the following command:

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

If you want to capture changes for tables that don't have primary keys, you should alter the table to add supplemental logging using the following command:

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

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

```
create table table_name(column data type, supplemental log data(ALL) columns);
```

If you create a table and later add a primary key, you need to add supplemental logging to the table. The following command alters the table to include supplemental logging:

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

- **Enable Automatic Backups** – For information on setting up automatic backups, see the *Amazon RDS User Guide*.

- **Set Up Archiving** – To retain archived redo logs of your Oracle database instance, which lets AWS DMS retrieve the log information using LogMiner, execute the following command. In this example, logs are kept for 24 hours.

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

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

Using a Microsoft SQL Server Database as a Source for AWS Database Migration Service

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

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

AWS DMS supports, as a source, Amazon RDS DB instance databases for SQL Server versions 2008R2, 2012, and 2014. The Enterprise and Standard editions are supported. Change data capture (CDC) is not supported for source databases on Amazon RDS. The Web, Workgroup, Developer, and Express editions are not supported.

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

You can use SSL to encrypt connections between your SQL Server endpoint and the replication instance. For more information on using SSL with a SQL Server endpoint, see Using SSL With AWS Database Migration Service (p. 57).

To capture changes from a source SQL Server database, the database must be configured for full backups and must be either the Enterprise, Developer, or Standard Edition. For a list of requirements and limitations when using CDC with SQL Server, see Special Limitations When Capturing Data Changes (CDC) from a SQL Server Source (p. 78).

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

Topics

General Limitations on Using SQL Server as a Source for AWS Database Migration Service

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

- The identity property for a column is not migrated to a target database column.
- Changes to rows with more than 8000 bytes of information, including header and mapping information, are not processed correctly due to limitations in the SQL Server TLOG buffer size.
- The Microsoft SQL Server endpoint does not support the use of sparse tables.
- Windows Authentication is not supported.
- Changes to computed fields in a Microsoft SQL Server are not replicated.
- Microsoft SQL Server partition switching is not supported.
- A clustered index on the source will be created as a non-clustered index on target.
- When using the WRITETEXT and UPDATETEXT utilities, AWS DMS does not capture events applied on the source database.
- The following data manipulation language (DML) pattern is not supported:

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

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

  For example, running the following command causes AWS Database Migration Service to fail:

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

Special Limitations When Capturing Data Changes (CDC) from a SQL Server Source

The following limitations apply specifically when trying to capture changes from a SQL Server database as a source for AWS DMS:

- You must use either the Enterprise, Standard, or Developer Edition.
- SQL Server must be configured for full backups and a backup must be made before beginning to replicate data.
- Recovery Model must be set to **Bulk logged** or **Full**.
- Microsoft SQL Server backup to multiple disks is not supported. If the backup is defined to write the database backup to multiple files over different disks, AWS DMS will not be able to read the data and the AWS DMS task will fail.
- Microsoft SQL Server Replication Publisher definitions for the source database used in a DMS CDC task are not removed when you remove a task. A Microsoft SQL Server system administrator must delete these definitions from Microsoft SQL Server.
- During CDC, AWS DMS needs to look up SQL Server transaction log backups to read changes. AWS DMS does not support using SQL Server transaction log backups that were created using third party backup software.
- SQL Server does not capture changes on newly created tables until they have been published. When tables are added to a SQL Server source, AWS DMS manages the creation of the publication. However,

this process might take several minutes. Operations made to newly created tables during this delay will not be captured or replicated to the target.

- The AWS DMS user account must have the **sysAdmin** fixed server role on the Microsoft SQL Server database you are connecting to.
- AWS DMS change data capture requires FULLOGGING to be turned on in SQL Server; the only way to turn on FULLLOGGING in SQL Server is to either enable MS-REPLICATION or CHANGE DATA CAPTURE (CDC).
- The SQL Server *tlog* cannot be reused until the changes have been processed.
- CDC operations are not supported on memory optimized tables. This limitation applies to Microsoft SQL Server 2014 (when the feature was first introduced) and above.

Supported Compression Methods

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

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

Note
Sparse columns and columnar structure compression are not supported.

Working with Microsoft SQL Server AlwaysOn Availability Groups

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

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

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

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

Configuring a Microsoft SQL Server Database as a Replication Source for AWS Database Migration Service

You can configure a Microsoft SQL Server database as a replication source for AWS Database Migration Service (AWS DMS). For the most complete replication of changes, we recommend that you use the Enterprise, Standard, or Developer edition of Microsoft SQL Server. One of these versions is required because these are the only versions that include MS-Replication(EE,SE) and MS-CDC(EE,DEV). The source SQL Server must also be configured for full backups. In addition, AWS DMS must connect with a user (a SQL Server instance login) that has the **sysAdmin** fixed server role on the SQL Server database you are connecting to.

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

Using MS-Replication to Capture Data Changes in Microsoft SQL Server

To use MS-REPLICATION to replicate changes, each source table must have a primary key. If a source table doesn't have a primary key, you can use MS-CDC for capturing changes. If you haven't previously enabled MS-REPLICATION, you must enable your SQL Server database to use MS-REPLICATION. To enable MS-REPLICATION, take the steps following or see the Microsoft SQL Server documentation. Setting up MS_REPLICATION adds a new SYSTEM database called `Distribution` to your source SQL Server database.

To open the Configure Distribution wizard from Microsoft SQL Server

1. In Microsoft SQL Server Management Studio, open the context (right-click) menu for the **Replication** folder, and then choose **Configure Distribution**.
2. In the **Distributor** step, choose **<Microsoft SQL Server Name> will act as its own distributor**. SQL Server creates a distribution database and log.

Using MS-CDC to Capture Data Changes in Microsoft SQL Server

If you need to replicate tables that don't have a primary key, the **Use MS-CDC** and **Do Not Use MS-Replication or MS-CDC** options are available, as described following.

Important
Replicating tables that don't have a primary key or a unique index can adversely affect performance, because additional database resources are required to capture the changes. However, you can prevent performance issues related to the absence of primary keys or a unique index by manually adding indexes to the target tables.

Note
SQL Server might not delete log entries. Log entries are not reused unless they are replicated and backed up.

Setting Up MS-CDC

To set up MS-CDC, you first need to enable MS-CDC for the database by running the following command.

```
use [DBname]
```

```
EXEC sys.sp_cdc_enable_db
```

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

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

For more information on setting up MS-CDC for specific tables, see the Microsoft SQL Server documentation.

Capturing Changes If You Can't Use MS-Replication or MS-CDC

If your database is not set up for MS-REPLICATION or MS-CDC, and it is listed as a supported configuration for CDC as discussed at the beginning of this section, you can still capture some changes to tables. However, such a setup only captures INSERT and DELETE data manipulation language (DML) events. UPDATE and TRUNCATE TABLE events are ignored. Depending on what operations occur, this setup can result in the target database no longer being consistent with the source.

Also in this setup, a DELETE event will not be applied to a row that has previously been modified on the source but not on the target (a row where a previous UPDATE event that was ignored).

Using a PostgreSQL Database as a Source for AWS Database Migration Service

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

You can use SSL to encrypt connections between your PostgreSQL endpoint and the replication instance. For more information on using SSL with a PostgreSQL endpoint, see Using SSL With AWS Database Migration Service (p. 57).

AWS DMS supports change data capture (CDC) for PostgreSQL tables with primary keys; if a table does not have a primary key, the WAL logs do not include a before image of the database row and AWS DMS cannot update the table.

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

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

Topics

Prerequisites for Using a PostgreSQL Database as a Source for AWS Database Migration Service

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

- Use a PostgreSQL database that is version 9.4.x or later.
- Grant superuser permissions for the user account specified for the PostgreSQL source database.
- Add the IP address of the AWS DMS replication server to the `pg_hba.conf` configuration file.
- Set the following parameters and values in the `postgresql.conf` configuration file:
  - Set `wal_level = logical`
  - Set `max_replication_slots >=1`

    The `max_replication_slots` value should be set according to the number of tasks that you want to run. For example, to run five tasks you need to set a minimum of five slots. Slots open automatically as soon as a task starts and remain open even when the task is no longer running. You need to manually delete open slots.
  - Set `max_wal_senders >=1`

    The `max_wal_senders` parameter sets the number of concurrent tasks that can run.
  - Set `wal_sender_timeout =0`

    The `wal_sender_timeout` parameter terminates replication connections that are inactive longer than the specified number of milliseconds. Although the default is 60 seconds, we recommend that you set this parameter to zero, which disables the timeout mechanism.

Security Requirements When Using a PostgreSQL Database as a Source for AWS Database Migration Service

The only security requirement when using PostgreSQL as a source is that the user account specified must be a registered user in the PostgreSQL database.

Limitations on Using a PostgreSQL Database as a Source for AWS Database Migration Service

The following change data capture (CDC) limitations apply when using PostgreSQL as a source for AWS DMS:

- A captured table must have a primary key. If a table doesn't have a primary key, AWS DMS ignores DELETE and UPDATE record operations for that table.
- 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, but since the results of updating a primary key in PostgreSQL is unpredictable, no records are written to the exceptions table.

.

- AWS DMS doesn't support the **Start Process Changes from Timestamp** run option.
- AWS DMS supports full load and change processing on Amazon RDS for PostgreSQL. For information on how to prepare a PostgreSQL DB instance and to set it up for using CDC, see Setting Up an Amazon RDS PostgreSQL DB Instance as a Source (p. 83).
- AWS DMS doesn't map some PostgreSQL data types, including the JSON data type. The JSON is converted to CLOB.
- Replication of multiple tables with the same name but where each name has a different case (for example table1, TABLE1, and Table1) can cause unpredictable behavior, and therefore AWS DMS doesn't support it.
- AWS DMS supports change processing of CREATE, ALTER, and DROP DDL statements for tables unless the tables are held in an inner function or procedure body block or in other nested constructs.

  For example, the following change is not captured:

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

- AWS DMS doesn't support change processing of TRUNCATE operations.
- AWS DMS doesn't support replication of partitioned tables. When a partitioned table is detected, the following occurs:
  - The endpoint reports a list of parent and child tables.
  - AWS DMS creates the table on the target as a regular table with the same properties as the selected tables.
  - If the parent table in the source database has the same primary key value as its child tables, a “duplicate key” error is generated.

  Note
  To replicate partitioned tables from a PostgreSQL source to a PostgreSQL target, you first need to manually create the parent and child tables on the target. Then you define a separate task to replicate to those tables. In such a case, you set the task configuration to **Truncate before loading**.

Setting Up an Amazon RDS PostgreSQL DB Instance as a Source

You can use the AWS master user account for the PostgreSQL DB instance as the user account for the PostgreSQL source endpoint for AWS DMS . The master user account has the required roles that allow it to set up change data capture (CDC). If you use an account other than the master user account, the account must have the rds_superuser role and the rds_replication role. The rds_replication role grants permissions to manage logical slots and to stream data using logical slots.

If you don't use the master user account for the DB instance, you must create several objects from the master user account for the account that you use. For information about creating the needed objects, see Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account (p. 84).

Using CDC with an Amazon RDS for PostgreSQL DB Instance

You can use PostgreSQL's native logical replication feature to enable CDC during a database migration of an Amazon RDS PostgreSQL DB instance. This approach reduces downtime and ensures that the target database is in sync with the source PostgreSQL database. Amazon RDS supports logical replication for a PostgreSQL DB instance version 9.4.9 and higher and 9.5.4 and higher.

To enable logical replication for an RDS PostgreSQL DB instance, do the following:

- In general, use the AWS master user account for the PostgreSQL DB instance as the user account for the PostgreSQL source endpoint. The master user account has the required roles that allow the it to set up CDC. If you use an account other than the master user account, you must create several objects from the master account for the account that you use. For more information, see Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account (p. 84).
- Set the `rds.logical_replication` parameter in your DB parameter group to 1. This is a static parameter that requires a reboot of the DB instance for the parameter to take effect. As part of applying this parameter, AWS DMS sets the `wal_level`, `max_wal_senders`, `max_replication_slots`, and `max_connections` parameters. Note that these parameter changes can increase WAL generation so you should only set the `rds.logical_replication` parameter when you are using logical slots.

Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account

If you don't use the master user account for the Amazon RDS PostgreSQL DB instance that you are using as a source, you need to create several objects to capture data definition language (DDL) events. You create these objects in the account other than the master account and then create a trigger in the master user account.

Note
If you set the `captureDDL` parameter to *N* on the source endpoint, you will not have to create the following table/trigger on the source database.

Use the following procedure to create these objects. The user account other than the master account is referred to as the NoPriv account in this procedure.

1. Choose a schema where the objects will be created. The default schema is `public`. Ensure that the schema exists and is accessible by the NoPriv account.
2. Log in to the PostgreSQL DB instance using the NoPriv account.
3. Create the table `awsdms_ddl_audit` by running the following command, replacing <objects_schema> in the code following with the name of the schema to use:

```
create table <objects_schema>.awsdms_ddl_audit
(
  c_key    bigserial primary key,
  c_time   timestamp,    -- Informational
  c_user   varchar(64),  -- Informational: current_user
  c_txn    varchar(16),  -- Informational: current transaction
  c_tag    varchar(24),  -- Either 'CREATE TABLE' or 'ALTER TABLE' or 'DROP TABLE'
  c_oid    integer,      -- For future use - TG_OBJECTID
  c_name   varchar(64),  -- For future use - TG_OBJECTNAME
  c_schema varchar(64),  -- For future use - TG_SCHEMANAME. For now - holds
 current_schema
  c_ddlqry  text         -- The DDL query associated with the current DDL event
)
```

4. Create the function `awsdms_intercept_ddl` by running the following command, replacing <objects_schema> in the code following with the name of the schema to use:

```
CREATE OR REPLACE FUNCTION <objects_schema>.awsdms_intercept_ddl()
  RETURNS event_trigger
LANGUAGE plpgsql
  AS $$
  declare _qry text;
BEGIN
  if (tg_tag='CREATE TABLE' or tg_tag='ALTER TABLE' or tg_tag='DROP TABLE') then
         SELECT current_query() into _qry;
         insert into <objects_schema>.awsdms_ddl_audit
         values
         (
         default,current_timestamp,current_user,cast(TXID_CURRENT()as
 varchar(16)),tg_tag,0,'',current_schema,_qry
         );
         delete from <objects_schema>.awsdms_ddl_audit;
end if;
END;
$$;
```

5. Log out of the NoPriv account and log in with an account that has the rds_superuser role assigned to it.
6. Create the event trigger `awsdms_intercept_ddl` by running the following command:

```
CREATE EVENT TRIGGER awsdms_intercept_ddl ON ddl_command_end
EXECUTE PROCEDURE <objects_schema>.awsdms_intercept_ddl();
```

When you have completed the procedure preceding, you can create the AWS DMS source endpoint using the NoPriv account.

Removing AWS Database Migration Service Artifacts from a PostgreSQL Source Database

To capture DDL events, AWS DMS creates various artifacts in the PostgreSQL database when a migration task starts. When the task completes, you might want to remove these artifacts. To remove the artifacts, issue the following statements (in the order they appear), where `{AmazonRDSMigration}` is the schema in which the artifacts were created:

```
drop event trigger awsdms_intercept_ddl;
```

Note that the event trigger doesn't belong to a specific schema.

```
drop function {AmazonRDSMigration}.awsdms_intercept_ddl()
drop table {AmazonRDSMigration}.awsdms_ddl_audit
drop schema {AmazonRDSMigration}
```

Note
Dropping a schema should be done with extreme caution, if at all. Never drop an operational schema, especially not a public one.

Additional Configuration Settings When Using a PostgreSQL Database as a Source for AWS Database Migration Service

You can add additional configuration settings when migrating data from a PostgreSQL database in two way.

- 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 PostgreSQL (p. 200).
- You can override connection string parameters. Select this option if you need to do either of the following:
  - Specify internal AWS DMS parameters. Such parameters are rarely required and are therefore not exposed in the user interface.
  - Specify pass-through (passthru) values for the specific database client. AWS DMS includes pass-through parameters in the connection sting passed to the database client.

Using a MySQL-Compatible Database as a Source for AWS Database Migration Service

You can migrate data from one or many MySQL, MariaDB, or Amazon Aurora databases using AWS Database Migration Service. With a MySQL-compatible database as a source, you can migrate data to either another MySQL-compatible database or one of the other supported databases. MySQL versions 5.5, 5.6, and 5.7, as well as MariaDB and Amazon Aurora, are supported for on-premises, Amazon RDS, and Amazon EC2 instance databases. To enable change data capture (CDC) with Amazon RDS MySQL, you must use Amazon RDS MySQL version 5.6 or higher.

> **Note**
> Regardless of the source storage engine (MyISAM, MEMORY, etc.), AWS DMS creates the MySQL-compatible target table as an InnoDB table by default. If you need to have a table that uses a storage engine other than InnoDB, you can manually create the table on the MySQL-compatible target and migrate the table using the "Do Nothing" mode. For more information about the "Do Nothing" mode, see Full Load Task Settings (p. 125).

You can use SSL to encrypt connections between your MySQL-compatible endpoint and the replication instance. For more information on using SSL with a MySQL-compatible endpoint, see Using SSL With AWS Database Migration Service (p. 57).

For additional details on working with MySQL-compatible databases and AWS Database Migration Service, see the following sections.

Topics

Prerequisites for Using a MySQL Database as a Source for AWS Database Migration Service

Before you begin to work with a MySQL database as a source for AWS DMS, make sure that you have the following prerequisites:

- A MySQL account with the required security settings. For more information, see Security Requirements for Using a MySQL Database as a Source for AWS Database Migration Service (p. 89).
- A MySQL-compatible database with the tables that you want to replicate accessible in your network.
  - MySQL Community Edition
  - MySQL Standard Edition
  - MySQL Enterprise Edition
  - MySQL Cluster Carrier Grade Edition
  - MariaDB
  - Amazon Aurora
- If your source is an Amazon RDS MySQL or MariaDB DB instance or an Amazon Aurora cluster, you must enable automatic backups. For more information on setting up automatic backups, see the *Amazon RDS User Guide*.
- If you use change data capture (CDC), you must enable and configure 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`.
`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`.

- In order to use change data capture (CDC) with an Amazon RDS MySQL DB instance as a source, AWS DMS needs access to the binary logs. Amazon RDS is fairly aggressive in clearing binary logs from the DB instance. To use CDC with a MySQL DB instance on Amazon RDS, you should increase the amount of time the binary logs remain on the MySQL DB instance. For example, to increase log retention to 24 hours, you would run the following command:

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

- To replicate clustered (NDB) tables using AWS Database Migration Service, the following parameters must be configured in MySQL's `my.ini` (Windows) or `my.cnf` (UNIX) file. Replicating clustered (NDB) tables is required only when you use CDC.

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.

Prerequisites for Using an Amazon RDS for MySQL Database as a Source for AWS Database Migration Service

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

- You must enable automatic backups. For more information on setting up automatic backups, see the *Amazon RDS User Guide*.
- In order to use change data capture (CDC) with an Amazon RDS MySQL DB instance as a source, AWS DMS needs access to the binary logs. Amazon RDS is fairly aggressive in clearing binary logs from the DB instance. To use CDC with a MySQL DB instance on Amazon RDS, you should increase the amount of time the binary logs remain on the MySQL DB instance to 24 hours or greater. For example, to increase log retention to 24 hours, you would run the following command:

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

- The `binlog_format` parameter should be set to "ROW."

Limitations on Using a MySQL Database as a Source for AWS Database Migration Service

When using a MySQL database as a source, AWS DMS doesn't support the following:

- The 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.
- Capturing changes from tables whose names contain both uppercase and lowercase characters if the source MySQL instance is installed on an OS with a file system where file names are treated as case in-sensitive, for example Windows and OS X using HFS+.
- The AR_H_USER header column.
- The AUTO_INCREMENT attribute on a column is not migrated to a target database column.
- Capturing changes when the binary logs are not stored on standard block storage. For example, CDC does not work when the binary logs are stored on Amazon S3.

Security Requirements for Using a MySQL Database as a Source for AWS Database Migration Service

As a security requirement, the AWS DMS user must have the ReplicationAdmin role with the following privileges:

- **REPLICATION CLIENT** – This privilege is required for change data capture (CDC) tasks only. In other words, full-load-only tasks don't require this privilege.
- **REPLICATION SLAVE** – This privilege is required for change data capture (CDC) tasks only. In other words, full-load-only tasks don't require this privilege.
- **SUPER** – This privilege is required only in MySQL versions prior to 5.6.6.

The AWS DMS user must also have SELECT privileges for the source tables designated for replication.

Configuring a MySQL Database as a Source for AWS Database Migration Service

You can use extra connection attributes to configure the MySQL source. For more information about MySQL extra connection attributes, see Using Extra Connection Attributes with AWS Database Migration Service (p. 199).

Using a SAP ASE Database as a Source for AWS Database Migration Service

You can migrate data from a SAP Adaptive Server Enterprise (ASE) database–formerly known as Sybase–using AWS Database Migration Service. With a SAP ASE database as a source, you can migrate data to any of the other supported AWS DMS target databases.

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

Topics

Prerequisites for Using a SAP ASE Database as a Source for AWS Database Migration Service

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

- SAP ASE replication must be enabled for tables by using the `sp_setreptable` command.
- `RepAgent` must be disabled on the SAP ASE database.

- When replicating to SAP ASE version 15.7 installed on a Windows EC2 instance configured with a non-Latin language (for example, Chinese), AWS DMS requires SAP ASE 15.7 SP121 to be installed on the target SAP ASE machine.

Limitations on Using SAP ASE as a Source for AWS Database Migration Service

The following limitations apply when using an SAP ASE database as a source for AWS Database Migration Service (AWS DMS):

- Only one AWS DMS task can be run per SAP ASE database.
- Rename table is not supported, for example: `sp_rename 'Sales.SalesRegion', 'SalesReg;`
- Rename column is not supported, for example: `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 will become `0x00000000000000000000000001000001` in the target table.
- AWS DMS creates the target table with columns that do not allow NULL values, if the database default is not to allow NULL values. Consequently, if a Full Load or CDC replication task contains empty values, errors will occur.

  You can prevent these errors from occurring by allowing nulls 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 is not supported.

User Account Permissions Required for Using SAP ASE as a Source for AWS Database Migration Service

To use an SAP ASE database as a source in an AWS DMS task, the user specified in the AWS DMS SAP ASE database definitions must be granted the following permissions in the SAP ASE database.

- sa_role
- replication_role
- sybase_ts_role
- If you have set the `enableReplication` connection property to `Y`, then your must also be granted the `sp_setreptable` permission. For more information on connection properties see Using Extra Connection Attributes with AWS Database Migration Service (p. 199).

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.

Once the `$replication_truncation_point` entry has been established, the AWS DMS task must be kept running at all times to prevent the database log from becoming excessively large. If you want to stop the AWS DMS task permanently, the replication truncation point must be removed by issuing the following command:

```
dbcc settrunc('ltm','ignore')
```

After the truncation point has been removed, the AWS DMS task cannot be resumed. The log continues to be truncated automatically at the checkpoints (if automatic truncation is set).

Using MongoDB as a Source for AWS Database Migration Service

AWS DMS supports MongoDB versions 2.6.x and 3.x as a database source. A MongoDB database is a JSON document database where there are multiple MongoDB collections made up of JSON documents. In MongoDB, a collection is somewhat equivalent to a relational database table and a JSON document is somewhat equivalent to a row in that relational database table. Internally, a JSON 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

In document mode, the MongoDB document is migrated “as is,” meaning that its JSON data becomes a single column in a target table named “_doc”.

You can optionally set the `extractDocID` parameter to *true* to create a second column named “_id” that will act as the primary key. You must set this parameter to *true* if you are going to use change data capture (CDC).

Document mode is the default setting when you use MongoDB as a source. To explicitly specify document mode, add `nestingLevel=NONE` to the extra connection attribute on the MongoDB source endpoint.

Here is how AWS DMS manages documents and collections in document mode:

- When adding a new collection, the collection is replication as a CREATE TABLE.
- Renaming a collection is not supported.

Table Mode

In table mode, AWS DMS scans a specified number of MongoDB documents and creates a set of all the keys and their types. This set is then used to create the columns of the target table. In this mode, a MongoDB document is transformed into a table data row. Each top level field is transformed into a column. For each MongoDB document, AWS DMS adds each key and type to the target table’s column set. Nested values are flattened into a column containing dot-separated key names. For example, a JSON document consisting of {"a" : {"b" : {"c": 1}}} is migrated into a column named a.b.c.

You can specify how many documents are scanned by setting the `docsToInvestigate` parameter. The default value is 1000. You can enable table mode by adding `nestingLevel=ONE` to the extra connection attributes of the MongoDB source endpoint.

Here is how AWS DMS manages documents and collections in table mode:

- When you add a document to an existing collection, the document (row) is replicated. If there are fields that do not exist in the collection, those fields are not replicated.
- When you update a document, the updated document is replicated. If there are fields that do not exist in the collection, those fields are not replicated.
- Deleting a document is fully supported.
- Adding a new collection will not result in a new table on the target when done during a CDC task.
- Renaming a collection is not supported.

Prerequisites When Using MongoDB as a Source for AWS Database Migration Service

The user account used for the MongoDB endpoint needs to have access to the operations log of the replica set you create.

Prerequisites When Using CDC with MongoDB as a Source for AWS Database Migration Service

To use change data capture (CDC) with a MongoDB source, you you'll need to do several things. First, you deploy the replica set to create the operations log. Next, you create the system user administrator. Finally, you set the extractDocID parameter to *true* to extract the document ID that is used during CDC.

The MongoDB operations log (oplog) is a special capped collection that keeps a rolling record of all operations that modify the data stored in your databases. MongoDB applies database operations on the **primary** and then records the operations on the primary's **oplog**. The secondary members then copy and apply these operations in an asynchronous process.

Deploying a Replica Set for Use with CDC

You need to deploy a replica set to use MongoDB as an AWS DMS source. When you deploy the replica set, you create the operations log that is used for CDC. Do the following steps to deploy the replica set. For more information, see the MongoDB documentation.

To deploy a replica set

1. Using the command line, connect to mongo.

```
mongo localhost
```

2. Run in mongo shell.

```
rs.initiate() root
```

3. Verify the deployment.

```
rs.conf()
```

4. Set a different database path. Run with --dbpath the-path.

Next, create the system user administrator role.

To create the root user

1. Create a user to be the root account, as shown in the following code. In this example, we call that user `root`.

```
use admin
db.createUser(
  {
    user: "root",
    pwd: "rootpass",
    roles: [ { role: "root", db: "admin" } ]
  }
)
```

2. Stop mongod.
3. Restart mongod using the following command:

```
mongod --replSet "rs0" --auth
```

4. Test that the operations log is readable using the following commands:

```
mongo localhost/admin -u root -p rootpass
mongo  --authenticationDatabase admin -u root -p rootpass
```

5. If you are unable to read from operations log, run the following command:

```
rs.initiate();
```

The final requirement to use CDC with MongoDB is to set the `extractDocID` parameter. Set the `extractDocID` parameter to *true* to create a second column named "_id" that will act as the primary key.

Security Requirements When Using MongoDB as a Source for AWS Database Migration Service

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:

- **MONOGODB-CR** — the default when using MongoDB 2.x authentication.
- **SCRAM-SHA-1** — the default when using MongoDB version 3.x authentication.

If an authentication method is not specified, AWS DMS uses the default method for the version of the MongoDB source. The two authentication methods are used to encrypt the password, so they are only used when the `authType` parameter is set to *password*.

Limitations When Using MongoDB as a Source for AWS Database Migration Service

The following are limitations when using MongoDB as a source for AWS DMS:

- When the `extractDocID` parameter is set to *true*, the ID string cannot exceed 200 characters.

- The MongoDB parameter `ObjectId` is a string with a limit of 200 characters. It includes the JSON structure of the id object: { "_id" : { "$oid" : "581730b9e85de3f180fd571e" } } and is used as the primary key in the target database.
- Collection names cannot include the dollar symbol ($).

Configuration Properties When Using MongoDB as a Source for AWS Database Migration Service

When you set up your MongoDB source endpoint, you can specify additional configuration settings attributes. Attributes are specified by key-value pairs and separated by semicolons. For example, the following code specifies that user name and password are not used for authentication, and that table mode is used.

The following table describes the configuration properties available when using MongoDB databases as an AWS Database Migration Service source database.

Attribute Name	Valid Values	Default Value and Description
authType	NO PASSWORD	PASSWORD – When NO is selected, user name and password parameters are not used and can be empty.
authMechanism	DEFAULT MONGODB_CR SCRAM_SHA_1	DEFAULT – For MongoDB version 2.x, use MONGODB_CR. For MongoDB version 3.x, use SCRAM_SHA_1. This attribute is not used when authType=No.
nestingLevel	NONE ONE	NONE – Specify NONE to use document mode. Specify ONE to use table mode.
extractDocID	true false	false – Use this attribute when `nestingLevel` is set to NONE.
docsToInvestigate	A positive integer greater than 0.	1000 – Use this attribute when `nestingLevel` is set to ONE.
authSource	A valid MongoDB database name.	admin – This attribute is not used when authType=No.

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.

> **Note**
> Regardless of the source storage engine (MyISAM, MEMORY, etc.), AWS DMS creates the MySQL-compatible target table as an InnoDB table by default. If you need to have a table that uses a storage engine other than InnoDB, you can manually create the table on the MySQL-compatible target and migrate the table using the "Do Nothing" mode. For more information about the "Do Nothing" mode, see Full Load Task Settings (p. 125).

The databases include the following:

On-premises and EC2 instance databases

- Oracle versions 10g, 11g, 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Microsoft SQL Server versions 2005, 2008, 2008R2, 2012, 2014 for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- MySQL versions 5.5, 5.6, and 5.7
- MariaDB (supported as a MySQL-compatible data target)
- PostgreSQL versions 9.3 and later
- SAP Adaptive Server Enterprise (ASE) 15.7 and later

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

- Amazon RDS Oracle versions 11g (versions 11.2.0.3.v1 and later) and 12c, for the Enterprise, Standard, Standard One, and Standard Two editions
- Amazon RDS Microsoft SQL Server versions 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions. The Web and Express editions are not supported.
- Amazon RDS MySQL versions 5.5, 5.6, and 5.7
- Amazon RDS MariaDB (supported as a MySQL-compatible data target)
- Amazon RDS PostgreSQL versions 9.3 and later

- Amazon Aurora
- Amazon Redshift
- Amazon S3
- Amazon DynamoDB

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 SSL to encrypt connections between your Oracle endpoint and the replication instance. For more information on using SSL with an Oracle endpoint, see Using SSL With AWS Database Migration Service (p. 57).

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

When using Oracle as a target, we assume the data should be migrated into the schema/user which is used for the target connection. If you want to migrate data to a different schema, you'll need to use a schema transformation to do so. For example, if my target endpoint connects to the user RDSMASTER and you wish to migrate from the user PERFDATA to PERFDATA, you'll need to create a transformation as follows:

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

For more information about transformations, see Selection and Transformation Table Mapping using JSON (p. 139).

Limitations on Oracle as a Target for AWS Database Migration Service

Limitations when using Oracle as a target for data migration include the following:

- AWS DMS does not create schema on the target Oracle database. You have to create any schemas you want on the target Oracle database. The schema name must already exist for the Oracle target. Tables from source schema are imported to user/schema, which AWS DMS uses to connect to the target instance. You must create multiple replication tasks if you have to migrate multiple schemas.

- AWS DMS doesn't support the `Use direct path full load` option for tables with INDEXTYPE CONTEXT. As a workaround, you can use array load.
- In Batch Optimized Apply mode, loading into the net changes table uses Direct Path, which doesn't support XMLType. As a workaround, you can use Transactional Apply mode.

User Account Privileges Required for Using Oracle as a Target

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

- SELECT ANY TRANSACTION
- SELECT on V$NLS_PARAMETERS
- SELECT on V$TIMEZONE_NAMES
- SELECT on ALL_INDEXES
- SELECT on ALL_OBJECTS
- SELECT on DBA_OBJECTS
- SELECT on ALL_TABLES
- SELECT on ALL_USERS
- SELECT on ALL_CATALOG
- SELECT on ALL_CONSTRAINTS
- SELECT on ALL_CONS_COLUMNS
- SELECT on ALL_TAB_COLS
- SELECT on ALL_IND_COLUMNS
- DROP ANY TABLE
- SELECT ANY TABLE
- INSERT ANY TABLE
- UPDATE ANY TABLE
- CREATE ANY VIEW
- DROP ANY VIEW
- CREATE ANY PROCEDURE
- ALTER ANY PROCEDURE
- DROP ANY PROCEDURE
- CREATE ANY SEQUENCE
- ALTER ANY SEQUENCE
- DROP ANY SEQUENCE

For the requirements specified following, grant the additional privileges named:

- To use a specific table list, grant SELECT on any replicated table and also ALTER on any replicated table.
- To allow a user to create a table in his default tablespace, grant the privilege GRANT UNLIMITED TABLESPACE.
- For logon, grant the privilege CREATE SESSION.
- If you are using a direct path, grant the privilege LOCK ANY TABLE.

- If the "DROP and CREATE table" or "TRUNCATE before loading" option is selected in the full load settings, and the target table schema is different from that for the AWS DMS user, grant the privilege DROP ANY TABLE.
- To store changes in change tables or an audit table when the target table schema is different from that for the AWS DMS user, grant the privileges CREATE ANY TABLE and CREATE ANY INDEX.

Read Privileges Required for AWS Database Migration Service on the Target Database

The AWS DMS user account must be granted read permissions for the following DBA tables:

- SELECT on DBA_USERS
- SELECT on DBA_TAB_PRIVS
- SELECT on DBA_OBJECTS
- SELECT on DBA_SYNONYMS
- SELECT on DBA_SEQUENCES
- SELECT on DBA_TYPES
- SELECT on DBA_INDEXES
- SELECT on DBA_TABLES
- SELECT on DBA_TRIGGERS

If any of the required privileges cannot be granted to V$xxx, then grant them to V_$xxx.

Configuring an Oracle Database as a Target for AWS Database Migration Service

Before using an Oracle database as a data migration target, you must provide an Oracle user account to AWS DMS. The user account must have read/write privileges on the Oracle database, as specified in the section User Account Privileges Required for Using Oracle as a Target (p. 97).

Using a Microsoft SQL Server Database as a Target for AWS Database Migration Service

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

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

For Amazon RDS instance databases, AWS DMS supports as a target SQL Server versions 2008R2, 2012, and 2014, for the Enterprise, Standard, Workgroup, and Developer editions are supported. The Web and Express editions are not supported.

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

Topics

Limitations on Using SQL Server as a Target for AWS Database Migration Service

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

- When you manually create a SQL Server target table with a computed column, full load replication is not supported when using the BCP bulk-copy utility. To use full load replication, disable the **Use BCP for loading tables** option in the console's **Advanced** tab.
- 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.

Security Requirements When Using SQL Server as a Target for AWS Database Migration Service

The following describes the security requirements for using AWS DMS with a Microsoft SQL Server target.

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

Using a PostgreSQL Database as a Target for AWS Database Migration Service

You can migrate data to PostgreSQL databases using AWS DMS, either from another PostgreSQL database or from one of the other supported databases.

PostgreSQL versions 9.3 and later are supported for on-premises, Amazon RDS, and EC2 instance databases.

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 will be 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 will 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.

Limitations on Using PostgreSQL as a Target for AWS Database Migration Service

The following limitations apply when using a PostgreSQL database as a target for AWS DMS:

- The JSON data type is converted to the Native CLOB data type.

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.

Using a MySQL-Compatible Database as a Target for AWS Database Migration Service

You can migrate data to MySQL databases using AWS DMS, either from another MySQL database or from one of the other supported databases.

MySQL versions 5.5, 5.6, and 5.7, as well as MariaDB and Amazon Aurora, are supported.

Prerequisites for Using a MySQL-Compatible Database as a Target for AWS Database Migration Service

Before you begin to work with a MySQL database as a target for AWS DMS, make sure that you have the following prerequisites:

- A MySQL account with the required security settings. For more information, see Security Requirements When Using MySQL as a Target for AWS Database Migration Service (p. 101).
- A MySQL database with the tables that you want to replicate accessible in your network. AWS DMS supports the following MySQL editions:
  - MySQL Community Edition
  - MySQL Standard Edition
  - MySQL Enterprise Edition
  - MySQL Cluster Carrier Grade Edition
  - MariaDB
  - Amazon Aurora

- During a load, you should consider disabling foreign keys. In order to disable foreign key checks on a MySQL-compatible database during a load, you can add the following command to the **Extra Connection Attributes** in the **Advanced** section of the target MySQL, Aurora, MariaDB endpoint connection information:

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

Limitations on Using MySQL as a Target for AWS Database Migration Service

When using a MySQL database as a target, AWS DMS doesn't support the following:

- The 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.

In addition, when you update a column's value to its existing value, MySQL returns a `0 rows affected` warning. In contrast, Oracle performs an update of one row in this case. The MySQL result generates an entry in the awsdms_apply_exceptions control table and 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.
```

Security Requirements When Using MySQL as a Target for AWS Database Migration Service

When using MySQL as a target for data migration, you must provide MySQL account access to the AWS DMS user account. This user must have read/write privileges in the MySQL database.

To create the necessary privileges, run the following commands:

```
CREATE USER '<user acct>'@'%' IDENTIFIED BY <user password>';
GRANT ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE, SELECT ON myschema.* TO '<user
 acct>'@'%';
GRANT ALL PRIVILEGES ON awsdms_control.* TO '<user acct>'@'%';
```

Using an Amazon Redshift Database as a Target for AWS Database Migration Service

You can migrate data to Amazon Redshift databases using AWS Database Migration Service. Amazon Redshift is a fully-managed, petabyte-scale data warehouse service in the cloud. With an Amazon Redshift database as a target, you can migrate data from all of the other supported source databases.

The Amazon Redshift cluster must be in the same AWS account and same AWS Region as the replication instance.

During a database migration to Amazon Redshift, AWS DMS first moves data to an S3 bucket. Once the files reside in an S3 bucket, AWS DMS then transfers them to the proper tables in the Amazon Redshift data warehouse. AWS DMS creates the S3 bucket in the same AWS Region as the Amazon Redshift database. The AWS DMS replication instance must be located in that same region.

If you use the AWS Command Line Interface (AWS CLI) or the AWS DMS API to migrate data to Amazon Redshift, you must set up an AWS Identity and Access Management (IAM) role to allow S3 access. For more information about creating this IAM role, see Creating the IAM Roles to Use With the AWS CLI and AWS DMS API (p. 45).

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.

Topics

Prerequisites for Using an Amazon Redshift Database as a Target for AWS Database Migration Service

The following list describe the prerequisites necessary for working with Amazon Redshift as a target for data migration:

- Use the AWS Management Console to launch an Amazon Redshift cluster. You should note the basic information about your AWS account and your Amazon Redshift cluster, such as your password, user name, and database name. You will 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 Redshift endpoint (hostname and port) that your cluster uses.

- AWS DMS uses an Amazon S3 bucket to transfer data to the Redshift database. For AWS DMS to create the bucket, the DMS console uses an Amazon IAM role, `dms-access-for-endpoint`. If you use the AWS CLI or DMS API to create a database migration with Amazon Redshift as the target database, you must create this IAM role. For more information about creating this role, see Creating the IAM Roles to Use With the AWS CLI and AWS DMS API (p. 45).

Limitations on Using Redshift as a Target for AWS Database Migration Service

When using a Redshift database as a target, AWS DMS doesn't support the following:

- When migrating from MySQL/Aurora to Redshift, you cannot use DDL to alter a column from the BLOB data type to the NVARCHAR data type.

  For example, the following DDL is not supported.

```
ALTER TABLE table_name MODIFY column_name NVARCHAR(n);
```

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 Amazon Redshift (p. 205).

Using Enhanced VPC Routing with an Amazon Redshift as a Target for AWS Database Migration Service

If you're using the *Enhanced VPC Routing* feature with your Amazon Redshift target, the feature forces all COPY traffic between your Redshift cluster and your data repositories through your Amazon VPC.

Because *Enhanced VPC Routing* affects the way that Amazon Redshift accesses other resources, COPY commands might fail if you haven't configured your VPC correctly.

AWS DMS can be affected by this behavior since it uses the COPY command to move data in S3 to a 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 Redshift to copy data from the CSV files in S3 to an appropriate table in 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.

Using a SAP ASE Database as a Target for AWS Database Migration Service

You can migrate data to SAP Adaptive Server Enterprise (ASE)–formerly known as Sybase–databases using AWS DMS, either from any of the supported database sources.

SAP ASE version 15.7 and later are supported.

Prerequisites for Using a SAP ASE Database as a Target for AWS Database Migration Service

Before you begin to work with a SAP ASE database as a target for AWS DMS, make sure that you have the following prerequisites:

- You must provide SAP ASE account access to the AWS DMS user. This user must have read/write privileges in the SAP ASE database.
- When replicating to SAP ASE version 15.7 installed on a Windows EC2 instance configured with a non-Latin language (for example, Chinese), AWS DMS requires SAP ASE 15.7 SP121 to be installed on the target SAP ASE machine.

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 S3 as a target in an AWS DMS task, both full load and change data capture (CDC) data is written to comma-separated-values (CSV) format. AWS DMS names files created during a full load using incremental counters, for example LOAD00001.csv, LOAD00002, and so on. AWS DMSnames CDC files using timestamps, for example 20141029-1134010000.csv. For each source table, AWS DMS creates a folder under the specified target folder. AWS DMS writes all full load and CDC files to the specified S3 bucket.

The parameter `bucketFolder` contains the location where the .csv files are stored before being uploaded to the S3 bucket. Table data is stored in the following format in the S3 bucket:

```
<schema_name>/<table_name>/LOAD001.csv
<schema_name>/<table_name>/LOAD002.csv
<schema_name>/<table_name>/<time-stamp>.csv
```

You can specify the column delimiter, row delimiter, null value indicator, and other parameters using the extra connection attributes. For more information on the extra connection attributes, see Extra Connection Attributes (p. 106) at the end of this section.

Prerequisites for Using Amazon S3 as a Target

The Amazon S3 bucket you are using as a target must be in the same region as the DMS replication instance you are using the migrate your data.

The AWS account you use for the migration must have write and delete access to the Amazon S3 bucket you are using as a target. The role assigned to the user account used to create the migration task must have the following set of permissions.

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

Limitations to Using Amazon S3 as a Target

The following limitations apply to a file in Amazon S3 that you are using as a target:

- Only the following data definition language (DDL) commands are supported: TRUNCATE TABLE, DROP TABLE, and CREATE TABLE.
- Full LOB mode is not supported.
- Changes to the source table structure during full load are not supported. Changes to the data is supported during full load.

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. You must specify the Amazon Resource Name (ARN) of an IAM role that has the permissions required to access Amazon S3.

Extra Connection Attributes

You can specify the following options as extra connection attributes.

Option	Description
csvRowDelimiter	The delimiter used to separate rows in the source files. The default is a carriage return (`\n`). **Example:** `csvRowDelimiter=\n;`
csvDelimiter	The delimiter used to separate columns in the source files. The default is a comma. **Example:** `csvDelimiter=,;`
bucketFolder	An optional parameter to set a folder name in the S3 bucket. If provided, tables are created in the path <bucketFolder>/<schema_name>/<table_name>/. If this parameter is not specified, then the path used is <schema_name>/<table_name>/. **Example:** `bucketFolder=testFolder;`
bucketName	The name of the S3 bucket. **Example:** `bucketName=buckettest;`
compressionType	An optional parameter to use GZIP to compress the target files. Set to NONE (the default) or do not use to leave the files uncompressed. **Example:** `compressionType=GZIP;`

Using an Amazon DynamoDB Database as a Target for AWS Database Migration Service

You can use AWS DMS to migrate data to an Amazon DynamoDB table. Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. AWS DMS supports using a relational database or MongoDB as a source.

In DynamoDB, tables, items, and attributes are the core components that you work with. A table is a collection of items, and each item is a collection of attributes. DynamoDB uses primary keys, called partition keys, to uniquely identify each item in a table. You can also use keys and secondary indexes to provide more querying flexibility.

You use object mapping to migrate your data from a source database to a target DynamoDB table. Object mapping lets you determine where the source data is located in the target.

Migrating from a Relational Database to a DynamoDB Table

AWS DMS supports migrating data to DynamoDB's scalar data types. When migrating from a relational database like Oracle or MySQL to DynamoDB, you may 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 would take data from a table and reformat it into DynamoDB scalar data type attributes. These attributes can accept data from multiple columns and you can map a column to an attribute directly.

AWS DMS supports the following DynamoDB scalar data types:

- String
- Number
- Boolean

Note
NULL data from the source are ignored on the target.

Prerequisites for Using a DynamoDB as a Target for AWS Database Migration Service

Before you begin to work with a DynamoDB database as a target for AWS DMS, make sure you create an IAM role that allows AWS DMS to assume and grants access to the DynamoDB tables that are being migrated into. The minimum set of access permissions is shown in the following sample role policy:

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

The user account that you use for the migration to DynamoDB must have the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:CreateTable",
        "dynamodb:DescribeTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:account-id:table/Name1",
        "arn:aws:dynamodb:us-west-2:account-id:table/OtherName*",
        ]
   },
{
      "Effect": "Allow",
      "Action": [
        "dynamodb:ListTables",
      ],
      "Resource": "*"
    }
  ]
}
```

Limitations When Using DynamoDB as a Target for AWS Database Migration Service

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

- DynamoDB limits the precision of the Number data type to 38 places. Store all data types with a higher precision as a String. You need to explicitly specify this using the object mapping feature.
- Because Amazon DynamoDB doesn't have a Date data type, data using the Date data type are converted to strings.
- AWS DMS only supports replication of tables with non-composite primary keys, unless you specify an object mapping for the target table with a custom partition key or sort key, or both.
- AWS DMS doesn't support LOB data unless it is a CLOB. AWS DMS converts CLOB data into a DynamoDB string when migrating data.

Using Object Mapping to Migrate Data to DynamoDB

AWS DMS uses table mapping rules to map data from the source to the target DynamoDB table. To map data to a DynamoDB target, you use a type of table mapping rule called *object-mapping*. Object mapping lets you define the attribute names and the data to be migrated to them. Amazon DynamoDB doesn't have a preset structure other than having a partition key and an optional sort key. If you have a non-composite primary key, AWS DMS will use it. If you have a composite primary key or you want to use a sort key, you need to define these keys and the other attributes in your target DynamoDB table.

To create an object mapping rule, you specify the `rule-type` as *object-mapping*. This rule specifies what type of object mapping you want to use.

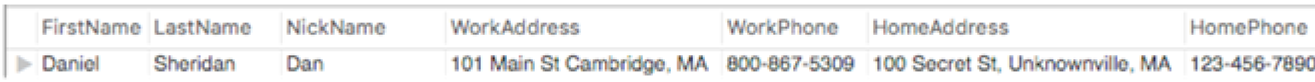
The structure for the rule is as follows:

```
{ "rules": [
    {
      "rule-type": "object-mapping",
      "rule-id": "<id>",
      "rule-name": "<name>",
      "rule-action": "<valid object-mapping rule action>",
      "object-locator": {
      "schema-name": "<case-sensitive schema name>",
      "table-name": ""
      },
      "target-table-name": "<table_name>",
      }
    }
  ]
}
```

AWS DMS currently supports *map-record-to-record* and *map-record-to-document* as the only valid values for the `rule-action` parameter. *map-record-to-record* and *map-record-to-document* specify what AWS DMS does by default to records that aren't excluded as part of the `exclude-columns` attribute list; these values don't affect the attribute-mappings in any way.

- *map-record-to-record* can be used when migrating from a relational database to DynamoDB. It uses the primary key from the relational database as the partition key in Amazon DynamoDB and creates an attribute for each column in the source database. When using `map-record-to-record`, for any column in the source table not listed in the `exclude-columns` attribute list, AWS DMS creates a corresponding attribute on the target DynamoDB instance regardless of whether that source column is used in an attribute mapping.
- *map-record-to-document* puts source columns into a single, flat, DynamoDB map on the target using the attribute name "_doc." When using `map-record-to-document`, for any column in the source table not listed in the `exclude-columns` attribute list, AWS DMS places the data into a single, flat, DynamoDB map attribute on the source called "_doc".

One way to understand the difference between the `rule-action` parameters *map-record-to-record* and *map-record-to-document* is to see the two parameters in action. For this example, assume you are starting with a relational database table row with the following structure and data:

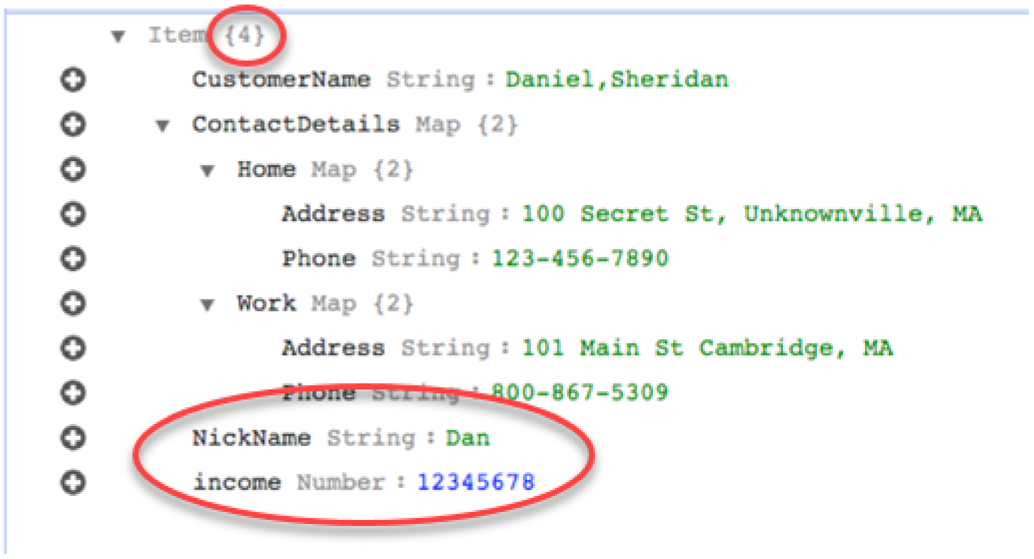


FirstName	LastName	NickName	WorkAddress	WorkPhone	HomeAddress	HomePhone
Daniel	Sheridan	Dan	101 Main St Cambridge, MA	800-867-5309	100 Secret St, Unknownville, MA	123-456-7890

To migrate this information to DynamoDB, you would 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. Note also that *NickName* and *income* are not excluded.

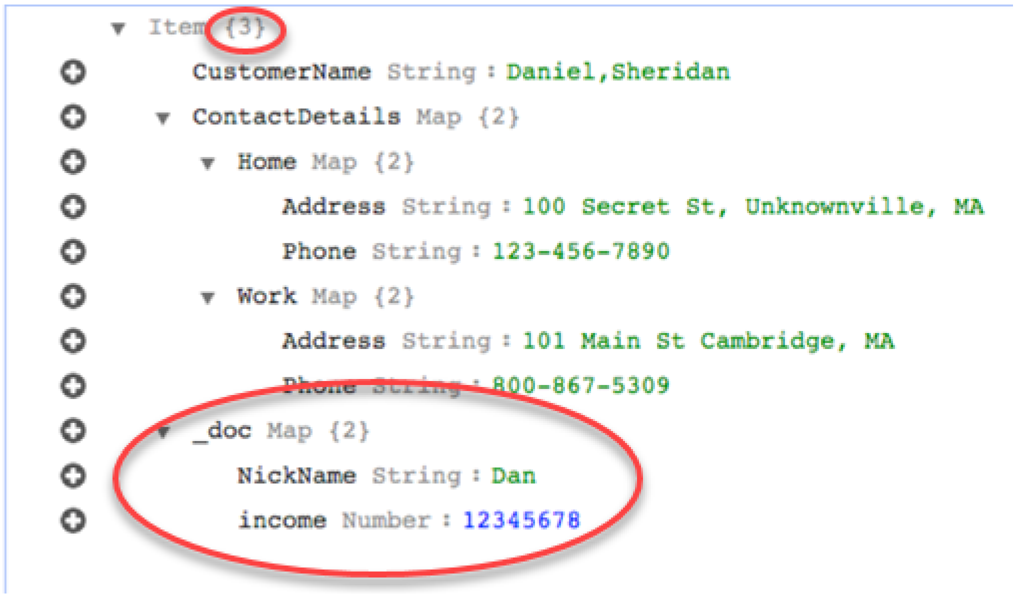
```

{
  "rules": [
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "test",
        "table-name": "customer",

      },
      "target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "exclude-columns": [
          "FirstName",
          "LastName",
          "HomeAddress",
          "HomePhone",
          "WorkAddress",
          "WorkPhone"
        ],
        "attribute-mappings": [
          {
            "target-attribute-name": "CustomerName",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": "${FirstName},${LastName}"
          },
          {
            "target-attribute-name": "ContactDetails",
            "attribute-type": "document",
            "attribute-sub-type": "dynamodb-map",
            "value": {
              "M": {
                “Home": {
                  "M": {
                    "Address": {
                      "S": "${HomeAddress}"
                    },
                    "Phone": {
                      "S": "${HomePhone}"
                    }
                  }
                },
                "Work": {
                  "M": {
                    "Address": {
                      "S": "${WorkAddress}"
                    },
                    "Phone": {
                      "S": "${WorkPhone}"
                    }
                  }
                }
              }
            }
          }
        ]
      }
    }
  ]
}
```

By using the `rule-action` parameter *map-record-to-record*, the data for *NickName* and *income* are mapped to items of the same name in the DynamoDB target.



```
▼ Item {4}
    CustomerName String : Daniel,Sheridan
  ▼ ContactDetails Map {2}
    ▼ Home Map {2}
        Address String : 100 Secret St, Unknownville, MA
        Phone String : 123-456-7890
    ▼ Work Map {2}
        Address String : 101 Main St Cambridge, MA
        Phone String : 800-867-5309
    NickName String : Dan
    income Number : 12345678
```

If you use the exact same rules but change the `rule-action` parameter to *map-record-to-document*, the columns not listed in the `exclude-columns` parameter, *NickName* and *income*, are mapped to a *_doc* item.



```
▼ Item {3}
    CustomerName String : Daniel,Sheridan
  ▼ ContactDetails Map {2}
    ▼ Home Map {2}
        Address String : 100 Secret St, Unknownville, MA
        Phone String : 123-456-7890
    ▼ Work Map {2}
        Address String : 101 Main St Cambridge, MA
        Phone String : 800-867-5309
  ▼ _doc Map {2}
      NickName String : Dan
      income Number : 12345678
```

Using Custom Condition Expressions with Object Mapping

You can use a feature of Amazon DynamoDB called conditional expressions to manipulate data that is being written to a DynamoDB table. For more information about condition expressions in DynamoDB, see Condition Expressions.

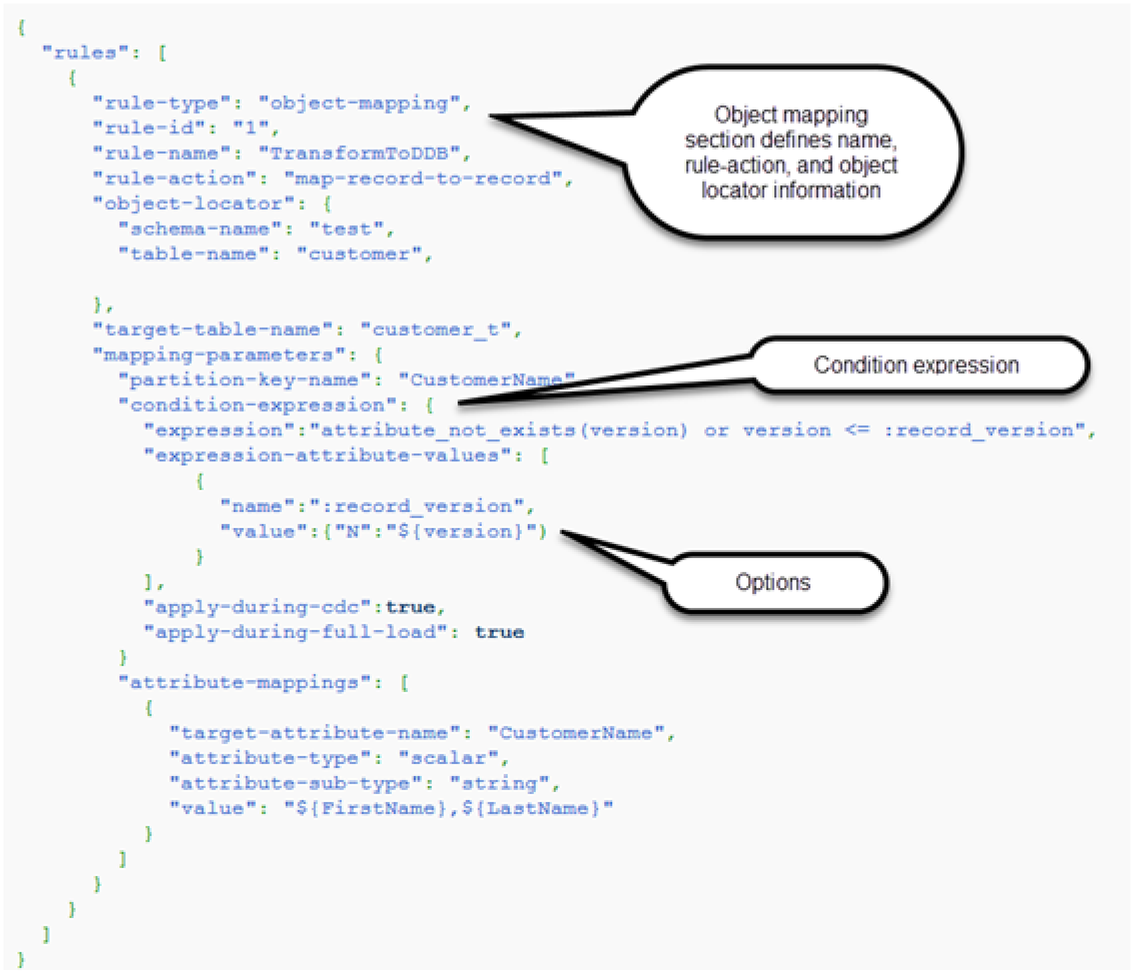
A condition expression member consists of:

- an expression (required)
- expression attribute values (optional) . Specifies a DynamoDB json structure of the attribute value
- expression attribute names (optional)
- options for when to use the condition expression (optional). The default is apply-during-cdc = false and apply-during-full-load = true

The structure for the rule is as follows:

```
"target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "condition-expression": {
          "expression":"<conditional expression>",
          "expression-attribute-values": [
              {
                "name":"<attribute name>",
                "value":<attribute value>
              }
          ],
          "apply-during-cdc":<optional Boolean value>,
          "apply-during-full-load": <optional Boolean value>
        }
```

The following sample highlights the sections used for condition expression.



```
{
  "rules": [
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "test",
        "table-name": "customer",

      },
      "target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "condition-expression": {
          "expression":"attribute_not_exists(version) or version <= :record_version",
          "expression-attribute-values": [
              {
                "name":":record_version",
                "value":{"N":"${version}"}
              }
          ],
          "apply-during-cdc":true,
          "apply-during-full-load": true
        }
        "attribute-mappings": [
          {
            "target-attribute-name": "CustomerName",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": "${FirstName},${LastName}"
          }
        ]
      }
    }
  ]
}
```

Object mapping section defines name, rule-action, and object locator information
Condition expression
Options

Using Attribute Mapping with Object Mapping

Attribute mapping lets you specify a template string using source column names to restructure data on the target. There is no formatting done other than what the user specifies in the template.

The following example shows the structure of the source database and the desired structure of the DynamoDB target. First is shown the structure of the source, in this case an Oracle database, and then the desired structure of the data in DynamoDB. The example concludes with the JSON used to create the desired target structure.

The structure of the Oracle data is as follows:

First	Last	Sto	HomeA	HomeP	WorkAddre	Work	DateOfBirth
Primary Key		N/A					
Randy	Marsh	5	221B Baker Street	1234567890	31 Spooner Street, Quahog	9876541230	02/29/1988

The structure of the DynamoDB data is as follows:

Custon	StoreId	ContactDetails	DateOfBirth
Partition Key	Sort Key	N/A	

`Randy,Marsh`	`5`	`{`
		`    "Name": "Randy",`
		`    "Home": {`
		`        "Address": "221B Baker Street",`
		`        "Phone": 1234567890`
		`    },`
		`    "Work": {`
		`        "Address": "31 Spooner Street,`
		` Quahog",`
		`        "Phone": 9876541230`
		`    }`
		`}`
			`02/29/1988`

The following JSON shows the object mapping and column mapping used to achieve the DynamoDB structure:

```

{
"rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "%"
            },
            "rule-action": "include"
        }
    ]
       {
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "owner-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}\"
                 }
              }”
          },
          {
            "target-attribute-name": "StoreId",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": {
              "${StoreId}"
            }
          }
        ]
      }
    }
  ]
}
```

Another way to use column mapping is to use DynamoDB format as your document type. The following code example uses *dynamodb-map* as the `attribute-sub-type` for attribute mapping.

```
{
  "rules": [
    {
      "rule-type": "object-mapping",
      "rule-id": "1",
      "rule-name": "TransformToDDB",
      "rule-action": "map-record-to-record",
      "object-locator": {
        "schema-name": "test",
        "table-name": "customer",

      },
      "target-table-name": "customer_t",
      "mapping-parameters": {
        "partition-key-name": "CustomerName",
        "sort-key-name": "StoreId",
        "exclude-columns": [
          "FirstName",
          "LastName",
          "HomeAddress",
          "HomePhone",
          "WorkAddress",
          "WorkPhone"
        ],
        "attribute-mappings": [
          {
            "target-attribute-name": "CustomerName",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": "${FirstName},${LastName}"
          },
          {
            "target-attribute-name": "StoreId",
            "attribute-type": "scalar",
            "attribute-sub-type": "string",
            "value": "${StoreId}"
          },
          {
            "target-attribute-name": "ContactDetails",
            "attribute-type": "document",
            "attribute-sub-type": "dynamodb-map",
            "value": {
              "M": {
                "Name": {
                  "S": "${FirstName}"
                }"Home": {
                  "M": {
                    "Address": {
                      "S": "${HomeAddress}"
                    },
                    "Phone": {
                      "S": "${HomePhone}"
                    }
                  }
                },
                "Work": {
                  "M": {
                    "Address": {
                      "S": "${WorkAddress}"
                    },
                    "Phone": {
                      "S": "${WorkPhone}"
                    }
                  }
```

```
                        }
                    }
                }
            }
        ]
    }
  }
 ]
}
```

Example 1: Using Attribute Mapping with Object Mapping

The following example migrates data from two MySQL database tables, *nfl_data* and *sport_team* , to two DynamoDB table called *NFLTeams* and *SportTeams*. The structure of the tables and the JSON used to map the data from the MySQL database tables to the DynamoDB tables are shown following.

The structure of the MySQL database table *nfl_data* is shown below:

```
mysql> desc nfl_data;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| Position      | varchar(5)  | YES  |     | NULL    |       |
| player_number | smallint(6) | YES  |     | NULL    |       |
| Name          | varchar(40) | YES  |     | NULL    |       |
| status        | varchar(10) | YES  |     | NULL    |       |
| stat1         | varchar(10) | YES  |     | NULL    |       |
| stat1_val     | varchar(10) | YES  |     | NULL    |       |
| stat2         | varchar(10) | YES  |     | NULL    |       |
| stat2_val     | varchar(10) | YES  |     | NULL    |       |
| stat3         | varchar(10) | YES  |     | NULL    |       |
| stat3_val     | varchar(10) | YES  |     | NULL    |       |
| stat4         | varchar(10) | YES  |     | NULL    |       |
| stat4_val     | varchar(10) | YES  |     | NULL    |       |
| team          | varchar(10) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
```

The structure of the MySQL database table *sport_team* is shown below:

```
mysql> desc sport_team;
+---------------------------+--------------+------+-----+---------+-----------------+
| Field                     | Type         | Null | Key | Default | Extra           |
+---------------------------+--------------+------+-----+---------+-----------------+
| id                        | mediumint(9) | NO   | PRI | NULL    | auto_increment  |
| name                      | varchar(30)  | NO   |     | NULL    |                 |
| abbreviated_name          | varchar(10)  | YES  |     | NULL    |                 |
| home_field_id             | smallint(6)  | YES  | MUL | NULL    |                 |
| sport_type_name           | varchar(15)  | NO   | MUL | NULL    |                 |
| sport_league_short_name   | varchar(10)  | NO   |     | NULL    |                 |
| sport_division_short_name | varchar(10)  | YES  |     | NULL    |                 |
```

The table mapping rules used to map the two tables to the two DynamoDB tables is shown below:

```
{
  "rules":[
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "dms_sample",
        "table-name": "nfl_data"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "selection",
      "rule-id": "2",
      "rule-name": "2",
      "object-locator": {
        "schema-name": "dms_sample",
        "table-name": "sport_team"
      },
      "rule-action": "include"
    },
    {
      "rule-type":"object-mapping",
      "rule-id":"3",
      "rule-name":"MapNFLData",
      "rule-action":"map-record-to-record",
      "object-locator":{
        "schema-name":"dms_sample",
        "table-name":"nfl_data"
      },
      "target-table-name":"NFLTeams",
      "mapping-parameters":{
        "partition-key-name":"Team",
        "sort-key-name":"PlayerName",
        "exclude-columns": [
          "player_number", "team", "Name"
        ],
        "attribute-mappings":[
          {
            "target-attribute-name":"Team",
            "attribute-type":"scalar",
            "attribute-sub-type":"string",
            "value":"${team}"
          },
          {
            "target-attribute-name":"PlayerName",
            "attribute-type":"scalar",
            "attribute-sub-type":"string",
            "value":"${Name}"
          },
          {
            "target-attribute-name":"PlayerInfo",
            "attribute-type":"scalar",
            "attribute-sub-type":"string",
            "value":"{\"Number\": \"${player_number}\",\"Position\": \"${Position}\",
\"Status\": \"${status}\",\"Stats\": {\"Stat1\": \"${stat1}:${stat1_val}\",\"Stat2\":
 \"${stat2}:${stat2_val}\",\"Stat3\": \"${stat3}:${
stat3_val}\",\"Stat4\": \"${stat4}:${stat4_val}\"}"
          }
        ]
      }
    },
    {
      "rule-type":"object-mapping",
      "rule-id":"4",
      "rule-name":"MapSportTeam",
      "rule-action":"map-record-to-record",
      "object-locator":{
        "schema-name":"dms_sample",
        "table-name":"sport_team"
      },
      "target-table-name":"SportTeams",
      "mapping-parameters":{
        "partition-key-name":"TeamName",
        "exclude-columns": [
          "name", "id"
        ],
        "attribute-mappings":[
          {
            "target-attribute-name":"TeamName",
            "attribute-type":"scalar",
            "attribute-sub-type":"string",
            "value":"${name}"
          },
          {
            "target-attribute-name":"TeamInfo",
            "attribute-type":"scalar",
            "attribute-sub-type":"string",
            "value":"{\"League\": \"${sport_league_short_name}\",\"Division\":
 \"${sport_division_short_name}\"}"
          }
        ]
      }
    }
  ]
}
```

The sample output for the *NFLTeams* DynamoDB table is shown below:

```
  "PlayerInfo": "{\"Number\": \"6\",\"Position\": \"P\",\"Status\": \"ACT\",\"Stats\":
 {\"Stat1\": \"PUNTS:73\",\"Stat2\": \"AVG:46\",\"Stat3\": \"LNG:67\",\"Stat4\": \"IN
 20:31\"}",
  "PlayerName": "Allen, Ryan",
  "Position": "P",
  "stat1": "PUNTS",
  "stat1_val": "73",
  "stat2": "AVG",
  "stat2_val": "46",
  "stat3": "LNG",
  "stat3_val": "67",
  "stat4": "IN 20",
  "stat4_val": "31",
  "status": "ACT",
  "Team": "NE"
}
```

The sample output for the SportsTeams *DynamoDB* table is shown below:

```
{
  "abbreviated_name": "IND",
  "home_field_id": 53,
  "sport_division_short_name": "AFC South",
  "sport_league_short_name": "NFL",
  "sport_type_name": "football",
  "TeamInfo": "{\"League\": \"NFL\",\"Division\": \"AFC South\"}",
  "TeamName": "Indianapolis Colts"
}
```

Working with AWS DMS Tasks

An AWS Database Migration Service (AWS DMS) task is where all the work happens. You use tasks to specify what tables and schemas to use for your migration and to apply specific replication requirements to a database migration. To create a replication task, you must have at least one source and one target database to work with AWS DMS. You must also have a replication instance set up. Then, you can configure the task using task settings and specify tables to be replicated.

Once 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 that the CDC replication process be started. You can monitor, stop, or restart replication tasks using the AWS DMS console, AWS CLI, or AWS DMS API.

The following are features you can use when performing an AWS DMS task

Task	Relevant Documentation
Choosing a Task Migration Method When you create a task, you specify the migration method, which includes either a full load of the existing data, a full load of the data plus ongoing changes, or just a replication of the ongoing changes.	Migration Methods for AWS Database Migration Service (p. 124)
Running Multiple Tasks In some migration scenarios, you might have to create several migration tasks.	Creating Multiple Tasks (p. 123)
Modifying a Task When a task is stopped, you can modify the settings for the task.	Modifying a Task (p. 121)
Reloading Tables During a Task You can reload a table during a task if an error occurs during the task.	Reloading Tables During a Task (p. 122)
Using Table Mapping	Selection Rules

Task	Relevant Documentation
Table mapping uses several types of rules to specify the data source, source schema, data, and any transformations that should occur during the task.	Selection Rules and Actions (p. 139) Transformation Rules Transformation Rules and Actions (p. 140)
Applying Filters You can use source filters to limit the number and type of records transferred from your source to your target. For example, you can specify that only employees with a location of headquarters are moved to the target database. You apply filters on a column of data.	Using Source Filters (p. 145)
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.	Task Settings for AWS Database Migration Service Tasks (p. 124)
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 Database Migration Service Tasks (p. 149)

For more information, see the following topics:

Topics

Modifying a Task

You can modify a task if you need to change the task settings, table mapping, or other settings. You modify a task in the DMS console by selecting the task and choosing **Modify**. You can also use the AWS CLI or AWS DMS API command `ModifyReplicationTask`.

There are a few limitations to modifying a task. These include:

- You cannot modify the source or target endpoint of a task.
- You cannot change the migration type from CDC to either Full_Load or Full_Load_and_CDC.
- You cannot change the migration type from Full Load to either CDC or Full_Load_and_CDC.
- A task that have been run must have a status of **Stopped** or **Failed** to be modified.

Reloading Tables During a Task

While a task is running, you can reload a target database table using data from the source. You may want to reload a table if, during the task, an error occurs or data changes due to partition operations (for example, when using Oracle). You can reload up to 10 tables from a task.

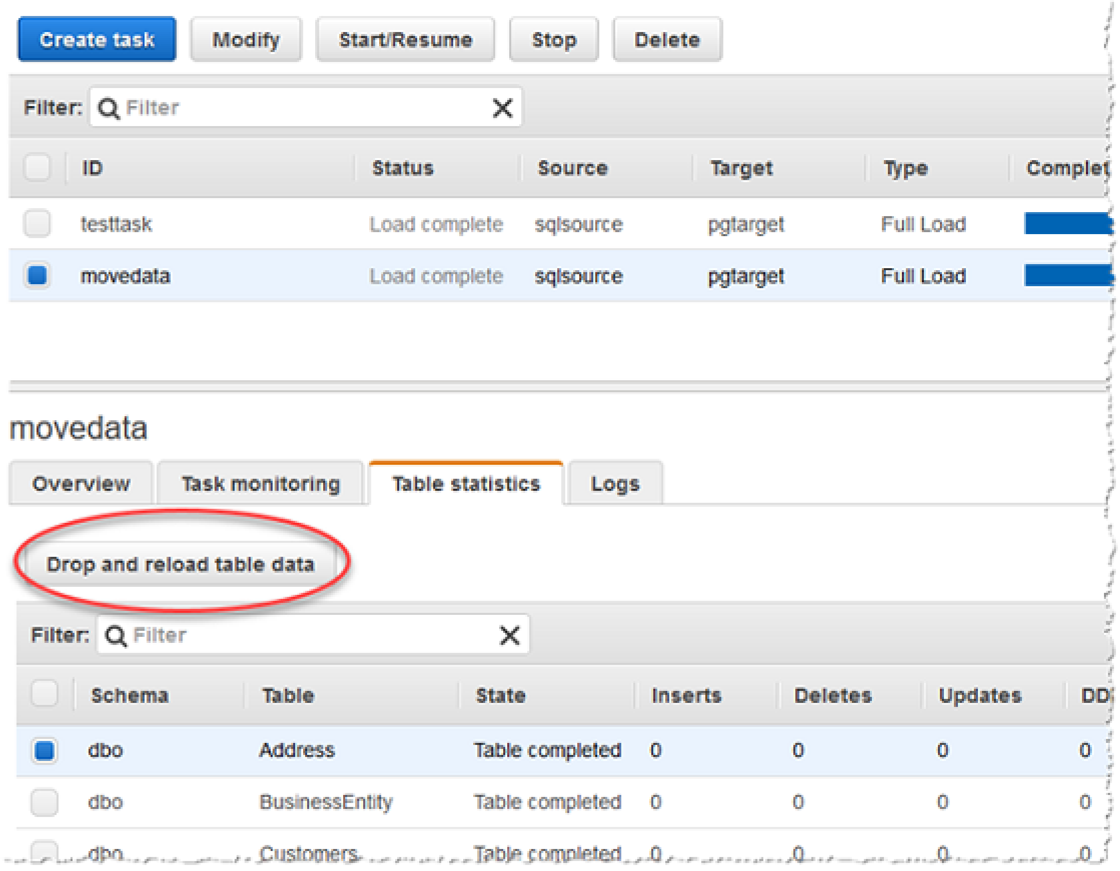
To reload a table, the following conditions must apply:

- The task must be running.
- The migration method for the task must be either Full Load or Full Load with CDC.
- Duplicate tables are not allowed.

AWS Management Console

To reload a table using the AWS DMS console

1. Sign in to the AWS Management Console and choose AWS DMS. Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see IAM Permissions Needed to Use AWS DMS (p. 43).
2. Select **Tasks** from the navigation pane.
3. Select the running task that has the table you want to reload.
4. Click the **Table Statistics** tab.

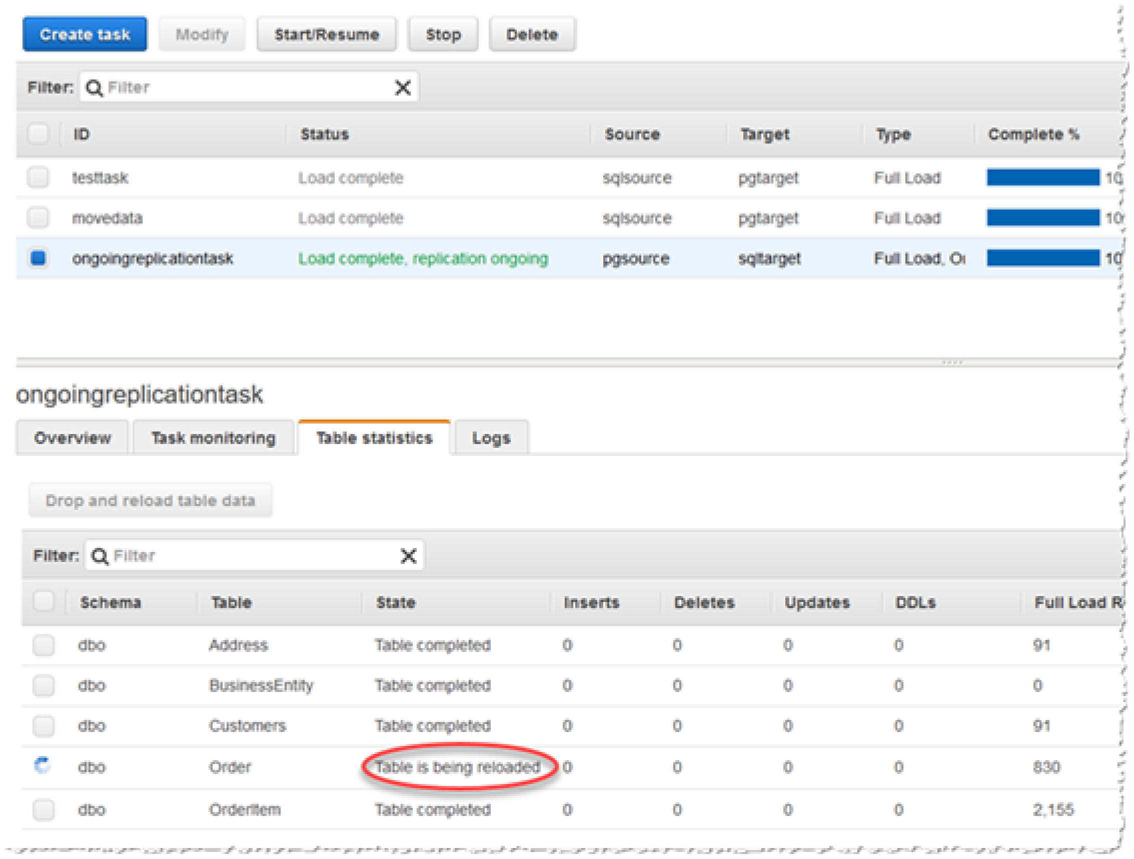


	ID	Status	Source	Target	Type	Complet
☐	testtask	Load complete	sqlsource	pgtarget	Full Load	
■	movedata	Load complete	sqlsource	pgtarget	Full Load	

	Schema	Table	State	Inserts	Deletes	Updates	DD
■	dbo	Address	Table completed	0	0	0	0
☐	dbo	BusinessEntity	Table completed	0	0	0	0
☐	dbo	Customers	Table completed	0	0	0	0

5. Select the table you want to reload. Note that if the task is no longer running, you will not be able to reload the table.
6. Choose **Drop and reload table data**.

When AWS DMS is preparing to reload a table, the console changes the table status to **The table is being reloaded**.



	ID	Status	Source	Target	Type	Complete %
☐	testtask	Load complete	sqlsource	pgtarget	Full Load	10
☐	movedata	Load complete	sqlsource	pgtarget	Full Load	10
■	ongoingreplicationtask	Load complete, replication ongoing	pgsource	sqltarget	Full Load, O	10

ongoingreplicationtask

	Schema	Table	State	Inserts	Deletes	Updates	DDLs	Full Load R
☐	dbo	Address	Table completed	0	0	0	0	91
☐	dbo	BusinessEntity	Table completed	0	0	0	0	0
☐	dbo	Customers	Table completed	0	0	0	0	91
	dbo	Order	Table is being reloaded	0	0	0	0	830
☐	dbo	OrderItem	Table completed	0	0	0	0	2,155

Creating Multiple Tasks

In some migration scenarios, you might have to create several migration tasks. Note that tasks work independently and can run concurrently; each task has its own initial load, CDC, and log reading process. Tables that are related through data manipulation language (DML) must be part of the same task.

Some reasons to create multiple tasks for a migration include the following:

- The target tables for the tasks reside on different databases, such as when you are fanning out or breaking a system into multiple systems.
- You want to break the migration of a large table into multiple tasks by using filtering.

Note
Because each task has its own change capture and log reading process, changes are *not* coordinated across tasks. Therefore, when using multiple tasks to perform a migration, make sure that source transactions are wholly contained within a single task.

Migration Methods for AWS Database Migration Service

AWS Database Migration Service can migrate your data in several ways:

- **Migrating Data to the Target Database** – This process creates files or tables in the target database, automatically defines the metadata that is required at the target, and populates the tables with data from the source. The data from the tables is loaded in parallel for improved efficiency. This process is the **Migrate existing data** option in the AWS console and is called `Full Load` in the API.
- **Capturing Changes During Migration** – This process captures changes to the source database that occur while the data is being migrated from the source to the target. When the migration of the originally requested data has completed, the change data capture (CDC) process then applies the captured changes to the target database. Changes are captured and applied as units of single committed transactions, and several different target tables can be updated as a single source commit. This approach guarantees transactional integrity in the target database. This process is the **Migrate existing data and replicate ongoing changes** option in the AWS console and is called `full-load-and-cdc` in the API.
- **Replicating Only Data Changes on the Source Database** – This process reads the recovery log file of the source database management system (DBMS) and groups together the entries for each transaction. If AWS DMS can't apply changes to the target within a reasonable time (for example, if the target is not accessible), 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.

Note
If you restart a task, any tables that have not completed their initial load are restarted.

Task Settings for AWS Database Migration Service Tasks

Each task has settings that you can configure according to the needs of your database migration. You create these settings in a JSON file or, with some settings, you can specify the settings using the AWS DMS console.

There are several main types of task settings:

- Target Metadata Task Settings (p. 125)
- Full Load Task Settings (p. 125)
- Logging Task Settings (p. 126)
- Control Table Task Settings (p. 127)
- Change Processing Tuning Settings (p. 130)
- Stream Buffer Task Settings (p. 129)
- Change Processing DDL Handling Policy Task Settings (p. 131)
- Error Handling Task Settings (p. 131)

To see an example JSON file with sample task settings, see Saving Task Settings (p. 133).

Target Metadata Task Settings

Target metadata settings include the following:

- `TargetSchema` – The target table schema name. If this metadata option is empty, the schema from the source table is used. AWS DMS automatically adds the owner prefix for the target database to all tables if no source schema is defined. This option should be left empty for MySQL-type target endpoints.
- `LOB settings` – Settings that determine how large objects (LOBs) are managed. If you set `SupportLobs=true`, you must set one of the following to `true`:
  - `FullLobMode` – If you set this option to `true`, then you must enter a value for the `LobChunkSize` option. Enter the size, in kilobytes, of the LOB chunks to use when replicating the data to the target. The `FullLobMode` option works best for very large LOB sizes but tends to cause slower loading.
  - `LimitedSizeLobMode` – If you set this option to `true`, then you must enter a value for the `LobMaxSize` option. Enter the maximum size, in kilobytes, for an individual LOB.
- `LoadMaxFileSize` – An option for PostgreSQL and MySQL target endpoints that defines the maximum size on disk of stored, unloaded data, such as .csv files. This option overrides the connection attribute. You can provide values from 0, which indicates that this option doesn't override the connection attribute, to 100,000 KB.
- `BatchApplyEnabled` – Determines if each transaction is applied individually or if changes are committed in batches. The default value is `false`.

  If set to `true`, AWS DMS commits changes in batches by a pre-processing action that groups the transactions into batches in the most efficient way. Setting this value to `true` can affect transactional integrity, so you must select `BatchApplyPreserveTransaction` in the `ChangeProcessingTuning` section to specify how the system handles referential integrity issues.

  If set to `false`, AWS DMS applies each transaction individually, in the order it is committed. In this case, strict referential integrity is ensured for all tables.

  When LOB columns are included in the replication, `BatchApplyEnabled`can only be used in **Limited-size LOB mode**.
- `ParallelLoadThreads` – Specifies the number of threads AWS DMS uses to load each table into the target database. The maximum value for a MySQL target is 16; the maximum value for a DynamoDB target is 32. The maximum limit can be increated upon request.
- `ParallelLoadBufferSize` – Specifies the maximum number of records to store in the buffer used by the parallel load threads to load data to the target. The default value is 50. Maximum value is 1000. This field is currently only valid when DynamoDB is the target. This parameter should be used in conjunction with `ParallelLoadThreads` and is valid only when ParallelLoadThreads > 1.

Full Load Task Settings

Full load settings include the following:

- To indicate how to handle loading the target at full-load startup, specify one of the following values for the `TargetTablePrepMode` option:
  - `DO_NOTHING` – Data and metadata of the existing target table are not affected.
  - `DROP_AND_CREATE` – The existing table is dropped and a new table is created in its place.
  - `TRUNCATE_BEFORE_LOAD` – Data is truncated without affecting the table metadata.
- To delay primary key or unique index creation until after full load completes, set the `CreatePkAfterFullLoad` option.
  When this option is selected, you cannot resume incomplete full load tasks.

- For full load and CDC-enabled tasks, you can set the following `Stop task after full load completes` options:
  - `StopTaskCachedChangesApplied` – Set this option to `true` to stop a task after a full load completes and cached changes are applied.
  - `StopTaskCachedChangesNotApplied` – Set this option to `true` to stop a task before cached changes are applied.
- `MaxFullLoadSubTasks` – Set this option to indicate the maximum number of tables to load in parallel. The default is 0; the maximum value is 49.
- To set the number of seconds that AWS DMS waits for transactions to close before beginning a full-load operation, if transactions are open when the task starts, set the `TransactionConsistencyTimeout` option. The default value is 600 (10 minutes). AWS DMS begins the full load after the timeout value is reached, even if there are open transactions.
- To indicate the maximum number of events that can be transferred together, set the `CommitRate` option.

Logging Task Settings

Logging task settings are written to a JSON file and they let you specify which component activities are logged and what amount of information is written to the log. The logging feature uses Amazon CloudWatch to log information during the migration process.

There are several ways to enable Amazon CloudWatch logging. You can select the `EnableLogging` option on the AWS Management Console when you create a migration task or set the `EnableLogging` option to `true` when creating a task using the AWS DMS API. You can also specify `"EnableLogging": true` in the JSON of the logging section of task settings.

You can specify logging for the following component activities:

- SOURCE_UNLOAD — Data is unloaded from the source database.
- SOURCE_CAPTURE — Data is captured from the source database.
- TARGET_LOAD — Data is loaded into the target database.
- TARGET_APPLY — Data and DDL are applied to the target database.
- TASK_MANAGER — The task manager triggers an event.

Once you specify a component activity, you can then specify the amount of information that is logged. The following list is in order from the lowest level of information to the highest level of information. The higher levels always include information from the lower levels. These severity values include:

- LOGGER_SEVERITY_ERROR — Error messages are written to the log.
- LOGGER_SEVERITY_WARNING — Warnings and error messages are written to the log.
- LOGGER_SEVERITY_INFO — Informational messages, warnings, and error messages are written to the log.
- LOGGER_SEVERITY_DEFAULT — Debug messages, informational messages, warnings, and error messages are written to the log.
- LOGGER_SEVERITY_DEBUG — Debug messages, informational messages, warnings, and error messages are written to the log.
- LOGGER_SEVERITY_DETAILED_DEBUG — All information is written to the log.

For example, the following JSON section gives task settings for logging for all component activities:

```
…
  "Logging": {
    "EnableLogging": true,
    "LogComponents": [{
        "Id": "SOURCE_UNLOAD",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
    },{
        "Id": "SOURCE_CAPTURE",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
    },{
        "Id": "TARGET_LOAD",
        "Severity": "LOGGER_SEVERITY_DEFAULT"
    },{
        "Id": "TARGET_APPLY",
        "Severity": "LOGGER_SEVERITY_INFO"
    },{
        "Id": "TASK_MANAGER",
        "Severity": "LOGGER_SEVERITY_DEBUG"
    }]
  },
…
```

Control Table Task Settings

Control tables provide information about the AWS DMS task, as well as useful statistics that you can use to plan and manage both the current migration task and future tasks. You can apply these task settings in a JSON file or using the **Advanced Settings** link on the **Create task** page in the AWS DMS console. In addition to the **Apply Exceptions (dmslogs.awsdms_apply_exceptions)** table, which is always created, you can choose to create additional tables including the following:

- **Replication Status (dmslogs.awsdms_status)** – This table provides details about the current task including task status, amount of memory consumed by the task, number of changes not yet applied to the target, and the position in the source database from which AWS DMS is currently reading. It also indicates if the task is a full load or change data capture (CDC).
- **Suspended Tables (dmslogs.awsdms_suspended_tables)** – This table provides a list of suspended tables as well as the reason they were suspended.
- **Replication History (dmslogs.awsdms_history)** – This table provides information about the replication history including the number and volume of records processed during the task, latency at the end of a CDC task, and other statistics.

The **Apply Exceptions (dmslogs.awsdms_apply_exceptions)** table contains the following parameters:

Column	Type	Description
TASK_NAME	nvchar	The name of the AWS DMS task.
TABLE_OWNER	nvchar	The table owner.
TABLE_NAME	nvchar	The table name.
ERROR_TIME	timestamp	The time the exception (error) occurred.
STATEMENT	nvchar	The statement that was being run when the error occurred.
ERROR	nvchar	The error name and description.

The **Replication History (dmslogs.awsdms_history)** table contains the following parameters:

Column	Type	Description
SERVER_NAME	nvchar	The name of the machine where the replication task is running.
TASK_NAME	nvchar	The name of the AWS DMS task.
TIMESLOT_TYPE	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.
TIMESLOT_LATENCY	int	The target latency at the end of the time slot. This value is only applicable to CDC time slots.
RECORDS	int	The number of records processed during the time slot.
TIMESLOT_VOLUME	int	The volume of data processed in MB.

The **Replication Status (dmslogs.awsdms_status)** table contains the current status of the task and the target database. It has the following settings:

Column	Type	Description
SERVER_NAME	nvchar	The name of the machine where the replication task is running.
TASK_NAME	nvchar	The name of the AWS DMS task.
TASK_STATUS	varchar	One of the following values: • FULL LOAD • CHANGE PROCESSING (CDC) Task status is set to FULL LOAD as long as there is at least one table in full load. After all tables have been loaded, the task status changes to CHANGE PROCESSING if CDC is enabled.
STATUS_TIME	timestamp	The timestamp of the task status.
PENDING_CHANGES	int	The number of change records that were not applied to the target.

Column	Type	Description
DISK_SWAP_SIZE	int	The amount of disk space used by old or offloaded transactions.
TASK_MEMORY	int	Current memory used, in MB.
SOURCE_CURRENT _POSITION	varchar	The position in the source database that AWS DMS is currently reading from.
SOURCE_CURRENT _TIMESTAMP	timestamp	The timestamp in the source database that AWS DMS is currently reading from.
SOURCE_TAIL _POSITION	varchar	The position of the oldest start transaction that is not committed. This value is the newest position that you can revert to without losing any changes.
SOURCE_TAIL _TIMESTAMP	timestamp	The timestamp of the oldest start transaction that is not committed. This value is the newest timestamp that you can revert to without losing any changes..
SOURCE_TIMESTAMP _APPLIED	timestamp	The timestamp of the last transaction commit. In a bulk apply process, this value is the timestamp for the commit of the last transaction in the batch.

Additional control table settings include the following:

- `ControlSchema` – Use this option to indicate the database schema name for the AWS DMS target Control Tables. If you do not enter any information in this field, then the tables are copied to the default location in the database.
- `HistoryTimeslotInMinutes` – Use this option to indicate the length of each time slot in the Replication History table. The default is 5 minutes.

Stream Buffer Task Settings

You can set stream buffer settings using the AWS CLI, include the following:

- `StreamBufferCount` – Use this option to specify the number of data stream buffers for the migration task. The default stream buffer number is 3. Increasing the value of this setting may 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 or if you receive a message in the log files stating 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. Value is in MB, and can be from 1 to 8. The default value is 5. You may need to increase this when working with a very large number of tables, such as tens of thousands of tables.

Change Processing Tuning Settings

The following settings determine how AWS DMS handles changes for target tables during change data capture (CDC). Several of these settings depend on the value of the target metadata parameter `BatchApplyEnabled`. For more information on the `BatchApplyEnabled` parameter, see Target Metadata Task Settings (p. 125).

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`.

  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 will be applied to the target in a single batch.
- `BatchApplyTimeoutMin` – Sets the minimum amount of time in seconds that AWS DMS waits between each application of batch changes. The default value is 1.
- `BatchApplyTimeoutMax` – Sets the maximum amount of time in seconds that AWS DMS waits between each application of batch changes before timing out. The default value is 30.
- `BatchApplyMemoryLimit` – Sets the maximum amount of memory in (MB) to use for pre-processing in **Batch optimized apply mode**. The default value is 500.
- `BatchSplitSize` – Sets the number of changes applied in a single change processing statement. Select the check box and then optionally change the default value. The default value is 10,000. A value of 0 means there is no limit applied.

The following settings apply only when the target metadata parameter `BatchApplyEnabled` is set to `false`.

- `MinTransactionSize` – Sets the minimum number of changes to include in each transaction. The default value is 1000.
- `CommitTimeout` – Sets the maximum time in seconds for AWS DMS to collect transactions in batches before declaring a timeout. The default value is 1.
- `HandleSourceTableAltered` – Set this option to `true` to alter the target table when the source table is altered.

AWS DMS attempts to keep transaction data in memory until the transaction is fully committed to the source and/or the target. However, transactions that are larger than the allocated memory or that are not committed within the specified time limit are written to disk.

The following settings apply to change processing tuning regardless of the change processing mode.

- `MemoryLimitTotal` – Sets the maximum size (in MB) that all transactions can occupy in memory before being written to disk. The default value is 1024.
- `MemoryKeepTime` – Sets the maximum time in seconds that each transaction can stay in memory before being written to disk. The duration is calculated from the time that AWS DMS started capturing the transaction. The default value is 60.

- `StatementCacheSize` – Sets the maximum number of prepared statements to store on the server for later execution when applying changes to the target. The default value is 50. The maximum value is 200.

Change Processing DDL Handling Policy Task Settings

The following settings determine how AWS DMS handles DDL changes for target tables during change data capture (CDC). Change processing DDL handling policy settings include the following:

- `HandleSourceTableDropped` – Set this option to `true` to drop the target table when the source table is dropped
- `HandleSourceTableTruncated` – Set this option to `true` to truncate the target table when the source table is truncated
- `HandleSourceTableAltered` – Set this option to `true` to alter the target table when the source table is altered.

Error Handling Task Settings

You can set the error handling behavior of your replication task during change data capture (CDC) using the following settings:

- `DataErrorPolicy` – Determines the action AWS DMS takes when there is an error related to data processing at the record level. Some examples of data processing errors include conversion errors, errors in transformation, and bad data. The default is `LOG_ERROR`.
  - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `DataErrorEscalationCount` property is incremented so that if you set a limit on errors for a table, this error will count toward that limit.
  - `LOG_ERROR` – The task continues and the error is written to the task log.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.
- `DataTruncationErrorPolicy` – Determines the action AWS DMS takes when data is truncated. The default is `LOG_ERROR`.
  - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `DataErrorEscalationCount` property is incremented so that if you set a limit on errors for a table, this error will count toward that limit.
  - `LOG_ERROR` – The task continues and the error is written to the task log.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.
- `DataErrorEscalationPolicy` – Determines the action AWS DMS takes when the maximum number of errors (set in the `DataErrorsEscalationCount` parameter) is reached. The default is `SUSPEND_TABLE`.
  - `LOG_ERROR` – The task continues and the error is written to the task log.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.
- `DataErrorEscalationCount` – Sets the maximum number of errors that can occur to the data for a specific record. When this number is reached, the data for the table that contains the error record is handled according to the policy set in the `DataErrorEscalationCount`. The default is 0.

- `TableErrorPolicy` – Determines the action AWS DMS takes when an error occurs when processing data or metadata for a specific table. This error only applies to general table data and is not an error that relates to a specific record. The default is `SUSPEND_TABLE`.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.
- `TableErrorEscalationPolicy` – Determines the action AWS DMS takes when the maximum number of errors (set using the `TableErrorEscalationCount` parameter). The default and only user setting is `STOP_TASK`, where the task is stopped and manual intervention is required.
- `TableErrorEscalationCount` – The maximum number of errors that can occur to the general data or metadata for a specific table. When this number is reached, the data for the table is handled according to the policy set in the `TableErrorEscalationPolicy`. The default is 0.
- `RecoverableErrorCount` – The maximum number of attempts made to restart a task when an environmental error occurs. After the system attempts to restart the task the designated number of times, the task is stopped and manual intervention is required. Set this value to -1 to attempt a restart six times. Set this value to 0 to never attempt to restart a task. The default is 1.
- `RecoverableErrorInterval` – The number of seconds that AWS DMS waits between attempts to restart a task. The default is 5.
- `RecoverableErrorThrottling` – When enabled, the interval between attempts to restart a task is increased each time a restart is attempted. The default is `true`.
- `RecoverableErrorThrottlingMax` – The maximum number of seconds that AWS DMS waits between attempts to restart a task if `RecoverableErrorThrottling` is enabled. The default is 1800.
- `ApplyErrorDeletePolicy` – Determines what action AWS DMS takes when there is a conflict with a DELETE operation. The default is `IGNORE_RECORD`.
  - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `ApplyErrorEscalationCount` property is incremented so that if you set a limit on errors for a table, this error will count toward that limit.
  - `LOG_ERROR` – The task continues and the error is written to the task log.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.
- `ApplyErrorInsertPolicy` – Determines what action AWS DMS takes when there is a conflict with an INSERT operation. The default is `LOG_ERROR`.
  - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `ApplyErrorEscalationCount` property is incremented so that if you set a limit on errors for a table, this error will count toward that limit.
  - `LOG_ERROR` – The task continues and the error is written to the task log.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.
  - `UPDATE_RECORD` – If there is an existing target record with the same primary key as the inserted source record, the target record is updated.
- `ApplyErrorUpdatePolicy` – Determines what action AWS DMS takes when there is a conflict with an UPDATE operation. The default is `LOG_ERROR`.
  - `IGNORE_RECORD` – The task continues and the data for that record is ignored. The error counter for the `ApplyErrorEscalationCount` property is incremented so that if you set a limit on errors for a table, this error will count toward that limit.
  - `LOG_ERROR` – The task continues and the error is written to the task log.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.

  - `UPDATE_RECORD` – If the target record is missing, the missing target record will be inserted into the target table. Selecting this option requires full supplemental logging to be enabled for all the source table columns when Oracle is the source database.
- `ApplyErrorEscalationPolicy` – Determines what action AWS DMS takes when the maximum number of errors (set using the `ApplyErrorsEscalationCount` parameter) is reached.
  - `LOG_ERROR` – The task continues and the error is written to the task log.
  - `SUSPEND_TABLE` – The task continues but data from the table with the error record is moved into an error state and the data is not replicated.
  - `STOP_TASK` – The task stops and manual intervention is required.
- `ApplyErrorEscalationCount` – Sets the maximum number of APPLY conflicts that can occur for a specific table during a change process operation. When this number is reached, the data for the table is handled according to the policy set in the `ApplyErrorEscalationPolicy` parameter. The default is 0.
- `FulloadIgnoreConflicts` – Determines if AWS DMS loads the conflicting data when carrying out a full-load operation after the change processing is complete.

Saving Task Settings

You can save the settings for a task as a JSON file, in case you want to reuse the settings for another task.

For example, the following JSON file contains settings saved for a task:

```

{
  "TargetMetadata": {
    "TargetSchema": "",
    "SupportLobs": true,
    "FullLobMode": false,
    "LobChunkSize": 64,
    "LimitedSizeLobMode": true,
    "LobMaxSize": 32,
    "BatchApplyEnabled": true
  },
  "FullLoadSettings": {
    "TargetTablePrepMode": "DO_NOTHING",
    "CreatePkAfterFullLoad": false,
    "StopTaskCachedChangesApplied": false,
    "StopTaskCachedChangesNotApplied": false,
    "MaxFullLoadSubTasks": 8,
    "TransactionConsistencyTimeout": 600,
    "CommitRate": 10000
  },
  "Logging": {
    "EnableLogging": false
  },
  "ControlTablesSettings": {
    "ControlSchema":"",
    "HistoryTimeslotInMinutes":5,
    "HistoryTableEnabled": false,
    "SuspendedTablesTableEnabled": false,
    "StatusTableEnabled": false
  },
  "StreamBufferSettings": {
    "StreamBufferCount": 3,
    "StreamBufferSizeInMB": 8
  },
  "ChangeProcessingTuning": {
    "BatchApplyPreserveTransaction": true,
    "BatchApplyTimeoutMin": 1,
    "BatchApplyTimeoutMax": 30,
    "BatchApplyMemoryLimit": 500,
    "BatchSplitSize": 0,
    "MinTransactionSize": 1000,
    "CommitTimeout": 1,
    "MemoryLimitTotal": 1024,
    "MemoryKeepTime": 60,
    "StatementCacheSize": 50
  },
  "ChangeProcessingDdlHandlingPolicy": {
    "HandleSourceTableDropped": true,
    "HandleSourceTableTruncated": true,
    "HandleSourceTableAltered": true
  },
  "ErrorBehavior": {
    "DataErrorPolicy": "LOG_ERROR",
    "DataTruncationErrorPolicy":"LOG_ERROR",
    "DataErrorEscalationPolicy":"SUSPEND_TABLE",
    "DataErrorEscalationCount": 50,
    "TableErrorPolicy":"SUSPEND_TABLE",
    "TableErrorEscalationPolicy":"STOP_TASK",
    "TableErrorEscalationCount": 50,
    "RecoverableErrorCount": 0,
    "RecoverableErrorInterval": 5,
    "RecoverableErrorThrottling": true,
    "RecoverableErrorThrottlingMax": 1800,
    "ApplyErrorDeletePolicy":"IGNORE_RECORD",
    "ApplyErrorInsertPolicy":"LOG_ERROR",
    "ApplyErrorUpdatePolicy":"LOG_ERROR",
    "ApplyErrorEscalationPolicy":"LOG_ERROR",
    "ApplyErrorEscalationCount": 0,
    "FullLoadIgnoreConflicts": true
  }
}
```

Using Table Mapping with an Task to Select and Filter Data

Table mapping uses several types of rules to specify the data source, source schema, data, and any transformations that should occur during the task. You can use table mapping to specify individual tables in a database to migrate and the schema to use for the migration. In addition, you can use filters to specify what data from a given table column you want replicated and you can use transformations to modify the data written to the target database.

Selection and Transformation Table Mapping using the AWS Console

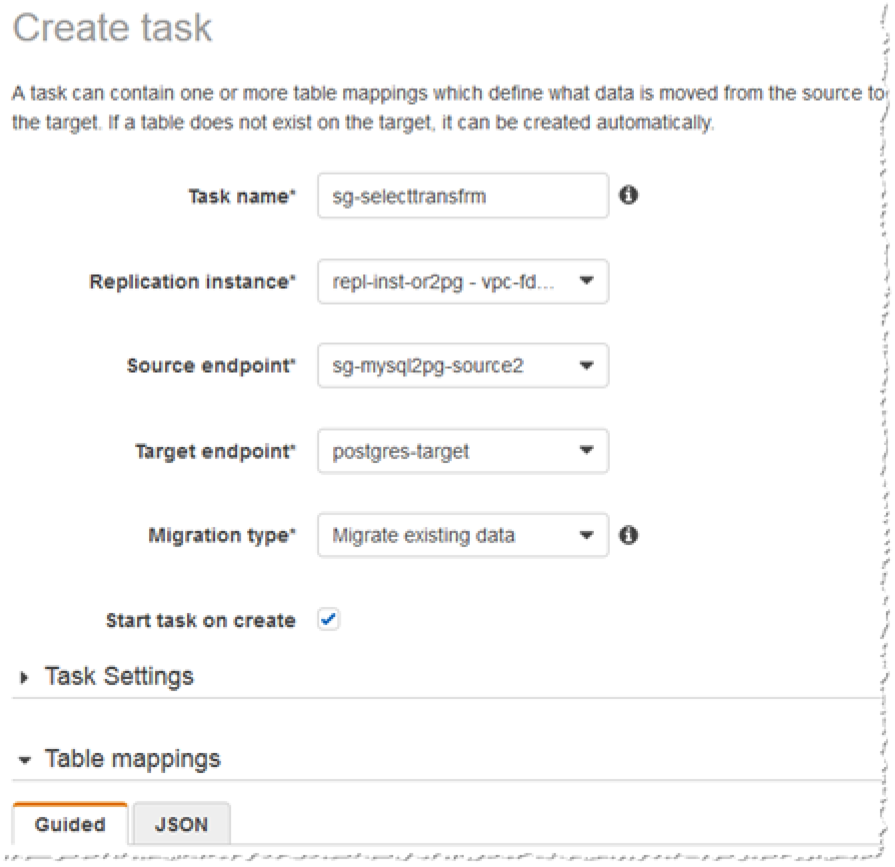
You can use the AWS console to perform table mapping, including specifying table selection and transformations. In the AWS Console user interface, you use the **Where** section to specify the schema, table, and action (include or exclude). You use the **Filter** section to specify the column name in a table and the conditions you want to apply to the replication task. Together these two actions create a selection rule.

Transformations can be included in a table mapping after you have specified at least one selection rule. Transformations can be used to rename a schema or table, add a prefix or suffix to a schema or table, or remove a table column.

The following example shows how to set up selection rules for a table called Customers in a schema called EntertainmentAgencySample. Note that the **Guided** tab, where you create selection rules and transformations, only appears when you have a source endpoint that has schema and table information.

To specify a table selection, filter criteria, and transformations using the AWS console

1. Sign in to the AWS Management Console and choose AWS DMS. Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see IAM Permissions Needed to Use AWS DMS (p. 43).
2. On the **Dashboard** page, choose **Tasks**.
3. Select **Create Task**.
4. Enter the task information, including **Task name**, **Replication instance**, **Source endpoint**, **Target endpoint**, and **Migration type**. Select **Guided** from the **Table mappings** section.

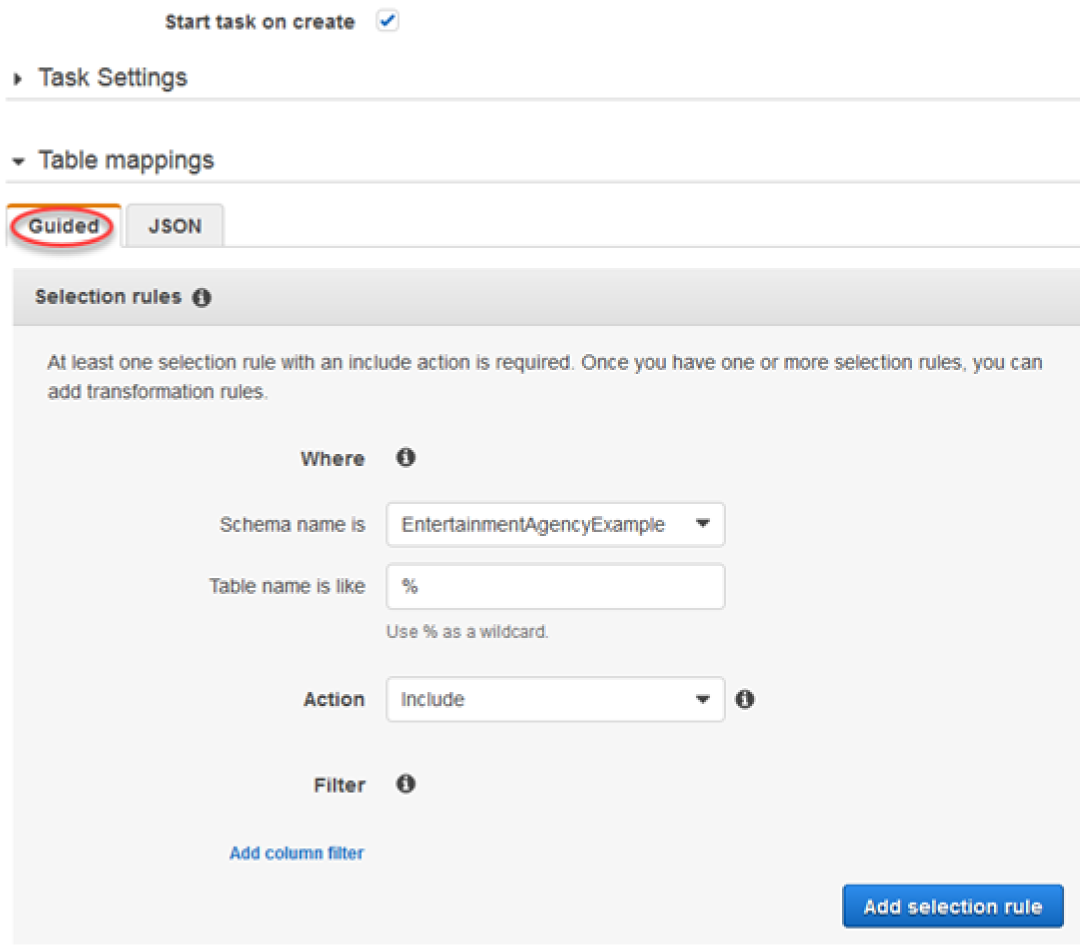


Create task

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

Task name*	sg-selecttransfrm
Replication instance*	repl-inst-or2pg - vpc-fd...
Source endpoint*	sg-mysql2pg-source2
Target endpoint*	postgres-target
Migration type*	Migrate existing data
Start task on create	✓

5. In the **Table mapping** section, select the schema name and table name. You can use the "%" as a wildcard value when specifying the table name. Specify the action to be taken, to include or exclude data defined by the filter.



Start task on create ✓

Task Settings

Table mappings

Selection rules

At least one selection rule with an include action is required. Once you have one or more selection rules, you can add transformation rules.

Where

Schema name is: EntertainmentAgencyExample

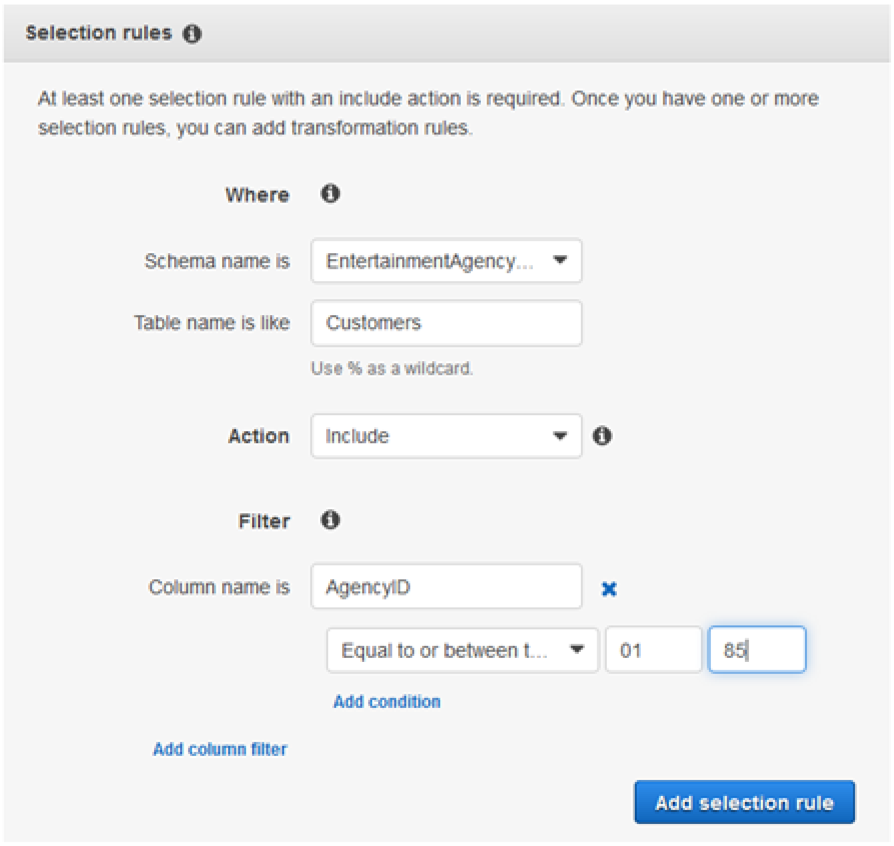
Table name is like: %

Use % as a wildcard.

Action: Include

Filter

6. You specify filter information using the **Add column filter** and the **Add condition** links. First, select **Add column filter** to specify a column and conditions. Select **Add condition** to add additional conditions. The following example shows a filter for the `Customers` table that includes `AgencyIDs` between `01` and `85`.



Selection rules

At least one selection rule with an include action is required. Once you have one or more selection rules, you can add transformation rules.

Where

Schema name is	EntertainmentAgency...
Table name is like	Customers
	Use % as a wildcard.
Action	Include

Filter

Column name is	AgencyID	
Equal to or between t...	01	85

7. When you have created the selections you want, select **Add selection rule**.
8. After you have created at least one selection rule, you can add a transformation to the task. Select **add transformation rule**.

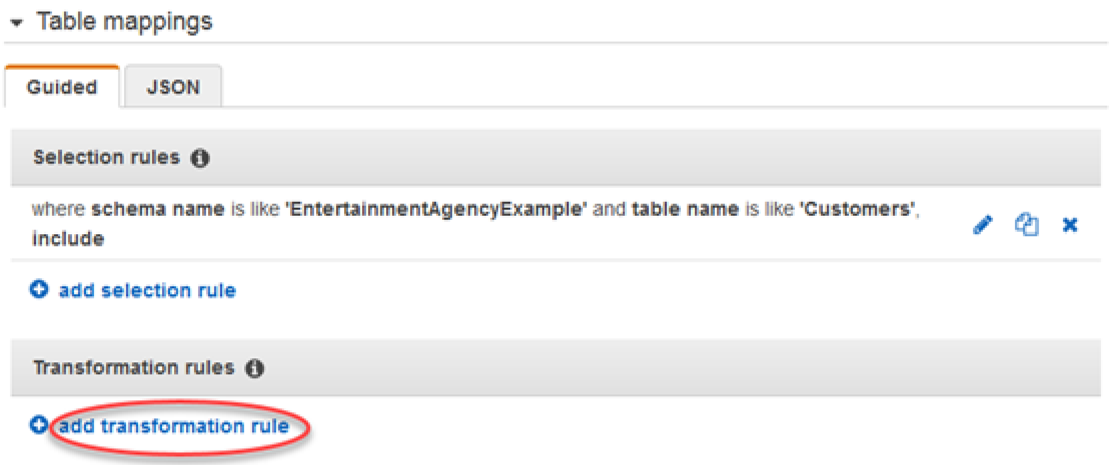


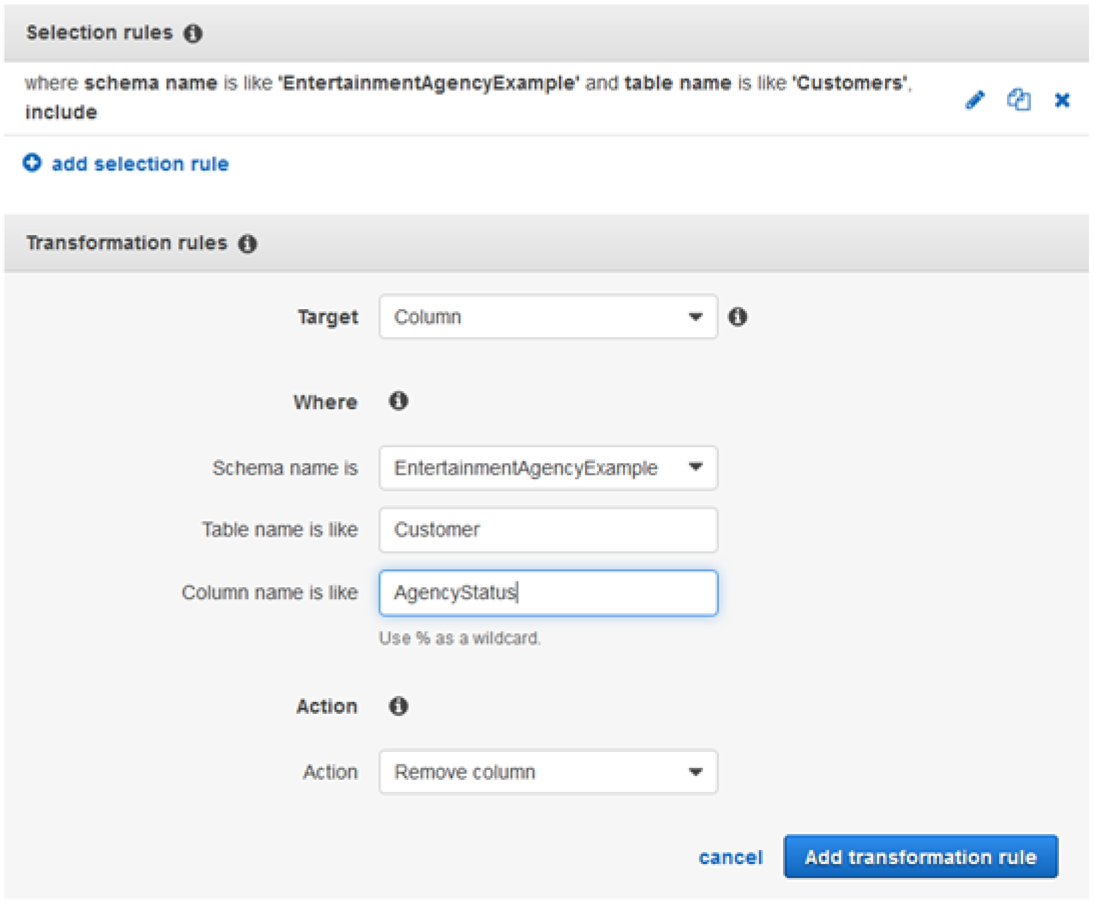
Table mappings

Selection rules

where **schema name** is like **'EntertainmentAgencyExample'** and **table name** is like **'Customers'**, **include**

Transformation rules

9. Select the target you want to transform and enter the additional information requested. The following example shows a transformation that deletes the `AgencyStatus` column from the `Customer` table.



Selection rules

where **schema name** is like **'EntertainmentAgencyExample'** and **table name** is like **'Customers'**, **include**

add selection rule

Transformation rules

Target: Column

Where

Schema name is: EntertainmentAgencyExample

Table name is like: Customer

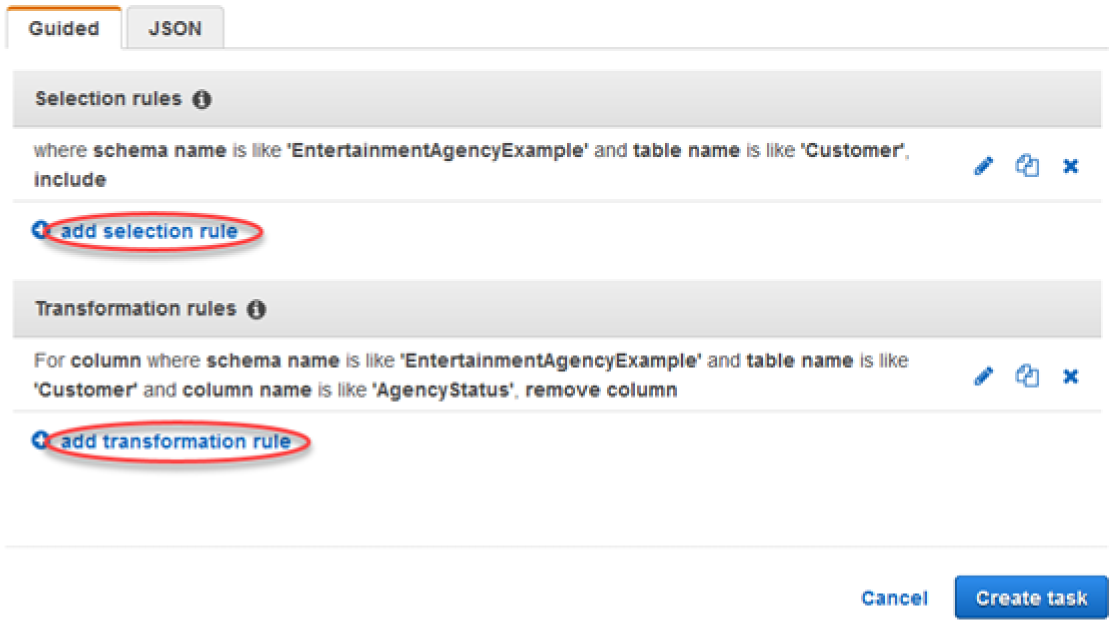
Column name is like: AgencyStatus

Use % as a wildcard.

Action

Action: Remove column

10. Choose **Add transformation rule**.
11. You can add additional selection rules or transformations by selecting the **add selection rule** or **add transformation rule**. When you are finished, choose **Create task**.



Selection rules

where **schema name** is like **'EntertainmentAgencyExample'** and **table name** is like **'Customer'**, **include**

add selection rule

Transformation rules

For **column** where **schema name** is like **'EntertainmentAgencyExample'** and **table name** is like **'Customer'** and **column name** is like **'AgencyStatus'**, **remove column**

add transformation rule

Selection and Transformation Table Mapping using JSON

Table mappings can be created in the JSON format. If you create a migration task using the AWS DMS Management Console, you can enter the JSON directly into the table mapping box. If you use the AWS Command Line Interface (AWS CLI) or AWS Database Migration Service API to perform migrations, you can create a JSON file to specify the table mappings that you want to occur during migration.

You can specify what tables or schemas you want to work with, and you can perform schema and table transformations. You create table mapping rules using the `selection` and `transformation` rule types.

Selection Rules and Actions

Using table mapping, you can specify what tables or schemas you want to work with by using selection rules and actions. For table mapping rules that use the selection rule type, the following values can be applied:

Parameter	Possible Values	Description
`rule-type`	`selection`	You must have at least one selection rule when specifying a table mapping.
`rule-id`	A numeric value.	A unique numeric value to identify the rule.
`rule-name`	An alpha-numeric value.	A unique name to identify the rule.
`rule-action`	`include`, `exclude`	Include or exclude the object selected by the rule.

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"
        }
    ]
}
```

Transformation Rules and Actions

You use the transformation actions to specify any transformations you want to apply to the selected schema or table. Transformation rules are optional.

For table mapping rules that use the transformation rule type, the following values can be applied:

Parameter	Possible Values	Description
`rule-type`	`transformation`	Applies a to the object specified by the selection rule.
`rule-id`	A numeric value.	A unique numeric value to identify the rule.
`rule-name`	An alpha-numeric value.	A unique name to identify the rule.
`object-locator`	`schema-name` The name of the schema. `table-name`The name of the table. You can use the "%" percent sign to be a wildcard.	The schema and table the rule applies to.
`rule-action`	• `rename` • `remove-column` • `convert-lowercase`, `convert-uppercase` • `add-prefix`, `remove-prefix`, `replace-prefix` • `add-suffix`, `remove-suffix`, `replace-suffix`	The transformation you want to apply to the object. All transformation rule actions are case sensitive.
`rule-target`	`schema`, `table`, `column`	The type of object you are transforming.

Parameter	Possible Values	Description
`value`	An alpha-numeric value that follows the naming rules for the target type.	The new value for actions that require input, such as `rename`.
`old-value`	An alpha-numeric value that follows the naming rules for the target type.	The old value for actions that require replacement, such as `replace-prefix`.

Example Rename a Schema

The following example renames a schema from **`Test`** in your source to **`Test1`** in your target endpoint:

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-action": "rename",
            "rule-target": "schema",
            "object-locator": {
                "schema-name": "Test"
            },
            "value": "Test1"
        }
    ]
}
```

Example Rename a Table

The following example renames a table from **`Actor`** in your source to **`Actor1`** in your target endpoint:

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-action": "rename",
            "rule-target": "table",
            "object-locator": {
                "schema-name": "Test",
                "table-name": "Actor"
            },
            "value": "Actor1"
        }
    ]
}
```

Example Rename a Column

The following example renames a column in table **`Actor`** from **`first_name`** in your source to **`fname`** in your target endpoint:

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "test",
                "table-name": "%"
            },
            "rule-action": "include"
        },
         {
            "rule-type": "transformation",
            "rule-id": "4",
            "rule-name": "4",
            "rule-action": "rename",
            "rule-target": "column",
            "object-locator": {
                "schema-name": "test",
                "table-name": "Actor",
                "column-name" : "first_name"
            },
            "value": "fname"
        }
    ]
}
```

Example Remove a Column

The following example transforms the table named **`Actor`** in your source to remove all columns starting with the characters **`col`** from it in your target endpoint:

```
{
  "rules": [{
  "rule-type": "selection",
  "rule-id": "1",
  "rule-name": "1",
  "object-locator": {
   "schema-name": "test",
   "table-name": "%"
  },
  "rule-action": "include"
 }, {
  "rule-type": "transformation",
  "rule-id": "2",
  "rule-name": "2",
  "rule-action": "remove-column",
  "rule-target": "column",
  "object-locator": {
   "schema-name": "test",
   "table-name": "Actor",
   "column-name": "col%"
  }
 }]
 }
```

Example Convert to Lowercase

The following example converts a table name from **`ACTOR`** in your source to **`actor`** in your target endpoint:

```
{
 "rules": [{
  "rule-type": "selection",
  "rule-id": "1",
  "rule-name": "1",
  "object-locator": {
   "schema-name": "test",
   "table-name": "%"
  },
  "rule-action": "include"
 }, {
  "rule-type": "transformation",
  "rule-id": "2",
  "rule-name": "2",
  "rule-action": "convert-lowercase",
  "rule-target": "table",
  "object-locator": {
   "schema-name": "test",
   "table-name": "ACTOR"
  }
 }]
}
```

Example Convert to Uppercase

The following example converts all columns in all tables and all schemas from lowercase in your source to uppercase in your target endpoint:

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "test",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-action": "convert-uppercase",
            "rule-target": "column",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%",
                "column-name": "%"
            }
        }
    ]
}
```

Example Add a Prefix

The following example transforms all tables in your source to add the prefix **`DMS_`** to them in your target endpoint:

```
{
  "rules": [{
  "rule-type": "selection",
  "rule-id": "1",
  "rule-name": "1",
  "object-locator": {
   "schema-name": "test",
   "table-name": "%"
  },
  "rule-action": "include"
 }, {
  "rule-type": "transformation",
  "rule-id": "2",
  "rule-name": "2",
  "rule-action": "add-prefix",
  "rule-target": "table",
  "object-locator": {
   "schema-name": "test",
   "table-name": "%"
  },
  "value": "DMS_"
 }]

}
```

Example Replace a Prefix

The following example transforms all columns containing the prefix **`Pre_`** in your source to replace the prefix with **`NewPre_`** in your target endpoint:

```
{
    "rules": [
        {
            "rule-type": "selection",
            "rule-id": "1",
            "rule-name": "1",
            "object-locator": {
                "schema-name": "test",
                "table-name": "%"
            },
            "rule-action": "include"
        },
        {
            "rule-type": "transformation",
            "rule-id": "2",
            "rule-name": "2",
            "rule-action": "replace-prefix",
            "rule-target": "column",
            "object-locator": {
                "schema-name": "%",
                "table-name": "%",
                "column-name": "%"
            },
            "value": "NewPre_",
            "old-value": "Pre_"
        }
    ]
}
```

Example Remove a Suffix

The following example transforms all tables in your source to remove the suffix **`_DMS`** from them in your target endpoint:

```
{
 "rules": [{
  "rule-type": "selection",
  "rule-id": "1",
  "rule-name": "1",
  "object-locator": {
   "schema-name": "test",
   "table-name": "%"
  },
  "rule-action": "include"
 }, {
  "rule-type": "transformation",
  "rule-id": "2",
  "rule-name": "2",
  "rule-action": "remove-suffix",
  "rule-target": "table",
  "object-locator": {
   "schema-name": "test",
   "table-name": "%"
  },
  "value": "_DMS"
 }]
}
```

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 will be 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 will be combined as if using an OR operator between the filter conditions.
- Filters are only applied when `rule-action = 'include'`.
- Filters require a column name and a list of filter conditions. Filter conditions must have a filter operator and a value.
- Column names, table names, and schema names are case sensitive.

Filtering by Time and Date

When selecting data to import, you can specify a date or time as part of your filter criteria. AWS DMS uses the date format YYYY-MM-DD and the time format YYYY-MM-DD HH:MM:SS for filtering. The AWS

DMS comparison functions follow the SQLite conventions. For more information about SQLite data types and date comparisons, see Datatypes In SQLite Version 3.

The following example shows how to filter on a date.

Example Single Date Filter

The following filter replicates all employees where `empstartdate >= January 1, 2002` to the target database.

```
 {
 "rules": [{
  "rule-type": "selection",
  "rule-id": "1",
  "rule-name": "1",
  "object-locator": {
   "schema-name": "test",
   "table-name": "employee"
  },
  "rule-action": "include",
  "filters": [{
   "filter-type": "source",
   "column-name": "empstartdate",
   "filter-conditions": [{
    "filter-operator": "gte",
    "value": "2002-01-01"
   }]
  }]
 }]
}
```

Creating Source Filter Rules in JSON

You can create source filters by specifying a column name, filter condition, filter operator, and a filter value.

The following table shows the parameters used for source filtering.

Parameter	Value
filter-type	source
column-name	The name of the source column you want the filter applied to. The name is case sensitive.
filter-conditions	
filter-operator	This parameter can be one of the following: • `ste` – less than or equal to • `gte` – greater than or equal to • `eq` – equal to • `between` – equal to or between two values
value	The value of the filter-operator parameter. If the filter-operator is `between`, provide two values, one for start-value and one for end-value.

The following examples show some common ways to use source filters.

Example Single Filter

The following filter replicates all employees where `empid >= 100` to the target database.

```
 {
 "rules": [{
  "rule-type": "selection",
  "rule-id": "1",
  "rule-name": "1",
  "object-locator": {
   "schema-name": "test",
   "table-name": "employee"
  },
  "rule-action": "include",
  "filters": [{
   "filter-type": "source",
   "column-name": "empid",
   "filter-conditions": [{
    "filter-operator": "gte",
    "value": "100"
   }]
  }]
 }]
}
```

Example Multiple Filter Operators

The following filter applies multiple filter operators to a single column of data. The filter replicates all employees where `(empid <=10)` OR `(empid is between 50 and 75)` OR `(empid >= 100)` to the target database.

```
{
  "rules": [{
   "rule-type": "selection",
   "rule-id": "1",
   "rule-name": "1",
   "object-locator": {
    "schema-name": "test",
    "table-name": "employee"
   },
   "rule-action": "include",
   "filters": [{
    "filter-type": "source",
    "column-name": "empid",
    "filter-conditions": [{
     "filter-operator": "ste",
     "value": "10"
    }, {
     "filter-operator": "between",
     "start-value": "50",
     "end-value": "75"
    }, {
     "filter-operator": "gte",
     "value": "100"
    }]
   }]
  }]
 }
```

Example Multiple Filters

The following filter applies multiple filters to two columns in a table. The filter replicates all employees where `(empid <= 100)` AND `(dept= tech)` to the target database.

```
{
 "rules": [{
  "rule-type": "selection",
  "rule-id": "1",
  "rule-name": "1",
  "object-locator": {
   "schema-name": "test",
   "table-name": "employee"
  },
  "rule-action": "include",
  "filters": [{
   "filter-type": "source",
   "column-name": "empid",
   "filter-conditions": [{
    "filter-operator": "ste",
    "value": "100"
   }]
  }, {
   "filter-type": "source",
   "column-name": "dept",
   "filter-conditions": [{
    "filter-operator": "eq",
    "value": "tech"
   }]
  }]
 }]
}
```

Monitoring AWS Database Migration Service Tasks

You can monitor the progress of your task by checking the task status and by monitoring the task's control table. For more information about control tables, see Control Table Task Settings (p. 127).

You can also monitor the progress of your tasks using Amazon CloudWatch. By using the AWS Management Console, the AWS Command Line Interface (CLI), or AWS DMS API, you can monitor the progress of your task and also the resources and network connectivity used.

Finally, you can monitor the status of your source tables in a task by viewing the table state.

Note that the "last updated" column the DMS console only indicates the time that AWS DMS last updated the table statistics record for a table. It does not indicate the time of the last update to the table.

For more information, see the following topics.

Topics

Task Status

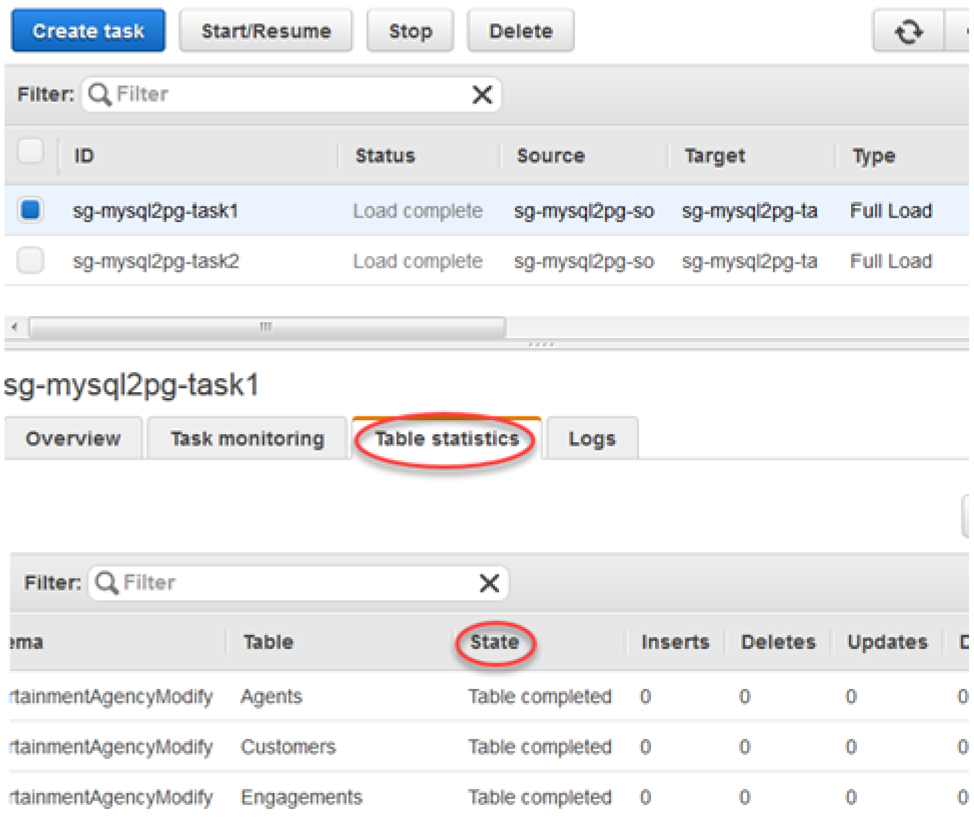
The task status indicated the condition of the task. The following table shows the possible statuses a task can have:

Task Status	Description
Creating	AWS DMS is creating the task.
Running	The task is performing the migration duties specified.
Stopped	The task is stopped.
Stopping	The task is being stopped. This is usually an indication of user intervention in the task.
Deleting	The task is being deleted, usually from a request for user intervention.
Failed	The task has failed. See the task log files for more information.
Starting	The task is connecting to the replication instance and to the source and target endpoints. Any filters and transformations are being applied.
Ready	The task is ready to run. This status usually follows the "creating" status.
Modifying	The task is being modified, usually due to a user action that modified the task settings.

The task status bar gives an estimation of the task's progress. The quality of this estimate depends on the quality of the source database’s table statistics; the better the table statistics, the more accurate the estimation. For tasks with only one table that has no estimated rows statistic, we are unable to provide any kind of percentage complete estimate. In this case, the task state and the indication of rows loaded can be used to confirm that the task is indeed running and making progress.

Table State During Tasks

The AWS DMS console updates information regarding the state of your tables during migration. The following table shows the possible state values:

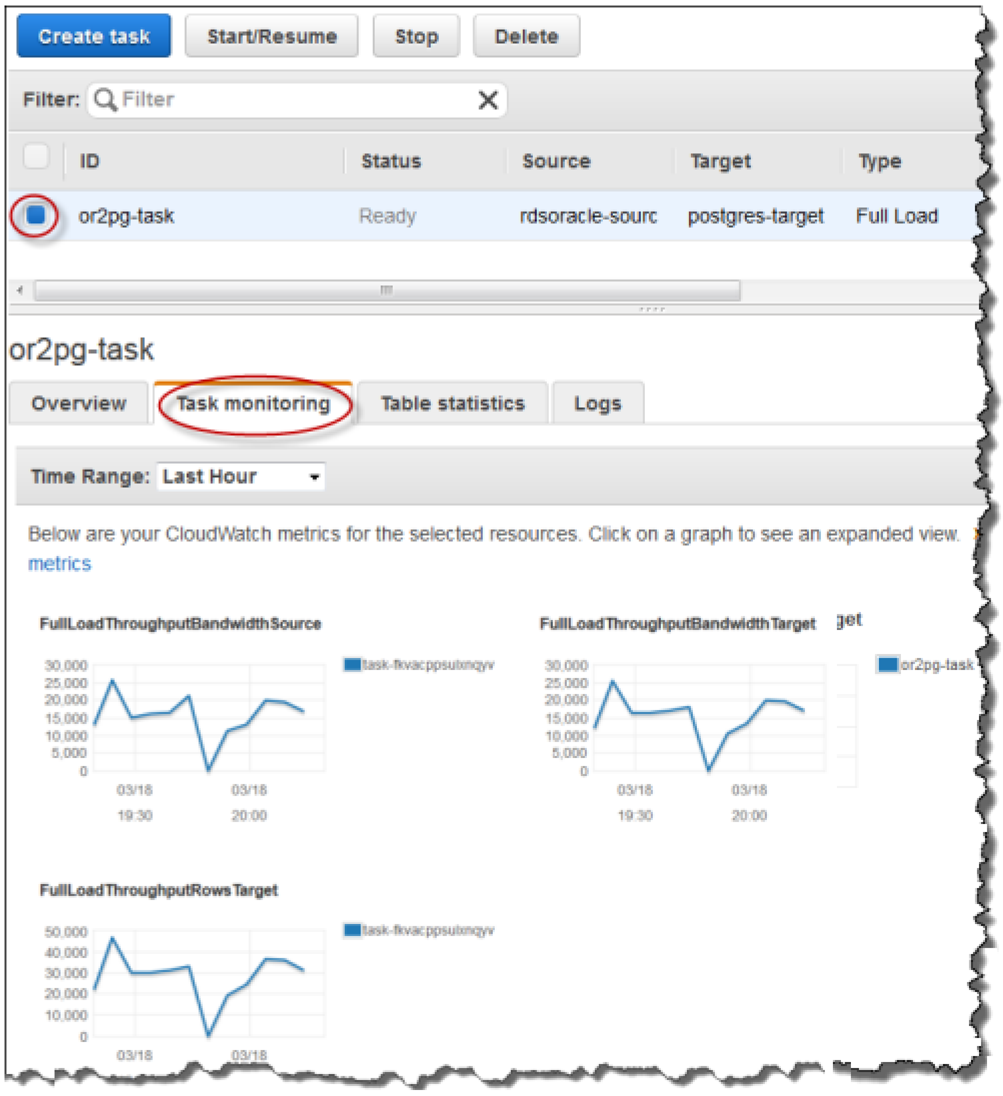


State	Description
Table does not exist	AWS DMS cannot find the table on the source endpoint.
Before load	The full load process has been enabled, but it hasn't started yet.
Full load	The full load process is in progress.
Table completed	Full load has completed.
Table cancelled	Loading of the table has been cancelled.
Table error	An error occurred when loading the table.

Monitoring 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

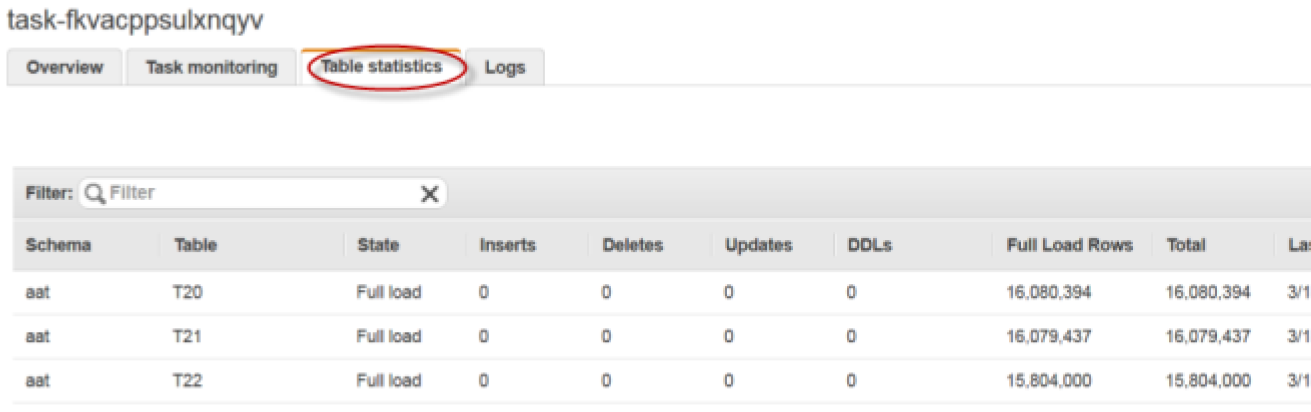
The AWS DMS console shows basic CloudWatch statistics for each task, including the task status, percent complete, elapsed time, and table statistics, as shown following. Select the replication task and then select the **Task monitoring** tab.


Create task
Start/Resume
Stop
Delete
Filter:
Filter

ID	Status	Source	Target	Type
or2pg-task	Ready	rdsoracle-sourc	postgres-target	Full Load

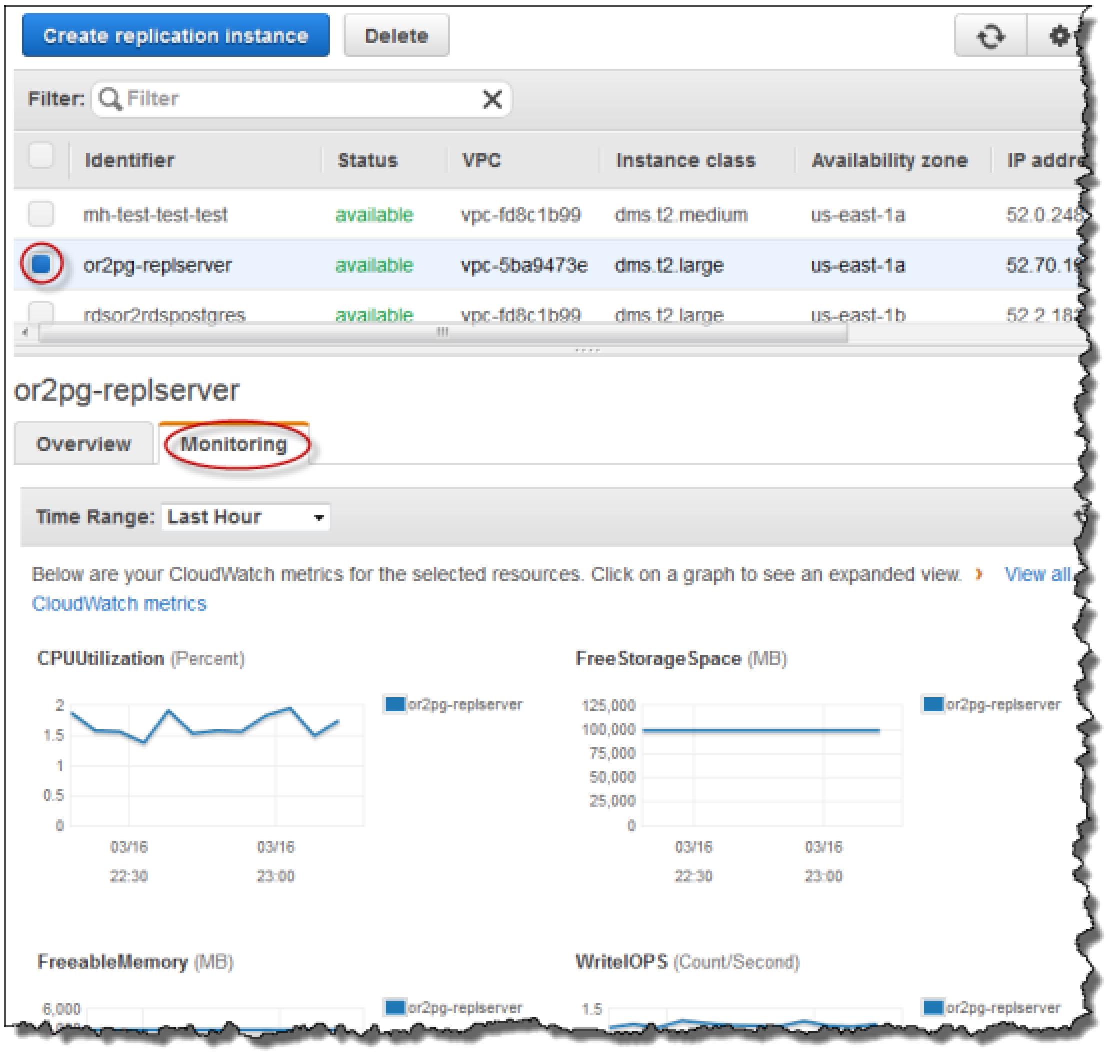
or2pg-task
Overview
Task monitoring
Table statistics
Logs
Time Range: Last Hour
Below are your CloudWatch metrics for the selected resources. Click on a graph to see an expanded view.
metrics
FullLoadThroughputBandwidthSource
task-fkvacppsulxnqyv
FullLoadThroughputBandwidthTarget
or2pg-task
FullLoadThroughputRowsTarget
task-fkvacppsulxnqyv

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.



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

In addition, if you select a replication instance from the **Replication Instance** page, you can view performance metrics for the instance by selecting the **Monitoring** tab.



	Identifier	Status	VPC	Instance class	Availability zone	IP addr
☐	mh-test-test-test	available	vpc-fd8c1b99	dms.t2.medium	us-east-1a	52.0.248
☑	or2pg-replserver	available	vpc-5ba9473e	dms.t2.large	us-east-1a	52.70.1
☐	rdsor2rdspostgres	available	vpc-fd8c1b99	dms.t2.large	us-east-1b	52.2.18

or2pg-replserver

Below are your CloudWatch metrics for the selected resources. Click on a graph to see an expanded view. › View all CloudWatch metrics

Data Migration Service Metrics

AWS DMS provides statistics for the following:

- **Host Metrics** – Performance and utilization statistics for the replication host, provided by Amazon CloudWatch. For a complete list of the available metrics, see Replication Instance Metrics (p. 154).
- **Replication Task Metrics** – Statistics for replication tasks including incoming and committed changes, and latency between the replication host and both the source and target databases. For a complete list of the available metrics, see Replication Task Metrics (p. 155).
- **Table Metrics** – Statistics for tables that are in the process of being migrated, including the number of insert, update, delete, and DDL statements completed.

Task metrics are divided into statistics between the replication host and the source endpoint, and statistics between the replication host and the target endpoint. You can determine the total statistic for a task by adding two related statistics together. For example, you can determine the total latency, or replica lag, for a task by combining the **CDCLatencySource** and **CDCLatencyTarget** values.

Task metric values can be influenced by current activity on your source database. For example, if a transaction has begun, but has not been committed, then the **CDCLatencySource** metric will continue 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.

Replication Instance Metrics

Replication instance monitoring include Amazon CloudWatch metrics for the following statistics:

CPUUtilization

The amount of CPU used.

Units: Bytes

FreeStorageSpace

The amount of available storage space.

Units: Bytes

FreeableMemory

The amount of available random access memory.

Units: Bytes

WriteIOPS

The average number of disk I/O operations per second.

Units: Count/Second

ReadIOPS

The average number of disk I/O operations per second.

Units: Count/Second

WriteThroughput

The average number of bytes written to disk per second.

Units: Bytes/Second

ReadThroughput

The average number of bytes read from disk per second.

Units: Bytes/Second

WriteLatency

The average amount of time taken per disk I/O operation.

Units: Seconds

ReadLatency

The average amount of time taken per disk I/O operation.

Units: Seconds

SwapUsage

The amount of swap space used on the replication instance.

Units: Bytes

NetworkTransmitThroughput

The outgoing (Transmit) network traffic on the replication instance, including both customer database traffic and AWS DMS traffic used for monitoring and replication.

Units: Bytes/second

NetworkReceiveThroughput

The incoming (Receive) network traffic on the replication instance, including both customer database traffic and AWS DMS traffic used for monitoring and replication.

Units: Bytes/second

Replication Task Metrics

Replication task monitoring includes metrics for the following statistics:

FullLoadThroughputBandwidthSource

Incoming network bandwidth from a full load from the source in kilobytes (KB) per second.

FullLoadThroughputBandwidthTarget

Outgoing network bandwidth from a full load for the target in KB per second.

FullLoadThroughputRowsSource

Incoming changes from a full load from the source in rows per second.

FullLoadThroughputRowsTarget

Outgoing changes from a full load for the target in rows per second.

CDCIncomingChanges

Total row count of changes for the task.

CDCChangesMemorySource

Amount of rows accumulating in a memory and waiting to be committed from the source.

CDCChangesMemoryTarget

Amount of rows accumulating in a memory and waiting to be committed to the target.

CDCChangesDiskSource

Amount of rows accumulating on disk and waiting to be committed from the source.

CDCChangesDiskTarget

Amount of rows accumulating on disk and waiting to be committed to the target.

CDCThroughputBandwidthTarget

Outgoing task network bandwidth for the target in KB per second.

CDCThroughputRowsSource

Incoming task changes from the source in rows per second.

CDCThroughputRowsTarget

Outgoing task changes for the target in rows per second.

CDCLatencySource

Latency reading from source in seconds.

CDCLatencyTarget

Latency writing to the target in seconds.

Logging AWS DMS API Calls Using AWS CloudTrail

The AWS CloudTrail service logs all AWS Database Migration Service (AWS DMS) API calls made by or on behalf of your AWS account. AWS CloudTrail stores this logging information in an S3 bucket. You can use the information collected by CloudTrail to monitor AWS DMS activity, such as creating or deleting a replication instance or an endpoint. For example, you can determine whether a request completed successfully and which user made the request. To learn more about CloudTrail, see the AWS CloudTrail User Guide.

If an action is taken on behalf of your AWS account using the AWS DMS console or the AWS DMS command line interface, then AWS CloudTrail logs the action as calls made to the AWS DMS API. For example, if you use the AWS DMS console to describe connections, or call the AWS CLI describe-connections command, then the AWS CloudTrail log shows a call to the AWS DMS API DescribeConnections action. For a list of the AWS DMS API actions that are logged by AWS CloudTrail, see the AWS DMS API Reference.

Configuring CloudTrail Event Logging

CloudTrail creates audit trails in each region separately and stores them in an S3 bucket. You can configure CloudTrail to use Amazon Simple Notification Service (Amazon SNS) to notify you when a log file is created, but that is optional. CloudTrail will notify you frequently, so we recommend that you use Amazon SNS with an Amazon Simple Queue Service (Amazon SQS) queue and handle notifications programmatically.

You can enable CloudTrail using the AWS Management Console, CLI, or API. When you enable CloudTrail logging, you can have the CloudTrail service create an S3 bucket for you to store your log files. For details, see Creating and Updating Your Trail in the *AWS CloudTrail User Guide*. The *AWS CloudTrail User Guide* also contains information on how to aggregate CloudTrail logs from multiple regions into a single S3 bucket.

There is no cost to use the CloudTrail service. However, standard rates for S3 usage apply, and also rates for Amazon SNS usage should you include that option. For pricing details, see the S3 and Amazon SNS pricing pages.

AWS Database Migration Service Event Entries in CloudTrail Log Files

CloudTrail log files contain event information formatted using JSON. An event record represents a single AWS API call and includes information about the requested action, the user that requested the action, the date and time of the request, and so on.

CloudTrail log files include events for all AWS API calls for your AWS account, not just calls to the AWS DMS API. However, you can read the log files and scan for calls to the AWS DMS API using the `eventName` element.

For more information about the different elements and values in CloudTrail log files, see CloudTrail Event Reference in the *AWS CloudTrail User Guide*.

You might also want to make use of one of the Amazon partner solutions that integrate with CloudTrail to read and analyze your CloudTrail log files. For options, see the AWS partners page.

Working with Events and Notifications

AWS Database Migration Service (AWS DMS) uses Amazon Simple Notification Service (Amazon SNS) to provide notifications when an AWS DMS event occurs, for example the creation or deletion of a replication instance. You can work with these notifications in any form supported by Amazon SNS for an AWS Region, such as an email message, a text message, or a call to an HTTP endpoint.

AWS DMS groups events into categories that you can subscribe to, so you can be notified when an event in that category occurs. For example, if you subscribe to the Creation category for a given replication instance, you are notified whenever a creation-related event occurs that affects your replication instance. If you subscribe to a Configuration Change category for a replication instance, you are notified when the replication instance's configuration is changed. You also receive notification when an event notification subscription changes. For a list of the event categories provided by AWS DMS, see AWS DMS Event Categories and Event Messages (p. 159), following.

AWS DMS sends event notifications to the addresses you provide when you create an event subscription. You might want to create several different subscriptions, such as one subscription receiving all event notifications and another subscription that includes only critical events for your production DMS resources. You can easily turn off notification without deleting a subscription by setting the **Enabled** option to **No** in the AWS DMS console or by setting the `Enabled` parameter to *false* using the AWS DMS API.

> **Note**
> AWS DMS event notifications using SMS text messages are currently available for AWS DMS resources in all regions where AWS DMS is supported. For more information on using text messages with SNS, see Sending and Receiving SMS Notifications Using Amazon SNS.

AWS DMS uses a subscription identifier to identify each subscription. You can have multiple AWS DMS event subscriptions published to the same Amazon SNS topic. When you use event notification, Amazon SNS fees apply; for more information on Amazon SNS billing, see Amazon SNS Pricing.

To subscribe to AWS DMS events, you use the following process:

1. Create an Amazon SNS topic. In the topic, you specify what type of notification you want to receive and to what address or number the notification will go to.
2. Create an AWS DMS event notification subscription by using the AWS Management Console, AWS CLI, or AWS DMS API.

3. AWS DMS sends an approval email or SMS message to the addresses you submitted with your subscription. To confirm your subscription, click the link in the approval email or SMS message.
4. When you have confirmed the subscription, the status of your subscription is updated in the AWS DMS console's **Event Subscriptions** section.
5. You then begin to receive event notifications.

For the list of categories and events that you can be notified of, see the following section. For more details about subscribing to and working with AWS DMS event subscriptions, see Subscribing to AWS DMS Event Notification (p. 161).

AWS DMS Event Categories and Event Messages

AWS DMS generates a significant number of events in categories that you can subscribe to using the AWS DMS console or the AWS DMS API. Each category applies to a source type; currently AWS DMS supports the replication instance and replication task source types.

The following table shows the possible categories and events for the replication instance source type.

Category	DMS Event ID	Description
Configuration Change	DMS-EVENT-0012	REP_INSTANCE_CLASS_CHANGING – The replication instance class for this replication instance is being changed.
Configuration Change	DMS-EVENT-0014	REP_INSTANCE_CLASS_CHANGE_COMPLETE – The replication instance class for this replication instance has changed.
Configuration Change	DMS-EVENT-0018	BEGIN_SCALE_STORAGE – The storage for the replication instance is being increased.
Configuration Change	DMS-EVENT-0017	FINISH_SCALE_STORAGE – The storage for the replication instance has been increased.
Configuration Change	DMS-EVENT-0024	BEGIN_CONVERSION_TO_HIGH_AVAILABILITY – The replication instance is transitioning to a Multi-AZ configuration.
Configuration Change	DMS-EVENT-0025	FINISH_CONVERSION_TO_HIGH_AVAILABILITY – The replication instance has finished transitioning to a Multi-AZ configuration.
Configuration Change	DMS-EVENT-0030	BEGIN_CONVERSION_TO_NON_HIGH_AVAILABILITY – The replication instance is transitioning to a Single-AZ configuration.
Configuration Change	DMS-EVENT-0029	FINISH_CONVERSION_TO_NON_HIGH_AVAILABILITY – The replication instance has finished transitioning to a Single-AZ configuration.
Creation	DMS-EVENT-0067	CREATING_REPLICATION_INSTANCE – A replication instance is being created.
Creation	DMS-EVENT-0005	CREATED_REPLICATION_INSTANCE – A replication instance has been created.

Category	DMS Event ID	Description
Deletion	DMS-EVENT-0066	DELETING_REPLICATION_INSTANCE – The replication instance is being deleted.
Deletion	DMS-EVENT-0003	DELETED_REPLICATION_INSTANCE – The replication instance has been deleted.
Maintenance	DMS-EVENT-0047	FINISH_PATCH_INSTANCE – Management software on the replication instance has been updated.
Maintenance	DMS-EVENT-0026	BEGIN_PATCH_OFFLINE – Offline maintenance of the replication instance is taking place. The replication instance is currently unavailable.
Maintenance	DMS-EVENT-0027	FINISH_PATCH_OFFLINE – Offline maintenance of the replication instance is complete. The replication instance is now available.
LowStorage	DMS-EVENT-0007	LOW_STORAGE – Free storage for the replication instance is low.
Failover	DMS-EVENT-0013	FAILOVER_STARTED – Failover started for a Multi-AZ replication instance.
Failover	DMS-EVENT-0049	FAILOVER_COMPLETED – Failover has been completed for a Multi-AZ replication instance.
Failover	DMS-EVENT-0050	MAZ_INSTANCE_ACTIVATION_STARTED – Multi-AZ activation has started.
Failover	DMS-EVENT-0051	MAZ_INSTANCE_ACTIVATION_COMPLETED – Multi-AZ activation completed.
Failure	DMS-EVENT-0031	REPLICATION_INSTANCE_FAILURE – The replication instance has gone into storage failure.
Failure	DMS-EVENT-0036	INCOMPATIBLE_NETWORK – The replication instance has failed due to an incompatible network.

The following table shows the possible categories and events for the replication task source type.

Category	DMS Event ID	Description
StateChange	DMS-EVENT-0069	REPLICATION_TASK_STARTED – The replication task has started.
StateChange	DMS-EVENT-0077	REPLICATION_TASK_STOPPED – The replication task has stopped.
Failure	DMS-EVENT-0078	REPLICATION_TASK_FAILED – A replication task has failed.
Deletion	DMS-EVENT-0073	REPLICATION_TASK_DELETED – The replication task has been deleted.
Creation	DMS-EVENT-0074	REPLICATION_TASK_CREATED – The replication task has been created.

Subscribing to AWS DMS Event Notification

You can create an AWS DMS event notification subscription so you can be notified when an AWS DMS event occurs. The simplest way to create a subscription is with the AWS DMS console. If you choose to create event notification subscriptions using AWS DMS API, you must create an Amazon SNS topic and subscribe to that topic with the Amazon SNS console or API. In this case, you also need to note the topic's Amazon Resource Name (ARN), because this ARN is used when submitting CLI commands or API actions. For information on creating an Amazon SNS topic and subscribing to it, see Getting Started with Amazon SNS.

In a notification subscription, you can specify the type of source you want to be notified of and the AWS DMS source that triggers the event. You define the AWS DMS source type by using a **SourceType** value. You define the source generating the event by using a **SourceIdentifier** value. If you specify both **SourceType** and **SourceIdentifier**, such as `SourceType = db-instance` and `SourceIdentifier = myDBInstance1`, you receive all the DB_Instance events for the specified source. If you specify **SourceType** but not **SourceIdentifier**, you receive notice of the events for that source type for all your AWS DMS sources. If you don't specify either **SourceType** or **SourceIdentifier**, you are notified of events generated from all AWS DMS sources belonging to your customer account.

AWS Management Console

To subscribe to AWS DMS event notification by using the console

1. Sign in to the AWS Management Console and choose AWS DMS. Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS.
2. In the navigation pane, choose **Event Subscriptions**.
3. On the **Event Subscriptions** page, choose **Create Event Subscription**.
4. On the **Create Event Subscription** page, do the following:

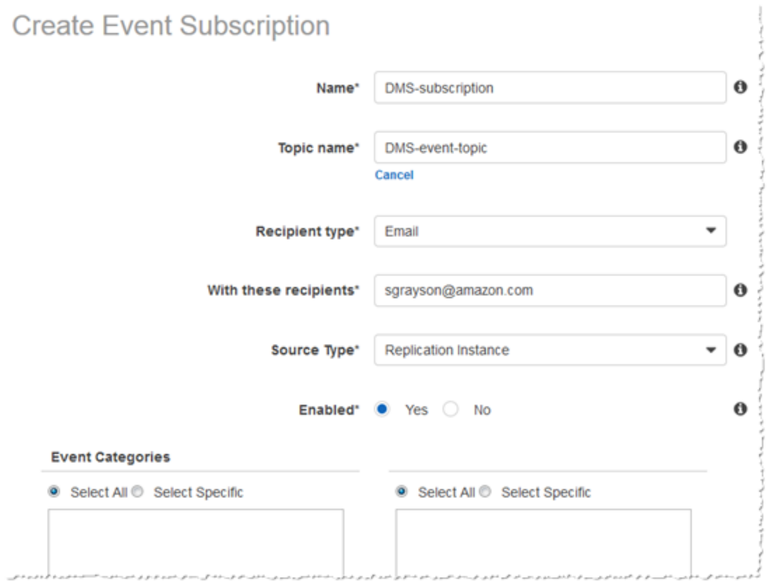
   a. For **Name**, type a name for the event notification subscription.

   b. Either choose an existing Amazon SNS topic for **Send notifications to**, or choose **create topic**. You must have either an existing Amazon SNS topic to send notices to or you must create the topic. If you choose **create topic**, you can enter an email address where notifications will be sent.

   c. For **Source Type**, choose a source type. The only option is **replication instance**.

   d. Choose **Yes** to enable the subscription. If you want to create the subscription but not have notifications sent yet, choose **No**.

   e. Depending on the source type you selected, choose the event categories and sources you want to receive event notifications for.



Create Event Subscription

Name*	DMS-subscription
Topic name*	DMS-event-topic
Recipient type*	Email
With these recipients*	sgrayson@amazon.com
Source Type*	Replication Instance
Enabled*	[✓ Yes] No

Event Categories

[✓ Select All] Select Specific — [✓ Select All] Select Specific

f. Choose **Create**.

The AWS DMS console indicates that the subscription is being created.

AWS DMS API

To subscribe to AWS DMS event notification by using the AWS DMS API

- Call `CreateEventSubscription`.

Troubleshooting Migration Tasks

The following sections provide information on troubleshooting issues with AWS Database Migration Service (AWS DMS).

Topics

Slow Running Migration Tasks

There are several issues that may cause a migration task to run slowly, or for subsequent tasks to run slower than the initial task. The most common reason for a migration task running slowly is that there are inadequate resources allocated to the AWS DMS replication instance. Check your replication instance's use of CPU, Memory, Swap, and IOPs to ensure that your instance has enough resources for the tasks you are running on it. For example, multiple tasks with Amazon Redshift as an endpoint are IO intensive. You can increase IOPS for your replication instance or split your tasks across multiple replication instances for a more efficient migration.

For more information about determining the size of your replication instance, see Determining the Optimum Size for a Replication Instance (p. 39)

You can increase the speed of an initial migration load by doing the following:

- If your target is an Amazon RDS DB instance, ensure that Multi-AZ is not enabled for the target DB instance.
- Turn off any automatic backups or logging on the target database during the load, and turn back on those features once the migration is complete.
- If the feature is available on the target, use Provisioned IOPs.
- If your migration data contains LOBs, ensure that the task is optimized for LOB migration. See Target Metadata Task Settings (p. 125) for more information on optimizing for LOBs.

Task Status Bar Not Moving

The task status bar gives an estimation of the task's progress. The quality of this estimate depends on the quality of the source database’s table statistics; the better the table statistics, the more accurate the estimation. For a task with only one table that has no estimated rows statistic, we are unable to provide any kind of percentage complete estimate. In this case, the task state and the indication of rows loaded can be used to confirm that the task is indeed running and making progress.

Missing Foreign Keys and Secondary Indexes

AWS DMS creates tables, primary keys, and in some cases unique indexes, but it doesn't create any other objects that are not required to efficiently migrate the data from the source. For example, it doesn't create secondary indexes, non-primary key constraints, or data defaults.

To migrate secondary objects from your database, use the database's native tools if you are migrating to the same database engine as your source database. Use the Schema Conversion Tool if you are migrating to a different database engine than that used by your source database to migrate secondary objects.

Amazon RDS Connection Issues

There can be several reasons why you are unable to connect to an Amazon RDS DB instance that you set as an endpoint. These include:

- Username and password combination is incorrect.
- Check that the endpoint value shown in the Amazon RDS console for the instance is the same as the endpoint identifier you used to create the AWS DMS endpoint.
- Check that the port value shown in the Amazon RDS console for the instance is the same as the port assigned to the AWS DMS endpoint.
- Check that the security group assigned to the Amazon RDS DB instance allows connections from the AWS DMS replication instance.

- If the AWS DMS replication instance and the Amazon RDS DB instance are not in the same VPC, check that the DB instance is publicly accessible.

Error Message: Incorrect thread connection string: incorrect thread value 0

This error can often occur when you are testing the connection to an endpoint. The error indicates that there is an error in the connection string, such as a space after the host IP address or a bad character was copied into the connection string.

Networking Issues

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

Other configuration related issues include:

- **Replication instance and both source and target endpoints in the same VPC** — The security group used by the endpoints must allow ingress on the database port from the replication instance. Ensure that the security group used by the replication instance has ingress to the endpoints, or you can create a rule in the security group used by the endpoints that allows the private IP address of the replication instance access.
- **Source endpoint is outside the VPC used by the replication instance (using Internet Gateway)** — The VPC security group must include routing rules that send traffic not destined for the VPC to the Internet Gateway. In this configuration, the connection to the endpoint appears to come from the public IP address on the replication instance.
- **Source endpoint is outside the VPC used by the replication instance (using NAT Gateway)** — You can configure a network address translation (NAT) gateway using a single Elastic IP Address bound to a single Elastic Network Interface which then receives a NAT identifier (nat-#####). If the VPC includes a default route to that NAT Gateway instead of the Internet Gateway, the replication instance will instead appear to contact the Database Endpoint using the public IP address of the Internet Gateway. In this case, the ingress to the Database Endpoint outside the VPC needs to allow ingress from the NAT address instead of the Replication Instance's public IP Address.

CDC Stuck After Full Load

Slow or stuck replication changes can occur after a full load migration when several AWS DMS settings conflict with each other. For example, if the **Target table preparation mode** parameter is set to **Do nothing** or **Truncate**, then you have instructed AWS DMS to do no setup on the target tables, including creating primary and unique indexes. If you haven't created primary or unique keys on the target tables, then AWS DMS must do a full table scan for each update, which can significantly impact performance.

Primary Key Violation Errors When Restarting a Task

This error can occur when data remains in the target database from a previous migration task. If the **Target table preparation mode** parameter is set to **Do nothing**, AWS DMS does not do any preparation on the target table, including cleaning up data inserted from a previous task. In order to restart your task and avoid these errors, you must remove rows inserted into the target tables from the previous running of the task.

Initial Load of Schema Fails

If your initial load of your schemas fails with an error of `Operation:getSchemaListDetails:errType=, status=0, errMessage=, errDetails=`, then the user account used by AWS DMS to connect to the source endpoint does not have the necessary permissions.

Tasks Failing With Unknown Error

The cause of these types of error can be varied, but often we find that the issue involves insufficient resources allocated to the AWS DMS replication instance. Check the replication instance's use of CPU, Memory, Swap, and IOPs to ensure your instance has enough resources to perform the migration. For more information on monitoring, see Data Migration Service Metrics (p. 153).

Task Restart Loads Tables From the Beginning

AWS DMS restarts table loading from the beginning when it has not finished the initial load of a table. When a task is restarted, AWS DMS does not reload tables that completed the initial load but will reload tables from the beginning when the initial load did not complete.

Number of Tables Per Task

While there is no set limit on the number of tables per replication task, we have generally found that limiting the number of tables in a task to less than 60,000 is a good rule of thumb. Resource use can often be a bottleneck when a single task uses more than 60,000 tables.

Troubleshooting Oracle Specific Issues

The following issues are specific to using AWS DMS with Oracle databases.

Topics

Pulling Data from Views

To be able to extract data from views, you must add the following code to the **Extra connection attributes** in the **Advanced** section of the Oracle source endpoint. Note that when you extract data from a view, the view is shown as a table on the target schema.

```
exposeViews=true
```

Migrating LOBs from Oracle 12c

AWS DMS can use two methods to capture changes to an Oracle database, BinaryReader and Oracle LogMiner. By default, AWS DMS uses Oracle LogMiner to capture changes. However, on Oracle 12c, Oracle LogMiner does not support LOB columns. To capture changes to LOB columns on Oracle 12c, use BinaryReader.

Switching Between Oracle LogMiner and BinaryReader

AWS DMS can use two methods to capture changes to a source Oracle database, BinaryReader and Oracle LogMiner. Oracle LogMiner is the default. To switch to using BinaryReader for capturing changes, do the following:

To use BinaryReader for capturing changes

1. Sign in to the AWS Management Console and select DMS.
2. Select **Endpoints**.
3. Select the Oracle source endpoint that you want to use BinaryReader.
4. Select **Modify**.
5. Select Advanced, and then add the following code to the Extra connection attributes text box:

   ```
   useLogminerReader=N
   ```

6. Use an Oracle developer tool such as SQL-Plus to grant the following additional privilege to the AWS DMS user account used to connect to the Oracle endpoint:

   ```
   SELECT ON V_$TRANSPORTABLE_PLATFORM
   ```

Error: Oracle CDC stopped 122301 Oracle CDC maximum retry counter exceeded.

This error occurs when the needed Oracle archive logs have been removed from your server before AWS DMS was able to use them to capture changes. Increase your log retention policies on your database server. For an Amazon RDS database, run the following procedure to increase log retention. For example, the following code increases log retention on an Amazon RDS DB instance to 24 hours.

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

Automatically Add Supplemental Logging to an Oracle Source Endpoint

By default, AWS DMS has supplemental logging turned off. To automatically turn on supplemental logging for a source Oracle endpoint, do the following:

To add supplemental logging to a source Oracle endpoint

1. Sign in to the AWS Management Console and select **DMS**.
2. Select **Endpoints**.
3. Select the Oracle source endpoint that you want to add supplemental logging to.
4. Select **Modify**.
5. Select **Advanced**, and then add the following code to the **Extra connection attributes** text box:

   ```
   addSupplementalLogging=Y
   ```
6. Choose **Modify**.

LOB Changes not being Captured

Currently, a table must have a primary key for AWS DMS to capture LOB changes. If a table that contains LOBs doesn't have a primary key, there are several actions you can take to capture LOB changes:

- Add a primary key to the table. This can be as simple as adding an ID column and populating it with a sequence using a trigger.
- Create a materialized view of the table that includes a system generated ID as the primary key and migrate the materialized view rather than the table.
- Create a logical standby, add a primary key to the table, and migrate from the logical standby.

Error: ORA-12899: value too large for column <column-name>

The error "ORA-12899: value too large for column <column-name>" is often caused by a mismatch in the character sets used by the source and target databases or when NLS settings differ between the two databases. A common cause of this error is when the source database NLS_LENGTH_SEMANTICS parameter is set to CHAR and the target database NLS_LENGTH_SEMANTICS parameter is set to BYTE.

NUMBER data type being misinterpreted

The Oracle NUMBER data type is converted into various AWS DMS datatypes, depending on the precision and scale of NUMBER. These conversions are documented here Using an Oracle Database as a Source for AWS Database Migration Service (p. 69). The way the NUMBER type is converted can also be affected by using Extra Connection Attributes for the source Oracle endpoint. These Extra Connection Attributes are documented in Using Extra Connection Attributes with AWS Database Migration Service (p. 199).

Troubleshooting MySQL Specific Issues

The following issues are specific to using AWS DMS with MySQL databases.

Topics

CDC Task Failing for Amazon RDS DB Instance Endpoint Because Binary Logging Disabled

This issue occurs with Amazon RDS DB instances because automated backups are disabled. Enable automatic backups by setting the backup retention period to a non-zero value.

Connections to a target MySQL instance are disconnected during a task

If you have a task with LOBs that is getting disconnected from a MySQL target with the following type of errors in the task log, you might need to adjust some of your task settings.

```
[TARGET_LOAD ]E: RetCode: SQL_ERROR SqlState: 08S01 NativeError:
2013 Message: [MySQL][ODBC 5.3(w) Driver][mysqld-5.7.16-log]Lost connection
to MySQL server during query [122502] ODBC general error.
```

```
 [TARGET_LOAD ]E: RetCode: SQL_ERROR SqlState: HY000 NativeError:
2006 Message: [MySQL][ODBC 5.3(w) Driver]MySQL server has gone away
[122502] ODBC general error.
```

To solve the issue where a task is being disconnected from a MySQL target, do the following:

- Check that you have your database variable `max_allowed_packet` set large enough to hold your largest LOB.
- Check that you have the following variables set to have a large timeout value. We suggest you use a value of at least 5 minutes for each of these variables.
  - `net_read_timeout`
  - `net_write_timeout`
  - `wait_timeout`
  - `interactive_timeout`

Adding Autocommit to a MySQL-compatible Endpoint

To add autocommit to a target MySQL-compatible endpoint

1. Sign in to the AWS Management Console and select **DMS**.
2. Select **Endpoints**.
3. Select the MySQL-compatible target endpoint that you want to add autocommit to.
4. Select **Modify**.
5. Select **Advanced**, and then add the following code to the **Extra connection attributes** text box:

```
Initstmt= SET AUTOCOMMIT=1
```

6. Choose **Modify**.

Disable Foreign Keys on a Target MySQL-compatible Endpoint

You can disable foreign key checks on MySQL by adding the following to the **Extra Connection Attributes** in the **Advanced** section of the target MySQL, Aurora, or MariaDB endpoint.

To disable foreign keys on a target MySQL-compatible endpoint

1. Sign in to the AWS Management Console and select **DMS**.
2. Select **Endpoints**.
3. Select the MySQL, Aurora, or MariaDB target endpoint that you want to disable foreign keys.
4. Select **Modify**.
5. Select **Advanced**, and then add the following code to the **Extra connection attributes** text box:

```
Initstmt=SET FOREIGN_KEY_CHECKS=0
```

6. Choose **Modify**.

Characters Replaced with Question Mark

The most common situation that causes this issue is when the source endpoint characters have been encoded by a character set that AWS DMS doesn't support. For example, AWS DMS does not support the UTF8MB4 character set.

"Bad event" Log Entries

"Bad event" entries in the migration logs usually indicate that an unsupported DDL operation was attempted on the source database endpoint. Unsupported DDL operations cause an event that the replication instance cannot skip so a bad event is logged. To fix this issue, restart the task from the beginning, which will reload the tables and will start capturing changes at a point after the unsupported DDL operation was issued.

Change Data Capture with MySQL 5.5

AWS DMS change data capture (CDC) for Amazon RDS MySQL-compatible databases requires full image row-based binary logging, which is not supported in MySQL version 5.5 or lower. To use AWS DMS CDC, you must 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 that perform an update of one row, even when the replacing value is the same as the current one.

Error: Identifier too long

The following error occurs when an identifier is too long:

```
TARGET_LOAD E: RetCode: SQL_ERROR SqlState: HY000 NativeError:
1059 Message: MySQLhttp://ODBC 5.3(w) Driverhttp://mysqld-5.6.10Identifier
name ‘<name>’ is too long 122502 ODBC general error. (ar_odbc_stmt.c:4054)
```

When AWS DMS is set to create the tables and primary keys in the target database, it currently does not use the same names for the Primary Keys that were used in the source database. Instead, AWS DMS creates the Primary Key name based on the tables name. When the table name is long, the auto-generated identifier created can be longer than the allowed limits for MySQL. The solve this issue, currently, pre-create the tables and Primary Keys in the target database and use a task with the task setting **Target table preparation mode** set to **Do nothing** or **Truncate** to populate the target tables.

Error: Unsupported Character Set Causes Field Data Conversion to Fail

The following error occurs when an unsupported character set causes a field data conversion to fail:

```
“[SOURCE_CAPTURE ]E: Column ‘<column name>' uses an unsupported character set [120112]
A field data conversion failed. (mysql_endpoint_capture.c:2154)
```

This error often occurs because of tables or databases using UTF8MB4 encoding. AWS DMS does not support the UTF8MB4 character set. In addition, check your database's parameters related to connections. The following command can be used to see these parameters:

```
SHOW VARIABLES LIKE '%char%';
```

Troubleshooting PostgreSQL Specific Issues

The following issues are specific to using AWS DMS with PostgreSQL databases.

Topics

Columns of a user defined data type not being migrated correctly

When replicating from a PostgreSQL source, AWS DMS creates the target table with the same data types for all columns, apart from columns with user-defined data types. In such cases, the data type is created as “character varying” in the target.

Error: No schema has been selected to create in

The error "SQL_ERROR SqlState: 3F000 NativeError: 7 Message: ERROR: no schema has been selected to create in" can occur when your JSON table mapping contains a wild card value for the schema but the source database doesn't support that value.

Deletes and updates to a table are not being replicated using CDC

Delete and Update operations during change data capture (CDC) are ignored if the source table does not have a primary key. AWS DMS supports change data capture (CDC) for PostgreSQL tables with primary keys; if a table does not have a primary key, the WAL logs do not include a before image of the database row and AWS DMS cannot update the table. Create a primary key on the source table if you want delete operations to be replicated.

Truncate statements are not being propagated

When using change data capture (CDC), TRUNCATE operations are not supported by AWS DMS.

Preventing PostgreSQL from capturing DDL

You can prevent a PostgreSQL target endpoint from capturing DDL statements by adding the following **Extra Connection Attribute** statement. The **Extra Connection Attribute** parameter is available in the **Advanced** tab of the source endpoint.

```
captureDDLs=N
```

Selecting the schema where database objects for capturing DDL are created

You can control what schema the database objects related to capturing DDL are created in. Add the following **Extra Connection Attribute** statement. The **Extra Connection Attribute** parameter is available in the **Advanced** tab of the target endpoint.

```
ddlArtifactsSchema=xyzddlschema
```

Oracle tables missing after migrating to PostgreSQL

Oracle defaults to uppercase table names while PostgreSQL defaults to lowercase table names. When performing a migration from Oracle to PostgreSQL you will most likely need to supply transformation rules under the table mapping section of your task to convert the case of your table names.

Your tables and data are still accessible; if you migrated your tables without using transformation rules to convert the case of your table names, you will need to enclose your table names in quotes when referencing them.

Task Using View as a Source Has No Rows Copied

A View as a PostgreSQL source endpoint is not supported by AWS DMS.

Troubleshooting Microsoft SQL Server Specific Issues

The following issues are specific to using AWS DMS with Microsoft SQL Server databases.

Topics

Special Permissions for AWS DMS user account to use CDC

The user account used with AWS DMS requires the SQL Server SysAdmin role in order to operate correctly when using change data capture (CDC). CDC for SQL Server is only available for on-premises databases or databases on an EC2 instance.

SQL Server Change Data Capture (CDC) and Amazon RDS

AWS DMS currently does not support change data capture (CDC) from an Amazon RDS SQL Server DB instance. CDC for SQL Server is only available for on-premises databases or databases on an Amazon EC2 instance.

Errors Capturing Changes for SQL Server Database

Errors during change data capture (CDC) can often indicate that one of the pre-requisites was not met. For example, the most common overlooked pre-requisite is a full database backup. The task log indicates this omission with the following error:

```
SOURCE_CAPTURE E: No FULL database backup found (under the 'FULL' recovery model).
To enable all changes to be captured, you must perform a full database backup.
120438 Changes may be missed. (sqlserver_log_queries.c:2623)
```

Review the pre-requisites listed for using SQL Server as a source in Using a Microsoft SQL Server Database as a Source for AWS Database Migration Service (p. 77).

Missing Identity Columns

AWS DMS does not support identity columns when you create a target schema. You must add them after the initial load has completed.

Error: SQL Server Does Not Support Publications

The following error is generated when you use SQL Server Express as a source endpoint:

```
RetCode: SQL_ERROR SqlState: HY000 NativeError: 21106
Message: This edition of SQL Server does not support publications.
```

AWS DMS currently does not support SQL Server Express as a source or target.

Changes Not Appearing in Target

AWS DMS requires that a source SQL Server database be in either ‘FULL’ or ‘BULK LOGGED’ data recovery model in order to consistently capture changes. The ‘SIMPLE’ model is not supported.

The SIMPLE recovery model logs the minimal information needed to allow users to recover their database. All inactive log entries are automatically truncated when a checkpoint occurs. All operations are still logged, but as soon as a checkpoint occurs the log is automatically truncated, which means that it becomes available for re-use and older log entries can be over-written. When log entries are overwritten, changes cannot be captured, and that is why AWS DMS doesn't support the SIMPLE data recovery model. For information on other required pre-requisites for using SQL Server as a source, see Using a Microsoft SQL Server Database as a Source for AWS Database Migration Service (p. 77).

Troubleshooting Amazon Redshift Specific Issues

The following issues are specific to using AWS DMS with Amazon Redshift databases.

Topics

Loading into a Amazon Redshift Cluster in a Different Region Than the AWS DMS Replication Instance

This can't be done. AWS DMS requires that the AWS DMS replication instance and a Redshift cluster be in the same region.

Error: Relation "awsdms_apply_exceptions" already exists

The error "Relation "awsdms_apply_exceptions" already exists" often occurs when a Redshift endpoint is specified as a PostgreSQL endpoint. To fix this issue, modify the endpoint and change the **Target engine** to "redshift."

Errors with Tables Whose Name Begins with "awsdms_changes"

Error messages that relate to tables with names that begin with "awsdms_changes" often occur when two tasks that are attempting to load data into the same Amazon Redshift cluster are running concurrently. Due to the way temporary tables are named, concurrent tasks can conflict when updating the same table.

Seeing Tables in Cluster with Names Like dms.awsdms_changes000000000XXXX

AWS DMS creates temporary tables when data is being loaded from files stored in S3. The name of these temporary tables have the prefix "dms.awsdms_changes." These tables are required so AWS DMS can store data when it is first loaded and before it is placed in its final target table.

Permissions Required to Work with Amazon Redshift

To use AWS DMS with Amazon Redshift, the user account you use to access Amazon Redshift must have the following permissions:

- CRUD (Select, Insert, Update, Delete)
- Bulk Load
- Create, Alter, Drop (if required by the task's definition)

To see all the pre-requisites required for using Amazon Redshift as a target, see Using an Amazon Redshift Database as a Target for AWS Database Migration Service (p. 101).

Troubleshooting Amazon Aurora Specific Issues

The following issues are specific to using AWS DMS with Amazon Aurora databases.

Topics

Error: CHARACTER SET UTF8 fields terminated by ',' enclosed by '"' lines terminated by '\n'

If you are using Amazon Aurora as a target and see an error like the following in the logs, this usually indicates that you have ANSI_QUOTES as part of the SQL_MODE parameter. Having ANSI_QUOTES as part of the SQL_MODE parameter causes double quotes to be handled like quotes and can create issues when you run a task. To fix this error, remove ANSI_QUOTES from the SQL_MODE parameter.

```
2016-11-02T14:23:48 [TARGET_LOAD ]E: Load data sql statement. load data local infile
"/rdsdbdata/data/tasks/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)
```

Reference for AWS Database Migration Service Including Data Conversion Reference and Additional Topics

This reference section includes additional information you may need when using AWS Database Migration Service (AWS DMS), including data type conversion information and additional procedures.

There are several ways to begin a database migration. You can select the AWS DMS console wizard that walks you through each step of the process, or you can do each step by selecting the appropriate task from the navigation pane. You can also use the AWS CLI; for information on using the AWS CLI with AWS DMS, see DMS.

There are a few important things to remember about data types when migrating a database:

- The FLOAT data type is inherently an approximation. The FLOAT data type is special in the sense that when you insert a specific value, it may be represented differently in the database, as it is not an accurate 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, so the migrated value of a FLOAT might not match exactly the value on the source database.

  Here are some articles on the topic:

  IEEE floating point https://en.wikipedia.org/wiki/IEEE_floating_point

  IEEE Floating-Point Representation https://msdn.microsoft.com/en-us/library/0b34tf65.aspx

  Why Floating-Point Numbers May Lose Precision https://msdn.microsoft.com/en-us/library/c151dt3s.aspx
- The UTF-8 4-byte character set (utf8mb4) is not supported and could cause unexpected behavior in a source database. Plan to convert any data using the UTF-8 4-byte character set before migrating.

Topics

Source Data Types

You can find data type conversion tables for databases used as a source for AWS Database Migration Service following.

Topics

Source Data Types for Oracle

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

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

Oracle Data Type	AWS DMS Data Type
BINARY_FLOAT	REAL4
BINARY_DOUBLE	REAL8
BINARY	BYTES
FLOAT (P)	If precision is less than or equal to 24, use REAL4. If precision is greater than 24, use REAL8.
NUMBER (P,S)	When scale is less than 0, use REAL8
NUMBER according to the "Expose number as" property in the Oracle source database settings.	When scale is 0: • And precision is 0, use REAL8. • And precision is greater than or equal to 2, use INT1. • And precision is greater than 2 and less than or equal to 4, use INT2. • And precision is greater than 4 and less than or equal to 9, use INT4. • And precision is greater than 9, use NUMERIC. • And precision is greater than or equal to scale, use NUMERIC. In all other cases, use REAL8.
DATE	DATETIME

Oracle Data Type	AWS DMS Data Type
INTERVAL_YEAR TO MONTH	STRING (with interval year_to_month indication)
INTERVAL_DAY TO SECOND	STRING (with interval day_to_second indication)
TIME	DATETIME
TIMESTAMP	DATETIME
TIMESTAMP WITH TIME ZONE	STRING (with timestamp_with_timezone indication)
TIMESTAMP WITH LOCAL TIME ZONE	STRING (with timestamp_with_local_ timezone indication)
CHAR	STRING
VARCHAR2	STRING
NCHAR	WSTRING
NVARCHAR2	WSTRING
RAW	BYTES
REAL	REAL8
BLOB	BLOB To use this data type with AWS DMS, you must enable the use of BLOB data types for a specific task. AWS DMS supports BLOB data types only in tables that include a primary key.
CLOB	CLOB To use this data type with AWS DMS, you must enable the use of CLOB data types for a specific task. During change data capture (CDC), AWS DMS supports CLOB data types only in tables that include a primary key.
NCLOB	NCLOB To use this data type with AWS DMS, you must enable the use of NCLOB data types for a specific task. During CDC, AWS DMS supports NCLOB data types only in tables that include a primary key.
LONG	CLOB The LONG data type is not supported in batch-optimized apply mode (TurboStream CDC mode). To use this data type with AWS DMS, you must enable the use of LOBs for a specific task. During CDC, AWS DMS supports LOB data types only in tables that have a primary key.
LONG RAW	BLOB The LONG RAW data type is not supported in batch-optimized apply mode (TurboStream CDC mode). To use this data type with AWS DMS, you must enable the use of LOBs for a specific task. During CDC, AWS DMS supports LOB data types only in tables that have a primary key.

Oracle Data Type	AWS DMS Data Type
XMLTYPE	CLOB Support for the XMLTYPE data type requires the full Oracle Client (as opposed to the Oracle Instant Client). When the target column is a CLOB, both full LOB mode and limited LOB mode are supported (depending on the target).

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

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

Note
Virtual columns are not supported.

Source Data Types for Microsoft SQL Server

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

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

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

SQL Server Data Types	AWS DMS Data Types
BIGINT	INT8
BIT	BOOLEAN
DECIMAL	NUMERIC
INT	INT4
MONEY	NUMERIC
NUMERIC (p,s)	NUMERIC
SMALLINT	INT2
SMALLMONEY	NUMERIC
TINYINT	UINT1
REAL	REAL4

SQL Server Data Types	AWS DMS Data Types
FLOAT	REAL8
DATETIME	DATETIME
DATETIME2 (SQL Server 2008 and later)	DATETIME
SMALLDATETIME	DATETIME
DATE	DATE
TIME	TIME
DATETIMEOFFSET	WSTRING
CHAR	STRING
VARCHAR	STRING
VARCHAR (max)	CLOB TEXT To use this data type with AWS DMS, you must enable the use of CLOB data types for a specific task. For SQL Server tables, AWS DMS updates LOB columns in the target even for UPDATE statements that don't change the value of the LOB column in SQL Server. During CDC, AWS DMS supports CLOB data types only in tables that include a primary key.
NCHAR	WSTRING
NVARCHAR (length)	WSTRING
NVARCHAR (max)	NCLOB NTEXT To use this data type with AWS DMS, you must enable the use of NCLOB 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.
BINARY	BYTES
VARBINARY	BYTES

SQL Server Data Types	AWS DMS Data Types
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.
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 do not support this data type.

AWS DMS does not 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.

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 additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

PostgreSQL Data Types	AWS DMS Data Types
INTEGER	INT4
SMALLINT	INT2
BIGINT	INT8
NUMERIC (p,s)	If precision is from 0 through 38, then use NUMERIC. If precision is 39 or greater, then use STRING.
DECIMAL(P,S)	If precision is from 0 through 38, then use NUMERIC. If precision is 39 or greater, then use STRING.
REAL	REAL4
DOUBLE	REAL8
SMALLSERIAL	INT2
SERIAL	INT4
BIGSERIAL	INT8
MONEY	NUMERIC(38,4) Note: The MONEY data type is mapped to FLOAT in SQL Server.
CHAR	WSTRING (1)
CHAR(N)	WSTRING (n)
VARCHAR(N)	WSTRING (n)
TEXT	NCLOB
BYTEA	BLOB
TIMESTAMP	TIMESTAMP
TIMESTAMP (z)	TIMESTAMP

PostgreSQL Data Types	AWS DMS Data Types
TIMESTAMP with time zone	Not supported
DATE	DATE
TIME	TIME
TIME (z)	TIME
INTERVAL	STRING (128)—1 YEAR, 2 MONTHS, 3 DAYS, 4 HOURS, 5 MINUTES, 6 SECONDS
BOOLEAN	CHAR (5) false or true
ENUM	STRING (64)
CIDR	STRING (50)
INET	STRING (50)
MACADDR	STRING (18)
BIT (n)	STRING (n)
BIT VARYING (n)	STRING (n)
UUID	STRING
TSVECTOR	CLOB
TSQUERY	CLOB
XML	CLOB
POINT	STRING (255) "(x,y)"
LINE	STRING (255) "(x,y,z)"
LSEG	STRING (255) "((x1,y1),(x2,y2))"
BOX	STRING (255) "((x1,y1),(x2,y2))"
PATH	CLOB "((x1,y1),(xn,yn))"
POLYGON	CLOB "((x1,y1),(xn,yn))"
CIRCLE	STRING (255) "(x,y),r"
JSON	NCLOB
ARRAY	NCLOB
COMPOSITE	NCLOB
INT4RANGE	STRING (255)
INT8RANGE	STRING (255)
NUMRANGE	STRING (255)
STRRANGE	STRING (255)

Source Data Types for MySQL

The following table shows the MySQL database source data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

> **Note**
> The UTF-8 4-byte character set (utf8mb4) is not supported and could cause unexpected behavior in a source database. Plan to convert any data using the UTF-8 4-byte character set before migrating.

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

MySQL Data Types	AWS DMS Data Types
INT	INT4
MEDIUMINT	INT4
BIGINT	INT8
TINYINT	INT1
DECIMAL(10)	NUMERIC (10,0)
BINARY	BYTES(1)
BIT	BOOLEAN
BIT(64)	BYTES(8)
BLOB	BYTES(66535)
LONGBLOB	BLOB
MEDIUMBLOB	BLOB
TINYBLOB	BYTES(255)
DATE	DATE
DATETIME	DATETIME
TIME	STRING
TIMESTAMP	DATETIME
YEAR	INT2
DOUBLE	REAL8
FLOAT	REAL(DOUBLE) The supported FLOAT range is -1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308 If the FLOAT values are not in the range specified here, map the FLOAT data type to the STRING data type.
VARCHAR (45)	WSTRING (45)

MySQL Data Types	AWS DMS Data Types
VARCHAR (2000)	WSTRING (2000)
VARCHAR (4000)	WSTRING (4000)
VARBINARY (4000)	BYTES (4000)
VARBINARY (2000)	BYTES (2000)
CHAR	WSTRING
TEXT	WSTRING (65535)
LONGTEXT	NCLOB
MEDIUMTEXT	NCLOB
TINYTEXT	WSTRING (255)
GEOMETRY	BLOB
POINT	BLOB
LINESTRING	BLOB
POLYGON	BLOB
MULTIPOINT	BLOB
MULTILINESTRING	BLOB
MULTIPOLYGON	BLOB
GEOMETRYCOLLECTION	BLOB

Note
If the DATETIME and TIMESTAMP data types are specified with a “zero” value (that is, 0000-00-00), you need to make sure that the target database in the replication task supports "zero" values for the DATETIME and TIMESTAMP data types. Otherwise, they are recorded as null on the target.

The following MySQL data types are supported in full load only:

MySQL Data Types	AWS DMS Data Types
ENUM	STRING
SET	STRING

Source Data Types for SAP ASE

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

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

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

SAP ASE Data Types	AWS DMS Data Types
BIGINT	INT8
BINARY	BYTES
BIT	BOOLEAN
CHAR	STRING
DATE	DATE
DATETIME	DATETIME
DECIMAL	NUMERIC
DOUBLE	REAL8
FLOAT	REAL8
IMAGE	BLOB
INT	INT4
MONEY	NUMERIC
NCHAR	WSTRING
NUMERIC	NUMERIC
NVARCHAR	WSTRING
REAL	REAL4
SMALLDATETIME	DATETIME
SMALLINT	INT2
SMALLMONEY	NUMERIC
TEXT	CLOB
TIME	TIME
TINYINT	UINT1
UNICHAR	UNICODE CHARACTER
UNITEXT	NCLOB
UNIVARCHAR	UNICODE
VARBINARY	BYTES
VARCHAR	STRING

AWS DMS does not support tables that include fields with the following data types:

- User-defined type (UDT)

Source Data Types for MongoDB

Data migration that uses MongoDB as a source for AWS DMS supports most MongoDB data types. The following table shows 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 https://docs.mongodb.com/manual/reference/bson-types.

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

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

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

Target Data Types

You can find data type conversion tables for databases used as a target for AWS Database Migration Service following.

Topics

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 In all other cases: VARCHAR2 (Length)
UINT1	NUMBER (3)
UINT2	NUMBER (5)
UINT4	NUMBER (10)
UINT8	NUMBER (19)
WSTRING	NVARCHAR2 (length)
BLOB	BLOB

AWS DMS Data Type	Oracle Data Type
	To use this data type with AWS DMS, you must enable the use of BLOBs for a specific task. BLOB data types are supported only in tables that include a primary key
CLOB	CLOB To use this data type with AWS DMS, you must enable the use of CLOBs for a specific task. During CDC, CLOB data types are supported only in tables that include a primary key.
NCLOB	NCLOB To use this data type with AWS DMS, you must enable the use of NCLOBs for a specific task. During CDC, NCLOB data types are supported only in tables that include a primary key.
XMLTYPE	The XMLTYPE target data type is only relevant in Oracle-to-Oracle replication tasks. When the source database is Oracle, the source data types are replicated "as is" to the Oracle target. For example, an XMLTYPE data type on the source is created as an XMLTYPE data type on the target.

Target Data Types for Microsoft SQL Server

The following table shows the Microsoft SQL Server target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types. For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

AWS DMS Data Type	SQL Server Data Type
BOOLEAN	TINYINT
BYTES	VARBINARY(length)
DATE	For SQL Server 2008 and later, 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 later, use DATETIME2 (%d). For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).
DATETIME	For SQL Server 2008 and later, use DATETIME2 (scale). For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37).
INT1	SMALLINT
INT2	SMALLINT
INT4	INT

AWS DMS Data Type	SQL Server Data Type
INT8	BIGINT
NUMERIC	NUMBER (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 later, use DATETIME2. • For earlier versions, if the scale is 3 or less use DATETIME. In all other cases, use VARCHAR (37). If the column is not a date or time column, use VARCHAR (length).
UINT1	TINYINT
UINT2	SMALLINT
UINT4	INT
UINT8	BIGINT
WSTRING	NVARCHAR (length)
BLOB	VARBINARY(max) IMAGE To use this data type with AWS DMS, you must enable the use of BLOBs for a specific task. AWS DMS supports BLOB data types only in tables that include a primary key.
CLOB	VARCHAR(max) To use this data type with AWS DMS, you must enable the use of CLOBs for a specific task. During CDC, AWS DMS supports CLOB data types only in tables that include a primary key.
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.

Target Data Types for PostgreSQL

The PostgreSQL database endpoint for AWS DMS supports most PostgreSQL database data types. The following table shows the PostgreSQL database target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types. Unsupported data types are listed following the table.

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

AWS DMS Data Type	PostgreSQL Data Type
BOOL	BOOL
BYTES	BYTEA
DATE	DATE
TIME	TIME
TIMESTAMP	If the scale is from 0 through 6, then use TIMESTAMP. If the scale is from 7 through 9, then use VARCHAR (37).
INT1	SMALLINT
INT2	SMALLINT
INT4	INTEGER
INT8	BIGINT
NUMERIC	DECIMAL (P,S)
REAL4	FLOAT4
REAL8	FLOAT8
STRING	If the length is from 1 through 21,845, then use VARCHAR (length in bytes). If the length is 21,846 through 2,147,483,647, then use VARCHAR (65535).
UINT1	SMALLINT
UINT2	INTEGER
UINT4	BIGINT
UINT8	BIGINT
WSTRING	If the length is from 1 through 21,845, then use VARCHAR (length in bytes). If the length is 21,846 through 2,147,483,647, then use VARCHAR (65535).
BCLOB	BYTEA
NCLOB	TEXT
CLOB	TEXT

Note
When replicating from a PostgreSQL source, AWS DMS creates the target table with the same data types for all columns, apart from columns with user-defined data types. In such cases, the data type is created as "character varying" in the target.

Target Data Types for MySQL

The following table shows the MySQL database target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

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)"
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 65535, then use TEXT. If the length is from 65536 through 2147483647, then use LONGTEXT. If the length is 0, then use LONGTEXT (full LOB support).

Target Data Types for SAP ASE

The following table shows the SAP ASE database target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

AWS DMS Data Types	SAP ASE Data Types
BOOLEAN	BIT
BYTES	VARBINARY (Length)
DATE	DATE
TIME	TIME
TIMESTAMP	If scale is => 0 and =< 6, then: BIGDATETIME If scale is => 7 and =< 9, then: VARCHAR (37)
INT1	TINYINT
INT2	SMALLINT
INT4	INTEGER

AWS DMS Data Types	SAP ASE Data Types
INT8	BIGINT
NUMERIC	NUMERIC (p,s)
REAL4	REAL
REAL8	DOUBLE PRECISION
STRING	VARCHAR (Length)
UINT1	TINYINT
UINT2	UNSIGNED SMALLINT
UINT4	UNSIGNED INTEGER
UINT8	UNSIGNED BIGINT
WSTRING	VARCHAR (Length)
BLOB	IMAGE
CLOB	UNITEXT
NCLOB	TEXT

AWS DMS does not support tables that include fields with the following data types. Replicated columns with these data types will show as null.

- User-defined type (UDT)

Target Data Types for Amazon Redshift

The Amazon Redshift endpoint for AWS DMS supports most Amazon Redshift data types. The following table shows the Amazon Redshift target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

AWS DMS Data Types	Amazon Redshift Data Types
BOOLEAN	BOOL
BYTES	VARCHAR (Length)
DATE	DATE
TIME	VARCHAR(20)
DATETIME	If the scale is => 0 and =< 6, then: TIMESTAMP (s) If the scale is => 7 and =< 9, then:

AWS DMS Data Types	Amazon Redshift Data Types
	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 and =< 65535, then: VARCHAR (Length in Bytes) If the length is => 65536 and =< 2147483647, then: VARCHAR (65535)
UINT1	INT2
UINT2	INT2
UINT4	INT4
UINT8	NUMERIC (20,0)
WSTRING	If the length is => 1 and =< 65535, then: NVARCHAR (Length in Bytes) If the length is => 65536 and =< 2147483647, then: NVARCHAR (65535)
BLOB	VARCHAR (Max LOB Size *2) Note: The maximum LOB size cannot exceed 31 KB.
NCLOB	NVARCHAR (Max LOB Size) Note: The maximum LOB size cannot exceed 63 KB.

AWS DMS Data Types	Amazon Redshift Data Types
CLOB	VARCHAR (Max LOB Size) Note: The maximum LOB size cannot exceed 63 KB.

Target Data Types for Amazon DynamoDB

The Amazon DynamoDB endpoint for Amazon AWS DMS supports most Amazon DynamoDB data types. The following table shows the Amazon AWS DMS target data types that are supported when using AWS DMS and the default mapping from AWS DMS data types.

For additional information about AWS DMS data types, see Data Types for AWS Database Migration Service (p. 198).

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

Data Types for AWS Database Migration Service

AWS Database Migration Service uses built-in data types to migrate data from one database to another. The following table shows the built-in data types and their descriptions.

AWS DMS Data Types	Description
STRING	A character string.
WSTRING	A double-byte character string.
BOOLEAN	A Boolean value.
BYTES	A binary data value.
DATE	A date value: year, month, day.
TIME	A time value: hour, minutes, seconds.
DATETIME	A timestamp value: year, month, day, hour, minute, second, fractional seconds. The fractional seconds have a maximum scale of 9 digits.
INT1	A one-byte, signed integer.
INT2	A two-byte, signed integer.
INT4	A four-byte, signed integer.
INT8	An eight-byte, signed integer.
NUMERIC	An exact numeric value with a fixed precision and scale.
REAL4	A single-precision floating-point value.
REAL8	A double-precision floating-point value.
UINT1	A one-byte, unsigned integer.
UINT2	A two-byte, unsigned integer.
UINT4	A four-byte, unsigned integer.
UINT8	An eight-byte, unsigned integer.
BLOB	Binary large object. This data type can be used only with Oracle endpoints.
CLOB	Character large object.
NCLOB	Native character large object.

Using Extra Connection Attributes with AWS Database Migration Service

You can specify additional connection attributes when creating an endpoint for AWS Database Migration Service. The following database engine specific sections show possible settings.

MySQL

Role	Name	Description
Source	`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: `eventsPollInterval=5`
	`initstmt=SET time-zone`	Specifies the time zone for the source MySQL database. Default value: UTC Valid values: Any three or four character abbreviation for the time zone you want to use. Valid values are the standard time zone abbreviations for the operating system hosting the source MySQL database. Example: `initstmt=SET time_zone=UTC`
	`afterConnectScript`	Specifies a script to run immediately after AWS DMS connects to the endpoint. Valid values: Any SQL statement separated by a semi-colon. Example: `afterConnectScript=ALTER SESSION SET CURRENT_SCHEMA = system;`
Target	`targetDbType`	Specifies where to migrate source tables on the target, either to a single database or multiple databases. Default value: `MULTIPLE_DATABASES` Valid values: {`SPECIFIC_DATABASE, MULTIPLE_DATABASES`} Example: `targetDbType=MULTIPLE_DATABASES`
	`parallelLoadThreads`	Improves performance when loading data into the MySQL target database. Specifies how many threads to use to load the data into the MySQL target database. Note that setting a large number of threads may have an adverse

Role	Name	Description
		effect on database performance since a separate connection is required for each thread. Default value: 2 Valid values: 1-5 Example: `parallelLoadThreads=1`
	`initstmt=SET FOREIGN_KEY_CHECKS=0`	Disables foreign key checks.
	`initstmt=SET time-zone`	Specifies the time zone for the target MySQL database. Default value: UTC Valid values: A three or four character abbreviation for the time zone you want to use. Valid values are the standard time zone abbreviations for the operating system hosting the target MySQL database. Example: `initstmt=SET time_zone=UTC`
	`afterConnectScript=SET character_set_connection='latin`	Improves the performance of certain operations on the target MySQL database when converting from Latin1 to UTF8.
	`maxFileSize`	Specifies the maximum size (in KB) of any CSV file used to transfer data to MySQL. Default value: 32768 KB (32 MB) Valid values: 1 - 1048576 Example: `maxFileSize=512`

PostgreSQL

Role	Name	Description
Source	`captureDDLs`	In order 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 the section Removing AWS Database Migration Service Artifacts from a PostgreSQL Source Database (p. 85). If this value is set to N, you do not have to create table/triggers on the source database. For more information, see Migrating an Amazon RDS for PostgreSQL Database Without Using the Master User Account (p. 84) Streamed DDL events are captured.

Role	Name	Description
		Default value: Y Valid values: Y/N Example: `captureDDLs=Y`
	`ddlArtifactsSchema`	The schema in which the operational DDL database artifacts are created. Default value: public Valid values: String Example: `ddlArtifactsSchema=xyzddlschema`
Target	`maxFileSize`	Specifies the maximum size (in KB) of any CSV file used to transfer data to PostgreSQL. Default value: 32768 KB (32 MB) Valid values: 1 - 1048576 Example: `maxFileSize=512`

Oracle

Role	Name	Description
Source	`addSupplementalLogging`	Set this attribute to automatically set up supplemental logging for the Oracle database. Default value: N Valid values: Y/N Example: `addSupplementalLogging=Y` **Note** If you use this option, you still need to enable supplemental logging at the database level using the following statement: `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA`
	`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 set to N, you must also add the setting useBfile=Y. For more information, see Using Oracle LogMiner or

Role	Name	Description
		Oracle Binary Reader for Change Data Capture (CDC) (p. 69). Default value: Y Valid values: Y/N Example: `useLogminerReader=N; useBfile=Y` If the Oracle source database is using Oracle ASM (Automatic Storage Management), the extra connection parameter needs to include the asm username and asm server address. The password field will also need to have both passwords, the source user password, as well as the ASM password. Example: `useLogminerReader=N;asm_user=<asm_username>;asm_server=<first_RAC_server_ip_address>/+ASM` If the Oracle source database is using Oracle ASM (Automatic Storage Management), the endpoint password field needs to have both the Oracle user password and the ASM password, separated by a comma. Example: <oracle_user_password>,<asm_user_password>
	`retryInterval`	Specifies the number of seconds that the system waits before resending a query. Default value: 5 Valid values: number starting from 1 Example: `retryInterval=6`
	`archivedLogDestId`	Specifies the destination of the archived redo logs. The value should be the same as the DEST_ID number in the $archived_log table. When working with multiple log destinations (DEST_ID), we recommended that you to specify an Archived redo logs location identifier. This will improve performance by ensuring that the correct logs are accessed from the outset. Default value:0 Valid values: Number Example: `archivedLogDestId=1`

Role	Name	Description
	`archivedLogsOnly`	When this field is set to Y, AWS DMS will only access the archived redo logs. If the archived redo logs ares stored on ASM only, the AWS DMS user needs to be granted the ASM privileges. Default value: N Valid values: Y/N Example: `archivedLogDestId=Y`
	`numberDataTypeScale`	Specifies the Number scale. You can select a scale up to 38 or you can select FLOAT. By default the NUMBER data type is converted to precision 38, scale 10. Default value: 10 Valid values: -1 to 38 (-1 for FLOAT) Example: `numberDataTypeScale =12`
	`afterConnectScript`	Specifies a script to run immediately after AWS DMS connects to the endpoint. Valid values: Any SQL statement separated by a semi-colon. Example: `afterConnectScript=ALTER SESSION SET CURRENT_SCHEMA = system;`
Target	`useDirectPathFullLoad`	Use direct path full load, specify this to enable/ disable the OCI direct path protocol for bulk loading Oracle tables. Default value: Y Valid values: Y/N Example: `useDirectPathFullLoad=N`
	`charLengthSemantics`	Column length semantics specifies whether the length of a column is in bytes or in characters. Set this value to `CHAR`. Example: `charLengthSemantics=CHAR`

SQL Server

Role	Name	Description
Source	`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 may not contain

Role	Name	Description
		all of the unread changes. When this occurs, AWS DMS accesses the backup log to capture the missing changes. To minimize the need to access the backup log, AWS DMS prevents truncation using one of the following methods: 1. **Start transactions in the database:** This is the default method. When this method is used, AWS DMS prevents TLOG truncation by mimicking a transaction in the database. As long as such a transaction is open, changes that appear after the transaction started will not be truncated. If you need Microsoft Replication to be enabled in your database, then you must choose this method. 2. **Exclusively use sp_repldone within a single task:** When this method is used, AWS DMS reads the changes and then uses sp_repldone to mark the TLOG transactions as ready for truncation. Although this method does not involve any transactional activities, it can only be used when Microsoft Replication is not running. Also, when using this method, only one AWS DMS task can access the database at any given time. Therefore, if you need to run parallel AWS DMS tasks against the same database, use the default method. Default value: `RELY_ON_SQL_SERVER_REPLICATION_AGENT` Valid values: {`EXCLUSIVE_AUTOMATIC_TRUNCATION`, `RELY_ON_SQL_SERVER_REPLICATION_AGENT`} Example: `safeguardPolicy= RELY_ON_SQL_SERVER_REPLICATION_AGENT`
Target	`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: Y Valid values: Y/N Example: `useBCPFullLoad=Y`
	`BCPPacketSize`	The maximum size of the packets (in bytes) used to transfer data using BCP. Default value: 16384 Valid values: 1 - 100000 Eg : `BCPPacketSize=16384`

Role	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) will be created on the specified filegroup.
		
		Default value: n/a
		
		Valid values: String
		
		Example: `controlTablesFileGroup=filegroup1`
		
		The following is an example of a command for creating a filegroup:
		
		`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`

Amazon Redshift

Role	Name	Description
Target	`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: `maxFileSize=512`
	`fileTransferUploadStreams`	Specifies the number of threads used to upload a single file. Default value: 10 Valid values: 1 - 64 Example: `fileTransferUploadStreams=20`

SAP Adaptive Server Enterprise (ASE)

Role	Name	Description
Target	`enableReplication`	Set to `Y` to automatically enable SAP ASE replication. This is only required if SAP ASE replication has not been enabled already.
	`additionalConnectionProperties`	Any additional ODBC connection parameters that you want to specify.

Note
If the user name or password specified in the connection string contains non-Latin characters (for example, Chinese), the following property is required: `charset=gb18030`

Using ClassicLink with AWS Database Migration Service

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

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

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

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

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

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

   1. Sign in to the AWS Management Console and choose AWS Database Migration Service. Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required for database migration, see IAM Permissions Needed to Use AWS DMS (p. 43).
   2. On the **Dashboard** page, choose **Replication Instance**. Follow the instructions at Step 2: Create a Replication Instance (p. 25) to create a replication instance.
   3. After you have created the AWS DMS replication instance, open the EC2 service console. Select **Network Interfaces** from the navigation pane.
   4. Select the *DMSNetworkInterface*, and then choose **Change Security Groups** from the **Actions** menu.

5. Select the security group you want to use for the replication instance and the target DB instance.

2. Step 2: Associate the security group from the last step with the target DB instance.

   To associate a security group with a DB instance

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

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

   To set up a proxy server on an EC2 Classic instance

   1. 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
```

   2. 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
```

3. 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>;
       }
   }
   ```

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

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

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

   Use ClassicLink to connect the proxy server with the target VPC

   1. Open the EC2 console and select the EC2 Classic instance that is running the proxy server.
   2. Select **ClassicLink** under **Actions**, then select **Link to VPC**.
   3. Select the security group you used earlier in this procedure.
   4. Select **Link to VPC**.
5. Step 5: Create AWS DMS endpoints using the procedure at Step 3: Specify Database Endpoints (p. 29). You must use the internal EC2 DNS hostname of the proxy as the server name when specifying the source endpoint.
6. Step 6: Create a AWS DMS task using the procedure at Step 4: Create a Task (p. 33).

Document History

The following table describes the important changes to the documentation since the last release of the AWS Database Migration Service.

- **API version:** 20160101
- **Latest documentation update:** July 11, 2017

Change	Description	Date Changed
New feature	Added support for AWS CloudFormation templates. For more information, see AWS DMS Support for AWS CloudFormation (p. 18).	July 11, 2017
New feature	Added support for using Amazon Dynamo as a target. For more information, see Using an Amazon DynamoDB Database as a Target for AWS Database Migration Service (p. 106).	April 10, 2017
New feature	Added support for using MongoDB as a source. For more information, see Using MongoDB as a Source for AWS Database Migration Service (p. 91).	April 10, 2017
New feature	Added support for using Amazon S3 as a target. For more information, see Using Amazon S3 as a Target for AWS Database Migration Service (p. 104).	March 27, 2017
New feature	Adds support for reloading database tables during a migration task. For more information, see Reloading Tables During a Task (p. 122).	March 7, 2017
New feature	Added support for events and event subscriptions. For more information, see Working with Events and Notifications (p. 158).	January 26, 2017
New feature	Added support for SSL endpoints for Oracle. For more information, see SSL Support for an Oracle Endpoint (p. 60).	December 5, 2016

Change	Description	Date Changed
New feature	Added support for using Change Data Capture (CDC) with an Amazon RDS PostgreSQL DB instance. For more information, see Setting Up an Amazon RDS PostgreSQL DB Instance as a Source (p. 83).	September 14, 2016
New region support	Added support for the Asia Pacific (Mumbai), Asia Pacific (Seoul), and South America (São Paulo) regions. For a list of supported regions, see What Is AWS Database Migration Service? (p. 1).	August 3, 2016
New feature	Added support for ongoing replication. For more information, see Ongoing Replication (p. 41).	July 13, 2016
New feature	Added support for secured connections using SSL. For more information, see Using SSL With AWS Database Migration Service (p. 57).	July 13, 2016
New feature	Added support for SAP Adaptive Server Enterprise (ASE) as a source or target endpoint. For more information, see Using a SAP ASE Database as a Source for AWS Database Migration Service (p. 89) and Using a SAP ASE Database as a Target for AWS Database Migration Service (p. 104).	July 13, 2016
New feature	Added support for filters to move a subset of rows from the source database to the target database. For more information, see Using Source Filters (p. 145).	May 2, 2016
New feature	Added support for Amazon Redshift as a target endpoint. For more information, see Using an Amazon Redshift Database as a Target for AWS Database Migration Service (p. 101).	May 2, 2016
General availability	Initial release of AWS Database Migration Service.	March 14, 2016
Public preview release	Released the preview documentation for AWS Database Migration Service.	January 21, 2016