AWS Database Migration Service Best Practices

August 2016

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

Today, as many companies move database workloads to Amazon Web Services (AWS), they are often also interested in changing their primary database engine. Most current methods for migrating databases to the cloud or switching engines require an extended outage. The AWS Database Migration Service helps organizations to migrate database workloads to AWS or change database engines while minimizing any associated downtime. This paper outlines best practices for using AWS DMS.

Introduction

AWS Database Migration Service allows you to migrate data from a source database to a target database. During a migration, the service tracks changes being made on the source database so that they can be applied to the target database to eventually keep the two databases in sync. Although the source and target databases can be of the same engine type, they don't need to be. The possible types of migrations are:

1. Homogenous migrations (migrations between the same engine types)
2. Heterogeneous migrations (migrations between different engine types)

At a high level, when using AWS DMS a user provisions a replication server, defines source and target endpoints, and creates a task to migrate data between the source and target databases. A typical task consists of three major phases: the full load, the application of cached changes, and ongoing replication.

During the full load, data is loaded from tables on the source database to tables on the target database, eight tables at a time (the default). While the full load is in progress, changes made to the tables that are being loaded are cached on the replication server; these are the cached changes. It's important to know that the capturing of changes for a given table doesn't begin until the full load for *that* table starts; in other words, the start of change capture for each individual table will be different. After the full load for a given table is complete, you can begin to apply the cached changes for that table immediately. When ALL tables are loaded, you begin to collect changes as transactions for the ongoing replication phase. After all cached changes are applied, your tables are consistent transactionally and you move to the ongoing replication phase, applying changes as transactions.

Upon initial entry into the ongoing replication phase, there will be a backlog of transactions causing some lag between the source and target databases. After working through this backlog, the system will eventually reach a steady state. At this point, when you're ready, you can:

- Shut down your applications.
- Allow any remaining transactions to be applied to the target.
- Restart your applications pointing at the new target database.

AWS DMS will create the target schema objects that are needed 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 will create tables, primary keys, and in some cases, unique indexes. It will not create secondary indexes, non-primary key constraints, data defaults, or other objects that are not required to efficiently migrate the data from the source system. 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, you can accomplish this by using your engine's native tools to perform a no-data export/import of the schema. If your migration is heterogeneous, you can use the AWS Schema Conversion Tool (AWS SCT) to generate a complete target schema for you.

> **Note** Any inter-table dependencies, such as foreign key constraints, must be disabled during the "full load" and "cached change application" phases of AWS DMS processing. Also, if performance is an issue, it will be beneficial to remove or disable secondary indexes during the migration process.

Provisioning a Replication Server

AWS DMS is a managed service that runs on an Amazon Elastic Compute Cloud (Amazon EC2) instance. The service connects to the source database, reads the source data, formats the data for consumption by the target database, and loads the data into the target database. Most of this processing happens in memory, however, large transactions may require some buffering on disk. Cached transactions and log files are also written to disk. The following sections describe what you should consider when selecting your replication server.

Instance Class

Some of the smaller instance classes are sufficient for testing the service or for small migrations. If your migration involves a large number of tables, or if you intend to run multiple concurrent replication tasks, you should consider using one of the larger instances because the service consumes a fair amount of memory and CPU.

> **Note** T2 type instances are designed to provide moderate baseline performance and the capability to burst to significantly higher performance, as required by your workload. They are intended for workloads that don't use the full CPU often or consistently, but that occasionally need to burst. T2 instances are well suited for general purpose workloads, such as web servers, developer environments, and small databases. If you're troubleshooting a slow migration and using a T2 instance type, look at the CPU Utilization host metric to see if you're bursting over the baseline for that instance type.

Storage

Depending on the instance class, your replication server will come with either 50 GB or 100 GB of data storage. This storage is used for log files and any cached changes that are collected during the load. If your source system is busy or takes large transactions, or if you're running multiple tasks on the replication server, you might need to increase this amount of storage. However, the default amount is usually sufficient.

> **Note** All storage volumes in AWS DMS are GP2 or General Purpose SSDs. GP2 volumes come with a base performance of three I/O Operations Per Second (IOPS), with abilities to burst up to 3,000 IOPS on a credit basis. As a rule of thumb, check the ReadIOPS and WriteIOPS metrics for the replication instance and be sure the sum of these values does not cross the base performance for that volume.

Multi-AZ

Selecting a Multi-AZ instance can protect your migration from storage failures. Most migrations are transient and not intended to run for long periods of time. If you're using AWS DMS for ongoing replication purposes, selecting a Multi-AZ instance can improve your availability should a storage issue occur.

Source Endpoint

The change capture process, used when replicating ongoing changes, collects changes from the database logs by using the database engines native API, no client side install is required. Each engine has specific configuration requirements for exposing this change stream to a given user account (for details, see the AWS Key Management Service documentation). 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.)

> **Note** When capturing changes from an Amazon Relational Database Service (Amazon RDS) source, ensure backups are enabled and the source is configured to retain change logs for a sufficiently long time (usually 24 hours).

Target Endpoint

Whenever possible, AWS DMS attempts to create the target schema for you, including underlying tables and primary keys. However, sometimes this isn't possible. For example, when the target is Oracle, AWS DMS doesn't create the target schema for security reasons. In MySQL, you have the option through extra connection parameters to have AWS DMS migrate objects to the specified database or to have AWS DMS create each database for you as it finds the database on the source.

> **Note** For the purposes of this paper, in Oracle a user and schema are synonymous. In MySQL, schema is synonymous with database. Both SQL Server and Postgres have a concept of database AND schema. In this paper, we're referring to the schema.

Task

The following section highlights common and important options to consider when creating a task.

Migration Type

- **Migrate existing data.** If you can afford an outage that's long enough to copy your existing data, this is a good option to choose. This option simply migrates the data from your source system to your target, creating tables as needed.
- **Migrate existing data and replicate ongoing changes.** This option performs a full data load while capturing changes on the source. After the full load is complete, captured changes are applied to the target. Eventually, the application of changes will reach a steady state. At that point, you can shut down your applications, let the remaining changes flow through to the target, and restart your applications to point at the target.
- **Replicate data changes only.** In some situations it may be more efficient to copy the existing data by using a method outside of AWS DMS. For example, in a homogeneous migration, using native export/import tools can be more efficient at loading the bulk data. When this is the case, you can use AWS DMS to replicate changes as of the point in time at which you started your bulk load to bring and keep your source and target systems in sync. When replicating data changes only, you need to specify a time from which AWS DMS will begin to read changes from the database change logs. It's important to keep these logs available on the server for a period of time to ensure AWS DMS has access to these changes. This is typically achieved by keeping the logs available for 24 hours (or longer) during the migration process.

Start Task on Create

By default, AWS DMS will start your task as soon as you create it. In some situations, it's helpful to postpone the start of the task. For example, using the AWS Command Line Interface (AWS CLI), you may have a process that creates a task and a different process that starts the task, based on some triggering event.

Target Table Prep Mode

Target table prep mode tells AWS DMS what to do with tables that already exist. If a table that is a member of a migration doesn't yet exist on the target, AWS DMS will create the table. By default, AWS DMS will drop and recreate any existing tables on the target in preparation for a full load or a reload. If you're pre-creating your schema, set your target table prep mode to truncate, causing AWS DMS to truncate existing tables prior to load or reload. When the table

prep mode is set to do nothing, any data that exists in the target tables is left as is. This can be useful when consolidating data from multiple systems into a single table using multiple tasks.

AWS DMS performs these steps when it creates a target table:

- The source database column data type is converted into an intermediate AWS DMS data type.
- The AWS DMS data type is converted into the target data type.

This data type conversion is performed for both heterogeneous and homogeneous migrations. In a homogeneous migration, this data type conversion may lead to target data types not matching source data types exactly. For example, in some situations it's necessary to triple the size of varchar columns to account for multi-byte characters. We recommend going through the AWS DMS documentation on source and target data types to see if all the data types you use are supported. If the resultant data types aren't to your liking when you're using AWS DMS to create your objects, you can pre-create those objects on the target database. If you do pre-create some or all of your target objects, be sure to choose the truncate or do nothing options for target table preparation mode.

LOB Controls

Due to their unknown and sometimes large size, large objects (LOBs) require more processing and resources than standard objects. To help with tuning migrations of systems that contain LOBs, AWS DMS offers the following options:

- **Don't include LOB columns.** When this option is selected, tables that include LOB columns are migrated in full, however, any columns containing LOBs will be omitted.
- **Full LOB mode.** When you select full LOB mode, AWS DMS assumes no information regarding the size of the LOB data. LOBs are migrated in full, in successive pieces, whose size is determined by the *LOB chunk size*. Changing the LOB chunk size affects the memory consumption of AWS DMS; a large LOB chunk size requires more memory and processing. Memory is consumed per LOB, per row. If you have a table containing three LOBs, and are moving data 1,000 rows at a time, an LOB chunk size of 32 k will require *3*32*1000 = 96,000 k* of memory for processing. Ideally, the LOB chunk size should be set to allow AWS DMS to retrieve the majority of LOBs in as few chunks as possible. For example, if 90 percent of your LOBs are less than 32 k, then setting the LOB chunk size to 32 k would be reasonable, assuming you have the memory to accommodate the setting.
- **Limited LOB mode.** When limited LOB mode is selected, any LOBs that are larger than *max LOB size* are truncated to *max LOB size* and a warning is issued to the log file. Using limited LOB mode is almost always more efficient and faster than full LOB mode. You can usually query your data dictionary to determine the size of the largest LOB in a table, setting *max LOB size* to something slightly larger than this (don't forget to account for multi-byte characters). If you have a table in which most LOBs are small, with a few

large outliers, it may be a good idea to move the large LOBs into their own table and use two tasks to consolidate the tables on the target.

LOB columns are transferred only if the source table has a primary key or a unique index on the table. Transfer of data containing LOBs is a two-step process:
1. The containing row on the target is created without the LOB data.
2. The table is updated with the LOB data.

The process was designed this way to accommodate the methods source database engines use to manage LOBs and changes to LOB data.

Enable Logging

It's always a good idea to enable logging because many informational and warning messages are written to the logs. However, be advised that you'll incur a small charge, as the logs are made accessible by using Amazon CloudWatch.

Find appropriate entries in the logs by looking for lines that start with the following:

- Lines starting with "**E:"** – Errors
- Lines starting with "**W:**" – Warnings
- Lines starting with "**I:**" – Informational messages

You can use grep (on UNIX-based text editors) or search (for Windows-based text editors) to find exactly what you're looking for in a huge task log.

Monitoring Your Tasks

There are several options for monitoring your tasks using the AWS DMS console.

Host Metrics

You can find host metrics on your replication instances monitoring tab. Here, you can monitor whether your replication instance is sized appropriately.

Replication Task Metrics

Metrics for replication tasks, including incoming and committed changes, and latency between the replication host and source/target databases can be found on the task monitoring tab for each particular task.

Table Metrics

Individual table metrics can be found under the table statistics tab for each individual task. These metrics include: the number of rows loaded during the full load; the number of inserts, updates, and deletes since the task started; and the number of DDL operations since the task started.

Performance Expectations

There are a number of factors that will affect the performance of your migration: resource availability on the source, available network throughput, resource capacity of the replication server, ability of the target to ingest changes, type and distribution of source data, number of objects to be migrated, and so on. In our tests, we have been able to migrate a terabyte of data in approximately 12–13 hours (under "ideal" conditions). Our tests were performed using source databases running on EC2, and in Amazon RDS with target databases in 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.

Increasing Performance

The performance of your migration will be limited by one or more bottlenecks you encounter along the way. The following are 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 may see some performance improvement by increasing this slightly when you're using a very large replication server; however, at some point increasing this parallelism will reduce performance. If your replication server is smaller, you should reduce this number.

Remove Bottlenecks on the Target

During the migration, try to remove any processes that would compete for write resources on your target database. This includes disabling unnecessary triggers, validation, secondary indexes, and so on. When migrating to an RDS database, it's a good idea to disable backups and Multi-AZ on the target until you're ready to cutover. Similarly, when migrating to non-RDS systems, disabling any logging on the target until cutover is usually a good idea.

Use Multiple Tasks

Sometimes using multiple tasks for a single migration can improve performance. If you have sets of tables that don't participate in common transactions, it may be possible to divide your migration into multiple tasks.

> **Note** Transactional consistency is maintained within a task. Therefore, it's important that tables in separate tasks don't participate in common transactions. Additionally, each task will independently read the transaction stream. Therefore, be careful not to put too much stress on the source system. For very large systems or systems with many LOBs, you may also consider using multiple replication servers, each containing one or more tasks. A review of the

host statistics of your replication server can help you determine whether this might be a good option.

Improving LOB Performance

Pay attention to the LOB parameters. Whenever possible, use limited LOB mode. If you have a table which consists of a few large LOBs and mostly smaller LOBs, consider breaking up the table into a table that contains the large LOBs and a table that contains the small LOBs prior to the migration. You can then use a task in limited LOB mode to migrate the table containing small LOBs, and a task in full LOB mode to migrate the table containing large LOBs.

Important In LOB processing, LOBs are migrated using a two-step process: first, the containing row is created without the LOB, and then the row is updated with the LOB data. Therefore, even if the LOB column is NOT NULLABLE on the source, it must be nullable on the target during the migration.

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 turn on *batch optimized apply*. Batch optimized apply groups transactions and applies them in batches for efficiency purposes.

Note Using batch optimized apply will almost certainly violate referential integrity constraints. Therefore, you should disable them during the migration process and enable them as part of the cutover process.

Reducing Load on Your Source System

During a migration, AWS DMS performs a full table scan of each source table being processed (usually in parallel). Additionally, each task periodically queries the source for change information. To perform change processing, you may be required to increase the amount of data written to your database's change log. If you find you are overburdening your source database, you can reduce the number of tasks or tables per task of your migration. If you prefer not to add load to your source, consider performing the migration from a read copy of your source system.

Note Using a read copy will increase the replication lag.

Frequently Asked Questions

What Are the Main Reasons for Performing a Database migration?

Would you like to move your database from a commercial engine to an open source alternative? Perhaps you want to move your on-premises database into the AWS Cloud. Would you like to divide your database into functional pieces? Maybe you'd like to move some of your data from RDS into Amazon Redshift. These and other similar scenarios can be considered "database migrations".

What Steps Does a Typical Migration Project Include?

This of course depends on the reason for and type of migration you choose to perform. At a minimum, you'll want to do the following.

Perform an Assessment

In an assessment, you determine the basic framework of your migration and discover things in your environment that you'll need to change to make a migration successful. The following are some questions to ask:

- Which objects do I want to migrate?
- Are my data types compatible with those covered by AWS DMS?
- Does my source system have the necessary capacity and is it configured to support a migration?
- What is my target and how should I configure it to get the required or desired capacity?

Prototype Migration Configuration

This is typically an iterative process. It's a good idea to use a small test migration consisting of a couple of tables to verify you've got things properly configured. Once you've verified your configuration, test the migration with any objects you suspect could be difficult. These can include LOB objects, character set conversions, complex data types, and so on. When you've worked out any kinks related to complexity, test your largest tables to see what sort of throughput you can achieve for them.

Design Your Migration

Concurrently with the prototyping stage, you should determine exactly how you intend to migrate your application. The steps can vary dramatically, depending on the type of migration you're performing.

Testing Your End-to-End Migration

After you have completed your prototyping, it's a good idea to test a complete migration. Are all objects accounted for? Does the migration fit within expected time limits? Are there any errors or warnings in the log files that are a concern?

Perform Your Migration

After you're satisfied that you've got a comprehensive migration plan and have tested your migration end-to-end, it's time to perform your migration!

How Much Load Will the Migration Process Add to My Source Database?

This a complex question with no specific answer. The load on a source database is dependent upon several things.

During a migration, AWS DMS performs a full table scan of the source table for each table processed in parallel. Additionally, each task periodically queries the source for change information. To perform change processing, you may be required to increase the amount of data written to your databases change log. If your tasks contain a Change Data Capture (CDC) component, the size, location, and retention of log files can have an impact on the load.

How Long Does a Typical Database Migration Take?

The following are items that determine the length of your migration: total amount of data being migrated, amount and size of LOB data, size of the largest tables, total number of objects being migrated, secondary indexes created on the target before the migration, resources available on the source system, resources available on the target system, resources available on the replication server, network throughput, and so on.

Clearly, there is no one formula that will predict how long your migration will take. The best way to gauge how long your particular migration will take is to test it.

I'm Changing Engines–How Can I Migrate My Complete Schema?

As previously stated, AWS DMS will only create those objects needed to perform an optimized migration of your data. You can use the free AWS Schema Conversion Tool (AWS SCT) to convert an entire schema from one database engine to another. The AWS SCT can be used with AWS DMS to facilitate the migration of your entire system.

Why Doesn't AWS DMS Migrate My Entire Schema?

All database engines supported by AWS DMS have native tools that you can use to export and import your schema in a homogeneous environment. Amazon has developed the AWS SCT to facilitate the migration of your schema in a heterogeneous environment. The AWS DMS is

intended to be used with one of these methods to perform a complete migration of your database.

Who Can Help Me with My Database Migration Project?

Most of Amazon's customers should be able to complete a database migration project by themselves. However, if your project is challenging, or you are short on resources, one of our migration partners should be able to help you out. For details, please visit https://aws.amazon.com/partners.

What Are the Main Reasons to Switch Database Engines?

There are two main reasons we see people switching engines:

- **Modernization.** The customer wants to use a modern framework or platform for their application portfolio, and these platforms are available only on more modern SQL or NoSQL database engines.
- **License fees.** The customer wants to migrate to an open source engine to reduce license fees.

How Can I Migrate from Unsupported Database Engine Versions?

Amazon has tried to make AWS DMS compatible with as many supported database versions as possible. However, some database versions don't support the necessary features required by AWS DMS, especially with respect to change capture and apply. Currently, to fully migrate from an unsupported database engine, you must first upgrade your database to a supported engine. Alternatively, you may be able to perform a complete migration from an "unsupported" version if you don't need the change capture, and apply capabilities of DMS. If you are performing a homogeneous migration, one of the following methods might work for you:

- MySQL: Importing and Exporting Data From a MySQL DB Instance
- Oracle: Importing Data Into Oracle on Amazon RDS
- SQL Server: Importing and Exporting SQL Server Databases
- PostgreSQL: Importing Data into PostgreSQL on Amazon RDS

When Should I NOT Use DMS?

Most databases offer a native method for migrating between servers or platforms. Sometimes, using a simple backup and restore or export/import is the most efficient way to migrate your data into AWS. If you're considering a homogeneous migration, you should first assess whether a suitable native option exists. In some situations, you might choose to use the native tools to perform the bulk load and use DMS to capture and apply changes that occur during the bulk load. For example when migrating between different flavors of MySQL or Amazon Aurora, creating and promoting a read replica is most likely your best option. See Importing and Exporting Data From a MySQL DB Instance.

When Should I Use a Native Replication Mechanism Instead of the DMS and the AWS Schema Conversion Tool?

This is very much related to the previous question. If you can successfully set up a replica of your primary database in your target environment by using native tools more easily than you can with DMS, you should consider using that native method for migrating your system. Some examples include:

- Read replicas – MySQL
- Standby databases – Oracle, Postgres
- AlwaysOn availability groups – SQL Server

> **Note** AlwaysOn is not supported in RDS.

What Is the Maximum Size of Database That DMS Can Handle?

This depends on your environment, the distribution of data, and how busy your source system is. The best way to determine whether your particular system is a candidate for DMS is to test it out. Start slowly, to get the configuration worked out, add some complex objects, and finally attempt a full load as a test. As a ballpark maximum figure: Under mostly ideal conditions (EC2 to RDS, cross region), over the course of a weekend (approximately 33 hours) we were able to migrate five terabytes of relatively evenly distributed data, including four large (250 GB) tables, a huge (1 TB) table, 1,000 small to moderately sized tables, three tables that contained LOBs varying between 25 GB and 75 GB, and 10,000 very small tables.

What if I Want to Migrate from Classic to VPC?

DMS can be used to help minimize database-related outages when moving a database from outside a VPC into a VPC. The following are the basic strategies for migrating into a VPC:

- Generic EC2 Classic to VPC Migration Guide: Migrating from a Linux Instance in EC2-Classic to a Linux Instance in a VPC
- Specific Procedures for RDS: Moving a DB Instance Not in a VPC into a VPC

Conclusion

This paper outlined best practices for using AWS DMS to migrate data from a source database to a target database, and offers answers to several frequently asked questions about migrations. As companies move database workloads to AWS, they are often also interested in changing their primary database engine. Most current methods for migrating databases to the cloud or switching engines require an extended outage. The AWS DMS helps to migrate database workloads to AWS or change database engines while minimizing any associated downtime.

Contributors

The following individuals and organizations contributed to this document:

- Ed Murray, Senior Database Engineer, Amazon RDS/AWS DMS
- Arun Thiagarajan, Cloud Support Engineer, AWS Premium Support