Migrating large, multi-terabyte MySQL or MariaDB databases to AWS

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Migrating large, multi-terabyte MySQL or MariaDB databases to AWS

Table of Contents

Migrating large, multi-terabyte MySQL or MariaDB databases to AWS

Babaiah Valluru and Ankur Bhanawat, Amazon Web Services

Many organizations that have on-premises MySQL and MariaDB database servers are interested in migrating their database workloads to the AWS Cloud. Many choose Amazon Relational Database Service (Amazon RDS) for MariaDB, Amazon RDS for MySQL, or Amazon Aurora MySQL-Compatible Edition. Amazon RDS is designed to make it easy to set up, operate, and scale relational databases in the cloud. Amazon Aurora is part of Amazon RDS, and it offers built-in security, continuous backups, server-free computing, up to 15 read replicas, automated multi-Region replication, and integration with other AWS services.

Although migrating to one of these AWS services can provide many benefits, database migration is one of the most time-consuming and critical tasks that database administrators must perform. It requires precise planning and implementation to migrate large databases and make sure that the performance of the migrated workload is equivalent or improved. In this guide, *large* databases can refer to a single, multi-terabyte database or refer to many large databases that add up to multiple terabytes of data. Selecting the right migration services and tools is key to the success of the migration. There are two common approaches for migrating a database: logical and physical. For more information about these approaches, see the MySQL and MariaDB documentation.

This guide discusses various open-source or third-party tools that you can use to migrate large, on-premises, multi-terabyte MySQL and MariaDB databases to Amazon RDS for MariaDB, Amazon RDS for MySQL, or Amazon Aurora MySQL-Compatible Edition. The options discussed in this guide use logical or physical migration approaches, and each option includes multiple approaches for transferring the large database backup files from the on-premises data center to the cloud, where you can restore the database from the backup file.

Intended audience

This guide is for program database administrators, database engineers, migration engineers, project managers, and operations or infrastructure managers who are planning to migrate their MySQL or MariaDB databases to the AWS Cloud.

Targeted business outcomes

The goal of this guide is to help you:

- Choose a migration approach for a large database that best fits your use case and environment.
- Avoid delays and financial losses that can occur when the migration strategy is flawed.
- Learn about the advantages and limitations of each migration option.
- Learn about different approaches you can use to transfer large database backup files from your on-premises data center to the AWS Cloud.
- Review overall best practices for migrating large databases and also review best practices for each tool, which can help you more efficiently migrate the database.

Migration options for large MySQL and MariaDB databases

You can choose from an extensive range of options to migrate from on-premises MySQL or MariaDB databases to Amazon Relational Database Service (Amazon RDS) or Amazon Aurora MySQL-Compatible Edition databases instances. Choosing the right migration approach and tool is essential for a successful migration, and in this guide, you evaluate the options based on your usability, data size, and downtime requirements.

The following are the common migration tools and approaches that are available to migrate multi-terabyte self-managed MySQL databases efficiently to Amazon RDS, Aurora, or Amazon Elastic Compute Cloud (Amazon EC2) database instances:

- Percona XtraBackup (Physical)
- MyDumper (Logical)
- mysqldump and mysqlpump (Logical)
- Split backup (Physical, logical, or both)

The following are the common migration tools and approaches that are available to migrate multi-terabyte MySQL-compatible (such as MariaDB) databases efficiently to Amazon RDS, Aurora, or Amazon EC2 database instances:

- MyDumper (Logical)
- mysqldump and mysqlpump (Logical)
- Split backup (Physical, logical, or both)

For each migration tool, there are several approaches you can use to transfer the large database backup file to the AWS CloudOptions are provided for each tool, and you can also use Amazon S3 File Gateway. For more information, see Amazon S3 File Gateway in this guide.

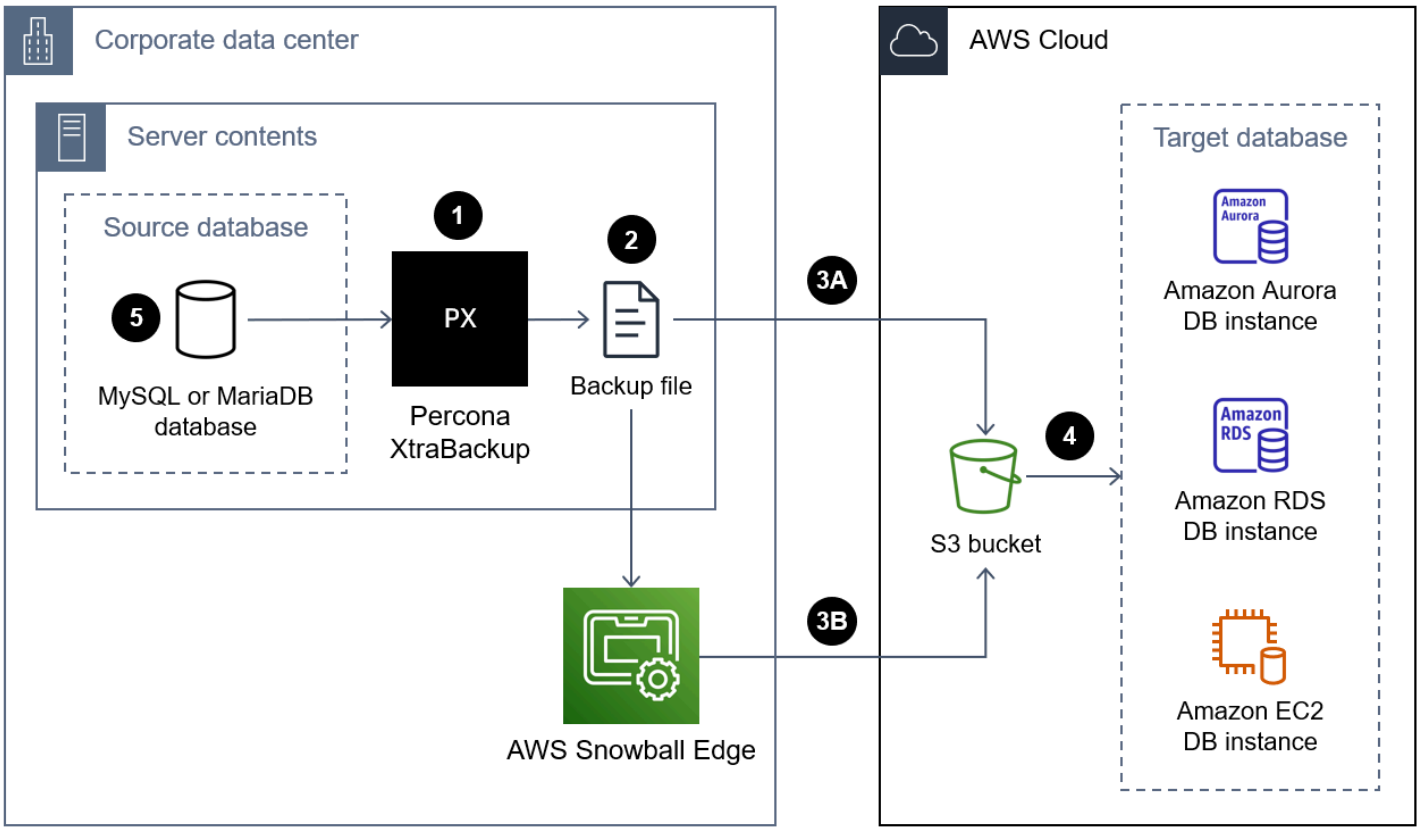
Percona XtraBackup

> **Important**
>
> Percona XtraBackup is not supported for MariaDB versions 10.3 or later and is only partially supported for versions 10.1 and 10.2.

Percona XtraBackup is a common open-source warm backup software for MySQL and MariaDB that makes non-blocking backups for InnoDB and XtraDB storage engines. It works with MySQL or MariaDB servers. For more information about the tool and some of its features and benefits, see About Percona XtraBackup in the Percona XtraBackup documentation.

This tool uses the physical migration approach. It directly copies the MySQL or MariaDB data directory and the files within it. For large databases, such as those larger than 100 GB, this can provide a significantly better restoration time than some other tools. You create a backup of the on-premises source database, migrate the backup files to the cloud, and then restore the backup on the new, target database instance.

The following diagram shows the high-level steps involved in migrating a database by using an Percona XtraBackup backup file. Depending on the size of the backup file, there are two options available for transferring the backup to an Amazon Simple Storage Service (Amazon S3) bucket in the AWS Cloud.



The following are the steps for using Percona XtraBackup to migrate a database to the AWS Cloud:

1. Install Percona XtraBackup on the on-premises server. If you're using Amazon Aurora MySQL version 2 or Amazon RDS, see Installing Percona XtraBackup 2.4. If you're using Amazon Aurora MySQL version 3, see Installing Percona XtraBackup 8.0 in the Percona XtraBackup documentation.
2. Create a full backup of the source MySQL or MariaDB database. For instructions for Percona XtraBackup 2.4, see Full backup. For instructions for Percona XtraBackup 8.0, see Create a full backup.
3. Transfer the backup files over the internet by using an approved service or tool in your organization, such as the following:
   - AWS Site-to-Site VPN
   - AWS Client VPN
   - AWS Direct Connect
   - Amazon S3 File Gateway (For more information, see Amazon S3 File Gateway in this guide.)
   - AWS Command Line Interface (AWS CLI)

4. From the Amazon S3 bucket, restore the backup files to the target database instance. For instructions, see the following:
   - For Aurora MySQL-Compatible Edition, see Migrating data from MySQL by using an Amazon S3 bucket in the Amazon RDS documentation.
   - For Amazon RDS for MySQL or for Amazon EC2, see Importing data into a MySQL DB instance.
   - For Amazon RDS for MariaDB or for Amazon EC2, see Importing data into a MariaDB DB instance.
5. (Optional) You can set up replication between the source database and the target database instance. You can use binary log (binlog) replication to reduce downtime. For more information, see the following:
   - Setting the replication source configuration in the MySQL documentation
   - For Amazon Aurora, see the following:
     - Synchronizing the Amazon Aurora MySQL DB cluster with the MySQL database using replication in the Aurora documentation
     - Using binlog replication in Amazon Aurora in the Aurora documentation
   - For Amazon RDS, see the following:
     - Working with MySQL replication in the Amazon RDS documentation
     - Working with MariaDB replication in the Amazon RDS documentation
   - For Amazon EC2, see the following:
     - Setting Up Binary Log File Position Based Replication in the MySQL documentation
     - Setting Up Replicas in the MySQL documentation
     - Setting Up Replication in the MariaDB documentation

Advantages

- Because Percona XtraBackup uses a physical migration approach, the restore process is typically faster than tools that use a logical migration approach. This is because the performance is limited by the disk or network throughput rather than the compute resources necessary for data processing.
- Because the restore process is a direct copy of the files from the S3 bucket to the target database instance, Percona XtraBackup files typically restore faster than backup files created with other tools.

- Percona XtraBackup is adaptable. For example, it supports multiple threads to help you copy files faster and supports compression to reduce the size of the backup.

Limitations

- Offline backup is not possible because Percona XtraBackup must have access to the source database server.
- Percona XtraBackup can be used only on systems with identical system architectures. For example, it is not possible to restore a backup of a source database running on Intel for Windows Server to an ARM for Linux target server.
- Percona XtraBackup isn't supported for MariaDB version 10.3 or later, and it is only partially supported for MariaDB version 10.2 and version 10.1. For more information, see Percona XtraBackup Overview: Compatibility with MariaDB in the MariaDB knowledge base.
- You cannot use Percona XtraBackup to restore a source MariaDB database to a target MySQL database instance, such as Amazon RDS for MySQL or Aurora MySQL-Compatible.
- The total volume of data and number of objects you can store in an S3 bucket are unlimited, however, the maximum file size is 5 TB. If your backup file exceeds 5 TB, you can split it up into multiple, smaller files.
- When the `innodb_file_per_table` setting is off, Percona XtraBackup doesn't support partial backups that use `--tables`, `--tables-exclude`, `--tables-file`, `--databases`, `--databases-exclude`, or `--databases-file`. For more information for Percona XtraBackup version 2.4, see Partial backups. For more information for Percona XtraBackup version 8.0, see Create a partial backup.

Best practices

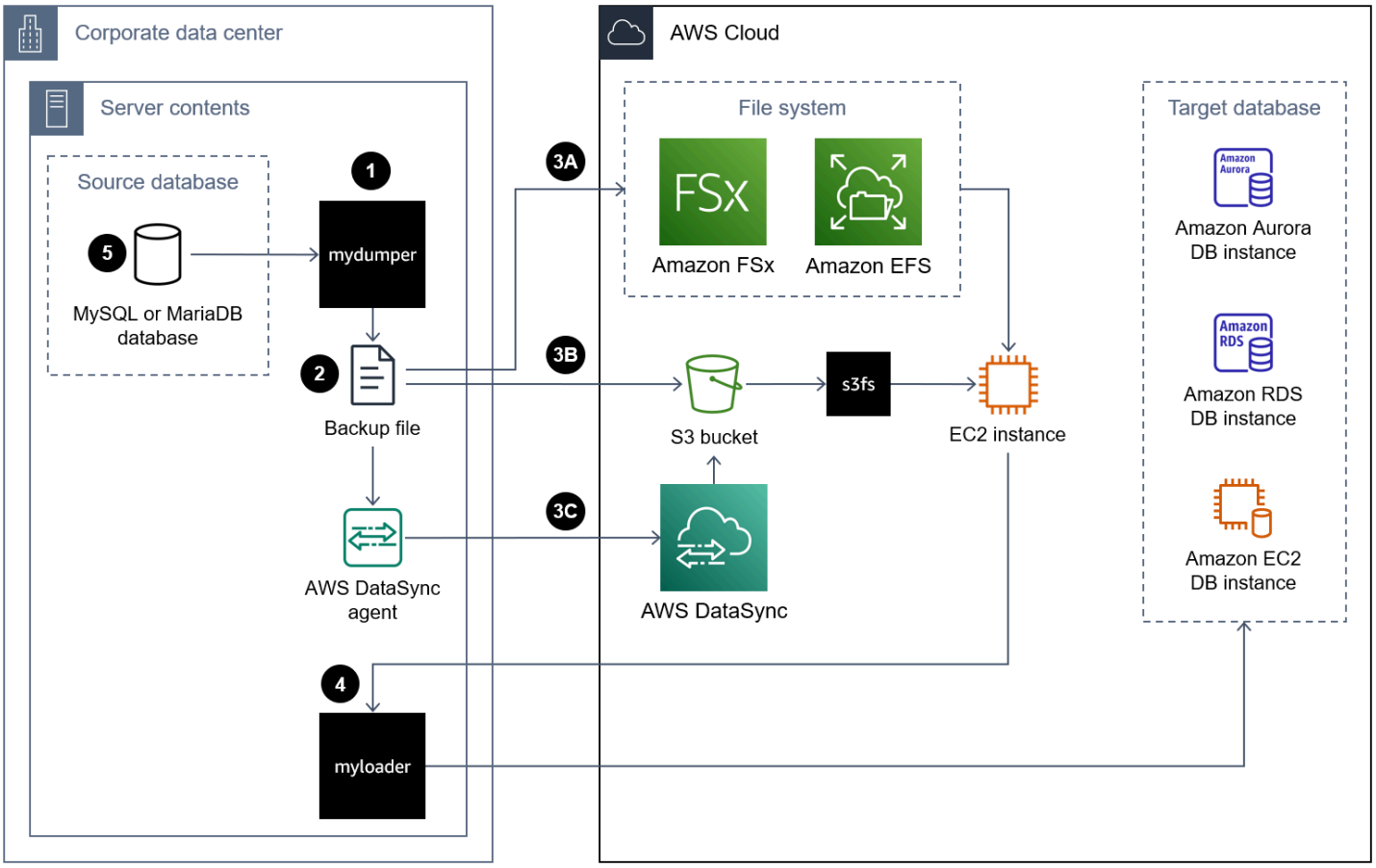
- To improve the performance of the backup process, do the following:
  - Copy multiple files in parallel by using --parallel=<threads>
  - Compress multiple files in parallel by using --compress-threads=<threads>
  - Increase memory by using --use-memory=<size>
  - Encrypt multiple files in parallel by using --encrypt-threads=<threads>
- Ensure that there is sufficient space on the source server to take the database backup files.
- Generate the database backup with the Percona xbstream (.xbstream) format file. For more information, see The xbstream binary overview in the Percona XtraBackup documentation.

MyDumper

MyDumper (GitHub) is an open-source, logical migration tool that consists of two utilities:

- MyDumper exports a consistent backup of MySQL databases. It supports backing up the database by using multiple parallel threads, up to one thread per available CPU core.
- myloader reads the backup files created by MyDumper, connects to the target database instance, and then restores the database.

The following diagram shows the high-level steps involved in migrating a database by using a MyDumper backup file. This architecture diagram includes three options for migrating the backup file from the on-premises data center to an Amazon EC2 instance in the AWS Cloud.



The following are the steps for using MyDumper to migrate a database to the AWS Cloud:

1. Install MyDumper and myloader. For instructions, see How to install mydumper/myloader (GitHub).

2. Use MyDumper to create a backup of the source MySQL or MariaDB database. For instructions, see How to use MyDumper.
3. Move the backup file to an EC2 instance in the AWS Cloud by using one of the following approaches:

   Approach 3A – Mount an Amazon FSx or Amazon Elastic File System (Amazon EFS) file system to the on-premises server that runs your database instance. You can use AWS Direct Connect or Site-to-Site VPN to establish the connection. You can directly back up the database to the mounted file share, or you can perform the backup in two steps by backing up the database to a local file system and then uploading it to the mounted FSx or EFS volume. Next, mount the Amazon FSx or Amazon EFS file system, which is also mounted on the on-premises server, on an EC2 instance.

   Approach 3B – Use the AWS CLI, AWS SDK, or Amazon S3 REST API to directly move the backup file from the on-premises server to an S3 bucket. If the target S3 bucket is in an AWS Region that is far away from the data center, you can use Amazon S3 Transfer Acceleration to transfer the file more quickly. Use the s3fs-fuse file system to mount the S3 bucket on the EC2 instance.

   Approach 3C – Install the AWS DataSync agent at the on-premises data center, and then use AWS DataSync to move the backup file to an Amazon S3 bucket. Use the s3fs-fuse file system to mount the S3 bucket on the EC2 instance.

   Note: You can also use Amazon S3 File Gateway to transfer the large database backup files to an S3 bucket in the AWS Cloud. For more information, see Amazon S3 File Gateway in this guide.

4. Use myloader to restore the backup on the target database instance. For instructions, see myloader usage (GitHub).
5. (Optional) You can set up replication between the source database and the target database instance. You can use binary log (binlog) replication to reduce downtime. For more information, see the following:
   - Setting the replication source configuration in the MySQL documentation
   - For Amazon Aurora, see the following:
     - Synchronizing the Amazon Aurora MySQL DB cluster with the MySQL database using replication in the Aurora documentation
     - Using binlog replication in Amazon Aurora in the Aurora documentation

- For Amazon RDS, see the following:
  - Working with MySQL replication in the Amazon RDS documentation
  - Working with MariaDB replication in the Amazon RDS documentation
- For Amazon EC2, see the following:
  - Setting Up Binary Log File Position Based Replication in the MySQL documentation
  - Setting Up Replicas in the MySQL documentation
  - Setting Up Replication in the MariaDB documentation

Advantages

- MyDumper supports parallelism by using multi-threading, which improves the speed of backup and restore operations.
- MyDumper avoids expensive character set conversion routines, which helps ensure the code is highly efficient.
- MyDumper simplifies the data view and parsing by using dumping separate files for tables and metadata.
- MyDumper maintains snapshots across all threads and provides accurate positions of primary and secondary logs.
- You can use Perl Compatible Regular Expressions (PCRE) to specify whether to include or exclude tables or databases.

Limitations

- You might choose a different tool if your data transformation processes require intermediate dump files in flat format instead of SQL format.
- myloader doesn't import database user accounts automatically. If you are restoring the backup to Amazon RDS or Aurora, recreate the users with the required permissions. For more information, see Master user account privileges in the Amazon RDS documentation. If you are restoring the backup to an Amazon EC2 database instance, you can manually export the source database user accounts and import them into the EC2 instance.

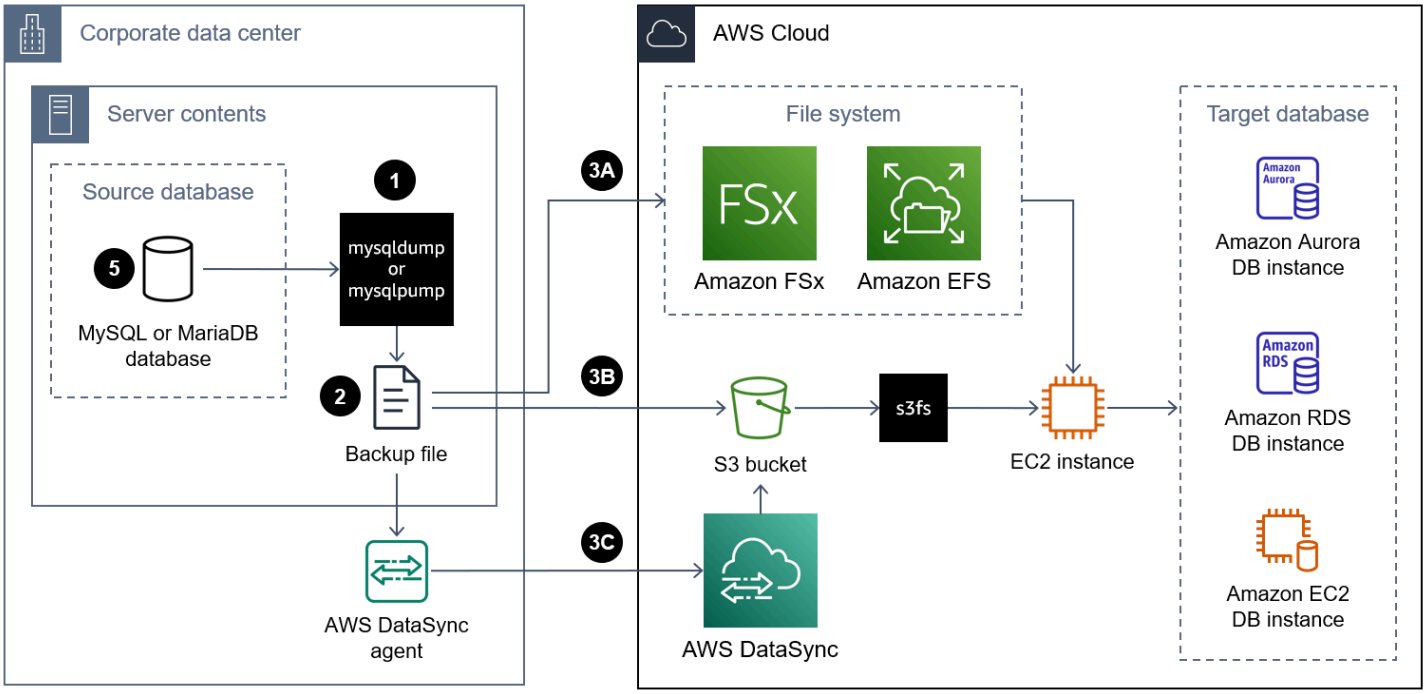
Best practices

- Configure MyDumper to divide each table into segments, such as 10,000 rows in each segment, and write each segment in a separate file. This makes it possible to import the data in parallel later.
- If you are using the InnoDB engine, use the `--trx-consistency-only` option to minimize locking.
- Using MyDumper to export the database can become read-intensive, and the process can impact overall performance of the production database. If you have a replica database instance, run the export process from the replica. Before you run the export from the replica, stop the replication SQL thread. This helps the export process run more quickly.
- Don't export the database during peak business hours. Avoiding peak hours can stabilize the performance of your primary production database during the database export.
- Amazon RDS for MySQL doesn't support the `keyring_aws` plugin. For more information, see Known issues and limitations. To migrate the on-premises encrypted tables to the Amazon RDS instance, in the backup scripts, you need to remove `ENCRYPTION` or `DEFAULT ENCRYPTION` from the `CREATE TABLE` syntax. For encryption at rest, you can use an AWS Key Management Service (AWS KMS) key. For more information, see Encrypting Amazon RDS resources.

mysqldump and mysqlpump

mysqldump and mysqlpump are native database backup tools for MySQL. MariaDB supports mysqldump but doesn't support mysqlpump. Both of these tools create logical backups and are part of the MySQL client programs. mysqldump supports single-threaded processing. mysqlpump supports parallel processing of databases and objects within databases, to speed up the dump process. It was introduced in MySQL version 5.7.8. mysqlpump was removed in MySQL version 8.4.

The following diagram shows the high-level steps involved in migrating a database by using a mysqldump or mysqlpump backup file.



The following are the steps for using mysqldump or mysqlpump to migrate a database to the AWS Cloud:

1. Install MySQL Shell on the on-premises server. For instructions, see Installing MySQL Shell in the MySQL documentation. This installs both mysqldump and mysqlpump.
2. Using mysqldump or mysqlpump, create a backup of the source, on-premises database. For instructions, see mysqldump and mysqlpump in the MySQL documentation, or see Making Backups with mysqldump in the MariaDB documentation. For more information about invoking MySQL programs and specifying options, see Using MySQL programs.
3. Move the backup file to an Amazon EC2 instance in the AWS Cloud by using one of the following approaches:

   Approach 3A – Mount an Amazon FSx or Amazon Elastic File System (Amazon EFS) file system to the on-premises server that runs your database instance. You can use AWS Direct Connect or Site-to-Site VPN to establish the connection. You can directly back up the database to the mounted file share, or you can perform the backup in two steps by backing up the database to a local file system and then uploading it to the mounted FSx or EFS volume. Next, mount the Amazon FSx or Amazon EFS file system, which is also mounted on the on-premises server, on an EC2 instance.

Approach 3B – Use the AWS CLI, AWS SDK, or Amazon S3 REST API to directly move the backup file from the on-premises server to an S3 bucket. If the target S3 bucket is in an AWS Region that is far away from the data center, you can use Amazon S3 Transfer Acceleration to transfer the file more quickly. Use the s3fs-fuse file system to mount the S3 bucket on the EC2 instance.

Approach 3C – Install the AWS DataSync agent at the on-premises data center, and then use AWS DataSync to move the backup file to an Amazon S3 bucket. Use the s3fs-fuse file system to mount the S3 bucket on the EC2 instance.

Note: You can also use Amazon S3 File Gateway to transfer the large database backup files to an S3 bucket in the AWS Cloud. For more information, see Amazon S3 File Gateway in this guide.

4. Use the native restore method to restore the backup on the target database. For instructions, see Reloading SQL-Format Backups in the MySQL documentation, or see Restoring Data from Dump Files in the MariaDB documentation.
5. (Optional) You can set up replication between the source database and the target database instance. You can use binary log (binlog) replication to reduce downtime. For more information, see the following:
   - Setting the replication source configuration in the MySQL documentation
   - For Amazon Aurora, see the following:
     - Synchronizing the Amazon Aurora MySQL DB cluster with the MySQL database using replication in the Aurora documentation
     - Using binlog replication in Amazon Aurora in the Aurora documentation
   - For Amazon RDS, see the following:
     - Working with MySQL replication in the Amazon RDS documentation
     - Working with MariaDB replication in the Amazon RDS documentation
   - For Amazon EC2, see the following:
     - Setting Up Binary Log File Position Based Replication in the MySQL documentation
     - Setting Up Replicas in the MySQL documentation
     - Setting Up Replication in the MariaDB documentation

Advantages

- mysqldump and mysqlpump are included in the MySQL Server installation
- The backup files generated by these tools are in a more readable format.
- Before restoring the backup file, you can modify the resultant .sql file by using a standard text editor.
- You can back up a specific table, database, or even a particular data selection.
- mysqldump and mysqlpump are machine-architecture independent.

Limitations

- mysqldump is a single-threaded backup process. Performance for taking a backup is good for small databases, but it can become inefficient when the backup size is larger than 10 GB.
- Backup files in logical format are voluminous, especially when saved as text, and often slow to create and restore.
- Data restoration can be slow because reapplying SQL statements in the target DB instance involves intensive disk I/O and CPU processing for insertion, index creation, and referential integrity constraints enforcement.
- The mysqlpump utility is not supported for MySQL versions earlier than 5.7.8 or for versions 8.4 and later.
- By default, mysqlpump does not take a backup of the system databases, such as `performance_schema` or `sys`. To backup part of the system database, explicitly name it in the command line.
- mysqldump does not backup InnoDB `CREATE TABLESPACE` statements.

Note: Backups of `CREATE TABLESPACE` statements and system databases are useful only when you are restoring MySQL or MariaDB database backups to an EC2 instance. These backups are not used for Amazon RDS or Aurora.

Best practices

- When you're restoring the database backup, disable the key checks, such as `FOREIGN_KEY_CHECKS`, at the session level in the target database . This increases the restoration speed.
- Make sure the database user has sufficient privileges to create and restore the backup.

Split backup

A *split backup* strategy is when you migrate a large database server by dividing the backup into multiple parts. You might use different approaches to migrate each part of the backup. This can be the best option for the following use cases:

- **Large database server but small individual databases** – This is a good approach when the size of the total database server is multiple TBs but the size of each individual, independent user database is less than 1 TB. To reduce the overall migration period, you can migrate individual database separately and in parallel.

  Let's use an example of an on-premises, 2 TB database server. This server consists of four databases that are each 0.5 TB. You can take backups of each individual database separately. When restoring the backup, you can either restore all databases on an instance in parallel, or if the databases are independent, you can restore each backup on a separate instance. It's a best practice to restore independent databases on separate instances, instead of restoring them on the same instance. For more information, see Best practices in this guide.
- **Large database server but small individual database tables** – This is a good approach when the size of the total database server is multiple TBs but the size of each independent database table is less than 1 TB. To reduce the overall migration period, you can migrate independent tables individually.

  Let's use an example of a single user database that is 1 TB, and it is the only database in an on-premises database server. There are 10 tables in the database, and each is 100 GB. You can take backups of each individual table separately. When restoring the backup, you can restore all tables on an instance in parallel.
- **A database contains both transactional and non-transactional workload tables** – Similar to the previous use case, you can use a split backup approach when you have both transactional and non-transactional workload tables in the same database.

Let's use an example of a 2 TB database that consists of 0.5 TB of critical workload tables used for online transaction processing (OLTP) and a single 1.5 TB table used for archiving old data. You can take the backup of all database objects except the archive table as a single-transaction and consistent backup. Then, you take another, separate backup of the archive table only. For the archive table backup, you can also consider taking multiple, parallel backups by using conditions to split the number of rows in the backup file. The following is an example:

```
mysqldump -p your_db1 --tables your_table1 --where="column1 between 1 and 1000000 " >
 your_table1_part1.sql
mysqldump -p your_db1 --tables your_table1 --where="column1 between 1000001 and
 2000000 " > your_table1_part2.sql
mysqldump -p your_db1 --tables your_table1 --where="column1 > 2000000 " >
 your_table1_part3.sql
```

When restoring the backup files, you can restore transactional workload backup and the archive table backup in parallel.

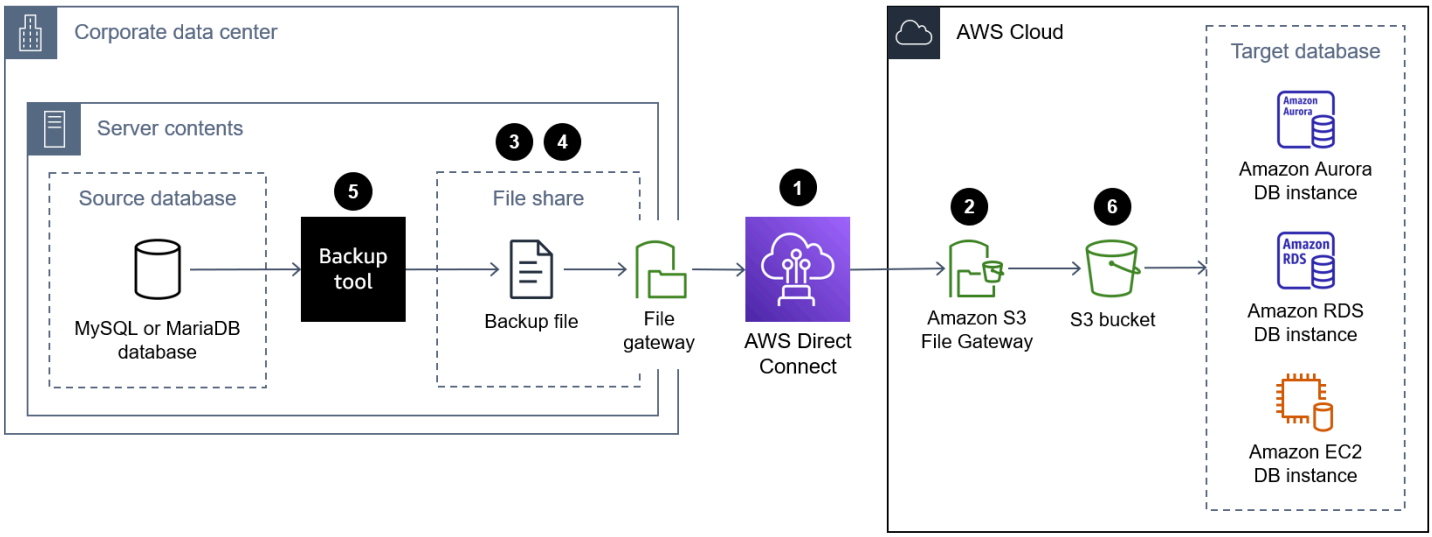
- **Compute resource limitations** – If you have limited compute resources in the on-premises server, such as CPU, memory, or disk I/O, this can affect stability and performance when taking the backup. Instead of taking a complete backup, you can divide it into parts.

  For example, an on-premises production server might be heavily loaded with workloads and have limited CPU resources. If you take a single-run backup of a multi-terabyte database on this server, it can consume additional CPU resources and adversely affect the production server. Instead of taking the complete database backup, divide the backup into multiple parts, such as 2–3 tables each.

Using Amazon S3 File Gateway to transfer backup files

Amazon S3 File Gateway connects your on-premises environment to Amazon Simple Storage Service (Amazon S3) through a file interface so that you can store and retrieve Amazon S3 objects by using industry-standard file protocols, such as Network File System (NFS) and Server Message Block (SMB). It is designed to be a cost-effective, scalable solution for storing data in the cloud. Because you can use it to store database backup files, this service can help you migrate large, on-premises databases to the AWS Cloud. For example, you could use Amazon S3 File Gateway and your preferred database backup tool to back up the large MySQL or MariaDB database directly to an Amazon S3 bucket. You can then mount the S3 bucket to the target instance and restore the backup.

The following diagram shows the high-level steps involved when using Amazon S3 File Gateway to transfer the backup file for an on-premises database to an S3 bucket in the AWS Cloud.



The following are the steps for using Amazon S3 File Gateway to transfer a database backup file from an on-premises data center to an S3 bucket in the AWS Cloud:

1. Connect the on-premises data center to the AWS Cloud by using a service such as AWS Direct Connect or AWS Site-to-Site VPN or by using a public internet connection.
2. Create an S3 File Gateway. For instructions, see Creating your gateway.
3. Create an NFS or SMB file share that is hosted by the S3 File Gateway. For instructions, see Create a file share.
4. Mount the NFS or SMB file share on the on-premises server that hosts your MySQL or MariaDB database. For instructions, see Mount and use your file share.

5. Back up the on-premises MySQL or MariaDB database to the directory where the NFS file share is mounted. You can use any of the backup tools discussed in this guide.
6. Restore the database backup on the target database instance by using any of the approaches discussed in this guide.

Advantages

- By producing database backups directly in the S3 bucket and restoring the backup on the target DB instance directly from the same S3 bucket, you can significantly accelerate the end-to-end migration process.
- Database backup files are stored durably in Amazon S3, and you choose the lifecycle management policy and S3 storage class.

Limitations

The following are limitations when using Amazon S3 File Gateway file shares:

- The maximum number of file shares per gateway is 50.
- To prevent read and write conflicts when multiple file shares use the same S3 bucket, you must configure each file share to use a unique prefix name.
- The maximum size of an individual file is 5 TB, which is the maximum size of any individual object in Amazon S3.
- The maximum path length is 1024 characters.
- Windows ACLs are supported only on file shares that are enabled for Active Directory when you use Windows SMB clients to access the file shares.
- Amazon S3 File Gateway supports a maximum of 10 ACL entries for each file and directory.
- The root ACL settings of SMB file shares are only on the gateway. These settings are persistent across gateway updates and restarts.

> Note: If you configure the ACLs on the root instead of the parent folder under the root, the ACL permissions aren't persistent in Amazon S3.

Best practices

For more information about the best practices for Amazon S3 File Gateway, see Best practices in the S3 File Gateway documentation.

Best practices for migrating large MySQL and MariaDB databases

In addition to the tool-specific best practices listed for each migration option, review the following general best practices. These best practices apply when migrating large, multi-terabyte MySQL and MariaDB databases, regardless of the tool you select:

- Make sure that there is sufficient space on the source and destination databases to take and restore the backup.
- Don't create secondary indexes on the target database instance until the migration is complete. Secondary indexes add additional maintenance overhead during import and can slow down the import process.
- If you use a multi-threaded approach, choose the right number of threads. For export, we recommend you use one thread for each CPU core. For import, we recommend you use one thread for every two CPU cores.
- Data dumps are often performed from active database servers that are part of a mission-critical production environment. If the data dump severely affects performance and this isn't acceptable in your environment, consider one of the following:
  - The source server has replicas, you can dump data from one of the replicas.
  - The source server is covered by regular backup procedures:
    - If the backup format is suitable for direct import into the target database, use the backup data as the input for the import process.
    - If the backup format isn't suitable for direct import into the target database, use the backup to provision a temporary database and dump data from it.
  - If replicas and backups aren't available:
    - Perform dumps during off-peak hours, when production traffic is at its lowest.
    - Reduce the concurrency of dump operations so that the server has enough spare capacity to handle production traffic.
- Create dumps of user-created databases only.
- Re-create the users on the target database and configure their permissions. For more information, see Identity and access management for Amazon RDS, Identity and access management for Amazon Aurora, or Identity and access management for Amazon EC2.

- When migrating a large database server that consists of multiple, independent databases, create a separate instance for each database. This helps you manage the database more efficiently and can improve resource provisioning, and the separate compute resources can improve database performance.

Resources

AWS Prescriptive Guidance

- Portfolio playbook for AWS large migrations
- Migrate an on-premises MySQL database to Amazon RDS for MySQL
- Set up data replication between Amazon RDS for MySQL and MySQL on Amazon EC2 using GTID
- Migrate an on-premises MariaDB database to Amazon RDS for MariaDB using native tools

AWS blog posts

- Security best practices for Amazon RDS for MySQL and MariaDB instances
- Migrate self-managed MariaDB to Amazon Aurora MySQL

Resources for restoring the backup

- Creating a bucket (Amazon S3 documentation)
- Connecting to your Linux instance using SSH (Amazon EC2 documentation)
- Configuring the AWS CLI (AWS CLI documentation)
- sync command (AWS CLI Command Reference)
- Creating an IAM policy to access Amazon S3 resources (Aurora documentation)
- DB cluster prerequisites (Aurora documentation)
- Working with DB subnet groups (Aurora documentation)
- Creating a VPC security group for a private DB cluster (Aurora documentation)
- Restoring an Aurora MySQL DB cluster from an Amazon S3 bucket (Aurora documentation)
- Setting up replication with MySQL or another Aurora DB cluster (Aurora documentation)
- rds_set_external_master procedure (Amazon RDS documentation)
- rds_start_replication procedure (Amazon RDS documentation)

AWS marketing

- Amazon Aurora
- Amazon RDS for MariaDB
- Amazon RDS for MySQL
- Amazon S3 File Gateway

Other resources

- Percona XtraBackup
- MyDumper
- mysqldump
- mysqlpump

Document history

The following table describes significant changes to this guide.

Change	Description	Date
mysqlpump availability	mysqlpump was removed in MySQL version 8.4. We updated the mysqldump and mysqlpump section to reflect this availability change.	November 21, 2024
MyDumper best practices	We updated the best practices for MyDumper to add information about migrating encrypted database tables.	October 24, 2024
Percona XtraBackup versions	In the Percona XtraBackup section, we updated the instructions to reflect the versions of Percona XtraBackup supported by Amazon Aurora MySQL and Amazon RDS.	August 3, 2023
Initial publication	—	April 6, 2023