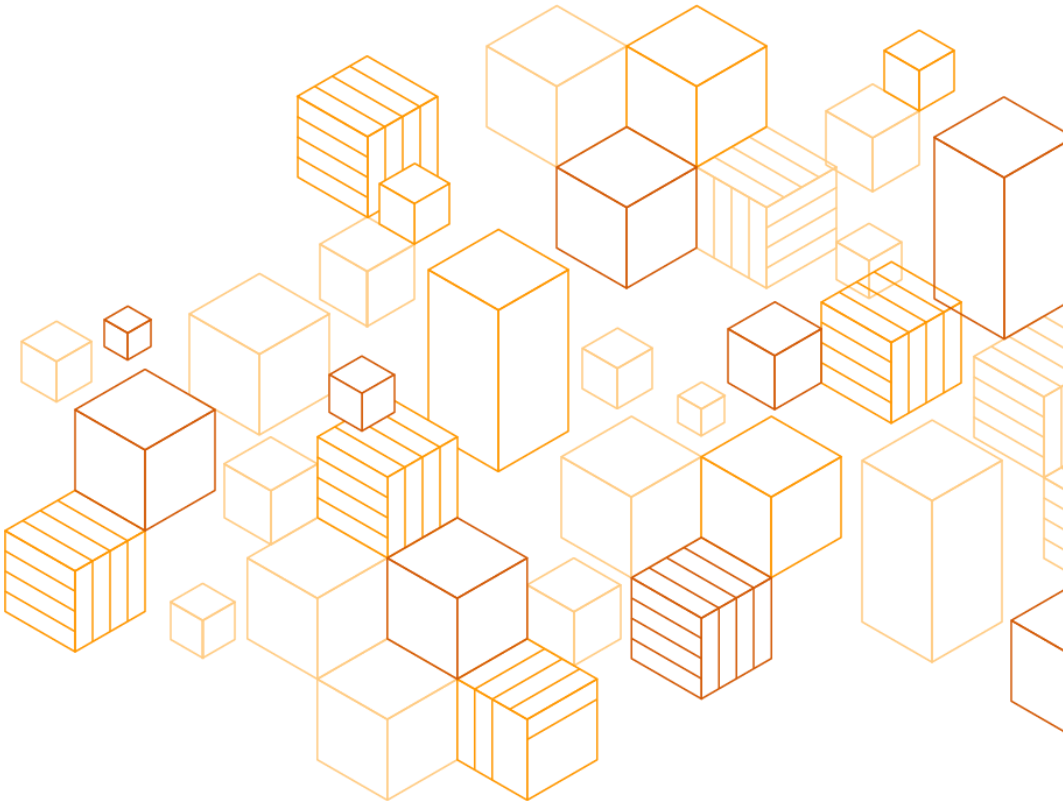
Migrating Applications Running Relational Databases to AWS

Best Practices Guide

First published December 2016

Updated March 9, 2021



Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. 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

About this Guide

The AWS Schema Conversion Tool (AWS SCT) and AWS Data Migration Service (AWS DMS) are essential tools used to migrate an on-premises database to Amazon Relational Database Service (Amazon RDS). This guide introduces you to the benefits and features of these tools and walks you through the steps required to migrate a database to Amazon RDS. Schema, data, and application code migration processes are discussed, regardless of whether your target database is PostgreSQL, MySQL, Amazon Aurora, MariaDB, Oracle, or SQL Server.

Introduction

Customers worldwide increasingly look at the cloud as a way to address their growing needs to store, process, and analyze vast amounts of data. Amazon Web Services (AWS) provides a modern, scalable, secure, and performant platform to address customer requirements. AWS makes it easy to develop applications deployed to the cloud using a combination of database, application, networking, security, compute, and storage services.

One of the most time-consuming tasks involved in moving an application to AWS is migrating the database schema and data to the cloud. The AWS Schema Conversion Tool (AWS SCT) and AWS Database Migration Service (AWS DMS) are invaluable tools to make this migration easier, faster, and less error-prone.

Amazon Relational Database Service (Amazon RDS) is a managed service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks. The simplicity and ease of management of Amazon RDS appeals to many customers who want to take advantage of the disaster recovery, high availability, redundancy, scalability, and time-saving benefits the cloud offers. Amazon RDS currently supports the MySQL, Amazon Aurora, MariaDB, PostgreSQL, Oracle, and Microsoft SQL Server database engines.

In this guide, we discuss how to migrate applications using a relational database management system (RDBMS), such as Oracle or Microsoft SQL Server, onto an Amazon RDS instance in the AWS Cloud using the AWS SCT and AWS DMS. This guide covers all major steps of application migration: database schema and data migration, SQL code conversion, and application code re-platforming.

Overview of Migrating Data-Centric Applications to AWS

Migration is the process of moving applications that were originally developed to run on-premises and need to be remediated for Amazon RDS.

During the migration process, a database application may be migrated between two databases of the same engine type (a homogeneous migration; for example, Oracle → Oracle, SQL Server → SQL Server, etc.) or between two databases that use different

engine types (a heterogeneous migration; for example, Oracle → PostgreSQL, SQL Server → MySQL, etc.). In this guide, we look at common migration scenarios regardless of the database engine, and touch on specific issues related to certain examples of heterogeneous conversions.

Migration Steps and Tools

Application migration to AWS involves the following steps, regardless of the database engine:

1. Migration assessment analysis
2. Schema conversion to a target database platform
3. SQL statement and application code conversion
4. Data migration
5. Testing of converted database and application code
6. Setting up replication and failover scenarios for data migration to the target platform
7. Setting up monitoring for a new production environment and go live with the target environment

Figure 1: Steps of application migration to AWS

Each application is different and may require extra attention to one or more of these steps. For example, a typical application contains the majority of complex data logic in database-stored procedures, functions, and so on. Other applications are heavier on logic in the application, such as ad hoc queries to support search functionality. On average, the percentage of time spent in each phase of the migration effort for a typical application breaks down as shown in Table 1.

Table 1: Time spent in each migration phase

Step	Percentage of Overall Effort
Migration Assessment	2%
Schema Conversion	30%
Embedded SQL and Application Code Conversion	15%
Data Migration	5%
Testing	45%
Data Replication	3%
Go Live	5%
Note: Percentages for data migration and replication are based on man-hours for configuration, and do not include hours needed for the initial load.	

To make the migration process faster, more predictable, and cost effective, AWS provides the following tools and methods to automate migration steps:

- AWS Schema Conversion Tool (AWS SCT) – a desktop tool that automates conversion of database objects from different database migration systems (Oracle, SQL Server, MySQL, PostgreSQL) to different RDS database targets (Amazon Aurora, PostgreSQL, Oracle, MySQL, SQL Server). This tool is invaluable during the Migration Assessment, Schema Conversion, and Application Code Conversion steps.
- AWS Database Migration Service (AWS DMS) – a service for data migration to and from AWS database targets. AWS DMS can be used for a variety of replication tasks, including continuous replication to offload reads from a primary production server for reporting or extract, transform, load (ETL); continuous replication for high availability; database consolidation; and temporary replication for data migrations. In this guide, we focus on the replication needed for data migrations. This service reduces time and effort during the Data Migration and Data Replication Setup steps.

Development Environment Setup Prerequisites

To prepare for the migration, you must set up a development environment to use for the iterative migration process. In most cases, it is desirable to have the development

environment mirror the production environment. Therefore, this environment is likely on-premises or running on an Amazon Elastic Compute Cloud (Amazon EC2) instance. Download and install the AWS SCT on a server in the development environment.

> If you are interested in changing database platforms, the New Project Wizard can help you determine the most appropriate target platform for the source database. See Step 1: Migration Assessment for more information.

Procure an Amazon RDS database instance to serve as the migration target and any necessary EC2 instances to run migration-specific utilities.

Step 1: Migration Assessment

During **Migration Assessment**, a team of system architects reviews the architecture of the existing application, produces an assessment report that includes a network diagram with all the application layers, identifies the application and database components that are not automatically migrated, and estimates the effort for manual conversion work. Although migration analysis tools exist to expedite the evaluation, the bulk of the assessment is conducted by internal staff or with help from AWS Professional Services. This effort is usually 2% of the whole migration effort.

One of the key tools in the assessment analysis is the Database Migration Assessment Report. This report provides important information about the conversion of the schema from your source database to your target RDS database instance. More specifically, the Assessment Report does the following:

- Identifies schema objects (e.g., tables, views, stored procedures, triggers, etc.) in the source database and the actions that are required to convert them (**Action Items**) to the target database (including fully automated conversion, small changes like selection of data types or attributes of tables, and rewrites of significant portions of the stored procedure)
- Recommends the best target engine, based on the source database and the features used
- Recommends other AWS services that can substitute for missing features
- Recommends unique features available in Amazon RDS that can save the customer licensing and other costs

- Recommends re-architecting for the cloud, for example, sharding a large database into multiple Amazon RDS instances such as sharding by customer or tenant, sharding by geography, or sharding by partition key

Report Sections

The database migration assessment report includes three main sections—executive summary, conversion statistics graph, conversion action items.

Executive Summary

The executive summary provides key migration metrics and helps you choose the best target database engine for your particular application.

Conversion Statistics Graph

The conversion statistics graph visualizes the schema objects and number of conversion issues (and their complexity) required in the migration project.

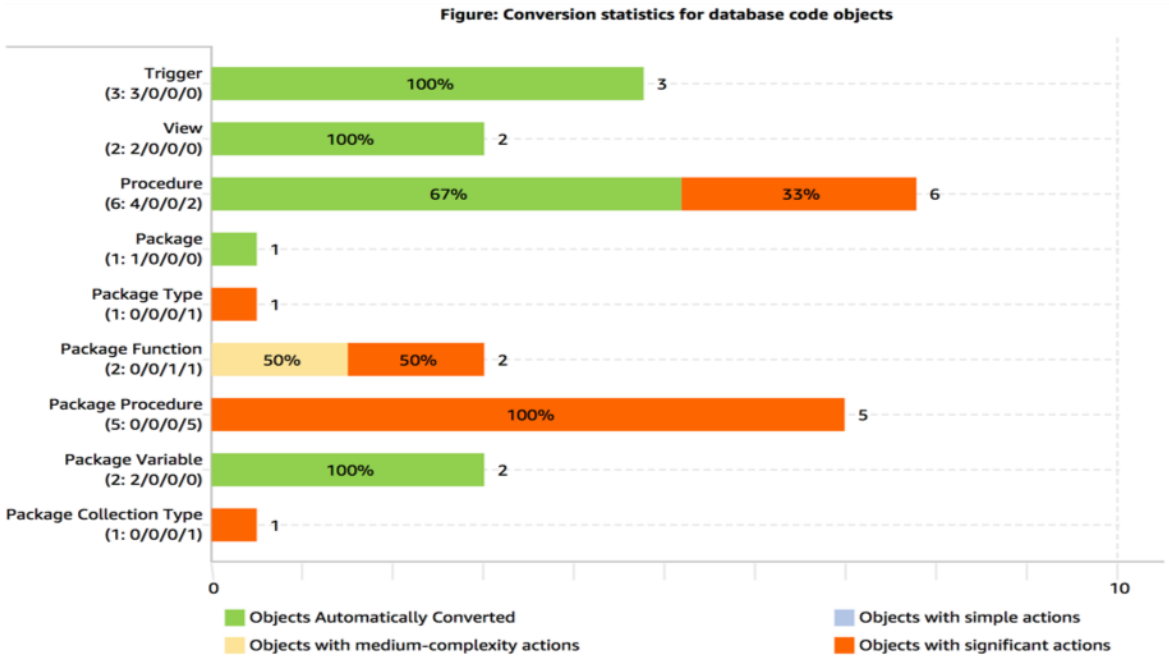


Figure 2: Graph of conversion statistics

Conversion Action Items

Conversion action items are presented in a detailed list with recommendations and their references in the database code.

The database migration assessment report shows conversion action items with three levels of complexity:

- **Simple** task that requires less than 1 hour to complete
- **Medium** task that requires 1 to 4 hours to complete
- **Significant** task that requires 4 or more hours to complete

Using the detailed report provided by the AWS SCT, skilled architects can provide a much more precise estimate for the efforts required to complete migration of the database schema code. For more information about how to configure and run the database migration assessment report, see Creating a Database Migration Assessment Report. All results of the assessment report calculations and the summary of conversion action items are saved inside the AWS SCT. This data is useful for the schema conversion step of the overall data migration.

Tips

- Before running the assessment report, you can restrict the database objects to evaluate by selecting or clearing the desired nodes in the source database tree.
- After running the initial assessment report, save the file as a PDF. Then, open the file in a PDF viewer to view the entire database migration assessment report. You can navigate the assessment report more easily if you convert it to a Microsoft Word document and use Word's Table of Contents Navigation pane.

Step 2: Schema Conversion

The **Schema Conversion** step consists of translating the data definition language (DDL) for tables, partitions, and other database storage objects from the syntax and features of the source database to the syntax and features of the target database.

Schema conversion in the AWS SCT is a two-step process:

1. Convert the schema.
2. Apply the schema to the target database.

AWS SCT also converts procedural application code in triggers, stored procedures, and functions from feature-rich languages (e.g., PLSQL, T-SQL) to the simpler procedural languages of MySQL and PostgreSQL. Schema conversion typically accounts for 30% of the whole migration effort.

The AWS SCT automatically creates DDL scripts for as many database objects on the target platform as possible. For the remaining database objects, the conversion action items describe why the object cannot be converted automatically and the manual steps required to convert the object to the target platform. References to articles that discuss the recommended solution on the target platform are included when available.

The translated DDL for database objects is also stored in the AWS SCT project file—both the DDL that is generated automatically by the AWS SCT and any custom or manual DDL for objects that could not convert automatically. The AWS SCT can also generate a DDL script file per object; this may come in handy for source code version control purposes. You have complete control over when the DDL is applied to the target database. For example, for a smaller database you can run the **Convert Schema** command to automatically generate DDL for as many objects as possible, then write code to handle manual conversion action items, and lastly apply all of the DDL to create all database objects at once. For a larger database that takes weeks or months to convert, it can be advantageous to generate the target database objects by executing the DDL selectively to create objects in the target database as needed.

The Step 6: Data Replication section discusses how you can also speed up the data migration process by applying secondary indexes and constraints as a separate step, after the initial data load. By selecting or clearing objects from the target database tree, you can save DDL scripts separately for tables and their corresponding foreign keys and secondary indexes. You can then use these scripts to generate tables, migrate data to those tables without performance slowdown, and then apply secondary indexes and foreign keys after the data is loaded.

After the database migration assessment report is created, the AWS SCT offers two views of the project: main view and assessment report view.

Tips for Navigating the AWS SCT in the Assessment Report View

See *Figure 3* and corresponding callouts in *Table 2* for tips on navigating the assessment report view.

Figure 3: AWS SCT in the assessment report view

Table 2: AWS SCT in assessment report view callouts

Callout	Description
1	Select a code object from the source database tree on the left to view the source code, DDL, and mappings to create the object in the target database. **Note:** Source code for tables is not displayed in the AWS SCT; however, the DDL to create tables in the target database is displayed. The AWS SCT displays both source and target DDL for other database objects.
2	Click the chevron (▶) next to an issue or double-click the issue message to expand the list of affected objects. Select the affected object to locate it in the source and target database trees, and view or edit the DDL script. Source database objects with an associated conversion action item are indicated with an exclamation icon: ▶ P_UTL_SMTP
3	When viewing the source SQL for objects, the AWS SCT highlights the lines of code that require manual intervention to convert to the target platform. Hovering over or double-clicking the highlighted source code displays the corresponding action item.
4	The target SQL includes comments with the Issue # for action items to be resolved in the converted SQL code.

Schema Mapping Rules

The AWS SCT allows you to create custom schema transformations and mapping rules to use during the conversion. Schema mapping rules can standardize the target schema naming convention, apply internal naming conventions, correct existing issues in the source schema, and so on. Transformations are applied to the target database, schema, table, or column DDL and currently include the following:

- Rename
- Add prefix
- Add suffix
- Remove prefix
- Remove suffix
- Replace prefix
- Replace suffix
- Convert uppercase (not available for columns)
- Convert lowercase (not available for columns)
- Move to (tables only)
- Change data type (columns only)

New transformations and mapping rules are being added to the AWS SCT with each release to increase the robustness of this valuable feature.

For example, *Figure 4* depicts a schema mapping rule that has been applied to standardize a table name and correct a typo. Notice the Source Name to Target Name mapping.

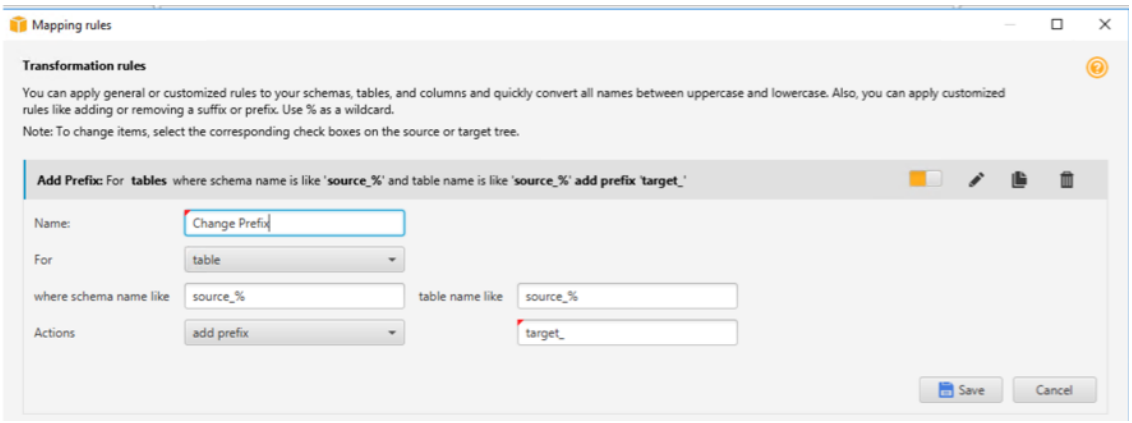


Figure 4: Schema mapping rule in AWS SCT

You can create as many schema mapping rules as you need by choosing **Settings**, and then **Mapping Rules** from the AWS SCT menu.

After schema mapping rules are created, you can export them for use by AWS DMS during the Data Migration step. Schema mapping rules are exported in JavaScript Object Notation (JSON) format. The Step 4: Data Migration section examines how AWS DMS uses this mapping.

Tips

- Before applying individual SQL objects to the target, carefully examine the SQL for the object to ensure that any dependent objects have already been created.

  If an error occurs while applying an object to the target database, check the error log for details. To find the location of the error log, from the AWS SCT menu, choose **Settings**, and then choose **Global Settings**.

Step 3: Conversion of Embedded SQL and Application Code

After you convert the database schema, the next step is to address any custom scripts with embedded SQL statements (e.g., ETL scripts, reports, etc.) and the application code so that they work with the new target database. This includes rewriting portions of application code written in Java, C#, C++, Perl, Python, etc., that relate to JDBC/ODBC driver usage, establishing connections, data retrieval, and iteration. AWS SCT scans a folder containing application code, extracts embedded SQL statements, converts as many as possible automatically, and flags the remaining statements for manual

conversion actions. Converting embedded SQL in application code typically accounts for 15% of the whole migration effort.

Some applications are more reliant on database objects, such as stored procedures, while other applications use more embedded SQL for database queries. In either case, these two efforts combined typically account for around 45%, or almost half, of the migration effort.

The workflow for application code conversion is similar to the workflow for the database migration:

1. Run an assessment report to understand the level of effort required to convert the application code to the target platform.
2. Analyze the code to extract embedded SQL statements.
3. Allow the AWS SCT to automatically convert as much code as possible.
4. Work through the remaining conversion Action Items manually.
5. Save code changes.

The AWS SCT uses a two-step process to convert application code:

1. Extract SQL statements from the surrounding application code.
2. Convert SQL statements.

An application conversion project is a subproject of a database migration project. One Database Migration Project can include one or more application conversion subprojects; for example, there may be a front end GUI application conversion, an ETL application conversion, and a reporting application conversion. All three applications can be attached to the parent database migration project and converted in the AWS SCT.

The AWS SCT can also standardize parameters in parameterized SQL statements to use named or positional styles, or keep parameters as they are. In the following example, the original application source code used the named (:name) style, and positional (?) style has been selected for the application conversion. Notice that AWS SCT replaced the named parameter `:id` with a positional `?` during conversion.

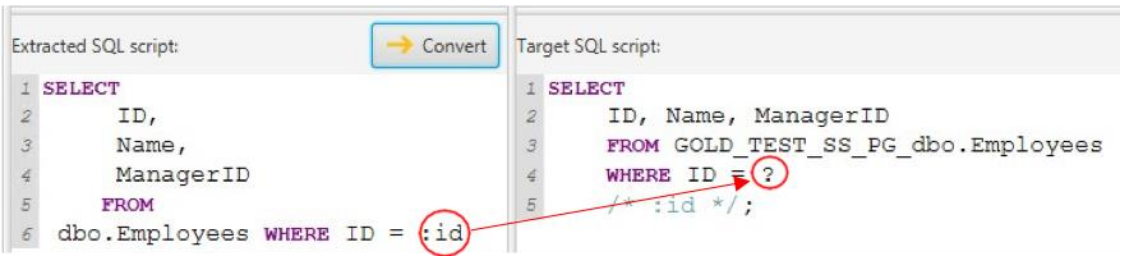


Figure 5: AWS SCT replaced named style with positional style

The application conversion workspace makes it easy to view and modify embedded SQL code and track changes that are yet to be made. Parsed SQL scripts and snippets appear in the bottom pane alongside their converted code. Selecting one of these parsed scripts highlights it in the application code so you can view the context, and the parsed script appears in the lower left pane, as shown in *Figure 6*.

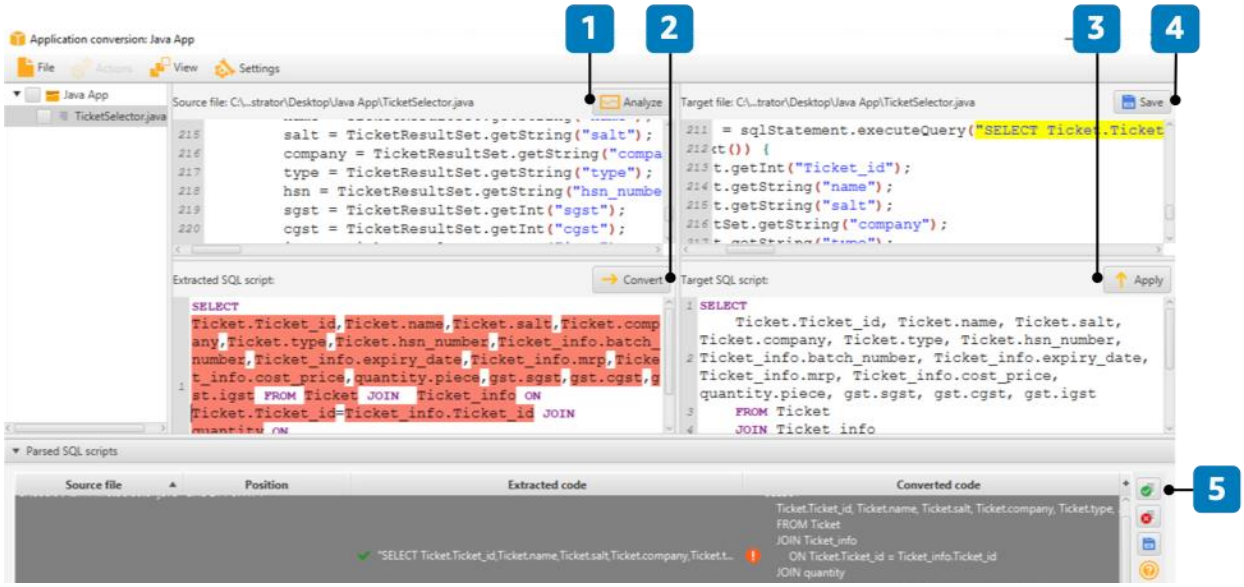


Figure 6: Selecting a parsed script highlights it in the application code

The embedded SQL conversion process consists of the following iterative steps:

1. Analyze the selected code folder to extract embedded SQL.
2. Convert the SQL to the target script. If the AWS SCT is able to convert the script automatically, it appears in the lower right pane. Any manual conversion code can also be entered here.
3. Apply the converted SQL to the source code base, swapping out the original snippet for the newly converted snippet.

4. Save the changes to the source code. A backup of the original source code is saved to your AWS SCT working directory with an extension of `.old`.
5. Click the green checkmark to the right of the Parsed SQL Script to validate the Target SQL script against the target database.

Tips

- AWS SCT can only convert or make recommendations for the SQL statements that it was able to extract. The application assessment report contains a SQL **Extraction Actions** tab. This tab lists conversion action items where AWS SCT detected SQL statements but was not able to accurately extract and parse them. Drill down through these issues to identify application code that must be manually evaluated by an application developer and converted manually, if needed.
- Drill into the issues on either the **SQL Extraction Actions** or the **SQL Conversion Actions** tab to locate the file and line number of the conversion item, then double-click the occurrence to view the extracted SQL.

Step 4: Data Migration

After the schema and application code are successfully converted to the target database platform, it is time to migrate data from the source database to the target database. You can easily accomplish this by using AWS DMS. After the data is migrated, you can perform testing on the new schema and application. Because much of the data mapping and transformation work has already been done in AWS SCT and AWS DMS manages the complexities of the data migration for you, configuring a new Data Migration Service is typically 5% of the whole migration effort.

> **Note:** AWS SCT and AWS DMS can be used independently. For example, AWS DMS can be used to synchronize homogeneous databases between environments, such as refreshing a test environment with production data. However, the tools are integrated so that the schema conversion and data migration steps can be used in any order. Later sections of this guide cover specific scenarios of integrating these tools.

AWS DMS works by setting up a replication server that acts as a middleman between the source and target databases. This instance is referred to as the AWS DMS replication instance (*Figure 7*). AWS DMS migrates data between source and target

instances and tracks which rows have been migrated and which rows have yet to be migrated.

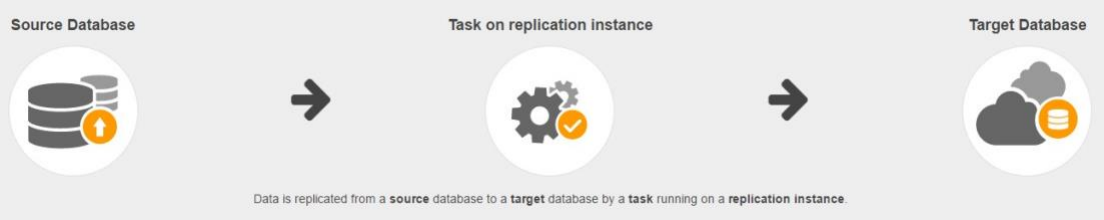


Figure 7: AWS DMS replication instance

AWS DMS provides a wizard to walk through the three main steps of getting the data migration service up and running:

1. Set up a replication instance.
2. Define connections for the source and target databases.
3. Define data replication tasks.

To perform a database migration, AWS DMS must be able to connect to the source and target databases and the replication instance. AWS DMS will automatically create the replication instance in the specified Amazon Virtual Private Cloud (Amazon VPC). The simplest database migration configuration is when the source and target databases are also AWS resources (Amazon EC2 or Amazon RDS) in the same VPC. For more information, see Setting Up a Network for Database Migration in the *AWS Database Migration Service User Guide*.

You can migrate data in two ways:

- As a full load of existing data
- As a full load of existing data, followed by continuous replication of data changes to the target

AWS DMS can be configured to drop and recreate the target tables or truncate existing data in the target tables before reloading data. AWS DMS will automatically create the target table on the target database according to the defined schema mapping rules with primary keys and required unique indexes, then migrate the data. However, AWS DMS 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 or other database objects such as stored procedures, views, functions, packages, and so on.

This is where the AWS SCT feature of saving SQL scripts separately for various SQL objects can be used, or these objects can be applied to the target database directly via the AWS SCT **Apply to Database** command after the initial load. Data can be migrated as-is (such as when the target schema is identical or compatible with the source schema), AWS DMS can use Schema Mapping Rules exported from the AWS SCT project, or custom mapping rules can be defined in AWS DMS via JSON. For example, the following JSON renames a table from `tbl_departmnet` to `department` and creates a mapping between these two tables.

```
{
"rules": [
{
"rule-type": "selection", "rule-id": "1",
"rule-name": "1", "object-locator": {
"schema-name": "HumanResources", "table-name": "%"
},
"rule-action": "include"
},
{
"rule-type": "transformation", "rule-id": "2",
"rule-name": "Rename tbl_Departmnet", "rule-action": "rename",
"rule-target": "table", "object-locator": {
"schema-name": "HumanResources", "table-name": "tbl_Departmnet"
},
"value": "Department"
}
]
```

Tips

For more information on AWS replication instance types and their capacities, see Working with an AWS DMS Replication Instance.

Step 5: Testing Converted Code

After schema and application code has been converted and the data successfully migrated onto the AWS platform, thoroughly test the migrated application. The focus of this testing is to ensure correct functional behavior on the new platform. Although best practices vary, it is generally accepted to aim for as much time in the testing phase as in the development phase, which is about 45% of the overall migration effort.

The goal of testing should be two-fold: exercising critical functionality in the application and verifying that converted SQL objects are functioning as intended. An ideal scenario

is to load the same test dataset into the original source database, load the converted version of the same dataset into the target database, and perform the same set of automated system tests in parallel on each system. The outcome of the tests on the converted database should be functionally equivalent to the source. Data rows affected by the tests should also be examined independently for equivalency. Analyzing the data independently from application functionality verifies there are no data issues lurking in the target database that are not obvious in the user interface (UI).

Step 6: Data Replication

Although a one-time full load of existing data is relatively simple to set up and run, many production applications with large database backends cannot tolerate a downtime window long enough to migrate all the data in a full load. For these databases, AWS DMS can use a proprietary Change Data Capture (CDC) process to implement ongoing replication from the source database to the target database. AWS DMS manages and monitors the ongoing replication process with minimal load on the source database, without platform-specific technologies and without components that need to be installed on either the source or target. Due to CDC's ease-of-use, setting up data replication typically accounts for 3% of the overall effort.

CDC offers two ways to implement ongoing replication:

- Migrate existing data and replicate ongoing changes – implements ongoing replication by:
    a. (Optional) Creating the target schema.
    b. Migrating existing data and caching changes to existing data as it is migrated.
    c. Applying those cached data changes until the database reaches a steady state.
    d. Lastly, applying current data changes to the target as soon as they are received by the replication instance.
- Replicate data changes only – replicate data changes only (no schema) from a specified point in time. This option is helpful when the target schema already exists and the initial data load is already completed. For example, using native export/import tools, ETL, or snapshots might be a more efficient method of loading the bulk data in some situations. In this case, AWS DMS can be used to replicate changes from when the bulk load process started to bring and keep the source and target databases in sync.

AWS DMS takes advantage of built-in functionality of the source database platform to implement the proprietary CDC process on the replication instance. This allows AWS DMS to manage, process, and monitor data replication with minimal impact to either the source or target databases. The following sections describe the source platform features and configurations needed by the DMS replication instance's CDC process.

MS SQL Server Sources

Replication. Replication must be enabled on the source server and a distribution database that acts as its own distributor configured.

Transaction logs. The source database must be in Full or Bulk Recovery Mode to enable transaction log backups.

Oracle Sources

BinaryReader or LogMiner. By default, AWS DMS uses LogMiner to capture changes from the source instance. For data migrations with a high volume of change and/or large object (LOB) data, using the proprietary Binary Reader may offer some performance advantages.

ARCHIVELOG. The source database must be in ARCHIVELOG mode.

Supplemental Logging. Supplemental logging must be turned on in the source database and in all tables that are being migrated.

PostgreSQL Sources

Write-Ahead Logging (WAL). In order for AWS DMS to capture changes from a PostgreSQL database:

- The `wal_level` must be set to logical.
- `max_replication_slots` must be >= 1.
- `max_wal_senders` must be >= 1.

Primary Key. Tables to be included in CDC must have a primary key.

MySQL Sources

Binary Logging. Binary logging must be enabled on the source database.

Automatic backups. Automatic backups must be enabled if the source is a MySQL, Amazon Aurora, or MariaDB Amazon RDS instance.

SAP ASE (Sybase) Sources

Replication. Replication must be enabled on the source, but RepAgent must be disabled.

MongoDB

Oplog. AWS DMS requires access to MongoDB oplog to enable ongoing replication.

IBM Db2 LUW

Either one or both of the database configuration parameters LOGARCHMETH1 and LOGARCHMETH2 should be set to ON

For additional information, including prerequisites and security configurations for each source platform, refer to the appropriate link in the Sources for Data Migration for AWS Database Migration Service section of the *AWS Database Migration Service User Guide*.

The basic setup of ongoing data replication is done in the **Task configuration** pane. *Table 3* describes the migration type options.

Table 3: Migration type options

Migration type	Description
Migrate existing data	Perform a one-time migration from the source endpoint to the target endpoint.
Migrate existing data and replicate ongoing changes	Perform a one-time migration from the source to the target, and then continue replicating data changes from the source to the target.
Replicate data changes only	Don't perform a one-time migration, but continue to replicate data changes from the source to the target.

Additional configurations for the data migration task are available in the **Task settings** pane (*Figure 8* and *Table 4*).

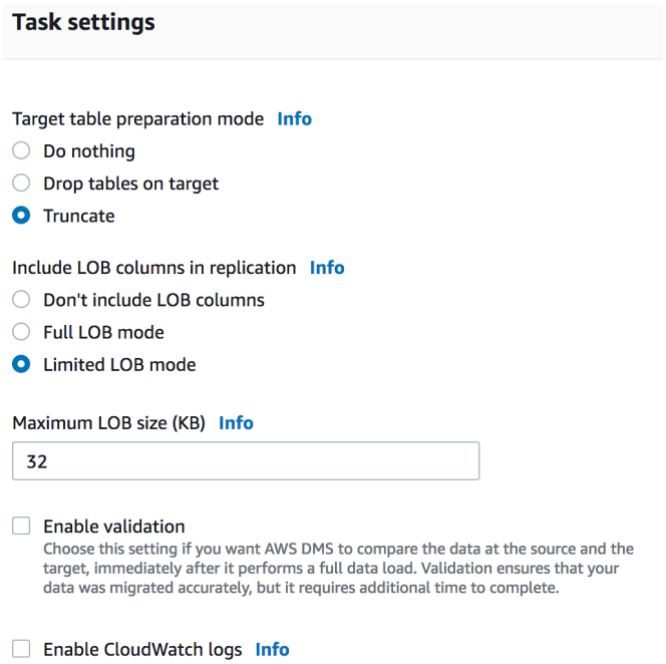


Figure 8: Data migration task settings

Table 4: Task setting options

Setting	Description
Target table preparation mode	
Do nothing	If the tables already exist at the target, they remain unaffected. Otherwise, AWS DMS creates new tables.
Drop tables on target	AWS DMS drops the tables, and creates new tables in their place.
Truncate	AWS DMS leaves the tables and their metadata in place, but removes the data from them.
Include LOB columns in replication	
Don't include LOB columns	AWS DMS ignores columns or fields that contain large objects (LOBs).
Full LOB mode	AWS DMS includes the complete LOB.
Limited LOB mode	AWS DMS truncates each LOB to the size defined by Max LOB size. (Limited LOB mode is faster than full LOB mode.)
Enable CloudWatch logs (check box)	AWS DMS publishes detailed task information to CloudWatch Logs.

Step 7: Deployment to AWS and Go Live

Test the data migration of the production database to ensure that all data can be successfully migrated during the allocated cutover window. Monitor the source and target databases to ensure that the initial data load is completed, cached transactions are applied, and data has reached a steady state before cutover. You can also use the **Enable Validation** option available in the **Task settings** pane of AWS DMS (*Figure 8*). If you select this option, AWS DMS validates the data migration by comparing the data in the source and the target databases

Design a simple rollback plan for the unlikely event that an unrecoverable error occurs during the Go Live window. The AWS SCT and AWS DMS work together to preserve the original source database and application, so the rollback plan will mainly consist of scripts to point connection strings back to the original source database.

Post-Deployment Monitoring

AWS DMS monitors the number of rows inserted, deleted, and updated, as well as the number of DDL statements issued per table while a task is running. You can view these statistics for the selected task on the **Table Statistics** pane of your migration task.

In the list of migration tasks in AWS DMS, choose your **Database migration task** (*Figure 9*).

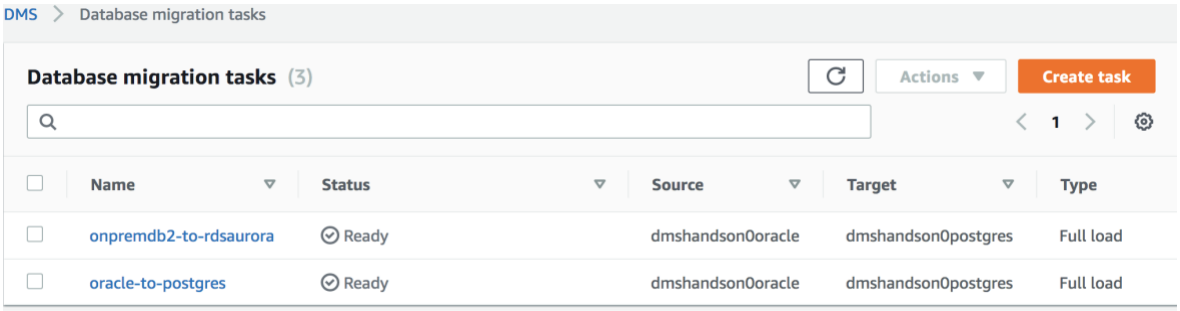


Figure 9: List of database migration tasks

On the detail page, scroll to the **Table Statistics** pane (*Figure 10*). You can monitor the number of rows inserted, deleted, and updated, as well as the number of DDL statements issued per table while a task is running.

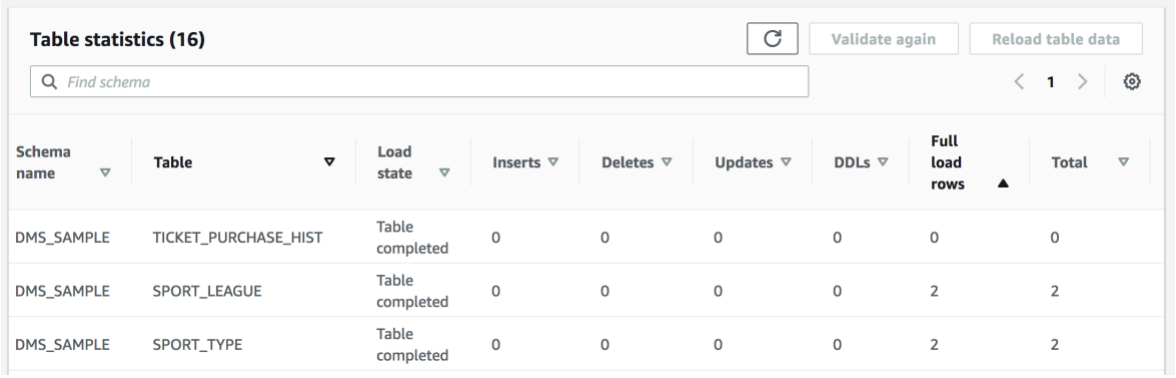


Figure 10: Table statistics monitoring

The most relevant metrics can be viewed for the selected task on the **Migration task metrics** pane (*Figure 11*).

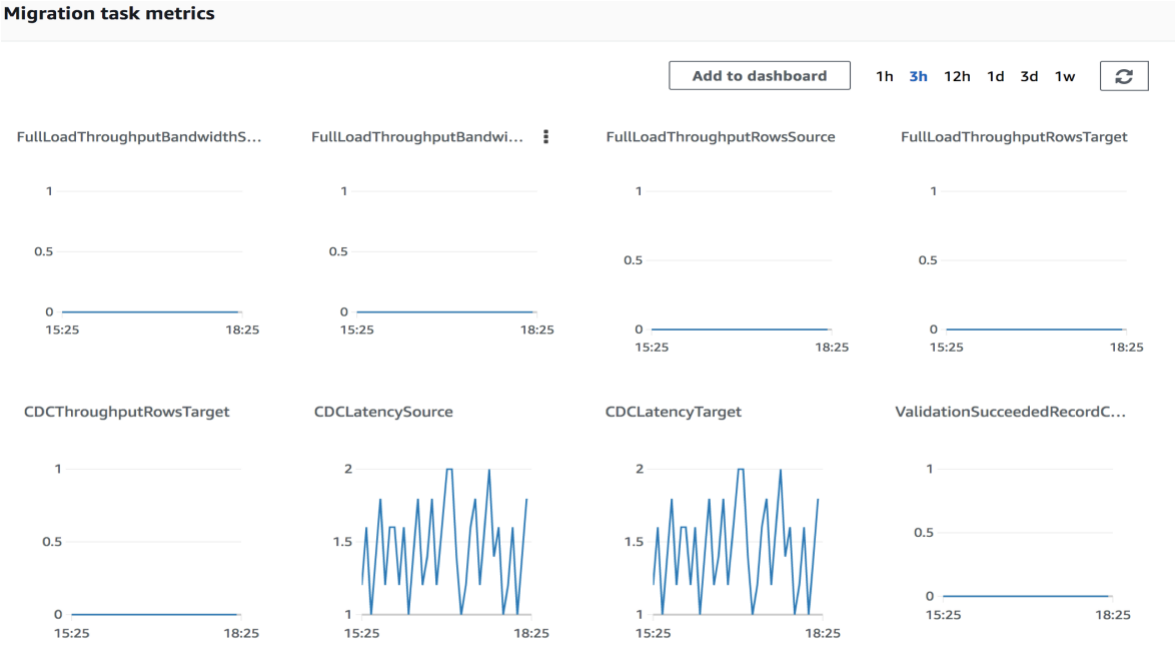


Figure 11: Relevant metrics for a task

Additional metrics are available from the Amazon CloudWatch Logs dashboard, accessible from the link on the **Overview details** pane, or by navigating in the AWS Management Console to **Services**, choosing **CloudWatch**, and then choosing **DMS**.

If logging is enabled for the task, review the Amazon CloudWatch Logs for any errors or warnings. You can enable logging for a task during task creation by selecting **Enable CloudWatch Logs** in **Task Settings** (*Figure 8*).

Best Practices

This section presents best practices for each of the seven major steps of migrating applications to AWS.

Schema Conversion Best Practices

- **Save the Database Migration Assessment Report.** After running the initial database migration assessment report, save it as a CSV and a PDF. As conversion action items are completed, they may no longer appear in the database migration assessment report if it is regenerated. Saving the initial assessment report can serve as a valuable project management tool, such as providing a history of conversion tasks and tracking the percentage of tasks completed. The CSV version is helpful because it can be imported into Excel for ease-of-use, such as the ability to search, filter, and sort conversion tasks.
- For most conversions, apply DDL to the target database in the following order to avoid dependency errors:

  a. Sequences

  b. Tables

  c. Views

  d. Procedures

  Functions should be applied to the target database in order of dependency. For example, a function might be referenced in a table column; therefore, the function must be applied before the table to avoid a dependency error. Another function might reference a table; therefore, the table must be created first.
- **Configure the AWS SCT with the memory performance settings you need.** Increasing memory speeds up the performance of your conversion but uses more memory resources on your desktop. On a desktop with limited memory, you can configure AWS SCT to use less memory, resulting in a slower conversion. You can change these settings by choosing **Settings**, **Global Settings**, and then **Performance and Memory**.

- **Apply the additional schema that AWS SCT creates to the target database.** For most conversion projects, AWS SCT creates an additional schema in the target database named `aw_[source platform]_ext`. This schema contains SQL objects to emulate features and functionality that are present in the source platform but not in the target platform. For example, when converting from Microsoft SQL Server to PostgreSQL, the `aws_sqlserver_ext` schema contains sequence definitions to replace SQL Server identity columns. Don't forget to apply this additional schema to the target database, as it will not have a direct mapping to a source object.
- **Use source code version control to track changes to target objects (both database and application code).** If you find bugs or data differences during testing or deployment, the history of changes is useful for debugging.

Application Code Conversion Best Practices

- **After running the initial application assessment report, save it as a CSV and a PDF.** As conversion tasks are completed, they no longer appear in the application assessment report if it is regenerated. The initial application assessment report serves as a history of tasks completed throughout the entire application conversion effort. The CSV file is also helpful because it can be imported into Excel for ease-of-use, such as the ability to search, filter, and sort conversion tasks.

Data Migration Best Practices

- **Choose a replication instance class large enough to support your database size and transactional load.** By default, AWS DMS loads eight tables at a time. On a large replication server, such as a dms.c4.xlarge or larger instance, you can improve performance by increasing the number of tables to load in parallel. On a smaller replication server, reduce the number of tables to load in parallel for improved performance.
- **On the target database, disable what isn't needed.** Disable unnecessary triggers, validation, foreign keys, and secondary indexes on the target databases, if possible. Disable unnecessary jobs, backups, and logging on the target databases.

- **Tables in the source database that do not participate in common transactions can be allocated to different tasks.** This allows multiple tasks to synchronize data for a single database migration, thereby improving performance in some instances.
- **Monitor performance of the source system to ensure it is able to handle the load of the database migration tasks.** Reducing the number of tasks and/or tables per task can reduce the load on the source system. Using a synchronized replica, mirror, or other read-only copy of the source database can also help reduce the load on the source system.
- **Enable logging using Amazon CloudWatch Logs.** Troubleshooting AWS DMS errors without the full logging captured in CloudWatch Logs can be difficult and time-consuming (if not impossible).
- If your source data contains Binary Large Objects (BLOBs) such as an image, XML, or other binary data, loading of these objects can be optimized using **Task Settings**. For more information, see Task Settings for AWS Database Migration Service Tasks in the *AWS Database Migration Service User Guide*.

Data Replication Best Practices

- **Achieve best performance by not applying indexes or foreign keys to the target database during the initial load.** The initial load of existing data comprises inserts into the target database. Therefore, you can get the best performance during the initial load if the target database does not have indexes or foreign keys applied. However, after the initial load, when cached data changes are applied, indexes can be useful for locating rows to update or delete.
- **Apply indexes and foreign keys to the target database before the application is ready to go live.**
- **For ongoing replication (such as for high availability), enable the Multi-AZ option on the replication instance.** The Multi-AZ option provides high availability and failover support for the replication instance.
- **Use the AWS API or AWS Command Line Interface (AWS CLI) for more advanced AWS DMS task settings**. The AWS API and/or AWS CLI offer more granular control over data replication tasks and additional settings not currently available in the AWS Management Console.

- **Disable backups on the target database during the full load for better performance.** Enable them during cutover.
- **Wait until cutover to make your target RDS instance Multi-AZ** for better performance.

Testing Best Practices

- **Have a test environment where full regression tests of the original application can be conducted.** The tests completed before conversion should work the same way for the converted database.
- **In the absence of automated testing, run "smoke" tests on the old and new applications**, comparing data values and UI functionality to ensure like behavior.
- **Apply standard practices for database-driven software testing regardless of the migration process.** The converted application must be fully retested.
- Have sample test data that is used only for testing.
- **Know your data logic and apply it to your test plans.** If you don't have correct test data, the tests might fail or not cover mission-critical application functionality.
- **Test using a dataset similar in size to the production dataset to expose performance bottlenecks**, such as missing or non-performant indexes.

Deployment and Go Live Best Practices

- **Have a rollback plan in place** should anything go wrong during the live migration. Since the original database and application code are still in place and not touched by AWS SCT or AWS DMS, this should be fairly straightforward.
- **Test the deployment on a staging or pre-production environment** to ensure that all needed objects, libraries, code, etc., are included in the deployment and created in the correct order of dependency (e.g., a sequence is created before the table that uses it).
- **Verify that AWS DMS has reached a steady state** and all existing data has been replicated to the new server before cutting off access to the old application in preparation for the cutover.
- **Verify that database maintenance jobs are in place**, such as backups and index maintenance.

- **Turn on Multi-AZ, if required.**
- **Verify that monitoring is in place.**
- **AWS provides several services to make deployments easier and trouble-free**, such as AWS CloudFormation, AWS OpsWorks, and AWS CodeDeploy. These services are especially helpful for deploying and managing stacks involving multiple AWS resources that must interact with each other, such as databases, web servers, load balancers, IP addresses, VPCs, and so on. These services enable you to create reusable templates to ensure that environments are identical. For example, when setting up the first development environment, you may complete some tasks manually, either via the AWS Management Console, AWS CLI, PowerShell, etc. Instead of tracking these items manually to ensure they are created in the staging environment, resources in the running development environment can be included in the template, then the template can be used for setting up the staging and production environments.

Post-Deployment Monitoring Best Practices

- **Create CloudWatch Logs alarms and notifications to monitor for unusual database activity, and send alerts to notify production staff if the AWS instance is not performing well.** High CPU utilization, disk latency, and high RAM usage can be indicators of missing indexes or other performance bottlenecks.
- **Monitor logs and exception reports** for unusual activity and errors.
- **Determine if there are additional platform-specific metrics to capture and monitor,** such as capturing locks from the `pg_locks` catalog table on the Amazon Redshift platform. Amazon Redshift also allows viewing running queries from the AWS Management Console.
- **Monitor instance health.** CloudWatch Logs provides more metrics on an RDS instance than an EC2 instance, and these may be sufficient for monitoring instance health. For an EC2 instance, consider installing a third-party monitoring tool to provide additional metrics.

Conclusion

The AWS Schema Conversion Tool (AWS SCT) and AWS Data Migration Service (AWS DMS) make the process of moving applications to the cloud much easier and

faster than manual conversion alone. Together, they save many hours of development during the migration effort, enabling you to reap the benefits of AWS more quickly.

Document Revisions

Date	Description
March 9, 2021	Reviewed for technical accuracy
November 2019	Updated to reflect latest features and functionality.
December 2016	First publication.