MIGRATION PLAYBOOK · DATABASES

How to migrate SQL Server and MySQL to Amazon RDS and Aurora

Pick the right method for each database (native backup and restore, physical backup to Aurora, or AWS DMS with change data capture), then test and cut over.

Updated 4 Nov 2026 6 phases 18 detailed steps 8 official sources

START HERE

How do you migrate SQL Server and MySQL to Amazon RDS and Aurora?

SHORT ANSWER

This moves SQL Server and MySQL databases into Amazon RDS or Aurora, where AWS runs the database engine, backups and patching for you. For each database you either restore a backup (simple and faithful, but offline while it copies) or use AWS DMS (Database Migration Service), which keeps copying changes so the outage is only minutes.

- **Recommended route:** Native backup and restore where the outage fits, AWS DMS where it does not
- **Order:** Assess each database and build the target first, script everything that lives outside the database, prove the method on a small database, then move the rest with the largest started earliest.
- **6 phases:** Assess each database, Build the target, Load with native tools, Or replicate with DMS, Test and cut over and Finish and clean up.
- **You're done when:** Every database and table is on the target, and row counts match for key tables at cutover.

The time is driven by database size, which sets how long the backup, copy and restore or the first full load take.

RECOMMENDED ROUTE

Native backup and restore where the outage fits, AWS DMS where it does not

Native restores give the most faithful copy, and DMS keeps the source online until a cutover measured in minutes.

▼ **Other options and when to use them**

- **Backup restore plus log or binlog replication**
  : Large databases: restore the bulk from a backup, then replicate changes to catch up before cutover.
- **Logical dump and import**
  : Small MySQL databases where a short outage is acceptable.

TOOLS FOR THIS PLAN

Not sure which route?
Answer 6 quick questions

Are you ready?
22-question readiness check

Plan your dates
Dated countdown you can add to your calendar

Compare options
Cloud platforms compared

Check your DNS
MX, SPF, DKIM and DMARC lookup

New to the jargon?
Plain-English migration glossary

ORDER TO MIGRATE IN, AND WHY

Assess each database and build the target first, script everything that lives outside the database, prove the method on a small database, then move the rest with the largest started earliest. Each step depends on the one before: a restore or DMS task needs the target, network and S3 bucket in place, and applications only connect if their logins already exist.

1. **Assessment: engine version, edition, size, change rate and features in use**
   Decides whether native restore, binlog or log catch-up, or DMS can move each database at all.
2. **Target: subnet group, security group, RDS instance or Aurora settings, S3 bucket and IAM role**
   Every load method needs somewhere to restore to and a path for the files or the replication.
3. **Scripts for logins, users, grants, jobs and routines**
   Several methods leave these behind, and applications cannot connect without them.
4. **Pilot: one small, non-critical database migrated and cut over**

Times backup, upload and restore (or DMS load) and finds missing objects while the stakes are low.

5 **Bulk loads, largest databases first**

The full copy takes longest for big databases; log, binlog or DMS replication then keeps them current until cutover.

6 **Cutover per application, with its databases in one window**

An application writing to two engines at once splits its data.

7 **Rollback window, then decommission the source and delete DMS resources**

The source is your fallback until owners sign off, and the DMS replication instance is billed while it exists.

SETTINGS CHECKLIST: DECIDE THESE BEFORE YOU COPY

Built-in and third-party tools treat these differently. Tick each one off once you have decided.

Download this checklist (Excel)

☐ **Engine version and edition**

Which target engine version each database restores to.

▼ How each tool handles it

Built-in method	Aurora MySQL physical restore (Percona XtraBackup files from S3): MySQL 5.7 or 8.0 only, into the same major version, never to an older version. MySQL 5.5 and 5.6 and Percona Server are not supported, and some early 8.0 builds (8.0.11, 8.0.13, 8.0.15) cannot go to Aurora MySQL 3.05 or later. RDS for SQL Server native restore: pick an engine version equal to or newer than the source.
AWS Database Migration Service	Copies between different versions and engines, with DMS Schema Conversion for engine changes.

Tip: Upgrade old MySQL 8.0 builds to 8.0.28 in their own change window before the migration.

AWS docs: Physical migration from MySQL by using Percona XtraBackup and Amazon S3

☐ **Logins, users and grants**

How every application login and user arrives on the target with the right permissions.

▼ How each tool handles it

Built-in method	RDS for SQL Server native restore: restores the database with its database users. Server logins are kept outside the database by SQL Server, so recreate them and check each user maps to its login. Aurora MySQL physical restore: does not restore user accounts; unsupported privileges are removed, and Aurora MySQL 3 does not import dynamic privileges.
AWS Database Migration Service	Does not create user accounts on the target.
MySQL Shell (util.dumpInstance)	Dumps user accounts, roles and grants with the instance. Aurora is not a documented target, so test the load.

Tip: Script logins and grants from the source before the pilot, and test each application login on the target.

AWS docs: Physical migration from MySQL by using Percona XtraBackup and Amazon S3

- [ ] **Stored procedures, functions, triggers and jobs**
Which code objects and scheduled jobs are recreated on the target, and when.

▼ How each tool handles it

Built-in method	RDS for SQL Server native restore: procedures, functions and triggers inside the database come with it; SQL Server Agent jobs live outside the database and must be recreated. Aurora MySQL physical restore: does not restore functions or stored procedures, and objects with the rdsadmin definer are not imported.
AWS Database Migration Service	Create triggers after the migration and before application cutover. Whether it copies stored procedures: not stated in AWS's guidance: test it in your pilot.
MySQL Shell (util.dumpInstance)	Includes routines, events and triggers. User-defined functions are not included.

Tip: List every job and routine in the assessment, and tick each one off on the target before cutover.

AWS docs: Physical migration from MySQL by using Percona XtraBackup and Amazon S3

- [ ] **Indexes, keys and defaults**

Who adds secondary indexes, foreign keys and default values on the target, and at what stage.

▼ How each tool handles it

Built-in method	Native restore (SQL Server .bak or Aurora physical restore): indexes, keys and defaults come with the database.
AWS Database Migration Service	Creates tables, primary keys and sometimes unique indexes, but not secondary indexes, other constraints or data defaults. With ongoing replication, add secondary indexes after the full load and foreign keys after cached changes are applied.

Tip: Script the schema from the source so DMS targets get every index and constraint at the right stage.

AWS re:Post: When can I add secondary objects to a target database during AWS DMS migration?

- [ ] **Large text and binary columns (LOBs)**

How large objects are copied, and what size limit applies.

▼ How each tool handles it

Built-in method	Native restore: LOB columns come with the database like any other data.
AWS Database Migration Service	Limited LOB mode truncates any LOB larger than the size you set. Full LOB mode copies them whole, more slowly. Inline LOB mode works only during the full load.

Tip: Measure the largest LOB in each table before choosing a DMS LOB mode.

AWS re:Post: How can I improve the speed of LOB data migration with AWS DMS?

- [ ] **Database and file size**

How backups are split, and which databases are too large for a method.

▼ How each tool handles it

Built-in method	RDS for SQL Server native restore: databases up to 64 TiB (10 GB on Express Edition), 5 TB per backup file, and at most 10 backup files at once, so use multifile backups for large databases. Aurora MySQL physical restore: up to 64 TiB uncompressed, 5 TB per S3 file and at most 1 million files.

AWS Database Migration Service	Not stated in AWS's DMS documentation: test it in your pilot.

Tip: Split XtraBackup streams with split, and give every database its own S3 folder (prefix).

AWS docs: Importing and exporting SQL Server databases using native backup and restore

- [ ] **Features the method cannot carry**

Which features are removed, replaced or moved by another method.

▼ **How each tool handles it**

Built-in method	RDS for SQL Server native restore: cannot restore a database with a FILESTREAM file group, or over an existing database with the same name. Aurora MySQL physical restore: no tables outside the default data directory, only one InnoDB system data file (ibdata1), and no plug-ins or user tables in the mysql schema.
AWS Database Migration Service	Does not support some SQL Server features, such as sparse columns, columnstore indexes, memory-optimised and temporal tables. Run the premigration assessment to find them.

Tip: Add a feature column to the assessment and resolve every flagged feature before the bulk load.

AWS docs: Importing and exporting SQL Server databases using native backup and restore

- [ ] **Encryption**

How backups and data are encrypted on the way and at rest.

▼ **How each tool handles it**

Built-in method	RDS for SQL Server native backup: files are encrypted with a symmetric AWS KMS key (asymmetric keys are not supported), and TDE databases (Transparent Data Encryption) can be restored. Aurora MySQL physical restore: cannot restore from an encrypted source database, but can restore files encrypted in S3 with SSE-S3 or SSE-KMS.
AWS Database Migration Service	Not stated in AWS's DMS documentation: test it in your pilot.

Tip: Turn on storage encryption when you create the RDS instance or Aurora cluster, because it is set at creation.

AWS docs: Physical migration from MySQL by using Percona XtraBackup and Amazon S3

- [ ] **Time zone and server settings**

Which server-level settings, such as time zone, must match the source.

▼ How each tool handles it

Built-in method	Aurora MySQL physical restore: time zone information is not restored; Aurora loads it from the cluster's own operating system. RDS for SQL Server: server-level settings are not part of a database backup, so compare them in the pilot.
AWS Database Migration Service	Not stated in AWS's DMS documentation: test it in your pilot.

Tip: Record the source time zone and key server settings in the assessment, and set them in the target parameter or option group.

AWS docs: Physical migration from MySQL by using Percona XtraBackup and Amazon S3

- [ ] **Catch-up copies before cutover**

How changes made after the bulk load reach the target, and how short the outage must be.

▼ How each tool handles it

Built-in method	RDS for SQL Server: restore the full backup with NORECOVERY, then apply differential and log backups. No snapshot may be taken between the full and differential backups, and Multi-AZ instances only restore databases in the full recovery model. Aurora MySQL: binlog replication from the source keeps the cluster current until cutover; incremental XtraBackup backups are also supported.
AWS Database Migration Service	Full load plus ongoing replication. SQL Server sources need MS-CDC or MS-REPLICATION; MySQL sources need binlog_format=ROW.

Tip: Keep binlogs or log backups for the full load time plus a margin, so replication never has to start again.

AWS docs: Importing and exporting SQL Server databases using native backup and restore

☐ **Rollback after cutover**

How the old server is kept current after the application starts writing to RDS, if at all.

▼ **How each tool handles it**

Built-in method	RDS for SQL Server cannot make native log backups, so changes cannot be shipped back with log restores. You can take a full native backup from RDS and restore it on the source.
AWS Database Migration Service (RDS for SQL Server as a source)	Replicates ongoing changes from RDS for SQL Server with MS-CDC or MS-REPLICATION on. Use DMS 3.5.3 or later.
AWS Database Migration Service (SQL Server as a target)	Writes to on-premises SQL Server. Start the reverse task at cutover, before the first application write.

Tip: Agree with the data owner, before cutover, how long the rollback window is and which method keeps it possible.

AWS docs: Importing and exporting SQL Server databases using native backup and restore

☐ **How many at once**

How many restores or DMS tasks run in parallel in each window.

▼ **How each tool handles it**

Built-in method	RDS for SQL Server native: up to two backup or restore tasks run at the same time on an instance. Aurora MySQL physical restore creates a new cluster each time and cannot load into an existing one.
AWS Database Migration Service	Tasks share the replication instance you size, so watch its CPU and memory in CloudWatch.

Tip: Sequence restores in the runsheet two at a time per instance, and track each with rds_task_status.

AWS docs: Importing and exporting SQL Server databases using native backup and restore

THE PLAN, PHASE BY PHASE

1 Assess each database

Before you build anything

- Record each database's engine version, edition, size and daily change rate.
- List features in use, such as SQL Agent jobs, linked servers, CLR, FILESTREAM and MyISAM tables.
- Check RDS offers an engine version equal to or newer than

the source.

Pick a method per database and write down its expected downtime.

> **Watch out:** Aurora MySQL physical restore: some older MySQL 8.0 builds (such as 8.0.11, 8.0.13 and 8.0.15) cannot move to Aurora MySQL 3.05 or higher.
>
> **WAYS AROUND IT**
>
> - Upgrade the source to 8.0.28 in its own change window before the migration.

> **Watch out:** Aurora MySQL physical restore does not support MySQL 5.5 or 5.6, Percona Server or encrypted source databases, and cannot go to an older version.
>
> **WAYS AROUND IT**
>
> - Upgrade the source, choose a compatible Aurora version, or use DMS.

Show the detailed steps (3) →

2 Build the target

Pick the AWS Region closest to your users that meets

your residency rules, such as Sydney (ap-southeast-2).

Create a DB subnet group with private subnets in at least two Availability Zones.

Create a security group that allows only the application subnets and the migration path on the database port.

Create the RDS instance with the size and Multi-AZ setting from your assessment (skip this for Aurora physical restores).

Create an S3 bucket (AWS file storage) and an IAM role that RDS can use to read it.

> **Watch out:** Restores fail if the S3 bucket is in a different Region from the database.

WAYS AROUND IT

- Create the bucket in the target Region and select that Region in the console before restoring.

Show the detailed steps (3) →

3 Load with native tools

If the outage window covers the copy

SQL Server: add the SQLSERVER_BACKUP_RESTORE option to the instance's option group.

Take a full backup with CHECKSUM, upload it to its own S3 folder, and restore it with NORECOVERY.

Apply differential or log

backups taken since, and track each task with rds_task_status.

MySQL to Aurora: back up with Percona XtraBackup, upload to S3 and use Restore from S3.

Start binlog replication from the source so Aurora catches up until cutover.

> ⚠ **Watch out:** RDS for SQL Server native restore cannot take FILESTREAM databases or overwrite a database with the same name, and S3 caps each file at 5 TB.

WAYS AROUND IT

- Use multifile SQL Server backups, or split the XtraBackup stream, for large databases.

Watch out: With a blank S3 prefix, RDS tries to restore every file in the bucket or its root.

WAYS AROUND IT

- Give each database its own folder and set the S3 prefix on every restore.

Watch out: Aurora MySQL physical restore (from XtraBackup files) does not restore user accounts, functions, stored procedures or time zone information.

WAYS AROUND IT

- Export users, grants and routines before the backup with SHOW GRANTS and mysqldump --no-data --routines, then apply them.

TOOLS THAT CAN DO IT

MySQL Shell (util.dumpInstance)
Dumps users, grants and routines in one go. User-defined functions are not included, and Aurora is not a documented target, so test the load.

Show the detailed steps (3) →

4 Or replicate with DMS

If the database must stay online

SQL Server: use the full recovery model, take a full backup, and turn on MS-REPLICATION

or MS-CDC.

MySQL: turn on binary logging with binlog_format=ROW and binlog_row_image=FULL, then create the DMS user.

Create a DMS replication instance and source and target endpoints, and test both connections.

Create a task to migrate existing data and replicate

ongoing changes, starting with the premigration assessment.

> **Watch out:** DMS does not support some SQL Server features, such as sparse columns, columnstore indexes, memory-optimised and temporal tables.
>
> **WAYS AROUND IT**
>
> - Run the premigration assessment, then fix or exclude what it reports before the full load.

> **Watch out:** If the source purges binlogs before the full load finishes, replication cannot catch up and you start again.
>
> **WAYS AROUND IT**
>
> - Keep binlogs for the full load time plus a margin (at least one day for DMS), with disk space to match.

Show the detailed steps (3) →

5 Test and cut over

In the agreed change window

Point a test copy of the application at the target, and run functional and performance tests.

Compare row counts for key tables in the data validation report.

Stop application writes, and wait until log restores, replication lag or DMS latency reach zero.

Finish the restore, stop replication or stop the DMS task.

Point connection strings or

the DNS alias at the RDS endpoint, and keep the source read-only.

> **Watch out:** RDS for SQL Server cannot make native log backups, so you cannot ship changes back to the old server after writes start.
>
> **WAYS AROUND IT**
>
> - Agree how long the source stays read-only, or take a full backup from RDS to restore on the source.
>
> **TOOLS THAT CAN DO IT**
>
> **AWS Database Migration Service (RDS for SQL Server as a source)**
> Replicates ongoing changes from RDS for SQL Server with MS-CDC or MS-REPLICATION on. Use DMS 3.5.3 or later.
>
> **AWS Database Migration Service (SQL Server as a target)**
> Writes to on-premises SQL Server. Start the reverse task at cutover, before the first application write.

Turn this phase into dated tasks with the cutover planner

Check your DNS records before and after the change

Show the detailed steps (3) →

6 Finish and clean up

After sign-off

Recreate logins, jobs and users that did

not migrate, and check each application login connects.

Turn on automated backups and Performance Insights, and set CloudWatch alarms.

Delete the DMS replication instance and endpoints.

Decommission the source database only after the rollback window closes.

> ⚠ **Watch out:** The DMS replication instance is billed for as long as it exists.
>
> **WAYS AROUND IT**

- Size it for the full load, watch it in CloudWatch, and delete it straight after cutover.

Show the detailed steps (3) →

✓ You are finished when

- Every database and table is on the target, and row counts match for key tables at cutover.
- Every application login connects to the RDS endpoint, and jobs and routines work.
- Automated backups have a retention period above zero and CloudWatch alarms are in OK state.
- The source has no application sessions or writes after cutover.

Tools named in this plan are independent third-party products, not endorsed by the platform vendor and not paid placements. Check the tool's own documentation and run a pilot first.

CLICK BY CLICK

What does each step involve?

Open a part when you reach it in the plan. Each step says where to go, what to do and what you should see.

About this migration

What it covers, limits at a glance −

For homogeneous moves (SQL Server to RDS for SQL Server, MySQL to RDS for MySQL or Aurora MySQL) native tools are usually the fastest and most faithful option. AWS Database Migration Service (DMS) adds continuous replication, so the source stays online until cutover, and it handles heterogeneous moves with DMS Schema Conversion.
Many teams combine them: restore a native backup for the bulk load, then use replication to catch up.

SQL SERVER TO RDS
Native .bak restore from Amazon S3 with

MYSQL TO AURORA MYSQL

MINIMAL DOWNTIME
AWS DMS full load plus change data capture

the SQLSERVER_BACKUP_RESTORE option

Percona XtraBackup to S3, then binlog replication to catch up

(CDC)

REGION RULE

S3 bucket and DB instance or cluster must be in the same AWS Region

CHOOSING A REGION

The Region closest to your users that meets your residency needs (for example Asia Pacific (Sydney) ap-southeast-2 or Asia Pacific (Melbourne) ap-southeast-4 in Australia)

1 What you need before step one

Need	Details	Where to get it
AWS permissions	To create RDS instances or Aurora clusters, option groups, S3 buckets, IAM roles (including iam:PassRole) and DMS resources.	AWS Management Console, https://console.aws.amazon.com/rds/
AWS CLI v2	For the S3, RDS and DMS commands below.	Install the AWS CLI, or use AWS CloudShell
Source database access	sysadmin on SQL Server; a MySQL account that can run backups and create replication users.	Your DBA team
Percona XtraBackup (MySQL to Aurora)	Version 2.4 for MySQL 5.7, version 8.0 for MySQL 8.0, installed on the source server.	https://www.percona.com/downloads
Network path	VPN, Direct Connect or a DMS replication instance that can reach both the source and the target.	Your network team

Need	Details	Where to get it
SQL client	SQL Server Management Studio for SQL Server; the mysql client for MySQL and Aurora.	Microsoft Learn (SSMS) or the MySQL website
Templates	Application inventory and dependency map, Data validation report, Cutover runsheet.	Templates linked at the end of this guide

2 Assess the source databases

3 steps

Source to target	Method	Downtime	Notes
SQL Server to RDS for SQL Server	Native backup and restore from S3	Backup, copy and restore time (less with log restores)	No FILESTREAM; up to 5 TB per backup file, use multifile backups for larger
SQL Server to RDS for SQL Server	AWS DMS full load plus CDC	Minutes at cutover	Needs full recovery model and MS-REPLICATION or MS-CDC on the source
MySQL 5.7 or 8.0 to Aurora MySQL	Percona XtraBackup to S3, restore to new cluster	Low when followed by binlog replication	New cluster only; same major version or newer
MySQL to RDS for MySQL or Aurora	AWS DMS full load plus CDC	Minutes at cutover	Needs binary logging in ROW format

1

Inventory databases and sizes

Where: SSMS (SQL Server) or the mysql client (MySQL) connected to each source, then your application inventory

Record engine version, edition, database sizes, daily change rate, features in use (SQL Agent jobs, linked servers, CLR, FILESTREAM, MyISAM tables) and connected applications. The queries below give version and size.

```
-- SQL Server
SELECT @@VERSION;
SELECT DB_NAME(database_id) AS db, SUM(size) * 8 / 1024 AS size_mb FROM sys.master_files GROUP BY
database_id;
-- MySQL
SELECT VERSION();
SELECT table_schema, ROUND(SUM(data_length + index_length) / 1024 / 1024) AS size_mb FROM
information_schema.tables GROUP BY table_schema;
SELECT table_schema, table_name FROM information_schema.tables WHERE engine = 'MyISAM';
```

✓ **You should see:** A row per database with version, size and features, and a list of any MyISAM tables.

2

Check target compatibility

Where: Amazon RDS console > Databases > Create database (engine version list), or a terminal with the AWS CLI

Choose an RDS engine version equal to or newer than the source. For Aurora MySQL physical migration, backward migration is not supported, so pick an Aurora version compatible with or newer than your MySQL version.

```
aws rds describe-db-engine-versions --engine sqlserver-se --region ap-southeast-2 --query
"DBEngineVersions[].EngineVersion"
```

✓ **You should see:** The list includes a version equal to or newer than your source.

3

Pick a method per database

Where: Your application inventory, using the table below

Use the table below. Prefer native methods when the downtime window fits, and DMS when it does not.

✓ **You should see:** Every database has a method and an estimated downtime.

3 Build the target and network

3 steps

1

Create the DB subnet group and security group

Where: Amazon RDS console > Subnet groups > Create DB subnet group; Amazon VPC console > Security groups > Create security group

Place the database in private subnets and allow inbound access only from application subnets and the migration path (VPN, Direct Connect or DMS replication instance), on the database port (1433 for SQL Server, 3306 for MySQL).

✓ **You should see:** A DB subnet group with private subnets in at least two Availability Zones, and a security group with only the inbound rules you listed.

2

Create the instance or cluster

Where: Amazon RDS console > Databases > Create database

Create the RDS instance with the engine version, instance class, storage and Multi-AZ setting from your assessment, using the subnet group and security group above. For Aurora physical migration, skip this: the restore from S3 creates the cluster.

✓ **You should see:** The instance status changes from Creating to Available.

3

Create the S3 bucket and IAM role

Where: A terminal with the AWS CLI for the bucket; the IAM role can be created for you by the RDS console (Create a new role) in Step 3a or 3b

Create an S3 bucket in the same Region as the database and an IAM role that RDS can assume with access to that bucket.

```
aws s3 mb s3://contoso-db-migration-syd --region ap-southeast-2
```

✓ **You should see:** make_bucket: contoso-db-migration-syd

4 **SQL Server: native backup and restore** 3 steps –

1

Enable native backup and restore

Where: Amazon RDS console > Option groups > your option group > Add option, or a terminal with the AWS CLI

Add the **SQLSERVER_BACKUP_RESTORE** option. For **IAM role** choose an existing role, or choose **Create a new role**, pick the **S3 bucket** and an optional **S3 prefix**, then choose **Add option**. Apply the option group to the instance by modifying the instance. No restart is needed.

```
aws rds add-option-to-option-group \
  --apply-immediately \
  --option-group-name sqlserver-se-backup \
  --options "OptionName=SQLSERVER_BACKUP_RESTORE,OptionSettings=
[{Name=IAM_ROLE_ARN,Value=arn:aws:iam::123456789012:role/rds-sql-backup}]"
aws rds modify-db-instance \
  --db-instance-identifier sql-sales-syd \
  --option-group-name sqlserver-se-backup \
  --apply-immediately
```

✓ **You should see:** The instance's Configuration tab shows the option group as in-sync.

2

Back up and upload

Where: SSMS query window on the source SQL Server, then a terminal with the AWS CLI on the same server

Take a full backup on the source (with CHECKSUM) and upload it to S3. For large databases split the backup across multiple files.

```
BACKUP DATABASE SalesDb TO DISK = 'D:\Backup\SalesDb.bak' WITH CHECKSUM, COMPRESSION;
aws s3 cp D:\Backup\SalesDb.bak s3://contoso-db-migration-syd/SalesDb.bak
```

✓ **You should see:** BACKUP reports success, and aws s3 cp prints upload: ... to s3://contoso-db-migration-syd/SalesDb.bak.

3

Restore with NORECOVERY, then apply logs

Where: SSMS connected to the RDS endpoint as the master user (endpoint from RDS console > Databases > the instance > Connectivity & security)

Restore the full backup without recovery, apply differential or log backups taken since, and finish the restore at cutover. Track each task with rds_task_status.

```
exec msdb.dbo.rds_restore_database
@restore_db_name='SalesDb',
@s3_arn_to_restore_from='arn:aws:s3:::contoso-db-migration-syd/SalesDb.bak',
@type='FULL',
@with_norecovery=1;

exec msdb.dbo.rds_restore_log
@restore_db_name='SalesDb',
@s3_arn_to_restore_from='arn:aws:s3:::contoso-db-migration-syd/SalesDb_log1.trn',
@with_norecovery=1;

exec msdb.dbo.rds_task_status @db_name='SalesDb';

exec msdb.dbo.rds_finish_restore @db_name='SalesDb';
```

✓ **You should see:** rds_task_status shows each task as SUCCESS, and after rds_finish_restore the database is online.

Native restore limits. AWS documents that you cannot restore a database with FILESTREAM file groups, cannot restore over an existing database with the same name, can run up to two backup or restore tasks at once, and on Multi-AZ instances can only restore databases backed up in the full recovery model.

5 MySQL: physical migration to Aurora MySQL

3 steps

1

Back up with Percona XtraBackup

Where: A shell on the source MySQL server with Percona XtraBackup installed (https://www.percona.com/downloads)

Use Percona XtraBackup 2.4 for MySQL 5.7 and 8.0 for MySQL 8.0. Split the stream so files are named in order, and do not use partial backup options such as --databases or --tables.

```
xtrabackup --backup --user=backup_user --password=<password> --stream=xbstream \
   --target-dir=/backup/s3-restore | split -d --bytes=500MB \
   - /backup/s3-restore/backup.xbstream
```

✓ **You should see:** Numbered files such as backup.xbstream00, backup.xbstream01 in /backup/s3-restore, and xtrabackup ends with completed OK!

2

Upload to S3 and restore

Where: A terminal with the AWS CLI, then Amazon RDS console (same Region as the bucket) > Databases > Restore from S3

Copy the files to the S3 bucket. On **Create database by restoring from S3**, under **S3 destination** choose the bucket and the **S3 folder path prefix**. Under **Engine options** choose **Amazon Aurora** and the Aurora MySQL version. For **IAM role** choose an existing role or **Create a new role**. Set the instance class, cluster identifier and credentials, then choose **Create database**.

```
aws s3 cp /backup/s3-restore/ s3://contoso-db-migration-syd/mysql/full/ --recursive
```

✓ **You should see:** The new cluster shows status Creating, then Available.

3

Catch up with binlog replication

Where: Amazon RDS console > Events (binlog position), the source MySQL server (replication user), then the mysql client connected to the Aurora writer endpoint

Find the binlog file and position in the **Recovered from Binary log filename** event. Create a replication user on the source with REPLICATION CLIENT and REPLICATION SLAVE, add the source server's IP to the cluster's VPC security group, and start replication on the Aurora cluster. On Aurora MySQL version 3 use rds_set_external_source.

```
CREATE USER 'repl_user'@'10.0.%' IDENTIFIED BY '<password>';
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'10.0.%';
-- On the Aurora cluster
CALL mysql.rds_set_external_source('mysql01.contoso.com', 3306, 'repl_user', '<password>', 'mysql-bin.000010', 120, 0);
CALL mysql.rds_start_replication;
SHOW REPLICA STATUS;
```

✓ **You should see:** SHOW REPLICA STATUS shows replication running, and Seconds Behind Master falls to 0.

Physical restore limits. AWS lists several restrictions: MySQL 5.5 and 5.6, Percona Server for MySQL and encrypted source databases are not supported, user accounts, functions and stored procedures may need recreating, and the innodb_data_file_path must use the default single ibdata1 file.

6 AWS DMS: full load plus change data capture

3 steps

1

Prepare the source for CDC

Where: The source database: SSMS for SQL Server; my.cnf and the mysql client for MySQL

For SQL Server, use the full (or bulk-logged) recovery model, take a full backup, and enable MS-REPLICATION or MS-CDC. For MySQL, enable binary logging with `binlog_format=ROW`, `binlog_row_image=FULL`, `binlog_checksum=NONE` and retention of at least one day, then create the DMS user.

```
CREATE USER 'dms_user'@'%' IDENTIFIED BY '<password>';
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'dms_user'@'%';
GRANT SELECT, RELOAD, LOCK TABLES, SHOW VIEW, EVENT, TRIGGER ON *.* TO 'dms_user'@'%';
```

✓ **You should see:** SHOW VARIABLES LIKE 'binlog_format'; returns ROW on MySQL, or the database shows the Full recovery model on SQL Server.

2

Create the replication instance and endpoints

Where: AWS DMS console > Replication instances > Create replication instance; then Endpoints > Create endpoint

Create a replication instance (or use DMS Serverless) in a subnet that can reach both databases. Create a **Source** endpoint with **Provide access information manually**, and a **Target endpoint** with **Select RDS DB instance**. On **Test endpoint connection (optional)** choose **Run test** for each.

```
aws dms create-endpoint --endpoint-identifier src-sales-mysql --endpoint-type source --engine-name
mysql --server-name mysql01.contoso.com --port 3306 --username dms_user --password <password>
aws dms test-connection --replication-instance-arn <replication-instance-arn> --endpoint-arn
<source-endpoint-arn>
```

✓ **You should see:** Both connection tests show successful.

3

Create and start a full load and CDC task

Where: AWS DMS console > Database migration tasks > Create task

Choose the replication instance and endpoints and the migration type **Migrate existing data and replicate on-going changes**. Under **Table mappings** > **Selection rules** choose **Add new selection rule** for each schema. Under **Migration task startup configuration** choose **Start with premigration assessments (recommended)** or **Start automatically on create**.

```
aws dms create-replication-task --replication-task-identifier sales-full-cdc --source-endpoint-arn
<source-endpoint-arn> --target-endpoint-arn <target-endpoint-arn> --replication-instance-arn
<replication-instance-arn> --migration-type full-load-and-cdc --table-mappings file://table-
mappings.json
aws dms start-replication-task --replication-task-arn <task-arn> --start-replication-task-type
start-replication
```

✓ **You should see:** The task shows Running, and Database migration tasks > the task > Table statistics shows each table's full load progress.

Check DMS limitations first. For SQL Server sources AWS lists unsupported items including sparse columns, columnstore indexes, memory-optimised tables, temporal tables and databases with accelerated database recovery. Review the full list for your engine before you choose DMS.

7 Test and cut over

3 steps

1

Test against a copy

Where: A test copy of the application, pointed at the target endpoint, and your Data validation report

Point a test copy of the application at the target, run functional and performance tests, and compare row counts for key tables.

✓ **You should see:** Tests pass and row counts match for every key table.

2

Cut over

Where: The application (stop writes), then the method you used: SSMS (rds_finish_restore), the mysql client on Aurora, or AWS DMS console > Database migration tasks

Stop application writes, wait for the last log restore, binlog replication lag or DMS CDC latency to reach zero, then finish the restore, stop replication or stop the DMS task. Point connection strings or DNS at the RDS endpoint.

```
CALL mysql.rds_stop_replication;
```

✓ **You should see:** The target accepts writes from the application and the source receives no new connections.

3

Post-cutover checks

Where: Amazon RDS console > Databases > the instance > Modify (backups, Performance Insights), and Amazon CloudWatch > Alarms > Create alarm

Recreate logins, jobs and users that did not migrate, enable automated backups and Performance Insights, and set CloudWatch alarms.

✓ **You should see:** The instance shows a backup retention period above zero, Performance Insights enabled, and alarms in the OK state.

Rollback. Keep the source database read-only and intact until sign-off. Once the application writes to RDS, rolling back requires moving those changes back.

WORTH KNOWING

What can go wrong, and how do you avoid it?

The detail behind the plan: options compared, and what experienced engineers check first. Tap any item.

Compare all the ways to do this 4 options −

Native SQL Server backup and restore from Amazon S3

LOW EFFORT

Best for: SQL Server databases moving to RDS for SQL Server where the outage window covers backup, copy and restore

+ The most faithful copy: a full .bak restored by RDS
+ Log restores with NORECOVERY shorten the final outage
– No FILESTREAM, and up to 5 TB per backup file (use multifile backups for larger databases)
– The S3 bucket must be in the same AWS Region as the DB instance

Read more →

Physical MySQL migration to Aurora with Percona XtraBackup

MEDIUM EFFORT

Best for: Large MySQL 5.7 or 8.0 databases moving to Aurora MySQL

+ Faster than a logical dump for large databases
+ Binlog replication catches up the changes, so downtime stays low
– Restores to a new cluster only, at the same major version or newer
– Users, routines and time zones need recreating on Aurora

Read more →

AWS DMS full load plus change data capture

MEDIUM EFFORT

Best for: Databases that must stay online until a cutover measured in minutes, and heterogeneous moves

+ The source stays live while changes replicate
+ Works for SQL Server and MySQL sources, with DMS Schema Conversion for engine changes
– SQL Server needs the full recovery model and MS-REPLICATION or MS-CDC; MySQL needs binary logging in ROW format
– A replication instance to size, monitor and stop afterwards

Read more →

Logical dump and import (small MySQL databases)

LOW EFFORT

Best for: Small MySQL databases where a short outage is acceptable

+ Standard MySQL tools, no extra AWS services
+ Works for RDS for MySQL and Aurora MySQL targets
– The database is offline from export until import finishes
– Slow for large databases

Read more →

OUR ADVICE

Pick a method per database. Use native tools when the outage window covers the backup, copy and restore, and DMS when it does not. Many teams combine them: a native backup or XtraBackup restore for the bulk load, then log or binlog replication to catch up before cutover.

Keep the S3 bucket in the same Region

SAVES TIME –

AWS requires the S3 bucket and the Aurora MySQL cluster to be in the same Region, and native SQL Server restore fails when the bucket is elsewhere.

WHAT TO DO

Create the migration bucket in the target Region (Step 2) and select that Region in the RDS console before you choose Restore from S3.

AWS docs: Physical migration from MySQL using Percona XtraBackup and Amazon S3

Split large backups into files under 5 TB

SAVES TIME

Amazon S3 limits a single uploaded file to 5 TB. AWS also limits an Aurora restore to 1 million source files.

WHAT TO DO

Use multifile BACKUP for big SQL Server databases and the split command with xtrabackup for MySQL. Use xbstream or tar archives if the backup would exceed a million files.

AWS docs: Physical migration from MySQL using Percona XtraBackup and Amazon S3

Users, routines and time zones need recreating on Aurora

REDUCES RISK

AWS says Aurora MySQL does not restore everything: user accounts, functions, stored procedures and time zone information should be saved and added after the restore. Objects with the rdsadmin definer are not imported.

WHAT TO DO

Export users, grants and routines from the source before the backup (for example with SHOW GRANTS and mysqldump --no-data --routines), and apply them to the new cluster before testing.

TOOLS THAT CAN GET AROUND THIS

MySQL Shell's instance dump includes user accounts, roles and grants as well as routines, events and triggers, so you can carry them across in one dump instead of rebuilding them by hand.

MySQL Shell (util.dumpInstance)

Users and routines are included by default. Use excludeUsers to leave out accounts the target will not accept; user-defined functions are not included. Oracle does not document Aurora as a target, so test the load.

Third-party tools, not endorsed by the platform vendor. Confirm in the tool's documentation and run a pilot first.

AWS docs: Physical migration from MySQL using Percona XtraBackup and Amazon S3

Upgrade old MySQL 8.0 builds first

REDUCES RISK

AWS says you cannot migrate to Aurora MySQL version 3.05 and higher from some older MySQL 8.0 versions, including 8.0.11, 8.0.13 and 8.0.15, and recommends upgrading to 8.0.28 before migrating.

WHAT TO DO

Check SELECT VERSION() on the source during assessment and upgrade the source in its own change window before the migration.

AWS docs: Physical migration from MySQL using Percona XtraBackup and Amazon S3

Use the S3 prefix to stop RDS reading the whole bucket

REDUCES RISK

AWS says that if you leave the prefix blank, a multiple-file SQL Server restore tries to restore every file in every folder of the bucket, and an Aurora restore uses all files in the bucket root.

WHAT TO DO

Give each database its own folder, set **S3 prefix** on the SQLSERVER_BACKUP_RESTORE option or **S3 folder path prefix** on Restore from S3, and do not mix backup sets in one folder.

AWS docs: Support for native backup and restore in SQL Server

Run the DMS premigration assessment

SAVES TIME

DMS has engine-specific limitations, such as sparse columns, columnstore indexes and temporal tables on SQL Server sources. Finding them after full load wastes the whole run.

WHAT TO DO

Choose **Start with premigration assessments (recommended)** when you create the task, fix or exclude what it reports, then start the task.

AWS docs: Set up replication for AWS DMS

Keep binlogs long enough for the whole load

REDUCES RISK

DMS CDC and Aurora binlog replication both read changes from the source binlog. If binlogs are purged before the full load finishes, replication cannot catch up and you start again.

WHAT TO DO

Set binlog retention to cover at least the full load time plus a margin (at least one day for DMS), and check free disk space on the source for the extra logs.

AWS docs: Using a MySQL-compatible database as a source (DMS)

Plan rollback without native log backups

REDUCES RISK

AWS states that native log backups from RDS for SQL Server are not supported. Once the application writes to RDS, you cannot ship logs back to the old server.

WHAT TO DO

Agree in the go or no-go meeting how long the source stays read-only, and if rollback after writes is required, plan a reverse DMS task or a full backup from RDS.

TOOLS THAT CAN GET AROUND THIS

AWS DMS can use RDS for SQL Server as a source with ongoing replication and an on-premises SQL Server as the target, so a reverse task can keep the old server up to date after cutover and keep rollback possible after the application starts writing to RDS.

AWS Database Migration Service (RDS for SQL Server as a source)

Supports CDC from Amazon RDS for SQL Server with MS-CDC or MS-REPLICATION enabled and the Full or Bulk logged recovery model. Use DMS 3.5.3 or later so changes are also read from log backups.

AWS Database Migration Service (SQL Server as a target)

AWS lists on-premises SQL Server (Enterprise, Standard, Workgroup and Developer) as a supported target. Start the reverse task at cutover, before the first application write.

Third-party tools, not endorsed by the platform vendor. Confirm in the tool's documentation and run a pilot first.

AWS docs: Importing and exporting SQL Server databases using native backup and restore

Right-size the replication instance and stop it afterwards

COST LEVER

The DMS replication instance runs, and is billed, for as long as it exists. Undersized instances slow full load; forgotten ones keep costing.

WHAT TO DO

Size it for the full load, watch CPU and memory in CloudWatch, consider DMS Serverless, and delete the instance and endpoints after cutover. Check the vendor's current pricing.

FINISH SAFELY

How do you check it worked, or roll back?

Full validation checklist

8 checks

Run these before you declare the migration complete. Record the results in the data validation report.

Check	How to check	What good looks like
Every database and table arrived	Rerun the Step 1 size queries against the RDS or Aurora endpoint and compare with the source. For DMS, open AWS DMS console > Database migration tasks > the task > Table statistics.	Every database and schema in scope exists on the target, and DMS shows every table as Table completed with no errors.
Row counts match for key tables	With writes stopped at cutover, run `SELECT COUNT(*)` on the key tables in your Data validation report on both source and target (SSMS for SQL Server, the mysql client for MySQL).	Counts match exactly for every key table at the cutover point.
Replication had fully caught up before cutover	For native SQL Server restores, check `exec msdb.dbo.rds_task_status` for the last log restore. For Aurora binlog replication, run `SHOW REPLICA STATUS;`. For DMS, check CDC latency on the task in Amazon CloudWatch.	The last restore task shows SUCCESS, replica lag is zero, or CDC latency is zero before you stopped the source.
Logins, users, jobs and routines work	Sign in to the target with each application login. On SQL Server, check that SQL Agent jobs you recreated run on schedule. On Aurora MySQL, run `SHOW GRANTS` for each user and call each stored procedure the application uses.	Every application login connects, jobs succeed, and grants and routines match the source export.
The application uses the new endpoint	Check connection strings or the DNS alias resolve to the RDS endpoint, and in the Amazon RDS console > the instance > Monitoring check DatabaseConnections.	All application servers connect to the RDS endpoint; connections appear in the RDS metrics.

Check	How to check	What good looks like
Performance is at or above the baseline	Run the same performance tests you ran against the test copy, and review Performance Insights and CloudWatch CPU, memory and IOPS for the instance during business hours.	Key transactions run within the agreed baseline and no resource stays saturated.
Backups and alarms are on	Run `aws rds describe-db-instances --db-instance-identifier sql-sales-syd --query "DBInstances[].BackupRetentionPeriod"` and check the CloudWatch alarms you created.	Automated backups have a retention period above zero and alarms are in OK state with a working notification target.
The source no longer receives writes	On the source, check active sessions (`sp_who2` in SSMS, or `SHOW PROCESSLIST;` in the mysql client) and confirm the database is read-only or stopped.	No application sessions or writes on the source after cutover.

Rollback plan

If it goes wrong —

ROLL BACK IF

- Row counts or checksums do not match and a final sync cannot fix it
- The application fails functional tests against RDS within the cutover window
- Performance under real load is far below the baseline and resizing does not fix it
- Logins or jobs that the business depends on cannot be recreated in time

POINT OF NO RETURN

Before the application writes to RDS, rollback is simple: point it back at the source, which you kept read-only and intact. Once the application writes to RDS, rolling back means moving those changes back. The native restore and binlog replication paths in this guide only run one way, into AWS, so the way back is a backup from RDS or a reverse DMS task. Deleting or decommissioning the source database is the final point of no return.

1

Stop writes to RDS

Where: The application (maintenance mode or stopped services)

Stop the application so no more data is written to the RDS or Aurora target.

✓ **You should see:** DatabaseConnections on the RDS instance drops to zero apart from admin sessions.

2

Make the source writable again

Where: SSMS or the mysql client connected to the source database

Reverse the read-only setting you applied at cutover (for SQL Server, `ALTER DATABASE SalesDb SET READ_WRITE;`) and restart any SQL Agent jobs or scheduled tasks you paused.

✓ **You should see:** The source database accepts writes from a test login.

3

Point the application back at the source

Where: Application configuration files or app settings, and your DNS alias

Restore the old connection strings or DNS alias, then restart the application.

✓ **You should see:** Application servers connect to the source and pass the smoke tests.

4

Bring back any data written to RDS

Where: SSMS connected to the RDS endpoint, or the AWS DMS console

For SQL Server, take a full backup from RDS to S3 with `exec msdb.dbo.rds_backup_database @source_db_name='SalesDb', @s3_arn_to_backup_to='arn:aws:s3:::contoso-db-migration-syd/SalesDb-rollback.bak';` and restore what you need on the source. For either engine you can create a reverse DMS task with RDS as the source. Agree with the data owner which approach to use before cutover.

✓ **You should see:** The source holds every transaction made on RDS since cutover, checked with row counts.

5

Stop the migration resources

Where: AWS DMS console, Amazon RDS console

Stop any DMS tasks still running and stop Aurora replication with `CALL mysql.rds_stop_replication;` so the two sides do not diverge further.

✓ **You should see:** No migration task or replication is running.

6

Record what cannot be undone

Where: Your Rollback plan and Decommission checklist

Note that once the source server is decommissioned, rollback means restoring a backup from RDS to a new server. Keep the source read-only and intact until sign-off.

✓ **You should see:** Source decommission is scheduled after sign-off and the rollback window.

Agree the triggers and who decides before the go / no-go meeting. Use the rollback plan template.

Common problems and fixes

6 errors

`rds_restore_database fails: database already exists`

Cause: RDS cannot restore a database with the same name as an existing database.

Fix: Drop or rename the existing database on the instance, or restore under a different name.

`Native restore fails with an S3 access error`

Cause: The IAM role on the SQLSERVER_BACKUP_RESTORE option lacks bucket permissions, or the bucket is in a different Region.

Fix: Grant the role access to the bucket and use a bucket in the same Region as the instance.

`rds_finish_restore fails`

Cause: The database is not in RESTORING state, a restore task is still pending, or you are not using the login that ran the last NORECOVERY restore.

Fix: Wait for pending tasks in rds_task_status and run it as the master login or the login that ran the last restore.

`SQL Server error 22838 during DMS CDC`

Cause: A column in a unique index allows NULLs, which MS-CDC does not support for this case.

Fix: Define every column in the unique index as NOT NULL, or exclude the table from CDC.

`DMS MySQL task fails: binary logging not enabled or wrong format`

Cause: The source is not writing binlogs in ROW format with FULL row images.

Fix: Set log-bin, binlog_format=ROW, binlog_row_image=FULL and binlog_checksum=NONE, restart MySQL and restart the task.

`Aurora restore from S3 fails on source version`

Cause: The source is MySQL 5.5 or 5.6, or newer than the selected Aurora version.

Fix: Upgrade the source to a supported version or choose a compatible Aurora MySQL version, or use DMS.

Search the full error library →

FAQ

Frequently asked questions

Can I restore a SQL Server .bak file directly into Amazon RDS?

Yes. Add the SQLSERVER_BACKUP_RESTORE option, upload the .bak file to S3 in the same Region and run msdb.dbo.rds_restore_database.

Can I take native log backups from RDS for SQL Server?

No. AWS states native log backups are not supported from RDS for SQL Server, which affects how you plan a rollback.

What is the fastest way to move a large MySQL database to Aurora?

A physical backup with Percona XtraBackup restored from S3 is usually faster than mysqldump, and binlog replication keeps the cluster in sync until cutover.

When should I use AWS DMS instead of native tools?

Use DMS when the downtime window is too short for backup and restore, or when changing engines with DMS Schema Conversion.

Which AWS Region should I use?

Use the Region closest to your users that meets your data residency rules and offers the services you need. In Australia that means Asia Pacific (Sydney) ap-southeast-2 or Asia Pacific (Melbourne) ap-southeast-4.

SOURCES

Official sources

8 official pages this guide is based on –

Vendors change limits and features, so confirm details on the source before you rely on them. Last checked 4 Nov 2026.

- AWS docs: Importing and exporting SQL Server databases using native backup and...
- AWS docs: Using native backup and restore (RDS for SQL Server)
- AWS docs: Support for native backup and restore in SQL Server
- AWS docs: Physical migration from MySQL using Percona XtraBackup and Amazon ...
- AWS docs: Using a Microsoft SQL Server database as a source for AWS DMS
- AWS docs: Using a MySQL-compatible database as a source (DMS)
- AWS: AWS Database Migration Service FAQs
- AWS docs: Set up replication for AWS DMS