

Amazon RDS for MySQL

Amazon RDS supports several versions of MySQL for DB instances. For complete information about the supported versions, see [MySQL on Amazon RDS versions](#).

To create an Amazon RDS for MySQL DB instance, use the Amazon RDS management tools or interfaces. You can then do the following:

- Resize your DB instance
- Authorize connections to your DB instance
- Create and restore from backups or snapshots
- Create Multi-AZ secondaries
- Create read replicas
- Monitor the performance of your DB instance

To store and access the data in your DB instance, you use standard MySQL utilities and applications.

Amazon RDS for MySQL is compliant with many industry standards. For example, you can use RDS for MySQL databases to build HIPAA-compliant applications. You can use RDS for MySQL databases to store healthcare related information, including protected health information (PHI) under a Business Associate Agreement (BAA) with AWS. Amazon RDS for MySQL also meets Federal Risk and Authorization Management Program (FedRAMP) security requirements. In addition, Amazon RDS for MySQL has received a FedRAMP Joint Authorization Board (JAB) Provisional Authority to Operate (P-ATO) at the FedRAMP HIGH Baseline within the AWS GovCloud (US) Regions. For more information on supported compliance standards, see [AWS cloud compliance](#).

For information about the features in each version of MySQL, see [The main features of MySQL](#) in the MySQL documentation.

Before creating a DB instance, complete the steps in [Setting up your Amazon RDS environment](#). When you create a DB instance, the RDS master user gets DBA privileges, with some limitations. Use this account for administrative tasks such as creating additional database accounts.

You can create the following:

- DB instances
- DB snapshots

- Point-in-time restores
- Automated backups
- Manual backups

You can use DB instances running MySQL inside a virtual private cloud (VPC) based on Amazon VPC. You can also add features to your MySQL DB instance by turning on various options. Amazon RDS supports Multi-AZ deployments for MySQL as a high-availability, failover solution.

Important

To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that need advanced privileges. You can access your database using standard SQL clients such as the mysql client. However, you can't access the host directly by using Telnet or Secure Shell (SSH).

Topics

- [MySQL feature support on Amazon RDS](#)
- [MySQL on Amazon RDS versions](#)
- [Connecting to your MySQL DB instance](#)
- [Securing MySQL DB instance connections](#)
- [Improving query performance for RDS for MySQL with Amazon RDS Optimized Reads](#)
- [Improving write performance with RDS Optimized Writes for MySQL](#)
- [Upgrades of the RDS for MySQL DB engine](#)
- [Upgrading a MySQL DB snapshot engine version](#)
- [Importing data into an Amazon RDS for MySQL DB instance](#)
- [Working with MySQL replication in Amazon RDS](#)
- [Configuring active-active clusters for RDS for MySQL](#)
- [Exporting data from a MySQL DB instance by using replication](#)
- [Options for MySQL DB instances](#)
- [Parameters for MySQL](#)
- [Common DBA tasks for MySQL DB instances](#)

- [Local time zone for MySQL DB instances](#)
- [Known issues and limitations for Amazon RDS for MySQL](#)
- [RDS for MySQL stored procedure reference](#)

MySQL feature support on Amazon RDS

RDS for MySQL supports most of the features and capabilities of MySQL. Some features might have limited support or restricted privileges.

You can filter new Amazon RDS features on the [What's New with Database?](#) page. For **Products**, choose **Amazon RDS**. Then search using keywords such as **MySQL 2022**.



The following lists are not exhaustive.

Topics

- [MySQL feature support on Amazon RDS for MySQL major versions](#)
- [Supported storage engines for RDS for MySQL](#)
- [Using memcached and other options with MySQL on Amazon RDS](#)
- [InnoDB cache warming for MySQL on Amazon RDS](#)
- [Inclusive language changes for RDS for MySQL 8.4](#)
- [MySQL features not supported by Amazon RDS](#)

MySQL feature support on Amazon RDS for MySQL major versions

In the following sections, find information about MySQL feature support on Amazon RDS for MySQL major versions:

Topics

- [MySQL 8.4 support on Amazon RDS](#)

For information about supported minor versions of Amazon RDS for MySQL, see [Supported MySQL minor versions on Amazon RDS](#).

MySQL 8.4 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MySQL version 8.4 or higher.

- **Cryptographic library** – RDS for MySQL replaced OpenSSL with AWS Libcrypto (AWS-LC), which is FIPS 140-3 certified. For more information, see the AWS-LC GitHub repository at <https://github.com/aws/aws-lc>.
- **TLS changes** – RDS for MySQL only supports TLS 1.2 and TLS 1.3. For more information, see [the section called "SSL/TLS support with MySQL"](#).
- **memcached support** – The memcached interface is no longer available on MySQL 8.4. For more information, see [MySQL memcached support](#).
- **Default authentication plugin** – The default authentication plugin is `caching_sha2_password`. For more information, see [the section called "MySQL default authentication plugin"](#).
- **mysqlpump client utility** – The mysqlpump client utility is no longer available in MySQL 8.4. For more information, see [Role-based privilege model for RDS for MySQL](#) and [mysqldump and mysqlpump](#) in the *AWS Prescriptive Guidance*.
- **Managed replication stored procedures** – When using stored procedures to manage replication with a replication user configured with `caching_sha2_password`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4.
- **Parameter behavior changes** – The following parameters changed for MySQL 8.4.
 - `innodb_dedicated_server` – This parameter is now enabled by default. For more information, see [Configuring buffer pool size and redo log capacity in MySQL 8.4](#).
 - `innodb_buffer_pool` – The database engine now calculates this parameter, but you can override this setting. For more information, see [Configuring buffer pool size and redo log capacity in MySQL 8.4](#).
 - `innodb_redo_log_capacity` – This parameter now controls the size of the redo log files. The database engine now calculates this parameter, but you can override this setting. For more information, see [Configuring buffer pool size and redo log capacity in MySQL 8.4](#).
- **Deprecated or removed parameters** – RDS for MySQL removed the following parameters from parameter groups for MySQL 8.4 DB instances. The `innodb_redo_log_capacity` parameter now controls the size of the redo log files.
 - `innodb_log_file_size`
 - `innodb_log_files_in_group`
- **New default values for parameters** – The following parameters have new default values for MySQL 8.4 DB instances:

- Various MySQL community parameters related to performance changed. For more information, see [What is New in MySQL 8.4 since MySQL 8.0](#).

We recommend that you test your application's performance on RDS for MySQL 8.4 before migrating a production instance.

- `innodb_purge_threads` – The default value is set to the formula `LEAST({DBInstanceVCPU/2}, 4)` to prevent the InnoDB history list length from growing too large.
- `group_replication_exit_state_action` – The default value is `OFFLINE_MODE`, which aligns with the default in MySQL Community. For more information, see [Considerations and best practices for RDS for MySQL active-active clusters](#).
- `binlog_format` – The default value is `ROW`, which aligns with the default in MySQL Community. You can modify the parameter for Single-AZ DB instances or Multi-AZ DB instances, but not for Multi-AZ DB clusters. Multi-AZ DB clusters use semisynchronous replication, and when `binlog_format` is set to `MIXED` or `STATEMENT`, replication fails.
- **Inclusive language changes** – RDS for MySQL 8.4 includes changes from RDS for MySQL 8.0 related to keywords and system schemas for inclusive language. For more information, see [Inclusive language changes for RDS for MySQL 8.4](#).

For a list of all MySQL 8.4 features and changes, see [What Is New in MySQL 8.4 since MySQL 8.0](#) in the MySQL documentation.

For a list of unsupported features, see [the section called “Features not supported”](#).

Supported storage engines for RDS for MySQL

While MySQL supports multiple storage engines with varying capabilities, not all of them are optimized for recovery and data durability. Amazon RDS fully supports the InnoDB storage engine for MySQL DB instances. Amazon RDS features such as Point-In-Time restore and snapshot restore require a recoverable storage engine and are supported for the InnoDB storage engine only. For more information, see [MySQL memcached support](#).

The Federated Storage Engine is currently not supported by Amazon RDS for MySQL.

For user-created schemas, the MyISAM storage engine does not support reliable recovery and can result in lost or corrupt data when MySQL is restarted after a recovery, preventing Point-In-Time

restore or snapshot restore from working as intended. However, if you still choose to use MyISAM with Amazon RDS, snapshots can be helpful under some conditions.

 **Note**

System tables in the mysql schema can be in MyISAM storage.

If you want to convert existing MyISAM tables to InnoDB tables, you can use the ALTER TABLE command (for example, `alter table TABLE_NAME engine=innodb;`). Bear in mind that MyISAM and InnoDB have different strengths and weaknesses, so you should fully evaluate the impact of making this switch on your applications before doing so.

MySQL 5.1, 5.5, and 5.6 are no longer supported in Amazon RDS. However, you can restore existing MySQL 5.1, 5.5, and 5.6 snapshots. When you restore a MySQL 5.1, 5.5, or 5.6 snapshot, the DB instance is automatically upgraded to MySQL 5.7.

Using memcached and other options with MySQL on Amazon RDS

Most Amazon RDS DB engines support option groups that allow you to select additional features for your DB instance. RDS for MySQL DB instances support the memcached option, a simple, key-based cache. For more information about memcached and other options, see [Options for MySQL DB instances](#). For more information about working with option groups, see [Working with option groups](#).

InnoDB cache warming for MySQL on Amazon RDS

InnoDB cache warming can provide performance gains for your MySQL DB instance by saving the current state of the buffer pool when the DB instance is shut down, and then reloading the buffer pool from the saved information when the DB instance starts up. This bypasses the need for the buffer pool to "warm up" from normal database use and instead preloads the buffer pool with the pages for known common queries. The file that stores the saved buffer pool information only stores metadata for the pages that are in the buffer pool, and not the pages themselves. As a result, the file does not require much storage space. The file size is about 0.2 percent of the cache size. For example, for a 64 GiB cache, the cache warming file size is 128 MiB. For more information on InnoDB cache warming, see [Saving and restoring the buffer pool state](#) in the MySQL documentation.

RDS for MySQL DB instances support InnoDB cache warming. To enable InnoDB cache warming, set the `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` parameters to 1 in the parameter group for your DB instance. Changing these parameter values in a parameter group will affect all MySQL DB instances that use that parameter group. To enable InnoDB cache warming for specific MySQL DB instances, you might need to create a new parameter group for those instances. For information on parameter groups, see [Parameter groups for Amazon RDS](#).

InnoDB cache warming primarily provides a performance benefit for DB instances that use standard storage. If you use PIOPS storage, you do not commonly see a significant performance benefit.

Important

If your MySQL DB instance does not shut down normally, such as during a failover, then the buffer pool state will not be saved to disk. In this case, MySQL loads whatever buffer pool file is available when the DB instance is restarted. No harm is done, but the restored buffer pool might not reflect the most recent state of the buffer pool prior to the restart. To ensure that you have a recent state of the buffer pool available to warm the InnoDB cache on startup, we recommend that you periodically dump the buffer pool "on demand." You can create an event to dump the buffer pool automatically and on a regular interval. For example, the following statement creates an event named `periodic_buffer_pool_dump` that dumps the buffer pool every hour.

```
CREATE EVENT periodic_buffer_pool_dump
ON SCHEDULE EVERY 1 HOUR
DO CALL mysql.rds_innodb_buffer_pool_dump_now();
```

For more information on MySQL events, see [Event syntax](#) in the MySQL documentation.

Dumping and loading the buffer pool on demand

You can save and load the InnoDB cache "on demand."

- To dump the current state of the buffer pool to disk, call the [`mysql.rds_innodb_buffer_pool_dump_now`](#) stored procedure.

- To load the saved state of the buffer pool from disk, call the [mysql.rds_innodb_buffer_pool_load_now](#) stored procedure.
- To cancel a load operation in progress, call the [mysql.rds_innodb_buffer_pool_load_abort](#) stored procedure.

Inclusive language changes for RDS for MySQL 8.4

RDS for MySQL 8.4 includes changes from the MySQL 8.4 community edition related to keywords and system schemas for inclusive language. For example, the SHOW REPLICA STATUS command replaces SHOW SLAVE STATUS.

Topics

- [Configuration parameter name changes](#)
- [Stored procedure name changes](#)

Configuration parameter name changes

The following configuration parameters have new names in RDS for MySQL 8.4.

For compatibility, you can check the parameter names in the mysql client by using either name. You can only use the new names when modifying values in a custom MySQL 8.4 parameter group. For more information, see [Default and custom parameter groups](#).

Name to be removed	New or preferred name
init_slave	init_replica
log_slave_updates	log_replica_updates
log_slow_slave_statements	log_slow_replica_statements
rpl_stop_slave_timeout	rpl_stop_replica_timeout
skip_slave_start	skip_replica_start

Name to be removed	New or preferred name
slave_checkpoint_group	replica_checkpoint_group
slave_checkpoint_period	replica_checkpoint_period
slave_compressed_protocol	replica_compressed_protocol
slave_exec_mode	replica_exec_mode
slave_load_tmpdir	replica_load_tmpdir
slave_max_allowed_packet	replica_max_allowed_packet
slave_net_timeout	replica_net_timeout
slave_parallel_type	replica_parallel_type
slave_parallel_workers	replica_parallel_workers
slave_pending_jobs_size_max	replica_pending_jobs_size_max
slave_preserve_commit_order	replica_preserve_commit_order
slave_skip_errors	replica_skip_errors
slave_sql_verify_checksum	replica_sql_verify_checksum
slave_transaction_retries	replica_transaction_retries

Name to be removed	New or preferred name
slave_type_conversations	replica_type_conversations
sql_slave_skip_counter	sql_replica_skip_counter

 **Note**

The parameter `replica_allow_batching` isn't available because Amazon RDS doesn't support NDB clusters.

Stored procedure name changes

The following stored procedures have new names in RDS for MySQL 8.4.

For compatibility, you can use either name in the initial RDS for MySQL 8.4 release. The old procedure names are to be removed in a future release. For more information, see [Configuring, starting, and stopping binary log \(binlog\) replication](#).

Name to be removed	New or preferred name
<code>mysql.rds_next_master_log</code>	<code>mysql.rds_next_source_log</code>
<code>mysql.rds_reset_external_master</code>	<code>mysql.rds_reset_external_source</code>
<code>mysql.rds_set_external_master</code>	<code>mysql.rds_set_external_source</code>
<code>mysql.rds_set_external_master_with_auto_position</code>	<code>mysql.rds_set_external_source_with_auto_position</code>

Name to be removed	New or preferred name
mysql.rds_set_exte rnal_master_with_d elay	mysql.rds_set_exte rnal_source_with_d elay
mysql.rds_set_mast er_auto_position	mysql.rds_set_sour ce_auto_position

MySQL features not supported by Amazon RDS

Amazon RDS doesn't currently support the following MySQL features:

- Authentication Plugin
- Error Logging to the System Log
- InnoDB Tablespace Encryption
- NDB clusters
- Password Strength Plugin
- Persisted system variables
- Rewriter Query Rewrite Plugin
- Semisynchronous replication, except for Multi-AZ DB clusters
- Transportable tablespace
- X Plugin

To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that require advanced privileges. Amazon RDS supports access to databases on a DB instance using any standard SQL client application. Amazon RDS doesn't allow direct host access to a DB instance by using Telnet, Secure Shell (SSH), or Windows Remote Desktop Connection. When you create a DB instance, you are assigned as *db_owner* for all databases on that instance, and you have all database-level permissions except for those used for backups. Amazon RDS manages backups for you.

MySQL on Amazon RDS versions

For MySQL, version numbers are organized as version = X.Y.Z. In Amazon RDS terminology, X.Y denotes the major version, and Z is the minor version number. For Amazon RDS implementations, a version change is considered major if the major version number changes—for example, going from version 5.7 to 8.0. A version change is considered minor if only the minor version number changes—for example, going from version 8.0.32 to 8.0.34.

Topics

- [Supported MySQL minor versions on Amazon RDS](#)
- [Supported MySQL major versions on Amazon RDS](#)
- [Amazon RDS Extended Support versions for RDS for MySQL](#)
- [Working with the Database Preview environment](#)
- [MySQL version 9.4 in the Database Preview environment](#)
- [MySQL version 9.3 in the Database Preview environment](#)
- [Deprecated versions for Amazon RDS for MySQL](#)

Supported MySQL minor versions on Amazon RDS

Amazon RDS currently supports the following minor versions of MySQL.

 **Note**

Amazon RDS Extended Support isn't available for minor versions.

The following table shows the minor versions of MySQL 8.4 that Amazon RDS currently supports.

MySQL engine version	Community release date	RDS release date	RDS end of standard support date
8.4.6	22 July 2025	1 August 2025	30 September 2026
8.4.5	15 April 2025	29 April 2025	30 September 2026

MySQL engine version	Community release date	RDS release date	RDS end of standard support date
8.4.4	21 January 2025	19 February 2025	31 March 2026
8.4.3	15 October 2024	21 November 2024	31 March 2026

The following table shows the minor versions of MySQL 8.0 that Amazon RDS currently supports.

Note

Minor versions can reach end of standard support before major versions do. For example, minor version 8.0.28 reached its end of standard support date on March 28, 2024 while major version 8.0 reaches this date on July 31, 2026. RDS will support additional 8.0.* minor versions that the MySQL community releases between these dates. We recommend that you upgrade to the latest available minor version as often as possible for all major versions.

MySQL engine version	Community release date	RDS release date	RDS end of standard support date
8.0.43	22 July 2025	1 August 2025	31 July 2026
8.0.42	15 April 2025	29 April 2025	31 July 2026
8.0.41	21 January 2025	19 February 2025	31 March 2026
8.0.40	15 October 2024	13 November 2024	31 March 2026
8.0.39	23 July 2024	12 August 2024	31 October 2025
8.0.37	30 April 2024	18 June 2024	31 October 2025

The following table shows the minor versions of MySQL 5.7 that are available under Amazon RDS Extended Support.

Note

Minor versions can reach end of Extended Support before major versions do. For example, minor version 5.7.44-RDS.20240529 reaches its end of Extended Support date in September 2025 while major version 5.7 reaches this date on February 28, 2027. RDS will generate and release additional 5.7.44-RDS.xxxyzz minor versions between these dates. We recommend that you upgrade to the latest available minor version as often as possible for all major versions.

MySQL engine version	Community release date	RDS release date	RDS end of Extended Support date
5.7.44-RDS.20250818*	Not applicable	15 September 2025	30 September 2026
5.7.44-RDS.20250508*	Not applicable	20 May 2025	30 September 2026
5.7.44-RDS.20250213*	Not applicable	12 March 2025	30 September 2026
5.7.44-RDS.20250103*	Not applicable	13 February 2025	31 March 2026
5.7.44-RDS.20240808*	Not applicable	28 August 2024	31 October 2025
5.7.44-RDS.20240529*	Not applicable	25 June 2024	31 October 2025
5.7.44-RDS.20240408*	Not applicable	17 May 2024	31 October 2025

* MySQL Community retired major version 5.7 and won't be releasing new minor versions. This is a minor version that Amazon RDS released with critical security patches and bug fixes for MySQL 5.7 databases that are covered under RDS Extended Support. For more information about these minor

versions, see [the section called “RDS Extended Support versions”](#). For more information about RDS Extended Support, see [Amazon RDS Extended Support with Amazon RDS](#).

You can specify any currently supported MySQL version when creating a new DB instance. You can specify the major version (such as MySQL 5.7), and any supported minor version for the specified major version. If no version is specified, Amazon RDS defaults to a supported version, typically the most recent version. If a major version is specified but a minor version is not, Amazon RDS defaults to a recent release of the major version you have specified. To see a list of supported versions, as well as defaults for newly created DB instances, run the [describe-db-engine-versions](#) AWS CLI command.

For example, to list the supported engine versions for RDS for MySQL, run the following CLI command:

```
aws rds describe-db-engine-versions --engine mysql --query "*[].  
{Engine:Engine,EngineVersion:EngineVersion}" --output text
```

The default MySQL version might vary by AWS Region. To create a DB instance with a specific minor version, specify the minor version during DB instance creation. You can determine the default minor version for an AWS Region by running the following AWS CLI command:

```
aws rds describe-db-engine-versions --default-only --engine mysql  
--engine-version major_engine_version --region region --query "*[].  
{Engine:Engine,EngineVersion:EngineVersion}" --output text
```

Replace *major_engine_version* with the major engine version, and replace *region* with the AWS Region. For example, the following AWS CLI command returns the default MySQL minor engine version for the 5.7 major version and the US West (Oregon) AWS Region (us-west-2):

```
aws rds describe-db-engine-versions --default-only --engine mysql --engine-version 5.7  
--region us-west-2 --query "*[].{Engine:Engine,EngineVersion:EngineVersion}" --output  
text
```

With Amazon RDS, you control when to upgrade your MySQL instance to a new major version supported by Amazon RDS. You can maintain compatibility with specific MySQL versions, test new versions with your application before deploying in production, and perform major version upgrades at times that best fit your schedule.

When automatic minor version upgrade is enabled, your DB instance will be upgraded automatically to new MySQL minor versions as they are supported by Amazon RDS. This patching occurs during your scheduled maintenance window. You can modify a DB instance to enable or disable automatic minor version upgrades.

If you opt out of automatically scheduled upgrades, you can manually upgrade to a supported minor version release by following the same procedure as you would for a major version update. For information, see [Upgrading a DB instance engine version](#).

Amazon RDS currently supports the following upgrades for major versions of the MySQL database engine:

- MySQL 5.7 to MySQL 8.0
- MySQL 8.0 to MySQL 8.4

Because major version upgrades involve some compatibility risk, they do not occur automatically; you must make a request to modify the DB instance. You should thoroughly test any upgrade before upgrading your production instances. For information about upgrading a MySQL DB instance, see [Upgrades of the RDS for MySQL DB engine](#).

You can test a DB instance against a new version before upgrading by creating a DB snapshot of your existing DB instance, restoring from the DB snapshot to create a new DB instance, and then initiating a version upgrade for the new DB instance. You can then experiment safely on the upgraded clone of your DB instance before deciding whether or not to upgrade your original DB instance.

MySQL minor versions on Amazon RDS

For the changes that the MySQL community made to the minor versions, see [Critical Patch Updates, Security Alerts and Bulletins](#) on the Oracle website. Under **Critical Patch Update**, choose the month when Oracle released the minor version. And then choose the MySQL minor version under **Affected Products and Versions**.

Minor versions

- [MySQL version 8.4.6](#)
- [MySQL version 8.4.5](#)
- [MySQL version 8.4.4](#)

- [MySQL version 8.0.43](#)
- [MySQL version 8.0.42](#)
- [MySQL version 8.0.41](#)
- [MySQL version 8.0.40](#)
- [MySQL version 8.0.39](#)
- [MySQL version 8.0.37](#)

MySQL version 8.4.6

MySQL version 8.4.6 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

MySQL version 8.4.5

MySQL version 8.4.5 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on tzdata2025b.

MySQL version 8.4.4

MySQL version 8.4.4 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on tzdata2025a.
- Fixed a bug that caused a collation error while executing the Amazon RDS stored procedures `mysql.rds_set_configuration` and `mysql.rds_kill`.

MySQL version 8.0.43

MySQL version 8.0.43 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

MySQL version 8.0.42

MySQL version 8.0.42 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on tzdata2025b.

MySQL version 8.0.41

MySQL version 8.0.41 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on tzdata2025a.
- Fixed a bug that caused a collation error while executing the Amazon RDS stored procedures mysql.rds_set_configuration and mysql.rds_kill.

MySQL version 8.0.40

MySQL version 8.0.40 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Fixed a bug that caused character set mismatch failures during database upgrades.

MySQL version 8.0.39

MySQL version 8.0.39 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Fixed a bug that prevented sql_log_off from working properly with the SESSION_VARIABLES_ADMIN privilege.
- Fixed a bug that prevented the master user from being able to grant the SESSION_VARIABLE_ADMIN privilege other database users.

- Fixed a bug that caused an illegal mix of collation while executing RDS-provided stored procedures.

MySQL version 8.0.37

MySQL version 8.0.37 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Fixed a bug with executing an instant DDL statement followed by an UPDATE that lead to an assertion failure.

Supported MySQL major versions on Amazon RDS

RDS for MySQL major versions are available under standard support at least until community end of life for the corresponding community version. You can continue running a major version past its RDS end of standard support date for a fee. For more information, see [Amazon RDS Extended Support with Amazon RDS](#) and [Amazon RDS for MySQL pricing](#).

You can use the following dates to plan your testing and upgrade cycles.

 **Note**

You can also view information about support dates for major engine versions by using the AWS CLI or the RDS API. For more information, see [Viewing support dates for engine versions in Amazon RDS Extended Support](#).

MySQL major version	Community release date	RDS release date	Community end of life date	RDS end of standard support date	RDS start of Extended Support year 1 pricing date	RDS start of Extended Support year 3 pricing date	RDS end of Extended Support date
MySQL 8.4	30 April 2024	21 November 2024	30 April 2029	31 July 2029	1 August, 2029	1 August 2031	31 July 2032
MySQL 8.0	19 April 2018	23 October 2018	30 April 2026	31 July 2026	1 August 2026	1 August 2028	31 July 2029
MySQL 5.7*	21 October 2015	22 February 2016	31 October 2023	29 February 2024	1 March 2024	1 March 2026	28 February 2027

* MySQL 5.7 is now only available under RDS Extended Support. For more information, see [Amazon RDS Extended Support with Amazon RDS](#).

Amazon RDS Extended Support versions for RDS for MySQL

The following content lists all releases of RDS Extended Support for RDS for MySQL versions.

Releases

- [RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250818](#)
- [RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250508](#)
- [RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250213](#)
- [RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250103](#)
- [RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240808](#)
- [RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240529](#)
- [RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240408](#)

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250818

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250818 is available.

Bugs fixed:

- Fixed an issue where the query rewrite plugin failed when the server operated with `autocommit=OFF`.
- Fixed a permission issue that prevented Debian and Ubuntu builds from running in rootless mode.
- Fixed missing update for bug 30875669.

CVEs fixed:

- [CVE-2025-50082](#)
- [CVE-2025-50083](#)
- [CVE-2025-50079](#)
- [CVE-2025-50084](#)
- [CVE-2025-50087](#)
- [CVE-2025-50091](#)
- [CVE-2025-50101](#)
- [CVE-2025-50102](#)
- [CVE-2025-50098](#)
- [CVE-2025-53023](#)
- [CVE-2025-50081](#)
- [CVE-2025-50085](#)
- [CVE-2025-50077](#)
- [CVE-2025-50088](#)
- [CVE-2025-50092](#)
- [CVE-2025-50099](#)
- [CVE-2025-50096](#)

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250508

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250508 is available.

Bugs fixed:

- Fixed virtual index unstable after rollback when `index_id` is greater than `max uint32`.
- Fixed Tests fails with memory issue.
- Fixed `<COMMAND_CLASS>` is empty for `<NAME>Execute</NAME>`.
- Fixed Compile MySQL with GCC 14 [noclose 5.7].

CVEs fixed:

- [CVE-2025-30682](#)
- [CVE-2025-30687](#)
- [CVE-2025-30688](#)
- [CVE-2025-21581](#)
- [CVE-2025-21585](#)
- [CVE-2025-30689](#)
- [CVE-2025-30722](#)
- [CVE-2025-21577](#)
- [CVE-2025-30693](#)
- [CVE-2025-30695](#)
- [CVE-2025-30703](#)

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250213

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250213 is available.

Bugs fixed:

- Fixed InnoDB failing assertion `result != FTS_INVALID`.
- Fixed crashing and widespread corruption of spatial indexes after `ALTER TABLE` operation rebuilds InnoDB table using the `INPLACE` algorithm.

- Fixed ON DELETE CASCADE with generated column crashes in innobase_get_computed_value.
- Fixed assertion failure in row_MySQL_pad_col.
- Fixed an issue where online DDL causes the following error: ERROR 1712 (HY000): Index PRIMARY is corrupted.
- Fixed crashes at Item_rollup_sum_switcher::current_arg.
- Fixed an issue where the database cache is not flushed on DROP USER.
- Fixed buffer overrun in my_print_help.
- Fixed an InnoDB issue where FULLTEXT index limits FTS_DOC_ID to max unsigned 32-bit value.

CVEs fixed:

- [CVE-2025-21497](#)
- [CVE-2025-21555](#)
- [CVE-2025-21559](#)
- [CVE-2025-21490](#)
- [CVE-2025-21491](#)
- [CVE-2025-21500](#)
- [CVE-2025-21501](#)
- [CVE-2025-21540](#)
- [CVE-2025-21543](#)
- [CVE-2025-21520](#)

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250103

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250103 is available.

Bugs fixed:

- Fixed FTS clean-up issue when dropping and adding a FULLTEXT index in the same transaction.
- Optimized the memory allocation timing in the MySQL client to prevent any potential leaks.
- Fixed the truncation of results at 34 bytes when using the UNION operator.

- Fixed potential out-of-bounds access due to ulong bitmask in the authorization code.

CVEs fixed:

- [CVE-2024-21230](#)
- [CVE-2024-21201](#)
- [CVE-2024-21241](#)
- [CVE-2024-21203](#)

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240808

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240808 is available.

Bugs fixed:

- Fixed assertion failure related to dictionary column index.
- Fixed issue with the `is_binlog_cache_empty()` function.
- Fixed heap-use-after-free errors in `sql/item.cc` files.
- Fixed several spatial index issues by disabling them for `index-only` reads.
- Fixed instrumentation issue with the `LOCK_ORDER: CONNECTION_CONTROL` plugin.
- Fixed threads getting stuck with the `CONNECTION_CONTROL` plugin.
- Fixed `PSI_THREAD_INFO` not updating for `PREPARED STATEMENTS`.
- Fixed double processing of FTS index words with `innodb_optimize_fulltext_only`.

CVEs fixed:

- [CVE-2024-21177](#)

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240529

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240529 is available.

Bugs fixed:

- Fixed `field.cc` assertion failure by implementing `fix_after_pullout`.

- Fixed a null pointer failure when returning metadata to the client for certain SQL queries. These queries contained dynamic parameters and subqueries in SELECT clauses.
- Fixed incorrect results when using GROUP BY for loose index scans, or scans of noncontiguous ranges of an index.
- Fixed loss of GTID information on MySQL crash during persistence.
- Fixed a race condition that could cause an InnoDB transaction to hang indefinitely.
- Fixed a race condition in Group Replication's certification information cleanup.
- Fixed backward index scan issue with concurrent page operations.
- Fixed an inconsistent full-text search (FTS) state issue in concurrent scenarios.
- Fixed assertion issue with change buffer on deleting tables.
- Unified behavior for calling deinit function across all plugin types.

CVEs fixed:

- [CVE-2024-20963](#)
- [CVE-2024-20993](#)
- [CVE-2024-20998](#)
- [CVE-2024-21009](#)
- [CVE-2024-21054](#)
- [CVE-2024-21055](#)
- [CVE-2024-21057](#)
- [CVE-2024-21062](#)
- [CVE-2024-21008](#)
- [CVE-2024-21013](#)
- [CVE-2024-21047](#)
- [CVE-2024-21087](#)
- [CVE-2024-21096](#)

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240408

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240408 is available.

This release contains patches for the following CVEs:

- [CVE-2024-20963](#)

Working with the Database Preview environment

In July 2023, Oracle announced a new release model for MySQL. This model includes two types of releases: Innovation Releases and LTS releases. Amazon RDS makes MySQL Innovation Releases available in the RDS Preview environment. To learn more about the MySQL Innovation releases, see [Introducing MySQL Innovation and Long-Term Support \(LTS\) versions](#).

RDS for MySQL DB instances in the Database Preview environment are functionally similar to other RDS for MySQL DB instances. However, you can't use the Database Preview environment for production workloads.

Preview environments have the following limitations:

- Amazon RDS deletes all DB instances 60 days after you create them, along with any backups and snapshots.
- You can only use General Purpose SSD and Provisioned IOPS SSD storage.
- You can't get help from Support with DB instances. Instead, you can post your questions to the AWS-managed Q&A community, [AWS re:Post](#).
- You can't copy a snapshot of a DB instance to a production environment.

The following options are supported by the preview.

- You can create DB instances using db.m6i, db.r6i, db.m6g, db.m5, db.t3, db.r6g, and db.r5 DB instance classes. For more information about RDS instance classes, see [DB instance classes](#).
- You can use both single-AZ and multi-AZ deployments.
- You can use standard MySQL dump and load functions to export databases from or import databases to the Database Preview environment.

Features not supported in the Database Preview environment

The following features aren't available in the Database Preview environment:

- Cross-Region snapshot copy
- Cross-Region read replicas

- RDS Proxy

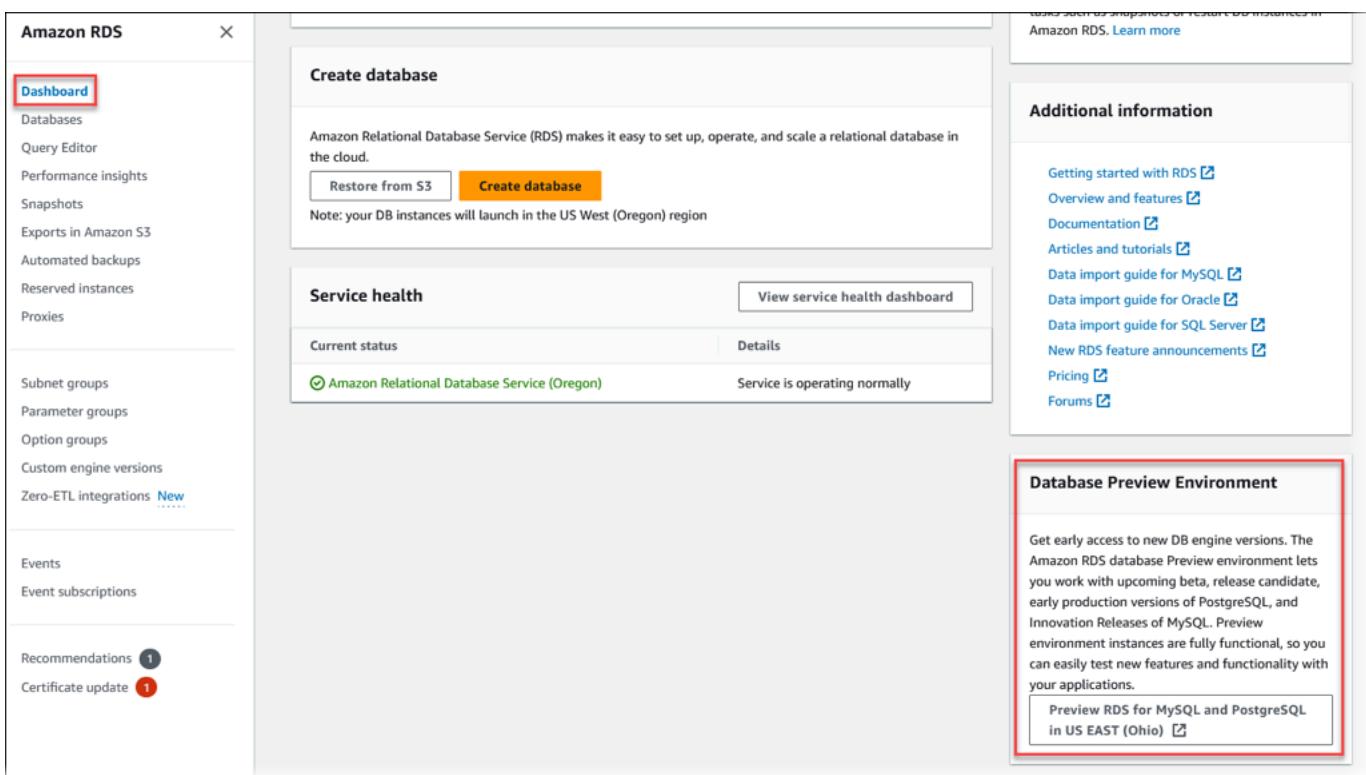
Creating a new DB instance in the Database Preview environment

You can create a DB instance in the Database Preview environment using the AWS Management Console, AWS CLI, or RDS API.

Console

To create a DB instance in the Database Preview environment

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Dashboard** from the navigation pane.
3. In the **Dashboard** page, locate the **Database Preview Environment** section, as shown in the following image.



You can navigate directly to the [Database Preview environment](#). Before you can proceed, you must acknowledge and accept the limitations.

Database Preview Environment Service Agreement X

The Amazon RDS Database Preview Environment is not covered by the Amazon RDS service level agreement (SLA), published at <https://aws.amazon.com/rds/sla> 

Do not use the Amazon RDS Database Preview Environment for production purposes. You should only use this environment for development and testing.

Certain use cases might fail in this environment - for example, upgrading from a previous version is not supported.

I acknowledge this limited service agreement for the Amazon RDS Database Preview Environment and that I should only use this environment for development and testing.

Cancel Accept

4. To create the RDS for MySQL DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the [Console](#) procedure in [Creating a DB instance](#).

AWS CLI

To create a DB instance in the Database Preview environment using the AWS CLI, use the following endpoint.

```
rds-preview.us-east-2.amazonaws.com
```

To create the RDS for MySQL DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the [AWS CLI](#) procedure in [Creating a DB instance](#).

RDS API

To create a DB instance in the Database Preview environment using the RDS API, use the following endpoint.

rds-preview.us-east-2.amazonaws.com

To create the RDS for MySQL DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the [RDS API](#) procedure in [Creating a DB instance](#).

MySQL version 9.4 in the Database Preview environment

MySQL version 9.4 is now available in the Amazon RDS Database Preview environment. MySQL version 9.4 contains several improvements that are described in [Changes in MySQL 9.4.0](#).

For information on the Database Preview environment, see [the section called “Database Preview environment”](#). To access the Preview Environment from the console, select <https://console.aws.amazon.com/rds-preview/>.

MySQL version 9.3 in the Database Preview environment

MySQL version 9.3 is now available in the Amazon RDS Database Preview environment. MySQL version 9.3 contains several improvements that are described in [Changes in MySQL 9.3.0](#).

For information on the Database Preview environment, see [the section called “Database Preview environment”](#). To access the Preview Environment from the console, select <https://console.aws.amazon.com/rds-preview/>.

Deprecated versions for Amazon RDS for MySQL

Amazon RDS for MySQL version 5.1, 5.5, and 5.6 are deprecated.

Amazon RDS for MySQL version 9.1 and 9.2 are deprecated in the Database Preview environment.

For information about the Amazon RDS deprecation policy for MySQL, see [Amazon RDS FAQs](#).

Connecting to your MySQL DB instance

Before you can connect to a DB instance running the MySQL database engine, you must create a DB instance. For information, see [Creating an Amazon RDS DB instance](#). After Amazon RDS provisions your DB instance, you can use any standard MySQL client application or utility to connect to the instance. In the connection string, you specify the DNS address from the DB instance endpoint as the host parameter, and specify the port number from the DB instance endpoint as the port parameter.

To authenticate to your RDS DB instance, you can use one of the authentication methods for MySQL and AWS Identity and Access Management (IAM) database authentication:

- To learn how to authenticate to MySQL using one of the authentication methods for MySQL, see [Authentication method](#) in the MySQL documentation.
- To learn how to authenticate to MySQL using IAM database authentication, see [IAM database authentication for MariaDB, MySQL, and PostgreSQL](#).

You can connect to a MySQL DB instance by using tools like the MySQL command-line client. For more information on using the MySQL command-line client, see [mysql - the MySQL command-line client](#) in the MySQL documentation. One GUI-based application you can use to connect is MySQL Workbench. For more information, see the [Download MySQL Workbench](#) page. For information about installing MySQL (including the MySQL command-line client), see [Installing and upgrading MySQL](#).

To connect to a DB instance from outside of its Amazon VPC, the DB instance must be publicly accessible, access must be granted using the inbound rules of the DB instance's security group, and other requirements must be met. For more information, see [Can't connect to Amazon RDS DB instance](#).

You can use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) encryption on connections to a MySQL DB instance. For information, see [SSL/TLS support for MySQL DB instances on Amazon RDS](#). If you are using AWS Identity and Access Management (IAM) database authentication, make sure to use an SSL/TLS connection. For information, see [IAM database authentication for MariaDB, MySQL, and PostgreSQL](#).

You can also connect to a DB instance from a web server. For more information, see [Tutorial: Create a web server and an Amazon RDS DB instance](#).

Note

For information on connecting to a MariaDB DB instance, see [Connecting to your MariaDB DB instance](#).

To find and connect to a RDS for MySQL DB instance, see the following topics.

Topics

- [Finding the connection information for an RDS for MySQL DB instance](#)
- [Installing the MySQL command-line client](#)
- [Connecting from the MySQL command-line client \(unencrypted\)](#)
- [Connecting from MySQL Workbench](#)
- [Connecting to RDS for MySQL with the AWS JDBC Driver, AWS Python Driver, and AWS ODBC Driver for MySQL](#)
- [Troubleshooting connections to your MySQL DB instance](#)

Finding the connection information for an RDS for MySQL DB instance

The connection information for a DB instance includes its endpoint, port, and a valid database user, such as the master user. For example, suppose that an endpoint value is mydb.123456789012.us-east-1.rds.amazonaws.com. In this case, the port value is 3306, and the database user is admin. Given this information, you specify the following values in a connection string:

- For host or host name or DNS name, specify mydb.123456789012.us-east-1.rds.amazonaws.com.
- For port, specify 3306.
- For user, specify admin.

To connect to a DB instance, use any client for the MySQL DB engine. For example, you might use the MySQL command-line client or MySQL Workbench.

To find the connection information for a DB instance, you can use the AWS Management Console, the AWS CLI [describe-db-instances](#) command, or the Amazon RDS API [DescribeDBInstances](#) operation to list its details.

Console

To find the connection information for a DB instance in the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases** to display a list of your DB instances.
3. Choose the name of the MySQL DB instance to display its details.
4. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.

RDS > Databases > mydb

mydb

Summary

DB identifier	mydb	CPU	2.33%
Role	Instance	Current activity	0 Connections

Connectivity & security Monitoring Logs & events Configuration

Connectivity & security

Endpoint	mydb. [REDACTED].us-east-1.rds.amazonaws.com	Network
Port	3306	Available
		us-east-1
		VPC
		vpc-61
		Subnet
		default

5. If you need to find the master user name, choose the **Configuration** tab and view the **Master username** value.

AWS CLI

To find the connection information for a MySQL DB instance by using the AWS CLI, run the [describe-db-instances](#) command. In the call, query for the DB instance ID, endpoint, port, and master user name.

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
--filters "Name=engine,Values=mysql" \
--query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

For Windows:

```
aws rds describe-db-instances ^
--filters "Name=engine,Values=mysql" ^
--query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

Your output should be similar to the following.

```
[  
 [  
   "mydb1",  
   "mydb1.123456789012.us-east-1.rds.amazonaws.com",  
   3306,  
   "admin"  
 ],  
 [  
   "mydb2",  
   "mydb2.123456789012.us-east-1.rds.amazonaws.com",  
   3306,  
   "admin"  
 ]  
]
```

RDS API

To find the connection information for a DB instance by using the Amazon RDS API, call the [DescribeDBInstances](#) operation. In the output, find the values for the endpoint address, endpoint port, and master user name.

Installing the MySQL command-line client

Most Linux distributions include the MariaDB client instead of the Oracle MySQL client. To install the MySQL command-line client on Amazon Linux 2023, run the following command:

```
sudo dnf install mariadb105
```

To install the MySQL command-line client on Amazon Linux 2, run the following command:

```
sudo yum install mariadb
```

To install the MySQL command-line client on most DEB-based Linux distributions, run the following command:

```
apt-get install mariadb-client
```

To check the version of your MySQL command-line client, run the following command:

```
mysql --version
```

To read the MySQL documentation for your current client version, run the following command:

```
man mysql
```

Connecting from the MySQL command-line client (unencrypted)

Important

Only use an unencrypted MySQL connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see [Connecting to your MySQL DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client \(encrypted\)](#).

To connect to a DB instance using the MySQL command-line client, enter the following command at the command prompt. For the -h parameter, substitute the DNS name (endpoint) for your DB instance. For the -P parameter, substitute the port for your DB instance. For the -u parameter,

substitute the user name of a valid database user, such as the master user. Enter the master user password when prompted.

```
mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com -P 3306 -  
u mymasteruser -p
```

After you enter the password for the user, you should see output similar to the following.

```
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 9738  
Server version: 8.0.28 Source distribution  
  
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.  
  
mysql>
```

Connecting from MySQL Workbench

To connect from MySQL Workbench

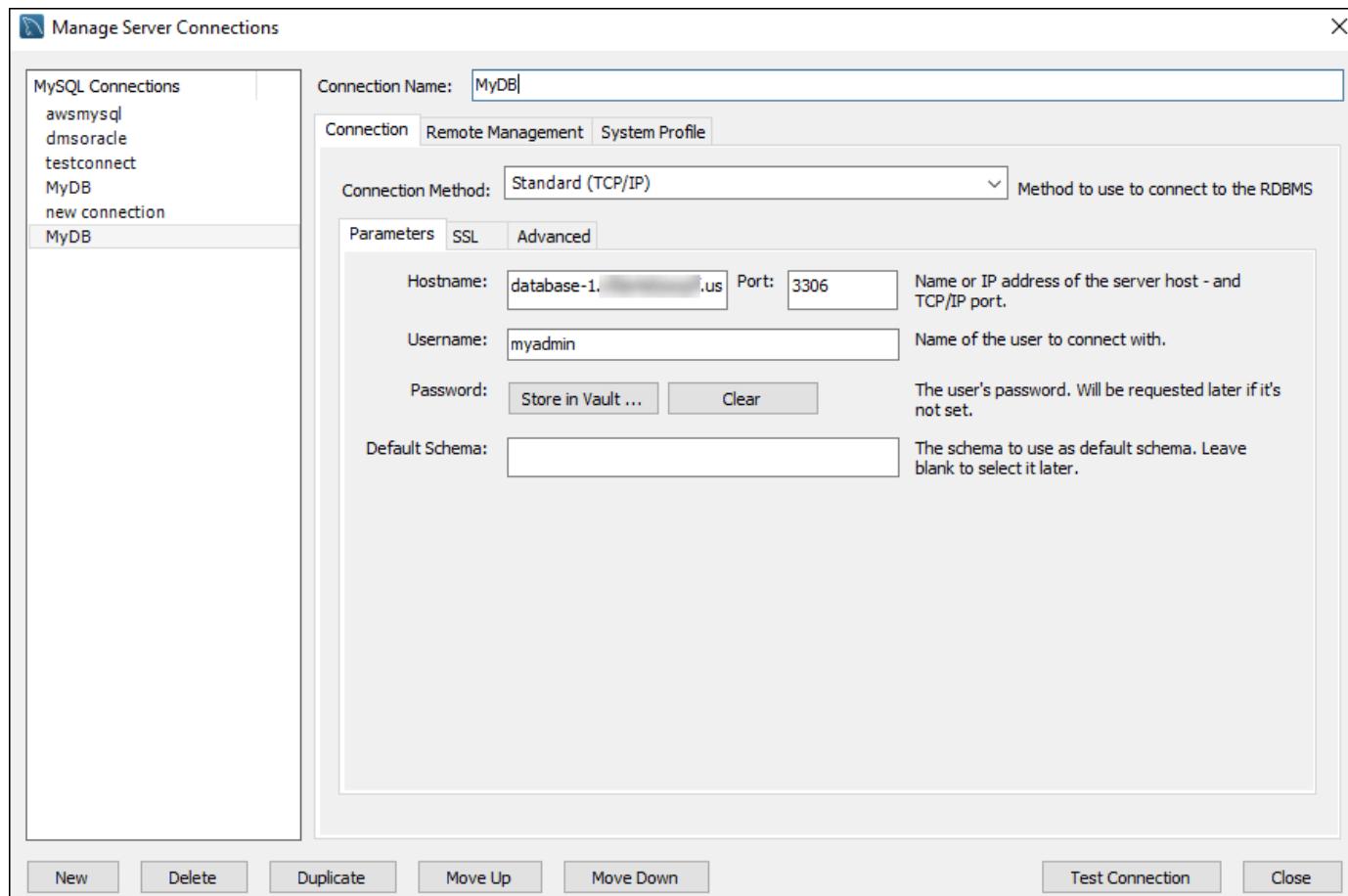
1. Download and install MySQL Workbench at [Download MySQL Workbench](#).
2. Open MySQL Workbench.



3. From **Database**, choose **Manage Connections**.
4. In the **Manage Server Connections** window, choose **New**.
5. In the **Connect to Database** window, enter the following information:

- **Stored Connection** – Enter a name for the connection, such as **MyDB**.
- **Hostname** – Enter the DB instance endpoint.
- **Port** – Enter the port used by the DB instance.
- **Username** – Enter the user name of a valid database user, such as the master user.
- **Password** – Optionally, choose **Store in Vault** and then enter and save the password for the user.

The window looks similar to the following:



You can use the features of MySQL Workbench to customize connections. For example, you can use the **SSL** tab to configure SSL/TLS connections. For information about using MySQL Workbench, see the [MySQL Workbench documentation](#). Encrypting client connections to MySQL DB instances with SSL/TLS, see [Encrypting client connections with SSL/TLS to MySQL DB instances on Amazon RDS](#).

6. Optionally, choose **Test Connection** to confirm that the connection to the DB instance is successful.
7. Choose **Close**.
8. From **Database**, choose **Connect to Database**.
9. From **Stored Connection**, choose your connection.
10. Choose **OK**.

Connecting to RDS for MySQL with the AWS JDBC Driver, AWS Python Driver, and AWS ODBC Driver for MySQL

Connect to RDS for MySQL DB instances with the AWS JDBC Driver, the AWS Python Driver, and the AWS ODBC Driver for MySQL. For more information, see the following topics.

Topics

- [Connecting to RDS for MySQL with the Amazon Web Services \(AWS\) JDBC Driver](#)
- [Connecting to RDS for MySQL with the Amazon Web Services \(AWS\) Python Driver](#)
- [Connecting to RDS for MySQL with the Amazon Web Services \(AWS\) ODBC Driver for MySQL](#)

Connecting to RDS for MySQL with the Amazon Web Services (AWS) JDBC Driver

The Amazon Web Services (AWS) JDBC Driver is designed as an advanced JDBC wrapper. This wrapper is complementary to and extends the functionality of an existing JDBC driver. The driver is drop-in compatible with the community MySQL Connector/J driver and the community MariaDB Connector/J driver.

To install the AWS JDBC Driver, append the AWS JDBC Driver .jar file (located in the application CLASSPATH), and keep references to the respective community driver. Update the respective connection URL prefix as follows:

- `jdbc:mysql://` to `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` to `jdbc:aws-wrapper:mariadb://`

For more information about the AWS JDBC Driver and complete instructions for using it, see the [Amazon Web Services \(AWS\) JDBC Driver GitHub repository](#).

Connecting to RDS for MySQL with the Amazon Web Services (AWS) Python Driver

The Amazon Web Services (AWS) Python Driver is designed as an advanced Python wrapper. This wrapper is complementary to and extends the functionality of the open-source Psycopg driver. The AWS Python Driver supports Python versions 3.8 and higher. You can install the `aws-advanced-python-wrapper` package using the pip command, along with the `psycopg` open-source packages.

For more information about the AWS Python Driver and complete instructions for using it, see the [Amazon Web Services \(AWS\) Python Driver GitHub repository](#).

Connecting to RDS for MySQL with the Amazon Web Services (AWS) ODBC Driver for MySQL

The AWS ODBC Driver for MySQL is a client driver designed for the high availability of RDS for MySQL. The driver can exist alongside the MySQL Connector/ODBC driver and is compatible with the same workflows.

For more information about the AWS ODBC Driver for MySQL and complete instructions for installing and using it, see the [Amazon Web Services \(AWS\) ODBC Driver for MySQL GitHub repository](#).

Troubleshooting connections to your MySQL DB instance

Two common causes of connection failures to a new DB instance are:

- The DB instance was created using a security group that doesn't authorize connections from the device or Amazon EC2 instance where the MySQL application or utility is running. The DB instance must have a VPC security group that authorizes the connections. For more information, see [Amazon VPC and Amazon RDS](#).

You can add or edit an inbound rule in the security group. For **Source**, choose **My IP**. This allows access to the DB instance from the IP address detected in your browser.

- The DB instance was created using the default port of 3306, and your company has firewall rules blocking connections to that port from devices in your company network. To fix this failure, recreate the instance with a different port.

For more information on connection issues, see [Can't connect to Amazon RDS DB instance](#).

Securing MySQL DB instance connections

You can implement robust security measures to protect MySQL DB instances from unauthorized access and potential threats. Security groups, SSL/TLS encryption, and IAM database authentication work together to create multiple layers of connection security for your MySQL DB instances. These security controls help you meet compliance requirements, prevent data breaches, and maintain secure communication channels between applications and databases. You can secure your MySQL DB instances by encrypting data in transit, restricting access to specific IP ranges, and managing user authentication through IAM roles rather than database passwords.

Security for MySQL DB instances is managed at three levels:

- AWS Identity and Access Management controls who can perform Amazon RDS management actions on DB instances. When you connect to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see [Identity and access management for Amazon RDS](#).
- When you create a DB instance, you use a VPC security group to control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance. These connections can be made using Secure Sockets Layer (SSL) and Transport Layer Security (TLS). In addition, firewall rules at your company can control whether devices running at your company can open connections to the DB instance.
- To authenticate login and permissions for a MySQL DB instance, you can take either of the following approaches, or a combination of them:
 - You can take the same approach as with a stand-alone instance of MySQL. Commands such as CREATE USER, RENAME USER, GRANT, REVOKE, and SET PASSWORD work just as they do in on-premises databases, as does directly modifying database schema tables. However, directly modifying the database schema tables isn't a best practice, and starting from RDS for MySQL version 8.0.36, it isn't supported. For information, see [Access control and account management](#) in the MySQL documentation.
 - You can also use IAM database authentication. With IAM database authentication, you authenticate to your DB instance by using an IAM user or IAM role and an authentication token. An *authentication token* is a unique value that is generated using the Signature Version 4 signing process. By using IAM database authentication, you can use the same credentials to control access to your AWS resources and your databases. For more information, see [IAM database authentication for MariaDB, MySQL, and PostgreSQL](#).

- Another option is Kerberos authentication for RDS for MySQL. The DB instance works with AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) to enable Kerberos authentication. When users authenticate with a MySQL DB instance joined to the trusting domain, authentication requests are forwarded. Forwarded requests go to the domain directory that you create with AWS Directory Service. For more information, see [Using Kerberos authentication for Amazon RDS for MySQL](#).

When you create an Amazon RDS DB instance, the master user has the following default privileges:

Engine version	System privilege	Database role
RDS for MySQL version 8.4.3 and higher	GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCE S ,INDEX, ALTER, SHOW DATABASES ,CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE ,REPLICATION CLIENT ,CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE ROLE, DROP ROLE, APPLICATION_PASSWORD_ADMIN ,FLUSH_OPTIMIZER_CO STS ,FLUSH_PRIVILEGES ,FLUSH_STATUS ,FLUSH_TABLES ,FLUSH_USER_RESOURCE ,ROLE_ADMIN ,SENSITIVE_VARIABLE S_OBSERVER ,SESSION_VARIABLES_ ADMIN ,SET_ANY_DEFINER ,SHOW_ROUTINE ,XA_RECOVER_ADMIN	rds_superuser_role For more information about rds_superuser_role , see Role-based privilege model for RDS for MySQL .
RDS for MySQL version 8.0.36 and higher	SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES , INDEX, ALTER, SHOW DATABASES ,CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT ,CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE ROLE, DROP ROLE, APPLICATION_PASSWORD_ADMIN ,	rds_superuser_role For more information about rds_superuser_role , see Role-based privilege model for RDS for MySQL .

Engine version	System privilege	Database role
	ROLE_ADMIN , SET_USER_ID , XA_RECOVE R_ADMIN	
RDS for MySQL versions lower than 8.0.36	SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES , INDEX, ALTER, SHOW DATABASES , CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION CLIENT , CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, REPLICATION SLAVE	None

 **Note**

Although it is possible to delete the master user on the DB instance, it is not recommended. To recreate the master user, use the [ModifyDBInstance](#) RDS API operation or run the [modify-db-instance](#) AWS CLI command and specify a new master user password with the appropriate parameter. If the master user does not exist in the instance, the master user is created with the specified password.

To provide management services for each DB instance, the `rdsadmin` user is created when the DB instance is created. Attempting to drop, rename, change the password, or change privileges for the `rdsadmin` account will result in an error.

To allow management of the DB instance, the standard `kill` and `kill_query` commands have been restricted. The Amazon RDS commands `rds_kill` and `rds_kill_query` are provided to allow you to end user sessions or queries on DB instances.

Password validation for RDS for MySQL

MySQL provides the `validate_password` plugin for improved security. The plugin enforces password policies using parameters in the DB parameter group for your MySQL DB instance. The plugin is supported for DB instances running MySQL version 5.7, 8.0, and 8.4. For more

information about the validate_password plugin, see [The Password Validation Plugin](#) in the MySQL documentation.

To enable the validate_password plugin for a MySQL DB instance

1. Connect to your MySQL DB instance and run the following command.

```
INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

2. Configure the parameters for the plugin in the DB parameter group used by the DB instance.

For more information about the parameters, see [Password Validation Plugin Options and Variables](#) in the MySQL documentation.

For more information about modifying DB instance parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

3. Restart the DB instance.

After enabling the validate_password plugin, reset existing passwords to comply with your new validation policies.

Your MySQL DB instance handles password validation for Amazon RDS. To change a password, you first submit a password update request through the AWS Management Console, modify-db-instance CLI command, or ModifyDBInstance API operation. RDS initially accepts your request, even if the password doesn't meet your policies. RDS then processes the request asynchronously. It updates the password in your MySQL DB instance only if the password meets your defined policies. If the password doesn't meet these policies, RDS keeps the existing password and logs an error event.

Unable to reset your password. Error information: Password failed to meet validation rules.

For more information about Amazon RDS events, see [Working with Amazon RDS event notification](#).

Encrypting client connections with SSL/TLS to MySQL DB instances on Amazon RDS

Secure Sockets Layer (SSL) is an industry-standard protocol for securing network connections between client and server. After SSL version 3.0, the name was changed to Transport Layer Security (TLS). Amazon RDS supports SSL/TLS encryption for MySQL DB instances. Using SSL/TLS, you can encrypt a connection between your application client and your MySQL DB instance. SSL/TLS support is available in all AWS Regions for MySQL.

With Amazon RDS, you can secure data in transit by encrypting client connections to MySQL DB instances with SSL/TLS, requiring SSL/TLS for all connections to a MySQL DB instance, and connecting from the MySQL command-line client with SSL/TLS (encrypted). The following sections provide guidance on configuring and utilizing SSL/TLS encryption for MySQL DB instances on Amazon RDS.

Topics

- [SSL/TLS support for MySQL DB instances on Amazon RDS](#)
- [Requiring SSL/TLS for specific user accounts to a MySQL DB instance on Amazon RDS](#)
- [Requiring SSL/TLS for all connections to a MySQL DB instance on Amazon RDS](#)
- [Connecting to your MySQL DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client \(encrypted\)](#)

SSL/TLS support for MySQL DB instances on Amazon RDS

Amazon RDS creates an SSL/TLS certificate and installs the certificate on the DB instance when Amazon RDS provisions the instance. These certificates are signed by a certificate authority. The SSL/TLS certificate includes the DB instance endpoint as the Common Name (CN) for the SSL/TLS certificate to guard against spoofing attacks.

An SSL/TLS certificate created by Amazon RDS is the trusted root entity and should work in most cases, but might fail if your application doesn't accept certificate chains. If your application doesn't accept certificate chains, try using an intermediate certificate to connect to your AWS Region. For example, you must use an intermediate certificate to connect to the AWS GovCloud (US) Regions with SSL/TLS.

For information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB instance or cluster](#). For more information about using SSL/TLS with MySQL, see [Updating applications to connect to MySQL DB instances using new SSL/TLS certificates](#).

For MySQL version 8.0 and lower, Amazon RDS for MySQL uses OpenSSL for secure connections. For MySQL version 8.4 and higher, Amazon RDS for MySQL uses AWS-LC. TLS support depends on the MySQL version. The following table shows the TLS support for MySQL versions.

MySQL version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
MySQL 8.4	Not supported	Not supported	Supported	Supported
MySQL 8.0	Not supported	Not supported	Supported	Supported
MySQL 5.7	Supported	Supported	Supported	Not supported

Requiring SSL/TLS for specific user accounts to a MySQL DB instance on Amazon RDS

You can require SSL/TLS encryption for specified user account connections to your MySQL DB instances on Amazon RDS. Protecting sensitive information from unauthorized access or interception is crucial to enforce security policies where data confidentiality is a concern.

To require SSL/TLS connections for specific users' accounts, use one of the following statements, depending on your MySQL version, to require SSL/TLS connections on the user account `encrypted_user`.

To do so, use the following statement.

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

For more information on SSL/TLS connections with MySQL, see the [Using encrypted connections](#) in the MySQL documentation.

Requiring SSL/TLS for all connections to a MySQL DB instance on Amazon RDS

Use the `require_secure_transport` parameter to require that all user connections to your MySQL DB instance use SSL/TLS. By default, the `require_secure_transport` parameter is set

to OFF. You can set the `require_secure_transport` parameter to ON to require SSL/TLS for connections to your DB instance.

You can set the `require_secure_transport` parameter value by updating the DB parameter group for your DB instance. You don't need to reboot your DB instance for the change to take effect.

When the `require_secure_transport` parameter is set to ON for a DB instance, a database client can connect to it if it can establish an encrypted connection. Otherwise, an error message similar to the following is returned to the client:

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --require_secure_transport=ON.
```

For information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

For more information about the `require_secure_transport` parameter, see the [MySQL documentation](#).

Connecting to your MySQL DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted)

The `mysql` client program parameters are slightly different depending on which version of MySQL or MariaDB you are using.

To find out which version you have, run the `mysql` command with the `--version` option. In the following example, the output shows that the client program is from MariaDB.

```
$ mysql --version
mysql Ver 15.1 Distrib 10.5.15-MariaDB, for osx10.15 (x86_64) using readline 5.1
```

Most Linux distributions, such as Amazon Linux, CentOS, SUSE, and Debian have replaced MySQL with MariaDB, and the `mysql` version in them is from MariaDB.

To connect to your DB instance using SSL/TLS, follow these steps:

To connect to a DB instance with SSL/TLS using the MySQL command-line client

1. Download a root certificate that works for all AWS Regions.

- For information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB instance or cluster](#).
2. Use a MySQL command-line client to connect to a DB instance with SSL/TLS encryption. For the `-h` parameter, substitute the DNS name (endpoint) for your DB instance. For the `--ssl-ca` parameter, substitute the SSL/TLS certificate file name. For the `-P` parameter, substitute the port for your DB instance. For the `-u` parameter, substitute the user name of a valid database user, such as the master user. Enter the master user password when prompted.

The following example shows how to launch the client using the `--ssl-ca` parameter using the MySQL 5.7 client or later:

```
mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-
bundle.pem --ssl-mode=REQUIRED -P 3306 -u myadmin -p
```

To require that the SSL/TLS connection verifies the DB instance endpoint against the endpoint in the SSL/TLS certificate, enter the following command:

```
mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-
bundle.pem --ssl-mode=VERIFY_IDENTITY -P 3306 -u myadmin -p
```

The following example shows how to launch the client using the `--ssl-ca` parameter using the MariaDB client:

```
mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-
bundle.pem --ssl -P 3306 -u myadmin -p
```

3. Enter the master user password when prompted.

You will see output similar to the following.

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 9738
Server version: 8.0.28 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Updating applications to connect to MySQL DB instances using new SSL/TLS certificates

As of January 13, 2023, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your RDS DB instances using Secure Socket Layer or Transport Layer Security (SSL/TLS). Following, you can find information about updating your applications to use the new certificates.

This topic can help you to determine whether any client applications use SSL/TLS to connect to your DB instances. If they do, you can further check whether those applications require certificate verification to connect.

Note

Some applications are configured to connect to MySQL DB instances only if they can successfully verify the certificate on the server. For such applications, you must update your client application trust stores to include the new CA certificates.

You can specify the following SSL modes: disabled, preferred, and required. When you use the preferred SSL mode and the CA certificate doesn't exist or isn't up to date, the connection falls back to not using SSL and connects without encryption.

We recommend avoiding preferred mode. In preferred mode, if the connection encounters an invalid certificate, it stops using encryption and proceeds unencrypted.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB instances. We strongly recommend testing these procedures in a development or staging environment before implementing them in your production environments.

For more information about certificate rotation, see [Rotating your SSL/TLS certificate](#). For more information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB instance or cluster](#). For information about using SSL/TLS with MySQL DB instances, see [SSL/TLS support for MySQL DB instances on Amazon RDS](#).

Topics

- [Determining whether any applications are connecting to your MySQL DB instance using SSL](#)
- [Determining whether a client requires certificate verification to connect](#)
- [Updating your application trust store](#)

- [Example Java code for establishing SSL connections](#)

Determining whether any applications are connecting to your MySQL DB instance using SSL

If you are using Amazon RDS for MySQL version 5.7, 8.0, or 8.4 and the Performance Schema is enabled, run the following query to check if connections are using SSL/TLS. For information about enabling the Performance Schema, see [Performance Schema quick start](#) in the MySQL documentation.

```
mysql> SELECT id, user, host, connection_type
    FROM performance_schema.threads pst
    INNER JOIN information_schema.processlist isp
    ON pst.processlist_id = isp.id;
```

In this sample output, you can see both your own session (`admin`) and an application logged in as `webapp1` are using SSL.

```
+----+-----+-----+-----+
| id | user           | host            | connection_type |
+----+-----+-----+-----+
|  8 | admin          | 10.0.4.249:42590 | SSL/TLS        |
|  4 | event_scheduler | localhost       | NULL           |
| 10 | webapp1        | 159.28.1.1:42189 | SSL/TLS        |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Determining whether a client requires certificate verification to connect

You can check whether JDBC clients and MySQL clients require certificate verification to connect.

JDBC

The following example with MySQL Connector/J 8.0 shows one way to check an application's JDBC connection properties to determine whether successful connections require a valid certificate. For

more information on all of the JDBC connection options for MySQL, see [Configuration properties](#) in the MySQL documentation.

When using the MySQL Connector/J 8.0, an SSL connection requires verification against the DB server certificate if your connection properties have `sslMode` set to `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

Note

If you use either the MySQL Java Connector v5.1.38 or later, or the MySQL Java Connector v8.0.9 or later to connect to your databases, even if you haven't explicitly configured your applications to use SSL/TLS when connecting to your databases, these client drivers default to using SSL/TLS. In addition, when using SSL/TLS, they perform partial certificate verification and fail to connect if the database server certificate is expired.

MySQL

The following examples with the MySQL Client show two ways to check a script's MySQL connection to determine whether successful connections require a valid certificate. For more information on all of the connection options with the MySQL Client, see [Client-side configuration for encrypted connections](#) in the MySQL documentation.

When using the MySQL Client version 5.7 and higher, an SSL connection requires verification against the server CA certificate if for the `--ssl-mode` option you specify `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
--ssl-mode=VERIFY_CA
```

Updating your application trust store

For information about updating the trust store for MySQL applications, see [Installing SSL certificates](#) in the MySQL documentation.

For information about downloading the root certificate, see [Using SSL/TLS to encrypt a connection to a DB instance or cluster](#).

For sample scripts that import certificates, see [Sample script for importing certificates into your trust store](#).

 **Note**

When you update the trust store, you can retain older certificates in addition to adding the new certificates.

If you are using the mysql JDBC driver in an application, set the following properties in the application.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

When you start the application, set the following properties.

```
java -Djavax.net.ssl.trustStore=/path_to_trust_store/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_trust_store_password com.companyName.MyApplication
```

 **Note**

Specify a password other than the prompt shown here as a security best practice.

Example Java code for establishing SSL connections

The following code example shows how to set up the SSL connection that validates the server certificate using JDBC.

```
public class MySQLSSLTest {

    private static final String DB_USER = "username";
    private static final String DB_PASSWORD = "password";
    // This trust store has only the prod root ca.
    private static final String TRUST_STORE_FILE_PATH = "file-path-to-trust-store";
    private static final String TRUST_STORE_PASS = "trust-store-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");

        System.setProperty("javax.net.ssl.trustStore", TRUST_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", TRUST_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            connection =
DriverManager.getConnection("jdbc:mysql://mydatabase.123456789012.us-
east-1.rds.amazonaws.com:3306",properties);
            stmt = connection.createStatement();
            rs=stmt.executeQuery("SELECT 1 from dual");
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                }
            }
            if (stmt != null) {
```

```
        try {
            stmt.close();
        } catch (SQLException e) {
        }
    }
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
return;
}
```

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-rsa2048-g1 certificates. For instructions, see step 3 in [Updating your CA certificate by modifying your DB instance or cluster](#).

Specify a password other than the prompt shown here as a security best practice.

Using Kerberos authentication for Amazon RDS for MySQL

You can use Kerberos authentication to authenticate users when they connect to your MySQL DB instance. The DB instance works with AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) to enable Kerberos authentication. When users authenticate with a MySQL DB instance joined to the trusting domain, authentication requests are forwarded. Forwarded requests go to the domain directory that you create with AWS Directory Service.

Keeping all of your credentials in the same directory can save you time and effort. With this approach, you have a centralized place for storing and managing credentials for multiple DB instances. Using a directory can also improve your overall security profile.

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of Amazon RDS with Kerberos authentication, see [Supported Regions and DB engines for Kerberos authentication in Amazon RDS](#).

Overview of Setting up Kerberos authentication for MySQL DB instances

To set up Kerberos authentication for a MySQL DB instance, complete the following general steps, described in more detail later:

1. Use AWS Managed Microsoft AD to create an AWS Managed Microsoft AD directory. You can use the AWS Management Console, the AWS CLI, or the AWS Directory Service to create the directory. For details about doing so, see [Create your AWS Managed Microsoft AD directory](#) in the *AWS Directory Service Administration Guide*.
2. Create an AWS Identity and Access Management (IAM) role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`. The role allows Amazon RDS to make calls to your directory.

For the role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see [Activating and deactivating AWS STS in an AWS Region](#) in the *IAM User Guide*.

3. Create and configure users in the AWS Managed Microsoft AD directory using the Microsoft Active Directory tools. For more information about creating users in your Active Directory, see [Manage users and groups in AWS managed Microsoft AD](#) in the *AWS Directory Service Administration Guide*.
4. Create or modify a MySQL DB instance. If you use either the CLI or RDS API in the create request, specify a domain identifier with the `Domain` parameter. Use the `d-*` identifier that was generated when you created your directory and the name of the role that you created.

If you modify an existing MySQL DB instance to use Kerberos authentication, set the domain and IAM role parameters for the DB instance. Locate the DB instance in the same VPC as the domain directory.

5. Use the Amazon RDS master user credentials to connect to the MySQL DB instance. Create the user in MySQL using the `CREATE USER` clause `IDENTIFIED WITH 'auth_pam'`. Users that you create this way can log in to the MySQL DB instance using Kerberos authentication.

Setting up Kerberos authentication for MySQL DB instances

You use AWS Managed Microsoft AD to set up Kerberos authentication for a MySQL DB instance. To set up Kerberos authentication, you take the following steps.

Step 1: Create a directory using AWS Managed Microsoft AD

AWS Directory Service creates a fully managed Active Directory in the AWS Cloud. When you create an AWS Managed Microsoft AD directory, AWS Directory Service creates two domain controllers and Domain Name System (DNS) servers on your behalf. The directory servers are created in different subnets in a VPC. This redundancy helps make sure that your directory remains accessible even if a failure occurs.

When you create an AWS Managed Microsoft AD directory, AWS Directory Service performs the following tasks on your behalf:

- Sets up an Active Directory within the VPC.
- Creates a directory administrator account with the user name Admin and the specified password. You use this account to manage your directory.

 **Note**

Be sure to save this password. AWS Directory Service doesn't store it. You can reset it, but you can't retrieve it.

- Creates a security group for the directory controllers.

When you launch an AWS Managed Microsoft AD, AWS creates an Organizational Unit (OU) that contains all of your directory's objects. This OU has the NetBIOS name that you typed when you created your directory and is located in the domain root. The domain root is owned and managed by AWS.

The Admin account that was created with your AWS Managed Microsoft AD directory has permissions for the most common administrative activities for your OU:

- Create, update, or delete users
- Add resources to your domain such as file or print servers, and then assign permissions for those resources to users in your OU
- Create additional OUs and containers

- Delegate authority
- Restore deleted objects from the Active Directory Recycle Bin
- Run AD and DNS Windows PowerShell modules on the Active Directory Web Service

The Admin account also has rights to perform the following domain-wide activities:

- Manage DNS configurations (add, remove, or update records, zones, and forwarders)
- View DNS event logs
- View security event logs

To create a directory with AWS Managed Microsoft AD

1. Sign in to the AWS Management Console and open the AWS Directory Service console at <https://console.aws.amazon.com/directorieservicev2/>.
2. In the navigation pane, choose **Directories** and choose **Set up Directory**.
3. Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option that you can currently use with Amazon RDS.
4. Enter the following information:

Directory DNS name

The fully qualified name for the directory, such as **corp.example.com**.

Directory NetBIOS name

The short name for the directory, such as **CORP**.

Directory description

(Optional) A description for the directory.

Admin password

The password for the directory administrator. The directory creation process creates an administrator account with the user name Admin and this password.

The directory administrator password and can't include the word "admin." The password is case-sensitive and must be 8–64 characters in length. It must also contain at least one character from three of the following four categories:

- Lowercase letters (a–z)
- Uppercase letters (A–Z)
- Numbers (0–9)
- Non-alphanumeric characters (~!@#\$%^&*_+=`|\(){}[],;"'<>,.?/)

Confirm password

The administrator password retyped.

5. Choose **Next**.
6. Enter the following information in the **Networking** section and then choose **Next**:

VPC

The VPC for the directory. Create the MySQL DB instance in this same VPC.

Subnets

Subnets for the directory servers. The two subnets must be in different Availability Zones.

7. Review the directory information and make any necessary changes. When the information is correct, choose **Create directory**.

Review & create

Review

Directory type

Microsoft AD

VPC

vpc-8b6b78e9 ([REDACTED])

Directory DNS name

corp.example.com

Subnets

subnet-75128d10 ([REDACTED], us-east-1a)

subnet-f51665dd ([REDACTED], us-east-1b)

Directory NetBIOS name

CORP

Directory description

My directory

Pricing

Edition

Standard

Free trial eligible [Learn more](#)

30-day limited trial

~USD [REDACTED] *

* Includes two domain controllers, USD [REDACTED]/mo for each additional domain controller.

Cancel

[Previous](#)

Create directory

It takes several minutes for the directory to be created. When it has been successfully created, the **Status** value changes to **Active**.

To see information about your directory, choose the directory name in the directory listing. Note the **Directory ID** value because you need this value when you create or modify your MySQL DB instance.

Directory Service > Directories > d-90670a8d36

Directory details		Reset user password	
Directory type	VPC	Status	
Microsoft AD	vpc-6594f31c	Active	
Edition	Subnets	Last updated	
Standard	subnet-7d36a227	Tuesday, January 7, 2020	
Directory ID	subnet-a2ab49c6	Launch time	
d-90670a8d36		Tuesday, January 7, 2020	
Directory DNS name	Availability zones		
corp.example.com	us-east-1c, us-east-1d		
Directory NetBIOS name	DNS address		
CORP			
Description - Edit			
My directory			

[Application management](#) [Scale & share](#) [Networking & security](#) [Maintenance](#)

Step 2: Create the IAM role for use by Amazon RDS

For Amazon RDS to call AWS Directory Service for you, an IAM role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess` is required. This role allows Amazon RDS to make calls to the AWS Directory Service.

When a DB instance is created using the AWS Management Console and the console user has the `iam:CreateRole` permission, the console creates this role automatically. In this case, the role name is `rds-directoryservice-kerberos-access-role`. Otherwise, you must create the

IAM role manually. When you create this IAM role, choose **Directory Service**, and attach the AWS managed policy `AmazonRDSDirectoryServiceAccess` to it.

For more information about creating IAM roles for a service, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

 **Note**

The IAM role used for Windows Authentication for RDS for SQL Server can't be used for RDS for MySQL.

Optionally, you can create policies with the required permissions instead of using the managed IAM policy `AmazonRDSDirectoryServiceAccess`. In this case, the IAM role must have the following IAM trust policy.

JSON

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

The role must also have the following IAM role policy.

JSON

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "ds:DescribeDirectories",  
                "ds:AuthorizeApplication",  
                "ds:UnauthorizeApplication",  
                "ds:GetAuthorizedApplicationDetails"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

Step 3: Create and configure users

You can create users with the Active Directory Users and Computers tool. This tool is part of the Active Directory Domain Services and Active Directory Lightweight Directory Services tools. Users represent individual people or entities that have access to your directory.

To create users in an AWS Directory Service directory, you must be connected to an Amazon EC2 instance based on Microsoft Windows. This instance must be a member of the AWS Directory Service directory and be logged in as a user that has privileges to create users. For more information, see [Manage users and groups in AWS Managed Microsoft AD](#) in the *AWS Directory Service Administration Guide*.

Step 4: Create or modify a MySQL DB instance

Create or modify a MySQL DB instance for use with your directory. You can use the console, CLI, or RDS API to associate a DB instance with a directory. You can do this in one of the following ways:

- Create a new MySQL DB instance using the console, the [create-db-instance](#) CLI command, or the [CreateDBInstance](#) RDS API operation.

For instructions, see [Creating an Amazon RDS DB instance](#).

- Modify an existing MySQL DB instance using the console, the [modify-db-instance](#) CLI command, or the [ModifyDBInstance](#) RDS API operation.

For instructions, see [Modifying an Amazon RDS DB instance](#).

- Restore a MySQL DB instance from a DB snapshot using the console, the [restore-db-instance-from-db-snapshot](#) CLI command, or the [RestoreDBInstanceFromDBSnapshot](#) RDS API operation.

For instructions, see [Restoring to a DB instance](#).

- Restore a MySQL DB instance to a point-in-time using the console, the [restore-db-instance-to-point-in-time](#) CLI command, or the [RestoreDBInstanceToPointInTime](#) RDS API operation.

For instructions, see [Restoring a DB instance to a specified time for Amazon RDS](#).

Kerberos authentication is only supported for MySQL DB instances in a VPC. The DB instance can be in the same VPC as the directory, or in a different VPC. The DB instance must use a security group that allows egress within the directory's VPC so the DB instance can communicate with the directory.

When you use the console to create, modify, or restore a DB instance, choose **Password and Kerberos authentication** in the **Database authentication** section. Choose **Browse Directory** and then select the directory, or choose **Create a new directory**.

Database authentication

Database authentication options [Info](#)

Password authentication
Authenticates using database passwords.

Password and IAM database authentication
Authenticates using the database password and user credentials through AWS IAM users and roles.

Password and Kerberos authentication
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

Directory

[Browse Directory](#)

When you use the AWS CLI or RDS API, associate a DB instance with a directory. The following parameters are required for the DB instance to use the domain directory you created:

- For the `--domain` parameter, use the domain identifier ("d-*" identifier) generated when you created the directory.
- For the `--domain-iam-role-name` parameter, use the role you created that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

For example, the following CLI command modifies a DB instance to use a directory.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
  --db-instance-identifier mydbinstance \
  --domain d-ID \
  --domain-iam-role-name role-name
```

For Windows:

```
aws rds modify-db-instance ^
  --db-instance-identifier mydbinstance ^
  --domain d-ID ^
  --domain-iam-role-name role-name
```

Important

If you modify a DB instance to enable Kerberos authentication, reboot the DB instance after making the change.

Step 5: Create Kerberos authentication MySQL logins

Use the Amazon RDS master user credentials to connect to the MySQL DB instance as you do any other DB instance. The DB instance is joined to the AWS Managed Microsoft AD domain. Thus, you can provision MySQL logins and users from the Active Directory users in your domain. Database permissions are managed through standard MySQL permissions that are granted to and revoked from these logins.

You can allow an Active Directory user to authenticate with MySQL. To do this, first use the Amazon RDS master user credentials to connect to the MySQL DB instance as with any other DB instance. After you're logged in, create an externally authenticated user with PAM (Pluggable Authentication Modules) in MySQL by running the following command. Replace `testuser` with the user name.

```
CREATE USER 'testuser'@'%' IDENTIFIED WITH 'auth_pam';
```

Users (both humans and applications) from your domain can now connect to the DB instance from a domain joined client machine using Kerberos authentication.

Important

We strongly recommended that clients use SSL/TLS connections when using PAM authentication. If they don't use SSL/TLS connections, the password might be sent as clear text in some cases. To require an SSL/TLS encrypted connection for your AD user, run the following command and replace `testuser` with the user name:

```
ALTER USER 'testuser'@'%' REQUIRE SSL;
```

For more information, see [SSL/TLS support for MySQL DB instances on Amazon RDS](#).

Managing a DB instance in a domain

You can use the CLI or the RDS API to manage your DB instance and its relationship with your managed Active Directory. For example, you can associate an Active Directory for Kerberos authentication and disassociate an Active Directory to disable Kerberos authentication. You can also move a DB instance to be externally authenticated by one Active Directory to another.

For example, using the Amazon RDS API, you can do the following:

- To reattempt enabling Kerberos authentication for a failed membership, use the `ModifyDBInstance` API operation and specify the current membership's directory ID.
- To update the IAM role name for membership, use the `ModifyDBInstance` API operation and specify the current membership's directory ID and the new IAM role.
- To disable Kerberos authentication on a DB instance, use the `ModifyDBInstance` API operation and specify none as the domain parameter.

- To move a DB instance from one domain to another, use the `ModifyDBInstance` API operation and specify the domain identifier of the new domain as the `domain` parameter.
- To list membership for each DB instance, use the `DescribeDBInstances` API operation.

Understanding domain membership

After you create or modify your DB instance, it becomes a member of the domain. You can view the status of the domain membership for the DB instance by running the [describe-db-instances](#) CLI command. The status of the DB instance can be one of the following:

- `kerberos-enabled` – The DB instance has Kerberos authentication enabled.
- `enabling-kerberos` – AWS is in the process of enabling Kerberos authentication on this DB instance.
- `pending-enable-kerberos` – The enabling of Kerberos authentication is pending on this DB instance.
- `pending-maintenance-enable-kerberos` – AWS will attempt to enable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `pending-disable-kerberos` – The disabling of Kerberos authentication is pending on this DB instance.
- `pending-maintenance-disable-kerberos` – AWS will attempt to disable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `enable-kerberos-failed` – A configuration problem has prevented AWS from enabling Kerberos authentication on the DB instance. Check and fix your configuration before reissuing the DB instance modify command.
- `disabling-kerberos` – AWS is in the process of disabling Kerberos authentication on this DB instance.

A request to enable Kerberos authentication can fail because of a network connectivity issue or an incorrect IAM role. For example, suppose that you create a DB instance or modify an existing DB instance and the attempt to enable Kerberos authentication fails. If this happens, re-issue the modify command or modify the newly created DB instance to join the domain.

Connecting to MySQL with Kerberos authentication

To connect to MySQL with Kerberos authentication, you must log in using the Kerberos authentication type.

To create a database user that you can connect to using Kerberos authentication, use an IDENTIFIED WITH clause on the CREATE USER statement. For instructions, see [Step 5: Create Kerberos authentication MySQL logins](#).

To avoid errors, use the MariaDB mysql client. You can download MariaDB software at <https://downloads.mariadb.org/>.

At a command prompt, connect to one of the endpoints associated with your MySQL DB instance. Follow the general procedures in [Connecting to your MySQL DB instance](#). When you're prompted for the password, enter the Kerberos password associated with that user name.

Restoring a MySQL DB instance and adding it to a domain

You can restore a DB snapshot or complete a point-in-time restore for a MySQL DB instance and then add it to a domain. After the DB instance is restored, modify the DB instance using the process explained in [Step 4: Create or modify a MySQL DB instance](#) to add the DB instance to a domain.

Kerberos authentication MySQL limitations

The following limitations apply to Kerberos authentication for MySQL:

- Only an AWS Managed Microsoft AD is supported. However, you can join RDS for MySQL DB instances to shared Managed Microsoft AD domains owned by different accounts in the same AWS Region.
- You must reboot the DB instance after enabling the feature.
- The domain name length can't be longer than 61 characters.
- You can't enable Kerberos authentication and IAM authentication at the same time. Choose one authentication method or the other for your MySQL DB instance.
- Don't modify the DB instance port after enabling the feature.
- Don't use Kerberos authentication with read replicas.
- If you have auto minor version upgrade turned on for a MySQL DB instance that is using Kerberos authentication, you must turn off Kerberos authentication and then turn it back on after an automatic upgrade. For more information about auto minor version upgrades, see [Automatic minor version upgrades for RDS for MySQL](#).
- To delete a DB instance with this feature enabled, first disable the feature. To do so, run the modify-db-instance CLI command for the DB instance and specify none for the --domain parameter.

- If you use the CLI or RDS API to delete a DB instance with this feature enabled, expect a delay.
- RDS for MySQL doesn't support Kerberos authentication across a forest trust between your on-premise or self-hosted AD and the AWS Managed Microsoft AD.

Improving query performance for RDS for MySQL with Amazon RDS Optimized Reads

You can achieve faster query processing for RDS for MySQL with Amazon RDS Optimized Reads. An RDS for MySQL DB instance or Multi-AZ DB cluster that uses RDS Optimized Reads can achieve up to 2x faster query processing compared to a DB instance or cluster that doesn't use it.

Topics

- [Overview of RDS Optimized Reads](#)
- [Use cases for RDS Optimized Reads](#)
- [Best practices for RDS Optimized Reads](#)
- [Using RDS Optimized Reads](#)
- [Monitoring DB instances that use RDS Optimized Reads](#)
- [Limitations for RDS Optimized Reads](#)

Overview of RDS Optimized Reads

When you use an RDS for MySQL DB instance or Multi-AZ DB cluster that has RDS Optimized Reads turned on, it achieves faster query performance through the use of an instance store. An *instance store* provides temporary block-level storage for your DB instance or Multi-AZ DB cluster. The storage is located on Non-Volatile Memory Express (NVMe) solid state drives (SSDs) that are physically attached to the host server. This storage is optimized for low latency, high random I/O performance, and high sequential read throughput.

RDS Optimized Reads is turned on by default when a DB instance or Multi-AZ DB cluster uses a DB instance class with an instance store, such as db.m5d or db.m6gd. With RDS Optimized Reads, some temporary objects are stored on the instance store. These temporary objects include internal temporary files, internal on-disk temp tables, memory map files, and binary log (binlog) cache files. For more information about the instance store, see [Amazon EC2 instance store](#) in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

The workloads that generate temporary objects in MySQL for query processing can take advantage of the instance store for faster query processing. This type of workload includes queries involving sorts, hash aggregations, high-load joins, Common Table Expressions (CTEs), and queries on unindexed columns. These instance store volumes provide higher IOPS and performance,

regardless of the storage configurations used for persistent Amazon EBS storage. Because RDS Optimized Reads offloads operations on temporary objects to the instance store, the input/output operations per second (IOPS) or throughput of the persistent storage (Amazon EBS) can now be used for operations on persistent objects. These operations include regular data file reads and writes, and background engine operations, such as flushing and insert buffer merges.

Note

Both manual and automated RDS snapshots only contain engine files for persistent objects. The temporary objects created in the instance store aren't included in RDS snapshots.

Use cases for RDS Optimized Reads

If you have workloads that rely heavily on temporary objects, such as internal tables or files, for their query execution, then you can benefit from turning on RDS Optimized Reads. The following use cases are candidates for RDS Optimized Reads:

- Applications that run analytical queries with complex common table expressions (CTEs), derived tables, and grouping operations
- Read replicas that serve heavy read traffic with unoptimized queries
- Applications that run on-demand or dynamic reporting queries that involve complex operations, such as queries with GROUP BY and ORDER BY clauses
- Workloads that use internal temporary tables for query processing

You can monitor the engine status variable `created_tmp_disk_tables` to determine the number of disk-based temporary tables created on your DB instance.

- Applications that create large temporary tables, either directly or in procedures, to store intermediate results
- Database queries that perform grouping or ordering on non-indexed columns

Best practices for RDS Optimized Reads

Use the following best practices for RDS Optimized Reads:

- Add retry logic for read-only queries in case they fail because the instance store is full during the execution.

- Monitor the storage space available on the instance store with the CloudWatch metric `FreeLocalStorage`. If the instance store is reaching its limit because of workload on the DB instance, modify the DB instance to use a larger DB instance class.
- When your DB instance or Multi-AZ DB cluster has sufficient memory but is still reaching the storage limit on the instance store, increase the `binlog_cache_size` value to maintain the session-specific binlog entries in memory. This configuration prevents writing the binlog entries to temporary binlog cache files on disk.

The `binlog_cache_size` parameter is session-specific. You can change the value for each new session. The setting for this parameter can increase the memory utilization on the DB instance during peak workload. Therefore, consider increasing the parameter value based on the workload pattern of your application and available memory on the DB instance.

- For MySQL 8.0 versions and lower, use the default value of `MIXED` for the `binlog_format` parameter. Depending on the size of the transactions, setting `binlog_format` to `ROW` can result in large binlog cache files on the instance store. For MySQL 8.4 and higher, use the default value of `ROW` for the `binlog_format` parameter.
- Set the [`internal_tmp_mem_storage_engine`](#) parameter to `TempTable`, and set the [`temptable_max_mmap`](#) parameter to match the size of the available storage on the instance store.
- Avoid performing bulk changes in a single transaction. These types of transactions can generate large binlog cache files on the instance store and can cause issues when the instance store is full. Consider splitting writes into multiple small transactions to minimize storage use for binlog cache files.
- Use the default value of `ABORT_SERVER` for the `binlog_error_action` parameter. Doing so avoids issues with the binary logging on DB instances with backups enabled.

Using RDS Optimized Reads

When you provision an RDS for MySQL DB instance with one of the following DB instance classes in a Single-AZ DB instance deployment, Multi-AZ DB instance deployment, or Multi-AZ DB cluster deployment, the DB instance automatically uses RDS Optimized Reads.

To turn on RDS Optimized Reads, do one of the following:

- Create an RDS for MySQL DB instance or Multi-AZ DB cluster using one of these DB instance classes. For more information, see [Creating an Amazon RDS DB instance](#).

- Modify an existing RDS for MySQL DB instance or Multi-AZ DB cluster to use one of these DB instance classes. For more information, see [Modifying an Amazon RDS DB instance](#).

RDS Optimized Reads is available in all AWS Regions RDS where one or more of the DB instance classes with local NVMe SSD storage are supported. For information about DB instance classes, see [the section called “DB instance classes”](#).

DB instance class availability differs for AWS Regions. To determine whether a DB instance class is supported in a specific AWS Region, see [the section called “Determining DB instance class support in AWS Regions”](#).

If you don't want to use RDS Optimized Reads, modify your DB instance or Multi-AZ DB cluster so that it doesn't use a DB instance class that supports the feature.

Monitoring DB instances that use RDS Optimized Reads

You can monitor DB instances that use RDS Optimized Reads with the following CloudWatch metrics:

- FreeLocalStorage
- ReadIOPSLocalStorage
- ReadLatencyLocalStorage
- ReadThroughputLocalStorage
- WriteIOPSLocalStorage
- WriteLatencyLocalStorage
- WriteThroughputLocalStorage

These metrics provide data about available instance store storage, IOPS, and throughput. For more information about these metrics, see [Amazon CloudWatch instance-level metrics for Amazon RDS](#).

Limitations for RDS Optimized Reads

The following limitations apply to RDS Optimized Reads:

- RDS Optimized Reads is supported for the following versions:
 - RDS for MySQL version 8.0.28 and higher major and minor versions

For information about RDS for MySQL versions, see [MySQL on Amazon RDS versions](#).

- You can't change the location of temporary objects to persistent storage (Amazon EBS) on the DB instance classes that support RDS Optimized Reads.
- When binary logging is enabled on a DB instance, the maximum transaction size is limited by the size of the instance store. In MySQL, any session that requires more storage than the value of `binlog_cache_size` writes transaction changes to temporary binlog cache files, which are created on the instance store.
- Transactions can fail when the instance store is full.

Improving write performance with RDS Optimized Writes for MySQL

You can improve the performance of write transactions with RDS Optimized Writes for MySQL. When your RDS for MySQL database uses RDS Optimized Writes, it can achieve up to two times higher write transaction throughput.

Topics

- [Overview of RDS Optimized Writes](#)
- [Using RDS Optimized Writes](#)
- [Enabling RDS Optimized Writes on an existing database](#)
- [Limitations for RDS Optimized Writes](#)

Overview of RDS Optimized Writes

When you turn on RDS Optimized Writes, your RDS for MySQL databases write only once when flushing data to durable storage without the need for the doublewrite buffer. The databases continue to provide ACID property protections for reliable database transactions, along with improved performance.

Relational databases, like MySQL, provide the *ACID properties* of atomicity, consistency, isolation, and durability for reliable database transactions. To help provide these properties, MySQL uses a data storage area called the *doublewrite buffer* that prevents partial page write errors. These errors occur when there is a hardware failure while the database is updating a page, such as in the case of a power outage. A MySQL database can detect partial page writes and recover with a copy of the page in the doublewrite buffer. While this technique provides protection, it also results in extra write operations. For more information about the MySQL doublewrite buffer, see [Doublewrite Buffer](#) in the MySQL documentation.

With RDS Optimized Writes turned on, RDS for MySQL databases write only once when flushing data to durable storage without using the doublewrite buffer. RDS Optimized Writes is useful if you run write-heavy workloads on your RDS for MySQL databases. Examples of databases with write-heavy workloads include ones that support digital payments, financial trading, and gaming applications.

These databases run on DB instance classes that use the AWS Nitro System. Because of the hardware configuration in these systems, the database can write 16-KiB pages directly to data files reliably and durably in one step. The AWS Nitro System makes RDS Optimized Writes possible.

You can set the new database parameter `rds.optimized_writes` to control the RDS Optimized Writes feature for RDS for MySQL databases. Access this parameter in the DB parameter groups of RDS for MySQL version 8.0 and RDS for MySQL version 8.4. Set the parameter using the following values:

- AUTO – Turn on RDS Optimized Writes if the database supports it. Turn off RDS Optimized Writes if the database doesn't support it. This setting is the default.
- OFF – Turn off RDS Optimized Writes even if the database supports it.

If you have an existing database with an engine version, DB instance class, and/or file system format that doesn't support RDS Optimized Writes, you can enable the feature by creating a blue/green deployment. For more information, see [the section called “Enabling on an existing database”](#).

If you migrate an RDS for MySQL database that is configured to use RDS Optimized Writes to a DB instance class that doesn't support the feature, RDS automatically turns off RDS Optimized Writes for the database.

When RDS Optimized Writes is turned off, the database uses the MySQL doublewrite buffer.

To determine whether an RDS for MySQL database is using RDS Optimized Writes, view the current value of the `innodb_doublewrite` parameter for the database. If the database is using RDS Optimized Writes, this parameter is set to FALSE (0).

Using RDS Optimized Writes

You can turn on RDS Optimized Writes when you create an RDS for MySQL database with the RDS console, the AWS CLI, or the RDS API. RDS Optimized Writes is turned on automatically when both of the following conditions apply during database creation:

- You specify a DB engine version and DB instance class that support RDS Optimized Writes.
 - RDS Optimized Writes is supported for RDS for MySQL version 8.0.30 and higher. For information about RDS for MySQL versions, see [MySQL on Amazon RDS versions](#).
 - RDS Optimized Writes is supported for RDS for MySQL databases that use the following DB instance classes:

- db.m7i
- db.m7g
- db.m6g
- db.m6gd
- db.m6i
- db.m5
- db.m5d
- db.r7i
- db.r7g
- db.r6g
- db.r6gd
- db.r6i
- db.r5
- db.r5b
- db.r5d
- db.x2idn
- db.x2iedn

For information about DB instance classes, see [the section called “DB instance classes”](#).

DB instance class availability differs for AWS Regions. To determine whether a DB instance class is supported in a specific AWS Region, see [the section called “Determining DB instance class support in AWS Regions”](#).

To upgrade your database to a DB instance class that supports RDS Optimized Writes, you can create a blue/green deployment. For more information, see [the section called “Enabling on an existing database”](#).

- In the parameter group associated with the database, the `rds.optimized_writes` parameter is set to AUTO. In default parameter groups, this parameter is always set to AUTO.

If you want to use a DB engine version and DB instance class that support RDS Optimized Writes, but you don't want to use this feature, then specify a custom parameter group when you create the database. In this parameter group, set the `rds.optimized_writes` parameter to OFF. If

you want the database to use RDS Optimized Writes later, you can set the parameter to AUTO to turn it on. For information about creating custom parameter groups and setting parameters, see [Parameter groups for Amazon RDS](#).

For information about creating a DB instance, see [Creating an Amazon RDS DB instance](#).

Console

When you use the RDS console to create an RDS for MySQL database, you can filter for the DB engine versions and DB instance classes that support RDS Optimized Writes. After you turn on the filters, you can choose from the available DB engine versions and DB instance classes.

To choose a DB engine version that supports RDS Optimized Writes, filter for the RDS for MySQL DB engine versions that support it in **Engine version**, and then choose a version.

Engine options

Engine type [Info](#)

- Aurora (MySQL Compatible)
- Aurora (PostgreSQL Compatible)
- MySQL
- MariaDB
- PostgreSQL
- Oracle
- Microsoft SQL Server
- IBM Db2

Edition

MySQL Community

Known issues/limitations
Review the [Known issues/limitations](#) to learn about potential compatibility issues with specific database versions.

Engine version [Info](#)
View the engine versions that support the following database features.

▼ Hide filters

Show versions that support the Multi-AZ DB cluster [Info](#)
Create a Multi-AZ DB cluster with one primary DB instance and two readable standby DB instances. Multi-AZ DB clusters provide up to 2x faster transaction commit latency and automatic failover in typically under 35 seconds.

Show versions that support the Amazon RDS Optimized Writes [Info](#)
Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

Engine Version

MySQL 8.0.31

In the **Instance configuration** section, filter for the DB instance classes that support RDS Optimized Writes, and then choose a DB instance class.

Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

Amazon RDS Optimized Writes - new [Info](#)
 Show instance classes that support Amazon RDS Optimized Writes

DB instance class [Info](#)
 Memory optimized classes (includes r and x classes)

db.r5b.large (supports Amazon RDS Optimized Writes)
2 vCPUs 16 GiB RAM Network: 10,000 Mbps

Include previous generation classes

After you make these selections, you can choose other settings that meet your requirements and finish creating the RDS for MySQL database with the console.

AWS CLI

To create a DB instance by using the AWS CLI, run the [create-db-instance](#) command. Make sure the --engine-version and --db-instance-class values support RDS Optimized Writes. In addition, make sure the parameter group associated with the DB instance has the `rds.optimized_writes` parameter set to AUTO. This example associates the default parameter group with the DB instance.

Example Creating a DB instance that uses RDS Optimized Writes

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --engine mysql \
    --engine-version 8.0.30 \
    --db-instance-class db.r5b.large \
    --manage-master-user-password \
    --master-username admin \
    --allocated-storage 200
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --engine mysql ^
```

```
--engine-version 8.0.30 ^
--db-instance-class db.r5b.large ^
--manage-master-user-password ^
--master-username admin ^
--allocated-storage 200
```

RDS API

You can create a DB instance using the [CreateDBInstance](#) operation. When you use this operation, make sure the EngineVersion and DBInstanceClass values support RDS Optimized Writes. In addition, make sure the parameter group associated with the DB instance has the rds.optimized_writes parameter set to AUTO.

Enabling RDS Optimized Writes on an existing database

In order to modify an existing RDS for MySQL database to turn on RDS Optimized Writes, the database must have been created with a supported DB engine version and DB instance class. In addition, the database must have been created *after* RDS Optimized Writes was released on November 27, 2022, as the required underlying file system configuration is incompatible with that of databases created before it was released. If these conditions are met, you can turn on RDS Optimized Writes by setting the rds.optimized_writes parameter to AUTO.

If your database was *not* created with a supported engine version, instance class, or file system configuration, you can use RDS Blue/Green Deployments to migrate to a supported configuration. While creating the blue/green deployment, do the following:

- Select **Enable Optimized Writes on green database**, then specify an engine version and DB instance class that supports RDS Optimized Writes. For a list of supported engine versions and instance classes, see [Using RDS Optimized Writes](#).
- Under **Storage**, choose **Upgrade storage file system configuration**. This option upgrades the database to a compatible underlying file system configuration.

When you create the blue/green deployment, if the rds.optimized_writes parameter is set to AUTO, RDS Optimized Writes will be automatically enabled on the green environment. You can then switch over the blue/green deployment, which promotes the green environment to be the new production environment.

For more information, see [the section called “Creating a blue/green deployment”](#).

Limitations for RDS Optimized Writes

When you're restoring an RDS for MySQL database from a snapshot, you can only turn on RDS Optimized Writes for the database if all of the following conditions apply:

- The snapshot was created from a database that supports RDS Optimized Writes.
- The snapshot was created from a database that was created *after* RDS Optimized Writes was released.
- The snapshot is restored to a database that supports RDS Optimized Writes.
- The restored database is associated with a parameter group that has the `rds.optimized_writes` parameter set to AUTO.

Upgrades of the RDS for MySQL DB engine

When Amazon RDS supports a new version of a database engine, you can upgrade your DB instances to the new version. There are two kinds of upgrades for MySQL databases: major version upgrades and minor version upgrades.

Major version upgrades

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, you must manually perform major version upgrades of your DB instances. You can initiate a major version upgrade by modifying your DB instance. Before you perform a major version upgrade, we recommend that you follow the instructions in [Major version upgrades for RDS for MySQL](#).

For major version upgrades of Multi-AZ DB instance deployments, Amazon RDS simultaneously upgrades the primary and standby replicas. Your DB instance won't be available until the upgrade completes. For major version upgrades of Multi-AZ DB cluster deployments, Amazon RDS upgrades the cluster member instances one at a time.

Tip

You can minimize the downtime required for a major version upgrade by using a blue/green deployment. For more information, see [Using Amazon RDS Blue/Green Deployments for database updates](#).

Minor version upgrades

Minor version upgrades include only changes that are backward-compatible with existing applications. You can initiate a minor version upgrade manually by modifying your DB instance. Or, you can enable the **Auto minor version upgrade** option when creating or modifying a DB instance. Doing so means that Amazon RDS automatically upgrades your DB instance after testing and approving the new version. For information about performing an upgrade, see [Upgrading a DB instance engine version](#).

When you perform a minor version upgrade of a Multi-AZ DB cluster, Amazon RDS upgrades the reader DB instances one at a time. Then, one of the reader DB instances switches to be the new writer DB instance. Amazon RDS then upgrades the old writer instance (which is now a reader instance).

Note

The downtime for a minor version upgrade of a Multi-AZ DB *instance* deployment can last for several minutes. Multi-AZ DB clusters typically reduce the downtime of minor version upgrades to approximately 35 seconds. When used with RDS Proxy, you can further reduce downtime to one second or less. For more information, see [Amazon RDS Proxy](#). Alternately, you can use an open source database proxy such as [ProxySQL](#), [PgBouncer](#), or the [AWS Advanced JDBC Wrapper Driver](#).

If your MySQL DB instance uses read replicas, then you must upgrade all of the read replicas before upgrading the source instance.

Topics

- [Considerations for MySQL upgrades](#)
- [Finding valid upgrade targets](#)
- [MySQL version numbers](#)
- [RDS version numbers in RDS for MySQL](#)
- [Major version upgrades for RDS for MySQL](#)
- [Testing an RDS for MySQL upgrade](#)
- [Upgrading a MySQL DB instance](#)
- [Automatic minor version upgrades for RDS for MySQL](#)
- [Using a read replica to reduce downtime when upgrading an RDS for MySQL database](#)
- [Monitoring RDS for MySQL engine upgrades with events](#)

Considerations for MySQL upgrades

Amazon RDS takes two or more DB snapshots during the upgrade process. Amazon RDS takes up to two snapshots of the DB instance *before* making any upgrade changes. If the upgrade doesn't work for your databases, you can restore one of these snapshots to create a DB instance running the old version. Amazon RDS takes another snapshot of the DB instance when the upgrade completes. Amazon RDS takes these snapshots regardless of whether AWS Backup manages the backups for the DB instance.

Note

Amazon RDS only takes DB snapshots if you have set the backup retention period for your DB instance to a number greater than 0. To change your backup retention period, see [Modifying an Amazon RDS DB instance](#).

After the upgrade is complete, you can't revert to the previous version of the database engine. If you want to return to the previous version, restore the first DB snapshot taken to create a new DB instance.

You control when to upgrade your DB instance to a new version supported by Amazon RDS. This level of control helps you maintain compatibility with specific database versions and test new versions with your application before deploying in production. When you are ready, you can perform version upgrades at the times that best fit your schedule.

If your DB instance uses read replication, then you must upgrade all of the read replicas before upgrading the source instance.

Finding valid upgrade targets

When you use the AWS Management Console to upgrade a DB instance, it shows the valid upgrade targets for the DB instance. You can also run the following AWS CLI command to identify the valid upgrade targets for a DB instance:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mysql \
--engine-version version_number \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mysql ^
--engine-version version_number ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For example, to identify the valid upgrade targets for a MySQL version 8.0.28 DB instance, run the following AWS CLI command:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mysql \
--engine-version 8.0.28 \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mysql ^
--engine-version 8.0.28 ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

MySQL version numbers

The version numbering sequence for the RDS for MySQL database engine is either in the form of *major.minor.patch.YYYYMMDD* or *major.minor.patch*, for example, 8.0.33.R2.20231201 or 5.7.44. The format used depends on the MySQL engine version. For information about RDS Extended Support version numbering, see [Amazon RDS Extended Support version naming](#).

major

The major version number is both the integer and the first fractional part of the version number, for example, 8.0. A major version upgrade increases the major part of the version number. For example, an upgrade from 5.7.44 to 8.0.33 is a major version upgrade, where 5.7 and 8.0 are the major version numbers.

minor

The minor version number is the third part of the version number, for example, the 33 in 8.0.33.

patch

The patch is the fourth part of the version number, for example, the R2 in 8.0.33.R2. An RDS patch version includes important bug fixes added to a minor version after its release.

YYYYMMDD

The date is the fifth part of the version number, for example, the 20231201 in 8.0.33.R2.20231201. An RDS date version is a security patch that includes important security fixes added to a minor version after its release. It doesn't include any fixes that might change an engine's behavior.

The following table explains the naming scheme for RDS for MySQL version 8.4.

8.4 minor version	Naming scheme
≥ 3	New DB instances use <i>major.minor.patch.YYMMDD</i> , for example, 8.4.3.R2.20241201. Existing DB instances might use <i>major.minor.patch</i> , for example, 8.4.3.R2, until your next major or minor version upgrade.

The following table explains the naming scheme for RDS for MySQL version 8.0.

8.0 minor version	Naming scheme
≥ 33	New DB instances use <i>major.minor.patch.YYMMDD</i> , for example, 8.0.33.R2.20231201. Existing DB instances might use <i>major.minor.patch</i> , for example, 8.0.33.R2, until your next major or minor version upgrade.
< 33	Existing DB instances use <i>major.minor.patch</i> , for example, 8.0.32.R2.

The following table explains the naming scheme for RDS for MySQL version 5.7.

5.7 minor version	Naming scheme
≥ 42	New DB instances use <i>major.minor.patch.YYMMDD</i> , for example, 5.7.42.R2.20231201. Existing DB instances might use <i>major.minor.patch</i> , for example, 5.7.42.R2, until your next major or minor version upgrade.

RDS version numbers in RDS for MySQL

RDS version numbers use either the *major.minor.patch* or the *major.minor.patch.YYYYMMDD* naming scheme. Amazon RDS Extended Support versions use the *minor-RDS.YYYYMMDD* minor version naming scheme.

An RDS patch version includes important bug fixes added to a minor version after its release. An RDS date version (*YYYYMMDD*) is a security patch. A security patch doesn't include any fixes that might change the engine's behavior. For information about RDS Extended Support version numbering, see [Amazon RDS Extended Support version naming](#).

You can find out the RDS version number of your RDS for MySQL database with the following SQL query:

```
mysql> select mysql.rds_version();
```

For example, querying an RDS for MySQL 8.0.34 database returns the following output:

```
+-----+
| mysql.rds_version() |
+-----+
| 8.0.34.R2.20231201 |
+-----+
1 row in set (0.01 sec)
```

Major version upgrades for RDS for MySQL

Amazon RDS supports the following in-place upgrades for major versions of the MySQL database engine:

- MySQL 5.7 to MySQL 8.0
- MySQL 8.0 to MySQL 8.4

Note

You can only create MySQL version 5.7, 8.0, and 8.4 DB instances with latest-generation and current-generation DB instance classes.

In some cases, you want to upgrade a DB instance running on a previous-generation DB instance class to a DB instance with a higher MySQL engine version. In these cases, first modify the DB instance to use a latest-generation or current-generation DB instance class. After you do this, you can then modify the DB instance to use the higher MySQL database engine version. For information on Amazon RDS DB instance classes, see [DB instance classes](#).

Topics

- [Overview of MySQL major version upgrades](#)
- [Prechecks for upgrades](#)
- [Rollback after failure to upgrade](#)

Overview of MySQL major version upgrades

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, Amazon RDS doesn't apply major version upgrades automatically; you must manually modify your DB instance. We recommend that you thoroughly test any upgrade before applying it to your production instances.

To perform a major version upgrade, first perform any available OS updates. After OS updates are complete, upgrade to each major version, for example, 5.7 to 8.0 and then 8.0 to 8.4. For information about upgrading an RDS for MySQL Multi-AZ DB cluster, see [Upgrading the engine version of a Multi-AZ DB cluster for Amazon RDS](#). MySQL DB instances created before April 24, 2014, show an available OS update until the update has been applied. For more information on OS updates, see [Applying updates to a DB instance](#).

During a major version upgrade of MySQL, Amazon RDS runs the MySQL binary `mysql_upgrade` to upgrade tables, if necessary. Also, Amazon RDS empties the `slow_log` and `general_log`

tables during a major version upgrade. To preserve log information, save the log contents before the major version upgrade.

MySQL major version upgrades typically complete in about 10 minutes. Some upgrades might take longer because of the DB instance class size or because the instance doesn't follow certain operational guidelines in [Best practices for Amazon RDS](#). If you upgrade a DB instance from the Amazon RDS console, the status of the DB instance indicates when the upgrade is complete. If you upgrade using the AWS Command Line Interface (AWS CLI), use the [describe-db-instances](#) command and check the Status value.

Prechecks for upgrades

Amazon RDS runs prechecks before upgrading to check for incompatibilities. These incompatibilities vary based on the MySQL version being upgraded to.

The prechecks include some that are included with MySQL and some that were created specifically by the Amazon RDS team. For information about the prechecks provided by MySQL, see [Upgrade checker utility](#).

The prechecks run before the DB instance is stopped for the upgrade, meaning that they don't cause any downtime when they run. If the prechecks find an incompatibility, Amazon RDS automatically cancels the upgrade before the DB instance is stopped. Amazon RDS also generates an event for the incompatibility. For more information about Amazon RDS events, see [Working with Amazon RDS event notification](#).

Amazon RDS records detailed information about each incompatibility in the log file PrePatchCompatibility.log. In most cases, the log entry includes a link to the MySQL documentation for correcting the incompatibility. For more information about viewing log files, see [Viewing and listing database log files](#).

Because of the nature of the prechecks, they analyze the objects in your database. This analysis results in resource consumption and increases the time for the upgrade to complete.

Topics

- [Prechecks for upgrades from MySQL 8.0 to 8.4](#)
- [Prechecks for upgrades from MySQL 5.7 to 8.0](#)

Prechecks for upgrades from MySQL 8.0 to 8.4

MySQL 8.4 includes a number of incompatibilities with MySQL 8.0. These incompatibilities can cause problems during an upgrade from MySQL 8.0 to MySQL 8.4. So, some preparation might be required on your database for the upgrade to be successful. The following is a general list of these incompatibilities:

- There must be no tables that use obsolete data types or functions.
- Triggers must not have a missing or empty definer or an invalid creation context.
- There must be no keyword or reserved word violations. Some keywords might be reserved in MySQL 8.4 that were not reserved previously.

For more information, see [Keywords and reserved words](#) in the MySQL documentation.

- There must be no tables in the MySQL 8.0 mysql system database that have the same name as a table used by the MySQL 8.4 data dictionary.
- There must be no obsolete SQL modes defined in your sql_mode system variable setting.
- There must be no tables or stored procedures with individual ENUM or SET column elements that exceed 255 characters or 1020 bytes in length.
- Your MySQL 8.0 installation must not use features that are not supported in MySQL 8.4.

For more information, see [Features removed in MySQL 8.4](#) in the MySQL documentation.

- There must be no foreign key constraint names longer than 64 characters.
- For improved Unicode support, review the following information:
 - Consider converting objects that use the utf8mb3 charset to use the utf8mb4 charset. The utf8mb3 character set is deprecated.
 - Consider using utf8mb4 for character set references instead of utf8, because currently utf8 is an alias for the utf8mb3 charset. If possible, change utf8 to utf8mb4 first, and then upgrade your database.
 - Because older clients can receive an unknown character set error for utf8mb3, upgrade your database clients before upgrading your database.

For more information, see [The utf8mb3 character set \(3-byte UTF-8 unicode encoding\)](#) in the MySQL documentation.

To change the character sets, you can manually perform a backup, restore, and replication of your database. Or you can use Amazon RDS Blue/Green Deployments. For more information, see [Using Amazon RDS Blue/Green Deployments for database updates](#).

When you start an upgrade from MySQL 8.0 to 8.4, Amazon RDS runs prechecks automatically to detect these incompatibilities. For information about upgrading to MySQL 8.4, see [Upgrading MySQL](#) in the MySQL documentation.

These prechecks are mandatory. You can't choose to skip them. The prechecks provide the following benefits:

- They enable you to avoid unplanned downtime during the upgrade.
- If there are incompatibilities, Amazon RDS prevents the upgrade and provides a log for you to learn about them. You can then use the log to prepare your database for the upgrade to MySQL 8.4 by reducing the incompatibilities. For detailed information about removing incompatibilities, see [Preparing your installation for upgrade](#) in the MySQL documentation.

Prechecks for upgrades from MySQL 5.7 to 8.0

MySQL 8.0 includes a number of incompatibilities with MySQL 5.7. These incompatibilities can cause problems during an upgrade from MySQL 5.7 to MySQL 8.0. So, some preparation might be required on your database for the upgrade to be successful. The following is a general list of these incompatibilities:

- There must be no tables that use obsolete data types or functions.
- There must be no orphan *.frm files.
- Triggers must not have a missing or empty definer or an invalid creation context.
- There must be no partitioned table that uses a storage engine that does not have native partitioning support.
- There must be no keyword or reserved word violations. Some keywords might be reserved in MySQL 8.0 that were not reserved previously.

For more information, see [Keywords and reserved words](#) in the MySQL documentation.

- There must be no tables in the MySQL 5.7 mysql system database that have the same name as a table used by the MySQL 8.0 data dictionary.
- There must be no obsolete SQL modes defined in your sql_mode system variable setting.

- There must be no tables or stored procedures with individual ENUM or SET column elements that exceed 255 characters or 1020 bytes in length.
- Before upgrading to MySQL 8.0.13 or higher, there must be no table partitions that reside in shared InnoDB tablespaces.
- There must be no queries and stored program definitions from MySQL 8.0.12 or lower that use ASC or DESC qualifiers for GROUP BY clauses.
- Your MySQL 5.7 installation must not use features that are not supported in MySQL 8.0.

For more information, see [Features removed in MySQL 8.0](#) in the MySQL documentation.

- There must be no foreign key constraint names longer than 64 characters.
- For improved Unicode support, review the following information:
 - Consider converting objects that use the utf8mb3 charset to use the utf8mb4 charset. The utf8mb3 character set is deprecated.
 - Consider using utf8mb4 for character set references instead of utf8, because currently utf8 is an alias for the utf8mb3 charset. If possible, change utf8 to utf8mb4 first, and then upgrade your database.
 - Because older clients can receive an unknown character set error for utf8mb3, upgrade your database clients before upgrading your database.

For more information, see [The utf8mb3 character set \(3-byte UTF-8 unicode encoding\)](#) in the MySQL documentation.

To change the character sets, you can manually perform a backup, restore, and replication of your database. Or you can use Amazon RDS Blue/Green Deployments. For more information, see [Using Amazon RDS Blue/Green Deployments for database updates](#).

When you start an upgrade from MySQL 5.7 to 8.0, Amazon RDS runs prechecks automatically to detect these incompatibilities. For information about upgrading to MySQL 8.0, see [Upgrading MySQL](#) in the MySQL documentation.

These prechecks are mandatory. You can't choose to skip them. The prechecks provide the following benefits:

- They enable you to avoid unplanned downtime during the upgrade.
- If there are incompatibilities, Amazon RDS prevents the upgrade and provides a log for you to learn about them. You can then use the log to prepare your database for the upgrade to MySQL.

8.0 by reducing the incompatibilities. For detailed information about removing incompatibilities, see [Preparing your installation for upgrade](#) in the MySQL documentation and [Upgrading to MySQL 8.0? Here is what you need to know...](#) on the MySQL Server Blog.

Rollback after failure to upgrade

When you upgrade a DB instance from MySQL version 5.7 to MySQL version 8.0 or from MySQL version 8.0 to 8.4, the upgrade can fail. In particular, it can fail if the data dictionary contains incompatibilities that weren't captured by the prechecks. In this case, the database fails to start up successfully in the new MySQL 8.0 or 8.4 version. At this point, Amazon RDS rolls back the changes performed for the upgrade. After the rollback, the MySQL DB instance is running the original version:

- MySQL version 8.0 (for a rollback from MySQL 8.4)
- MySQL version 5.7 (for a rollback from MySQL 8.0)

When an upgrade fails and is rolled back, Amazon RDS generates an event with the event ID RDS-EVENT-0188.

Typically, an upgrade fails because there are incompatibilities in the metadata between the databases in your DB instance and the target MySQL version. When an upgrade fails, you can view the details about these incompatibilities in the `upgradeFailure.log` file. Resolve the incompatibilities before attempting to upgrade again.

During an unsuccessful upgrade attempt and rollback, your DB instance is restarted. Any pending parameter changes are applied during the restart and persist after the rollback.

For more information about upgrading to MySQL 8.0, see the following topics in the MySQL documentation:

- [Preparing Your Installation for Upgrade](#)
- [Upgrading to MySQL 8.0? Here is what you need to know...](#)

For more information about upgrading to MySQL 8.4, see [Preparing Your Installation for Upgrade](#) in the MySQL documentation.

Testing an RDS for MySQL upgrade

Before you perform a major version upgrade on your DB instance, thoroughly test your database for compatibility with the new version. In addition, thoroughly test all applications that access the database for compatibility with the new version. We recommend that you use the following procedure.

To test a major version upgrade

1. Review the upgrade documentation for the new version of the database engine to see if there are compatibility issues that might affect your database or applications:
 - [Changes in MySQL 5.7](#)
 - [Changes in MySQL 8.0](#)
 - [Changes in MySQL 8.4](#)
2. If your DB instance is a member of a custom DB parameter group, create a new DB parameter group with your existing settings that is compatible with the new major version. Specify the new DB parameter group when you upgrade your test instance, so your upgrade testing ensures that it works correctly. For more information about creating a DB parameter group, see [Parameter groups for Amazon RDS](#).
3. Create a DB snapshot of the DB instance to be upgraded. For more information, see [Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS](#).
4. Restore the DB snapshot to create a new test DB instance. For more information, see [Restoring to a DB instance](#).
5. Modify this new test DB instance to upgrade it to the new version, using one of the methods detailed following. If you created a new parameter group in step 2, specify that parameter group.
6. Evaluate the storage used by the upgraded instance to determine if the upgrade requires additional storage.
7. Run as many of your quality assurance tests against the upgraded DB instance as needed to ensure that your database and application work correctly with the new version. Implement any new tests needed to evaluate the impact of any compatibility issues that you identified in step 1. Test all stored procedures and functions. Direct test versions of your applications to the upgraded DB instance.

8. If all tests pass, then perform the upgrade on your production DB instance. We recommend that you don't allow write operations to the DB instance until you confirm that everything is working correctly.

Upgrading a MySQL DB instance

For information about manually or automatically upgrading a MySQL DB instance, see [Upgrading a DB instance engine version](#).

Automatic minor version upgrades for RDS for MySQL

If you specify the following settings when creating or modifying a DB instance, you can have your DB instance automatically upgraded.

- The **Auto minor version upgrade** setting is enabled.
- The **Backup retention period** setting is greater than 0.

In the AWS Management Console, these settings are under **Additional configuration**. The following image shows the **Auto minor version upgrade** setting.

Maintenance

Auto minor version upgrade [Info](#)

Enable auto minor version upgrade
Enabling auto minor version upgrade will automatically upgrade to new minor versions as they are released. The automatic upgrades occur during the maintenance window for the database.

Maintenance window [Info](#)
Select the period you want pending modifications or maintenance applied to the database by Amazon RDS.

Select window
 No preference

Start day	Start time	Duration
Monday ▾	00 ▾ : 00 ▾ UTC	0.5 ▾ hours

For more information about these settings, see [Settings for DB instances](#).

For some RDS for MySQL major versions in some AWS Regions, one minor version is designated by RDS as the automatic upgrade version. After a minor version has been tested and approved by

Amazon RDS, the minor version upgrade occurs automatically during your maintenance window. RDS doesn't automatically set newer released minor versions as the automatic upgrade version. Before RDS designates a newer automatic upgrade version, several criteria are considered, such as the following:

- Known security issues
- Bugs in the MySQL community version
- Overall fleet stability since the minor version was released

You can run the following AWS CLI command to determine the current automatic minor upgrade target version for a specified MySQL minor version in a specific AWS Region.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mysql \
--engine-version minor_version \
--region region \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade, EngineVersion:EngineVersion}" \
--output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mysql ^
--engine-version minor_version ^
--region region ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade, EngineVersion:EngineVersion}" ^
--output text
```

For example, the following AWS CLI command determines the automatic minor upgrade target for MySQL minor version 8.0.11 in the US East (Ohio) AWS Region (us-east-2).

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mysql \
```

```
--engine-version 8.0.11 \
--region us-east-2 \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mysql ^
--engine-version 8.0.11 ^
--region us-east-2 ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output table
```

Your output is similar to the following.

```
-----
|   DescribeDBEngineVersions   |
+-----+-----+
|   AutoUpgrade |   EngineVersion   |
+-----+-----+
|   False      |   8.0.15      |
|   False      |   8.0.16      |
|   False      |   8.0.17      |
|   False      |   8.0.19      |
|   False      |   8.0.20      |
|   False      |   8.0.21      |
|   True      |   8.0.23      |
|   False      |   8.0.25      |
+-----+-----+
```

In this example, the AutoUpgrade value is True for MySQL version 8.0.23. So, the automatic minor upgrade target is MySQL version 8.0.23, which is highlighted in the output.

A MySQL DB instance is automatically upgraded during your maintenance window if the following criteria are met:

- The **Auto minor version upgrade** setting is enabled.
- The **Backup retention period** setting is greater than 0.

- The DB instance is running a minor DB engine version that is less than the current automatic upgrade minor version.

For more information, see [Automatically upgrading the minor engine version](#).

Using a read replica to reduce downtime when upgrading an RDS for MySQL database

In most cases, a blue/green deployment is the best option to reduce downtime when upgrading a MySQL DB instance. For more information, see [Using Amazon RDS Blue/Green Deployments for database updates](#).

If you can't use a blue/green deployment and your MySQL DB instance is currently in use with a production application, you can use the following procedure to upgrade the database version for your DB instance. This procedure can reduce the amount of downtime for your application.

By using a read replica, you can perform most of the maintenance steps ahead of time and minimize the necessary changes during the actual outage. With this technique, you can test and prepare the new DB instance without making any changes to your existing DB instance.

The following procedure shows an example of upgrading from MySQL version 5.7 to MySQL version 8.0. You can use the same general steps for upgrades to other major versions. You can use the same general steps for upgrades to other major versions.

Note

When you are upgrading from MySQL version 5.7 to MySQL version 8.0, or from MySQL version 8.0 to MySQL version 8.4, complete the prechecks before performing the upgrade.

For more information, see [Prechecks for upgrades from MySQL 5.7 to 8.0](#) and [Prechecks for upgrades from MySQL 8.0 to 8.4](#).

To upgrade a MySQL database while a DB instance is in use

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Create a read replica of your MySQL 5.7 DB instance. This process creates an upgradable copy of your database. Other read replicas of the DB instance might also exist.

- a. In the console, choose **Databases**, and then choose the DB instance that you want to upgrade.
 - b. For **Actions**, choose **Create read replica**.
 - c. Provide a value for **DB instance identifier** for your read replica and ensure that the **DB instance class** and other settings match your MySQL 5.7 DB instance.
 - d. Choose **Create read replica**.
3. (Optional) When the read replica has been created and **Status** shows **Available**, convert the read replica into a Multi-AZ deployment and enable backups.

By default, a read replica is created with backups disabled. Because the read replica ultimately becomes the production DB instance, it is a best practice to configure a Multi-AZ deployment and enable backups.

- a. In the console, choose **Databases**, and then choose the read replica that you just created.
 - b. Choose **Modify**.
 - c. For **Multi-AZ deployment**, choose **Create a standby instance**.
 - d. For **Backup Retention Period**, choose a positive nonzero value, such as 3 days, and then choose **Continue**.
 - e. For **Scheduling of modifications**, choose **Apply immediately**.
 - f. Choose **Modify DB instance**.
4. When the read replica **Status** shows **Available**, upgrade the read replica to MySQL 8.0:
- a. In the console, choose **Databases**, and then choose the read replica that you just created.
 - b. Choose **Modify**.
 - c. For **DB engine version**, choose the MySQL 8.0 version to upgrade to, and then choose **Continue**.
 - d. For **Scheduling of modifications**, choose **Apply immediately**.
 - e. Choose **Modify DB instance** to start the upgrade.
5. When the upgrade is complete and **Status** shows **Available**, verify that the upgraded read replica is up-to-date with the source MySQL 5.7 DB instance. To verify, connect to the read replica and run the `SHOW REPLICAS STATUS` command. If the `Seconds_Behind_Master` field is `0`, then replication is up-to-date.

Note

Previous versions of MySQL used SHOW SLAVE STATUS instead of SHOW REPLICA STATUS. If you are using a MySQL version before 8.0.23, then use SHOW SLAVE STATUS.

6. (Optional) Create a read replica of your read replica.

If you want the DB instance to have a read replica after it is promoted to a standalone DB instance, you can create the read replica now.

- a. In the console, choose **Databases**, and then choose the read replica that you just upgraded.
- b. For **Actions**, choose **Create read replica**.
- c. Provide a value for **DB instance identifier** for your read replica and ensure that the **DB instance class** and other settings match your MySQL 5.7 DB instance.
- d. Choose **Create read replica**.

7. (Optional) Configure a custom DB parameter group for the read replica.

If you want the DB instance to use a custom parameter group after it is promoted to a standalone DB instance, you can create the DB parameter group now and associate it with the read replica.

- a. Create a custom DB parameter group for MySQL 8.0. For instructions, see [Creating a DB parameter group in Amazon RDS](#).
- b. Modify the parameters that you want to change in the DB parameter group you just created. For instructions, see [Modifying parameters in a DB parameter group in Amazon RDS](#).
- c. In the console, choose **Databases**, and then choose the read replica.
- d. Choose **Modify**.
- e. For **DB parameter group**, choose the MySQL 8.0 DB parameter group you just created, and then choose **Continue**.
- f. For **Scheduling of modifications**, choose **Apply immediately**.
- g. Choose **Modify DB instance** to start the upgrade.

8. Make your MySQL 8.0 read replica a standalone DB instance.

Important

When you promote your MySQL 8.0 read replica to a standalone DB instance, it is no longer a replica of your MySQL 5.7 DB instance. We recommend that you promote your MySQL 8.0 read replica during a maintenance window when your source MySQL 5.7 DB instance is in read-only mode and all write operations are suspended. When the promotion is completed, you can direct your write operations to the upgraded MySQL 8.0 DB instance to ensure that no write operations are lost.

In addition, we recommend that, before promoting your MySQL 8.0 read replica, you perform all necessary data definition language (DDL) operations on your MySQL 8.0 read replica. An example is creating indexes. This approach avoids negative effects on the performance of the MySQL 8.0 read replica after it has been promoted. To promote a read replica, use the following procedure.

- a. In the console, choose **Databases**, and then choose the read replica that you just upgraded.
 - b. For **Actions**, choose **Promote**.
 - c. Choose **Yes** to enable automated backups for the read replica instance. For more information, see [Introduction to backups](#).
 - d. Choose **Continue**.
 - e. Choose **Promote Read Replica**.
9. You now have an upgraded version of your MySQL database. At this point, you can direct your applications to the new MySQL 8.0 DB instance.

Monitoring RDS for MySQL engine upgrades with events

When you upgrade the engine version of a RDS for MySQL database, Amazon RDS emits a specific event during each phase of the process. To track the progress of an upgrade, you can view or subscribe to these events.

For more information about RDS events, see [Monitoring Amazon RDS events](#).

For detailed information about a specific Amazon RDS event that occurs during your engine upgrade, see [Amazon RDS event categories and event messages](#).

Upgrading a MySQL DB snapshot engine version

With Amazon RDS, you can create a storage volume DB snapshot of your MySQL DB instance. When you create a DB snapshot, the snapshot is based on the engine version used by your DB instance. You can upgrade the engine version for your DB snapshots.

For RDS for MySQL, you can upgrade a version 5.7 snapshot to version 8.0, or a version 8.0 snapshot to version 8.4. You can upgrade encrypted or unencrypted DB snapshots.

To view the available engine versions for your RDS for MySQL DB snapshot, use the following AWS CLI example.

```
aws rds describe-db-engine-versions --engine mysql --include-all --engine-version example-engine-version --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --output text
```

If you don't see results for your snapshot, your engine version might be deprecated. If your engine version is deprecated, we recommend that you upgrade to the newest major version upgrade target or to one of the other available upgrade targets for that version. For more information, see [Upgrade options for DB snapshots with unsupported engine versions for RDS for MySQL](#).

After restoring a DB snapshot upgraded to a new engine version, make sure to test that the upgrade was successful. For more information about a major version upgrade, see [the section called "Upgrades of the MySQL DB engine"](#). To learn how to restore a DB snapshot, see [the section called "Restoring to a DB instance"](#).

Note

You can't upgrade automated DB snapshots that were created during the automated backup process.

You can upgrade a DB snapshot using the AWS Management Console, AWS CLI, or RDS API.

Console

To upgrade a DB snapshot engine version using the AWS Management Console, use the following procedure.

To upgrade a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot that you want to upgrade.
4. For **Actions**, choose **Upgrade snapshot**. The **Upgrade snapshot** page appears.
5. Choose the **New engine version** to upgrade to.
6. Choose **Save changes** to upgrade the snapshot.

During the upgrade process, all snapshot actions are disabled for this DB snapshot. Also, the DB snapshot status changes from **Available** to **Upgrading**, and then changes to **Active** upon completion. If the DB snapshot can't be upgraded because of snapshot corruption issues, the status changes to **Unavailable**. You can't recover the snapshot from this state.

 **Note**

If the DB snapshot upgrade fails, the snapshot is rolled back to the original state with the original version.

AWS CLI

To upgrade a DB snapshot to a new database engine version, run the AWS CLI [modify-db-snapshot](#) command.

Options

- **--db-snapshot-identifier** – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see [Amazon Resource Names \(ARNs\) in Amazon RDS](#).
- **--engine-version** – The engine version to upgrade the DB snapshot to.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot \
```

```
--db-snapshot-identifier my_db_snapshot \
--engine-version new_version
```

For Windows:

```
aws rds modify-db-snapshot ^
--db-snapshot-identifier my_db_snapshot ^
--engine-version new_version
```

Amazon RDS API

To upgrade a DB snapshot to a new database engine version, call the RDS API [ModifyDBSnapshot](#) operation.

Parameters

- **DBSnapshotIdentifier** – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see [Amazon Resource Names \(ARNs\) in Amazon RDS](#).
- **EngineVersion** – The engine version to upgrade the DB snapshot to.

Upgrade options for DB snapshots with unsupported engine versions for RDS for MySQL

The following table shows which engine versions you can upgrade to from an unsupported engine version for RDS for MySQL DB snapshots.

Note

You might have to upgrade your DB snapshot more than once to upgrade to your chosen engine version.

DB snapshot engine version	Engine versions available for upgrade
5.5.8	5.5.62, 5.6.51
5.5.12	5.5.62, 5.6.51

DB snapshot engine version	Engine versions available for upgrade
5.5.20	5.5.62, 5.6.51
5.5.23	5.5.62, 5.6.51
5.5.25a	5.5.62, 5.6.51
5.5.27	5.5.62, 5.6.51
5.5.31	5.5.62, 5.6.51
5.5.33	5.5.62, 5.6.51
5.5.37	5.5.62, 5.6.51
5.5.38	5.5.62, 5.6.51
5.5.40	5.5.62, 5.6.51
5.5.40a	5.5.62, 5.6.51
5.5.40b	5.5.62, 5.6.51
5.5.41	5.5.62, 5.6.51
5.5.42	5.5.62, 5.6.51
5.5.59	5.5.62, 5.6.51
5.6.12	5.6.51, 5.7.44
5.6.13	5.6.51, 5.7.44
5.6.17	5.6.51, 5.7.44
5.6.19	5.6.51, 5.7.44
5.6.19a	5.6.51, 5.7.44
5.6.19b	5.6.51, 5.7.44

DB snapshot engine version	Engine versions available for upgrade
5.6.21	5.6.51, 5.7.44
5.6.21b	5.6.51, 5.7.44
5.6.22	5.6.51, 5.7.44
5.6.23	5.6.51, 5.7.44
5.6.27	5.6.51, 5.7.44
5.6.27a	5.6.51, 5.7.44
5.7.10	5.7.44, 5.7.44-rds.20240408, 5.7.44-rds.20240529, 5.7.44-rds.20250103, 5.7.44-rds.20250213, 8.0.32, 8.0.33, 8.0.34, 8.0.35, 8.0.36, 8.0.37, 8.0.39, 8.0.40, 8.0.41
5.7.11	5.7.44, 5.7.44-rds.20240408, 5.7.44-rds.20240529, 5.7.44-rds.20250103, 5.7.44-rds.20250213, 8.0.32, 8.0.33, 8.0.34, 8.0.35, 8.0.36, 8.0.37, 8.0.39, 8.0.40, 8.0.41
5.7.12	5.7.44, 5.7.44-rds.20240408, 5.7.44-rds.20240529, 5.7.44-rds.20250103, 5.7.44-rds.20250213, 8.0.32, 8.0.33, 8.0.34, 8.0.35, 8.0.36, 8.0.37, 8.0.39, 8.0.40, 8.0.41

Importing data into an Amazon RDS for MySQL DB instance

You can use several different techniques to import data into an RDS for MySQL DB instance. The best approach depends on a number of factors:

- Source of the data
- Amount of data
- One-time import or ongoing
- Amount of downtime

If you are also migrating an application with the data, the amount of downtime is important to consider.

The following table lists techniques to importing data into an RDS for MySQL DB instance:

Source	Amount of data	One time or ongoing	Application downtime	Technique	More information
Existing MySQL database on premises or on Amazon EC2	Any	One time	Some	Create a backup of your on-premises database, store it on Amazon S3, and then restore the backup file to a new Amazon RDS DB instance running MySQL.	Restoring a backup into an Amazon RDS for MySQL DB instance
Existing MySQL database on premises or on Amazon EC2	Any	Ongoing	Minimal	Configure replication with an existing MySQL database as the replication source.	Configuring binary log file position replication with an

Source	Amount of data	One time or ongoing	Application downtime	Technique	More information
					external source instance Importing data to an Amazon RDS for MySQL database with reduced downtime

Source	Amount of data	One time or ongoing	Application downtime	Technique	More information
Any existing database	Any	One time or ongoing	Minimal	Use AWS Database Migration Service to migrate the database with minimal downtime and, for many database DB engines, continue ongoing replication.	What is AWS Database Migration Service and Using a MySQL-compatible database as a target for AWS DMS in the <i>AWS Database Migration Service User Guide</i>
Existing MySQL DB instance	Any	One time or ongoing	Minimal	Create a read replica for ongoing replication. Promote the read replica for one-time creation of a new DB instance.	Working with DB instance read replicas

Source	Amount of data	One time or ongoing	Application downtime	Technique	More information
Existing MySQL database	Small	One time	Some	Copy the data directly to your MySQL DB instance using a command-line utility.	Importing data from an external MySQL database to an Amazon RDS for MySQL DB instance
Data not stored in an existing database	Medium	One time	Some	Create flat files and import them using MySQL LOAD DATA LOCAL INFILE statements.	Importing data from any source to an Amazon RDS for MySQL DB instance

Note

The mysql system database contains authentication and authorization information required to log in to your DB instance and access your data. Dropping, altering, renaming, or truncating tables, data, or other contents of the mysql database in your DB instance can result in error and might render the DB instance and your data inaccessible. If this occurs, you can restore the DB instance from a snapshot using the AWS CLI [restore-db-instance-](#)

[from-db-snapshot](#) command. You can recover the DB instance using the AWS CLI [restore-db-instance-to-point-in-time](#) command.

Importing data considerations for MySQL

The following content contains technical information related to loading data into MySQL. This content is aimed at users who are familiar with the MySQL server architecture.

Binary logging

Enabling binary logging reduces data load performance and requires up to four times additional disk space compared to disabled logging. The transaction size used to load the data directly affects system performance and disk space needs—larger transactions require more resources.

Transaction size

Transaction size influences the following aspects of MySQL data loads:

- Resource consumption
- Disk space utilization
- Resume process
- Time to recover
- Input format (flat files or SQL)

This section describes how transaction size affects binary logging and makes the case for disabling binary logging during large data loads. You can enable and disable binary logging by setting the Amazon RDS automated backup retention period. Non-zero values enable binary logging, and zero disables it. For more information, see [Backup retention period](#).

This section also describes the impact of large transactions on InnoDB and why it's important to keep transaction sizes small.

Small transactions

For small transactions, binary logging doubles the number of disk writes required to load the data. This effect can severely degrade performance for other database sessions and increase the time required to load the data. The degradation experienced depends in part on the following factors:

- Upload rate
- Other database activity taking place during the load
- Capacity of your Amazon RDS DB instance

The binary logs also consume disk space roughly equal to the amount of data loaded until the logs are backed up and removed. Amazon RDS minimizes this by frequently backing up and removing binary logs.

Large transactions

For large transactions, binary logging triples IOPS and disk usage for the following reasons:

- The binary log cache stores transaction data temporarily on disk.
- This cache grows with the transaction size, which consumes disk space.
- When the transaction (commit or rollback) completes, the system copies the cache to the binary log.

This process creates three copies of the data:

- The original data
- The cache on disk
- The final binary log entry

Each write operation incurs additional IO, further impacting performance.

Because of this, binary logging requires triple the disk space compared to disabled logging. For example, loading 10 GiB of data as a single transaction creates three copies:

- 10 GiB for the table data
- 10 GiB for the binary log cache
- 10 GiB for the binary log file

The total temporary disk space required is 30 GiB.

Important disk space considerations:

- The cache file persists until either the session ends or a new transaction creates another cache.
- The binary log remains until it's backed up, potentially holding 20 GiB (cache and log) for an extended period.

If you use `LOAD DATA LOCAL INFILE` to load the data, data recovery creates a fourth copy in case the database has to be recovered from a backup made before the load. During recovery, MySQL extracts the data from the binary log into a flat file. MySQL then runs `LOAD DATA LOCAL INFILE`. Building on the preceding example, this recovery requires a total temporary disk space of 40 GiB, or 10 GiB each for table, cache, log, and local file. Without at least 40 GiB of free disk space, recovery fails.

Optimizing large data loads

For large data loads, disable binary logging to reduce overhead and disk space requirements. You can disable binary logging by setting the backup retention period to 0. After loading completes, restore the backup retention period to the appropriate non-zero value. For more information, see [Modifying an Amazon RDS DB instance](#) and [Backup retention period](#) in the settings table.

Note

If the DB instance is a source DB instance for read replicas, then you can't set the backup retention period to 0.

Before loading the data, we recommend that you create a DB snapshot. For more information, see [Managing manual backups](#).

InnoDB

The following information about undo logging and recovery options supports keeping InnoDB transactions small to optimize database performance.

Understanding InnoDB undo logging

Undo is a logging mechanism that enables transaction rollback and supports multi-version concurrency control (MVCC).

For MySQL 5.7 and lower versions, undo logs are stored in the InnoDB system tablespace (usually `ibdata1`) and are retained until the purge thread removes them. As a result, large data load

transactions can cause the system tablespace to become quite large and consume disk space that you can't reclaim unless you recreate the database.

For all MySQL versions, the purge thread must wait to remove any undo logs until the oldest active transaction either commits or rolls back. If the database is processing other transactions during the load, their undo logs also accumulate and can't be removed, even if the transactions commit and no other transaction needs the undo logs for MVCC. In this situation, all transactions—including read-only transactions—slow down. This slowdown occurs because all transactions access all rows that any transaction—not just the load transaction—changes. In effect, transactions must scan through undo logs that long-running load transactions prevented from being purged during an undo log cleanup. This affects performance for any operation accessing modified rows.

InnoDB transaction recovery options

Although InnoDB optimizes commit operations, large transaction rollbacks are slow. For faster recovery, perform a point-in-time recovery or restore a DB snapshot. For more information, see [Point-in-time recovery](#) and [Restoring to a DB instance](#).

Data import formats

MySQL supports two data import formats: flat files and SQL. Review the information about each format to determine the best option for your needs.

Flat files

For small transactions, load flat files with `LOAD DATA LOCAL INFILE`. This data import format can provide the following benefits over using SQL:

- Less network traffic
- Lower data transmission costs
- Decreased database processing overhead
- Faster processing

`LOAD DATA LOCAL INFILE` loads the entire flat file as one transaction. Keep the size of the individual files small for the following advantages:

- **Resume capability** – You can keep track of which files have been loaded. If a problem arises during the load, you can pick up where you left off. You might need to retransmit some data to Amazon RDS, but with small files, the amount retransmitted is minimal.

- **Parallel data loading** – If you have sufficient IOPS and network bandwidth for a single file load, loading in parallel could save time.
- **Load rate control** – If your data load has a negative impact on other processes, you can control the load rate by increasing the interval between files.

Large transactions reduce the benefits of using `LOAD DATA LOCAL INFILE` to import data. When you can't break a large amount of data into smaller files, consider using SQL.

SQL

SQL has one main advantage over flat files: you can easily keep transaction sizes small. However, SQL can take significantly longer to load than flat files. Also, after a failure, it can be difficult to determine where to resume—you can't restart mysqldump files. If a failure occurs while loading a mysqldump file, you must modify or replace the file before the load can resume. Or, alternatively, after you correct the cause of the failure, you can restore to the point in time before the load and resend the file. For more information, see [Point-in-time recovery](#).

Using Amazon RDS DB snapshots for database checkpoints

If you load data over long durations—such as hours or days—with binary logging, use DB snapshots to provide periodic checkpoints for data safety. Each DB snapshot creates a consistent copy of your database instance that serves as a recovery point during system failures or data corruption events. Because DB snapshots are fast, frequent checkpointing has minimal impact on load performance. You can delete previous DB snapshots without impacting database durability or recovery capabilities. For more information about DB snapshots, see [Managing manual backups](#).

Reducing database load times

The following items are additional tips to reduce load times:

- Create all secondary indexes before loading data into MySQL databases. Unlike other database systems, MySQL rebuilds the entire table when adding or modifying secondary indexes. This process creates a new table with index changes, copies all data, and drops the original table.
- Load data in primary key order. For InnoDB tables, this can reduce load times by 75%–80% and reduce data file size by 50%.
- Disable foreign key constraints by setting `foreign_key_checks` to 0. This is often required for flat files loaded with `LOAD DATA LOCAL INFILE`. For any load, disabling foreign key

checks accelerates data loading. After loading completes, re-enable constraints by setting `foreign_key_checks` to 1 and verify the data.

- Load data in parallel unless approaching a resource limit. To enable concurrent loading across multiple table segments, use partitioned tables when appropriate.
- To reduce SQL execution overhead, combine multiple `INSERT` statements into single multi-value `INSERT` operations. `mysqldump` implements this optimization automatically.
- Reduce InnoDB log IO operations by setting `innodb_flush_log_at_trx_commit` to 0. After loading completes, restore `innodb_flush_log_at_trx_commit` to 1.

 **Warning**

Setting `innodb_flush_log_at_trx_commit` to 0 causes InnoDB to flush its logs every second instead of at each commit. This setting increases performance but can risk transaction loss during system failures.

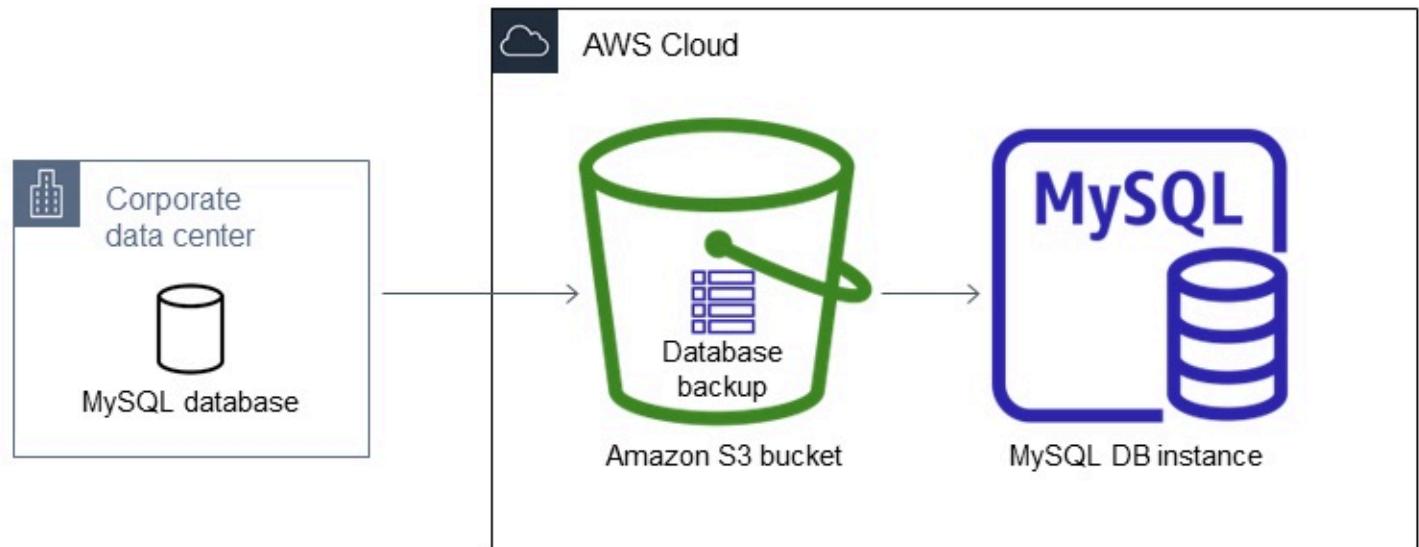
- If you are loading data into a DB instance that doesn't have read replicas, set `sync_binlog` to 0. After loading completes, restore `sync_binlog` parameter to 1.
- Load data into a Single-AZ DB instance before converting the DB instance to a Multi-AZ deployment. If the DB instance already uses a Multi-AZ deployment, we don't recommend switching to a Single-AZ deployment for data loading. Doing so only provides marginal improvements.

Restoring a backup into an Amazon RDS for MySQL DB instance

Amazon RDS supports importing MySQL databases with backup files. You can create a backup of your database, store the backup file on Amazon S3, and then restore the backup file to a new Amazon RDS DB instance running MySQL. Amazon RDS supports importing backup files from Amazon S3 in all AWS Regions.

The scenario described in this section restores a backup of an on-premises database. As long as the database is accessible, you can use this technique for databases in other locations, such as Amazon EC2 or other cloud services.

The following diagram shows the supported scenario.



If your on-premises database can be offline while you create, copy, and restore backup files, then we recommend that you use backup files to import your database to Amazon RDS. If your database can't be offline, then you can use one of the following methods:

- **Binary logs** – First, import backup files from Amazon S3 and to Amazon RDS, as explained in this topic. Then use binary log (binlog) replication to update your database. For more information, see [Configuring binary log file position replication with an external source instance](#).
- **AWS Database Migration Service** – Use AWS Database Migration Service to migrate your database to Amazon RDS. For more information, see [What is AWS Database Migration Service?](#)

Overview of setup to import backup files from Amazon S3 to Amazon RDS

To import backup files from Amazon S3 to Amazon RDS, you need the following components:

- An Amazon S3 bucket to store your backup files.

If you already have an Amazon S3 bucket, you can use that bucket. If you don't have an Amazon S3 bucket, create a new one. For more information, see [Creating a bucket](#).

- A backup of your on-premises database created by Percona XtraBackup.

For more information, see [Creating your database backup](#).

- An AWS Identity and Access Management (IAM) role to allow Amazon RDS to access the S3 bucket.

If you already have an IAM role, you can use that role and attach trust and permissions policies to it. For more information, see [Creating an IAM role manually](#).

If you don't have an IAM role, you have two options:

- You can manually create a new IAM role. For more information, see [Creating an IAM role manually](#).
- You can choose for Amazon RDS to create a new IAM role for you. If you want Amazon RDS to create a new IAM role for you, follow the procedure that uses the AWS Management Console in [Importing data from Amazon S3 to a new MySQL DB instance](#) section.

Creating your database backup

Use the Percona XtraBackup software to create your backup. We recommend that you use the latest version of Percona XtraBackup. You can install Percona XtraBackup from [Software Downloads](#) on the Percona website.

Warning

When creating a database backup, XtraBackup might save credentials in the `xtrabackup_info` file. Make sure to confirm that the `tool_command` setting in the `xtrabackup_info` file doesn't contain any sensitive information.

The Percona XtraBackup version that you use depends on the MySQL version that you are backing up.

- **MySQL 8.4** – Use Percona XtraBackup version 8.4.
- **MySQL 8.0** – Use Percona XtraBackup version 8.0.

Note

Percona XtraBackup 8.0.12 and higher versions support migration of all versions of MySQL 8.0. If you are migrating to RDS for MySQL 8.0.32 or higher, you must use Percona XtraBackup 8.0.12 or higher.

- **MySQL 5.7** – Use Percona XtraBackup version 2.4.

You can use Percona XtraBackup to create a full backup of your MySQL database files. Alternatively, if you already use Percona XtraBackup to back up your MySQL database files, you can upload your existing full and incremental backup directories and files.

For more information about backing up your database with Percona XtraBackup, see [Percona XtraBackup - Documentation](#) on the Percona website.

Creating a full backup with Percona XtraBackup

To create a full backup of your MySQL database files that Amazon RDS can restore from Amazon S3, use the Percona XtraBackup utility (`xtrabackup`).

For example, the following command creates a backup of a MySQL database and stores the files in the folder `/on-premises/s3-restore/backup`.

```
xtrabackup --backup --user=myuser --password=password --target-dir=/on-premises/s3-restore/backup
```

If you want to compress your backup into a single file—which you can split into multiple files later, if needed—you can save your backup in one of the following formats based on your MySQL version:

- **Gzip (.gz)** – For MySQL 5.7 and lower versions
- **tar (.tar)** – For MySQL 5.7 and lower versions
- **Percona xbstream (.xbstream)** – For all MySQL versions

Note

Percona XtraBackup 8.0 and higher only supports Percona xbstream for compression.

MySQL 5.7 and lower versions

The following command creates a backup of your MySQL database split into multiple Gzip files. Replace values with your own information.

```
xtrabackup --backup --user=my_user --password=password --stream=tar \  
 --target-dir=/on-premises/s3-restore/backup | gzip - | split -d --bytes=500MB \
```

- */on-premises/s3-restore/backup/backup.tar.gz*

MySQL 5.7 and lower versions

The following command creates a backup of your MySQL database split into multiple tar files. Replace values with your own information.

```
xtrabackup --backup --user=my_user --password=password --stream=tar \  
--target-dir=/on-premises/s3-restore/backup | split -d --bytes=500MB \  
- /on-premises/s3-restore/backup/backup.tar
```

All MySQL versions

The following command creates a backup of your MySQL database split into multiple xbstream files. Replace values with your own information.

```
xtrabackup --backup --user=myuser --password=password --stream=xbstream \  
--target-dir=/on-premises/s3-restore/backup | split -d --bytes=500MB \  
- /on-premises/s3-restore/backup/backup.xbstream
```

Note

If you see the following error, it might be because you mixed file formats in your command:

```
ERROR:/bin/tar: This does not look like a tar archive
```

Using incremental backups with Percona XtraBackup

If you already use Percona XtraBackup to perform full and incremental backups of your MySQL database files, you don't need to create a full backup and upload the backup files to Amazon S3. Instead, to save time, copy your existing backup directories and files to your Amazon S3 bucket. For more information about creating incremental backups using Percona XtraBackup, see [Create an incremental backup](#) on the Percona website.

When copying your existing full and incremental backup files to an Amazon S3 bucket, you must recursively copy the contents of the base directory. Those contents include both the full backup and all incremental backup directories and files. This copy must preserve the directory structure

in the Amazon S3 bucket. Amazon RDS iterates through all files and directories. Amazon RDS uses the `xtrabackup-checkpoints` file that is included with each incremental backup to identify the base directory and to order incremental backups by log sequence number (LSN) range.

Backup considerations for Percona XtraBackup

Amazon RDS consumes your backup files based on the file name. Name your backup files with the appropriate file extension based on the file format. For example, use `.xbstream` for files stored using the Percona xbstream format.

Amazon RDS consumes your backup files in alphabetical order and also in natural number order. To ensure that your backup files are written and named in the proper order, use the `split` option when you issue the `xtrabackup` command.

Amazon RDS doesn't support partial backups created using Percona XtraBackup. You can't use the following options to create a partial backup when you back up the source files for your database:

- `--tables`
- `--tables-exclude`
- `--tables-file`
- `--databases`
- `--databases-exclude`
- `--databases-file`

Creating an IAM role manually

If you don't have an IAM role, you can create a new one manually. However, if you restore the database by using the AWS Management Console, we recommend that you choose to have Amazon RDS create this new IAM role for you. For Amazon RDS to create this role for you, follow the procedure in the [Importing data from Amazon S3 to a new MySQL DB instance](#) section.

To create a new IAM role manually for importing your database from Amazon S3, create a role to delegate permissions from Amazon RDS to your Amazon S3 bucket. When you create an IAM role, you attach trust and permissions policies. To import your backup files from Amazon S3, use trust and permissions policies similar to the following examples. For more information about creating the role, see [Creating a role to delegate permissions to an AWS service](#).

The trust and permissions policies require that you provide an Amazon Resource Name (ARN). For more information about ARN formatting, see [Amazon Resource Names \(ARNs\) and AWS service namespaces](#).

Example trust policy for importing from Amazon S3

JSON

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

Example permissions policy for importing from Amazon S3 — IAM user permissions

In the following example, replace *iam_user_id* with your own value.

JSON

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowS3AccessRole",  

```

{

Example permissions policy for importing from Amazon S3 — role permissions

In the following example, replace *amzn-s3-demo-bucket* and *prefix* with your own values.

JSON

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListBucket",  
                "s3:GetBucketLocation"  
            ],  
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3GetObject"  
            ],  
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/prefix*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kmsDecrypt"  
            ],  
            "Resource": [  
                "arn:aws:kms:us-east-1:111122223333:key/key_id*"  
            ]  
        }  
    ]  
}
```

Note

If you include a file name prefix, include the asterisk (*) after the prefix. If you don't want to specify a prefix, specify only an asterisk.

Importing data from Amazon S3 to a new MySQL DB instance

You can import data from Amazon S3 to a new MySQL DB instance using the AWS Management Console, AWS CLI, or RDS API.

Console

To import data from Amazon S3 to a new MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where you want to create your DB instance. Choose the same AWS Region as the Amazon S3 bucket that contains your database backup.
3. In the navigation pane, choose **Databases**.
4. Choose **Restore from S3**.

The **Create database by restoring from S3** page appears.

RDS > Databases > Restore from S3

Create database by restoring from S3

S3 source

S3 bucket
Choose the Amazon S3 bucket that contains your database backup files.

S3 prefix - optional | [Info](#)
Enter the file path prefix for the files stored in your Amazon S3 bucket.

Engine options

Engine type | [Info](#)
 Aurora (MySQL Compatible)

 MySQL


Edition
 MySQL Community

Source engine version | [Info](#)

Engine version

5. Under **S3 source**:

- a. Choose the **S3 bucket** that contains the backup.
- b. (Optional) For **S3 prefix**, enter the file path prefix for the files stored in your Amazon S3 bucket.

If you don't specify a prefix, then Amazon RDS creates your DB instance using all of the files and folders in the root folder of the S3 bucket. If you do specify a prefix, then Amazon RDS creates your DB instance using the files and folders in the S3 bucket where the path for the file begins with the specified prefix.

For example, you store your backup files on S3 in a subfolder named backups, and you have multiple sets of backup files, each in its own directory (gzip_backup1, gzip_backup2, and so on). In this case, to restore from the files in the gzip_backup1 folder, you specify the prefix backups/gzip_backup1.

6. Under **Engine options:**
 - a. For **Engine type**, choose **MySQL**.
 - b. For **Source engine version**, choose the MySQL major version of your source database.
 - c. For **Engine Version**, choose the default minor version of your MySQL major version in your AWS Region.

In the AWS Management Console, only the default minor version is available. After you complete the import, you can upgrade your DB instance.

- 7. For **IAM role**, create or choose IAM role with the required trust policy and permissions policy that allows Amazon RDS to access your Amazon S3 bucket. Perform one of the following actions:
 - (Recommended) Choose **Create a new role**, and enter the **IAM role name**. With this option, Amazon RDS automatically creates the role with the trust policy and permissions policy for you.
 - Choose an existing IAM role. Make sure that this role meets all of the criteria in [the section called "Creating an IAM role manually"](#).
- 8. Specify your DB instance information. For information about each setting, see [Settings for DB instances](#).

 **Note**

Be sure to allocate enough storage for your new DB instance so that the restore operation can succeed.

To allow for future growth automatically, under **Additional storage configuration**, choose **Enable storage autoscaling**.

9. Choose additional settings as needed.
10. Choose **Create database**.

AWS CLI

To import data from Amazon S3 to a new MySQL DB instance by using the AWS CLI, run the [restore-db-instance-from-s3](#) command with the following options. For information about each setting, see [Settings for DB instances](#).

Note

Be sure to allocate enough storage for your new DB instance so that the restore operation can succeed.

To enable storage autoscaling and allow for future growth automatically, use the `--max-allocated-storage` option.

- `--allocated-storage`
- `--db-instance-identifier`
- `--db-instance-class`
- `--engine`
- `--master-username`
- `--manage-master-user-password`
- `--s3-bucket-name`
- `--s3-ingestion-role-arn`
- `--s3-prefix`
- `--source-engine`
- `--source-engine-version`

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-s3 \
  --allocated-storage 250 \
  --db-instance-identifier my_identifier \
  --db-instance-class db.m5.large \
  --engine mysql \
  --master-username admin \
  --manage-master-user-password \
  --s3-bucket-name amzn-s3-demo-bucket \
  --s3-ingestion-role-arn arn:aws:iam::account-number:role/rolename \
  --s3-prefix bucket_prefix \
  --source-engine my_sql \
  --source-engine-version 8.0.32
```

```
--max-allocated-storage 1000
```

For Windows:

```
aws rds restore-db-instance-from-s3 ^
--allocated-storage 250 ^
--db-instance-identifier my_identifier ^
--db-instance-class db.m5.large ^
--engine mysql ^
--master-username admin ^
--manage-master-user-password ^
--s3-bucket-name amzn-s3-demo-bucket ^
--s3-ingestion-role-arn arn:aws:iam::account-number:role/rolename ^
--s3-prefix bucket_prefix ^
--source-engine mysql ^
--source-engine-version 8.0.32 ^
--max-allocated-storage 1000
```

RDS API

To import data from Amazon S3 to a new MySQL DB instance by using the Amazon RDS API, call the [RestoreDBInstanceFromS3](#) operation.

Limitations and considerations for importing backup files from Amazon S3 to Amazon RDS

The following limitations and considerations apply to importing backup files from Amazon S3 to an RDS for MySQL DB instance:

- You can only migrate your data to a new DB instance, not to an existing DB instance.
- You must use Percona XtraBackup to back up your data to Amazon S3. For more information, see [Creating your database backup](#).
- The Amazon S3 bucket and the RDS for MySQL DB instance must be in the same AWS Region.
- You can't restore from the following sources:
 - A DB instance snapshot export to Amazon S3. You also can't migrate data from a DB instance snapshot export to your Amazon S3 bucket.
 - An encrypted source database. However, you can encrypt the data being migrated. You can also leave the data unencrypted during the migration process.
 - A MySQL 5.5 or 5.6 database.

- RDS for MySQL doesn't support Percona Server for MySQL as a source database because it can contain `compression_dictionary*` tables in the `mysql` schema.
- RDS for MySQL doesn't support backward migration for either major versions or minor versions. For example, you can't migrate from MySQL version 8.0 to RDS for MySQL 5.7, and you can't migrate from MySQL version 8.0.32 to RDS for MySQL version 8.0.26.
- Amazon RDS doesn't support importing on the db.t2.micro DB instance class from Amazon S3. However, you can restore to a different DB instance class, and then change the DB instance class later. For more information about instance classes, see [Hardware specifications for DB instance classes](#).
- Amazon S3 limits the size of a file uploaded to an Amazon S3 bucket to 5 TB. If a backup file exceeds 5 TB, then you must split the backup file into smaller files.
- Amazon RDS limits the number of files uploaded to an Amazon S3 bucket to 1 million. If the backup data for your database, including all full and incremental backups, exceeds 1 million files, use a Gzip (.gz), tar (.tar.gz), or Percona xbstream (.xbstream) file to store full and incremental backup files in the Amazon S3 bucket. Percona XtraBackup 8.0 only supports Percona xbstream for compression.
- To provide management services for each DB instance, Amazon RDS creates the `rdsadmin` user when it creates the DB instance. Because `rdsadmin` is a reserved user in Amazon RDS, the following limitations apply:
 - Amazon RDS doesn't import functions, procedures, views, events, and triggers with the '`rdsadmin`'@'`localhost`' definer. For more information, see [Stored objects with 'rdsadmin'@'localhost' as the definer](#) and [Master user account privileges](#).
 - When creating the DB instance, Amazon RDS creates a master user with the maximum supported privileges. When restoring from backup, Amazon RDS automatically removes any unsupported privileges assigned to users being imported.

To identify users that might be affected by this, see [User accounts with unsupported privileges](#). For more information on supported privileges in RDS for MySQL, see [Role-based privilege model for RDS for MySQL](#).

- Amazon RDS doesn't migrate user-created tables in the `mysql` schema.
- You must configure the `innodb_data_file_path` parameter with only one data file that uses the default data file name `ibdata1:12M:autoextend`. You can migrate databases with two data files, or with a data file with a different name, using this method.

The following examples are file names that Amazon RDS doesn't allow:

- innodb_data_file_path=ibdata1:50M
- ibdata2:50M:autoextend
- innodb_data_file_path=ibdata01:50M:autoextend
- You can't migrate from a source database that has tables defined outside of the default MySQL data directory.
- The maximum supported size for uncompressed backups using this method is limited to 64 TiB. For compressed backups, this limit is lower to account for uncompression space requirements. In such cases, the maximum supported backup size is 64 TiB - compressed backup size.

For information about the maximum supported database size that RDS for MySQL supports, see [General Purpose SSD storage](#) and [Provisioned IOPS SSD storage](#).

- Amazon RDS doesn't support the importing of MySQL and other external components and plugins.
- Amazon RDS doesn't restore everything from your database. We recommend that you save the database schema and values for the following items from your source MySQL system database, and then add them to your restored RDS for MySQL DB instance after it has been created:
 - User accounts
 - Functions
 - Stored procedures
 - Time zone information. Time zone information is loaded from the local operating system of your RDS for MySQL DB instance. For more information, see [Local time zone for MySQL DB instances](#).

Stored objects with 'rdsadmin'@'localhost' as the definer

Amazon RDS doesn't import functions, procedures, views, events, and triggers with 'rdsadmin'@'localhost' as the definer.

You can use the following SQL script on your source MySQL database to list the stored objects that have the unsupported definer.

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.  
  
SELECT  
    ROUTINE_SCHEMA,  
    ROUTINE_NAME
```

```
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.

SELECT
    TABLE_SCHEMA,
    TABLE_NAME
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

User accounts with unsupported privileges

User accounts with privileges that RDS for MySQL doesn't support are imported without the unsupported privileges. For the list of supported privileges, see [Role-based privilege model for RDS for MySQL](#).

You can run the following SQL query on your source database to list the user accounts that have unsupported privileges.

```
SELECT
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Importing data from an external MySQL database to an Amazon RDS for MySQL DB instance

You can import data from an existing MySQL database to an RDS for MySQL DB instance. You do so by copying the database with [mysqldump](#) and piping it directly into the RDS for MySQL DB instance. The mysqldump command line utility is commonly used to make backups and transfer data from one MySQL server to another. It's included with MySQL client software.

Note

If you are importing or exporting large amounts of data with a MySQL DB instance, it's more reliable and faster to move data in and out of Amazon RDS by using xtrabackup backup files and Amazon S3. For more information, see [Restoring a backup into an Amazon RDS for MySQL DB instance](#).

A typical mysqldump command to move data from an external database to an Amazon RDS DB instance looks similar to the following example. Replace values with your own information.

```
mysqldump -u local_user \
--databases database_name \
--single-transaction \
--compress \
--order-by-primary \
--routines=0 \
--triggers=0 \
--events=0 \
-plocal_password | mysql -u RDS_user \
```

```
--port=port_number \
--host=host_name \
-pRDS_password
```

Important

Make sure not to leave a space between the `-p` option and the entered password. As a security best practice, specify credentials other than the prompts shown in this example.

Make sure that you're aware of the following recommendations and considerations:

- Exclude the following schemas from the dump file:
 - `sys`
 - `performance_schema`
 - `information_schema`

The `mysqldump` utility excludes these schemas by default.

- If you need to migrate users and privileges, consider using a tool that generates the data control language (DCL) for recreating them, such as the [pt-show-grants](#) utility.
- To perform the import, make sure the user doing so has access to the DB instance. For more information, see [Controlling access with security groups](#).

The parameters used are as follows:

- `-u local_user` – Use to specify a user name. In the first usage of this parameter, specify the name of a user account on the local MySQL database that you identify with the `--databases` parameter.
- `--databases database_name` – Use to specify the name of the database on the local MySQL instance that you want to import into Amazon RDS.
- `--single-transaction` – Use to ensure that all of the data loaded from the local database is consistent with a single point in time. If there are other processes changing the data while `mysqldump` is reading it, using this parameter helps maintain data integrity.
- `--compress` – Use to reduce network bandwidth consumption by compressing the data from the local database before sending it to Amazon RDS.

- **--order-by-primary** – Use to reduce load time by sorting each table's data by its primary key.
- **--routines** – Use if routines such as stored procedures or functions exist in the database that you are copying. Set the parameter to *0*, which excludes the routines during the import process. Then later manually recreate the routines in the Amazon RDS database.
- **--triggers** – Use if triggers exist in the database that you are copying. Set the parameter to *0*, which excludes the triggers during the import process. Then later manually recreate the triggers in the Amazon RDS database.
- **--events** – Use if events exist in the database that you are copying. Set the parameter to *0*, which excludes the events during the import process. Then later manually recreate the events in the Amazon RDS database.
- **-p*local_password*** – Use to specify a password. In the first usage of this parameter, specify the password for the user account that you identify with the first **-u** parameter.
- **-u *RDS_user*** – Use to specify a user name. In the second usage of this parameter, specify the name of a user account on the default database for the MySQL DB instance that you identify with the **--host** parameter.
- **--port *port_number*** – Use to specify the port for your MySQL DB instance. By default, this is *3306* unless you changed the value when creating the DB instance.
- **--host *host_name*** – Use to specify the Domain Name System (DNS) name from the Amazon RDS DB instance endpoint, for example, *myinstance.123456789012.us-east-1.rds.amazonaws.com*. You can find the endpoint value in the DB instance details in the Amazon RDS console.
- **-p*RDS_password*** – Use to specify a password. In the second usage of this parameter, you specify the password for the user account identified by the second **-u** parameter.

Make sure to create any stored procedures, triggers, functions, or events manually in your Amazon RDS database. If you have any of these objects in the database that you are copying, then exclude them when you run `mysqldump`. To do so, include the following parameters with your `mysqldump` command:

- **--routines=0**
- **--triggers=0**
- **--events=0**

Example

The following example copies the `world` sample database on the local host to an RDS for MySQL DB instance. Replace values with your own information.

For Linux, macOS, or Unix:

```
sudo mysqldump -u local_user \
--databases world \
--single-transaction \
--compress \
--order-by-primary \
--routines=0 \
--triggers=0 \
--events=0 \
-plocal_password | mysql -u rds_user \
--port=3306 \
--host=my_instance.123456789012.us-east-1.rds.amazonaws.com \
-pRDS_password
```

For Windows:

Run the following command in a command prompt that has been opened by right-clicking **Command Prompt** on the Windows programs menu and choosing **Run as administrator**. Replace values with your own information.

```
mysqldump -u local_user ^
--databases world ^
--single-transaction ^
--compress ^
--order-by-primary ^
--routines=0 ^
--triggers=0 ^
--events=0 ^
-plocal_password | mysql -u RDS_user ^
--port=3306 ^
--host=my_instance.123456789012.us-east-1.rds.amazonaws.com ^
-pRDS_password
```

Note

As a security best practice, specify credentials other than the prompts shown in the example.

Importing data to an Amazon RDS for MySQL database with reduced downtime

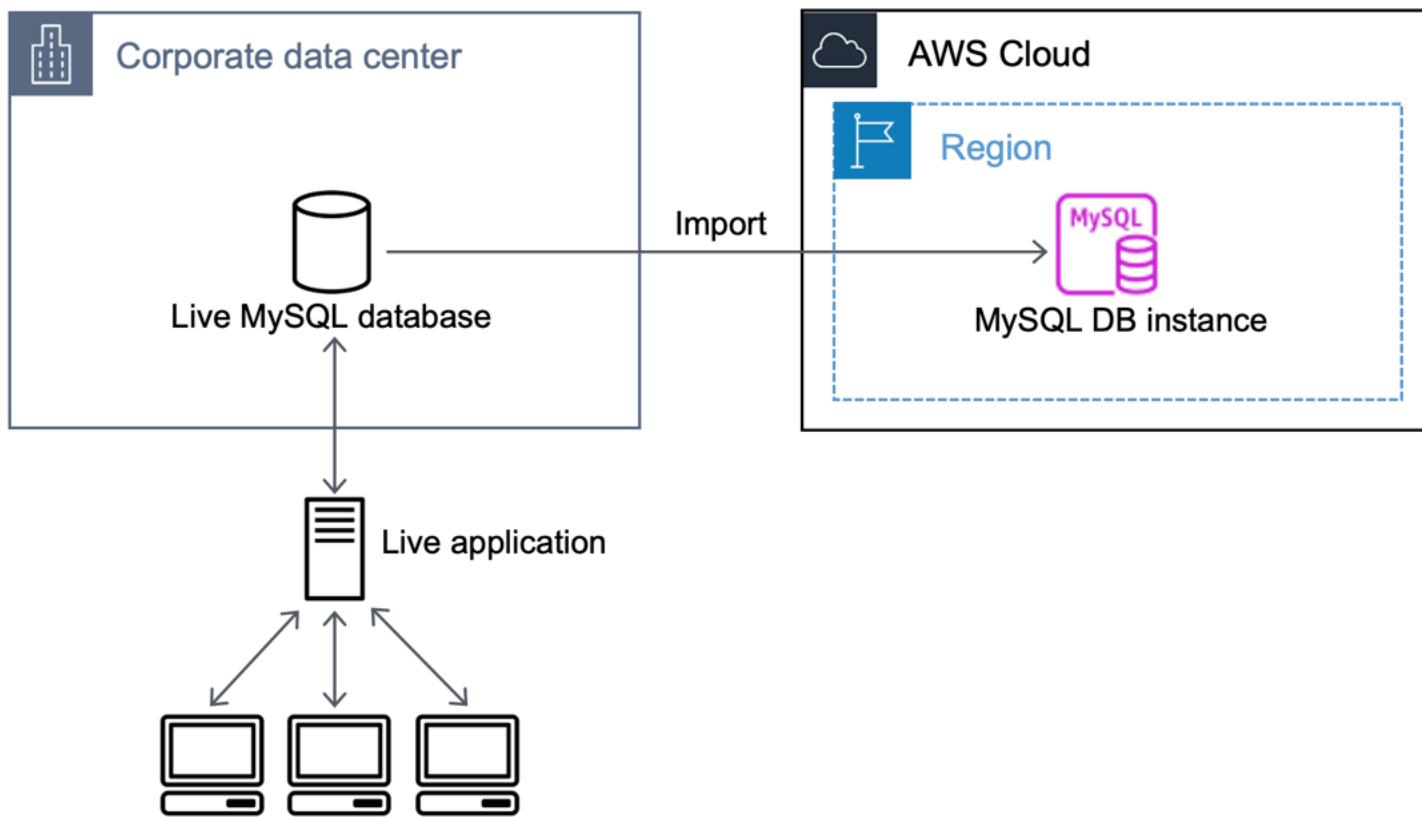
In some cases, you might need to import data from an external MySQL database that supports a live application to an RDS for MySQL DB instance or an RDS for MySQL Multi-AZ DB cluster. Use the following procedure to minimize the impact on availability of applications. This procedure can also help if you are working with a very large database. Using this procedure, you can reduce the cost of the import by reducing the amount of data that is passed across the network to AWS.

In this procedure, you transfer a copy of your database data to an Amazon EC2 instance and import the data into a new Amazon RDS database. You then use replication to bring the Amazon RDS database up-to-date with your live external instance, before redirecting your application to the Amazon RDS database. Configure replication based on binary log coordinates.

Note

If you want to import data into an RDS for MySQL DB instance and your scenario supports it, we recommend moving data in and out of Amazon RDS by using backup files and Amazon S3. For more information, see [Restoring a backup into an Amazon RDS for MySQL DB instance](#).

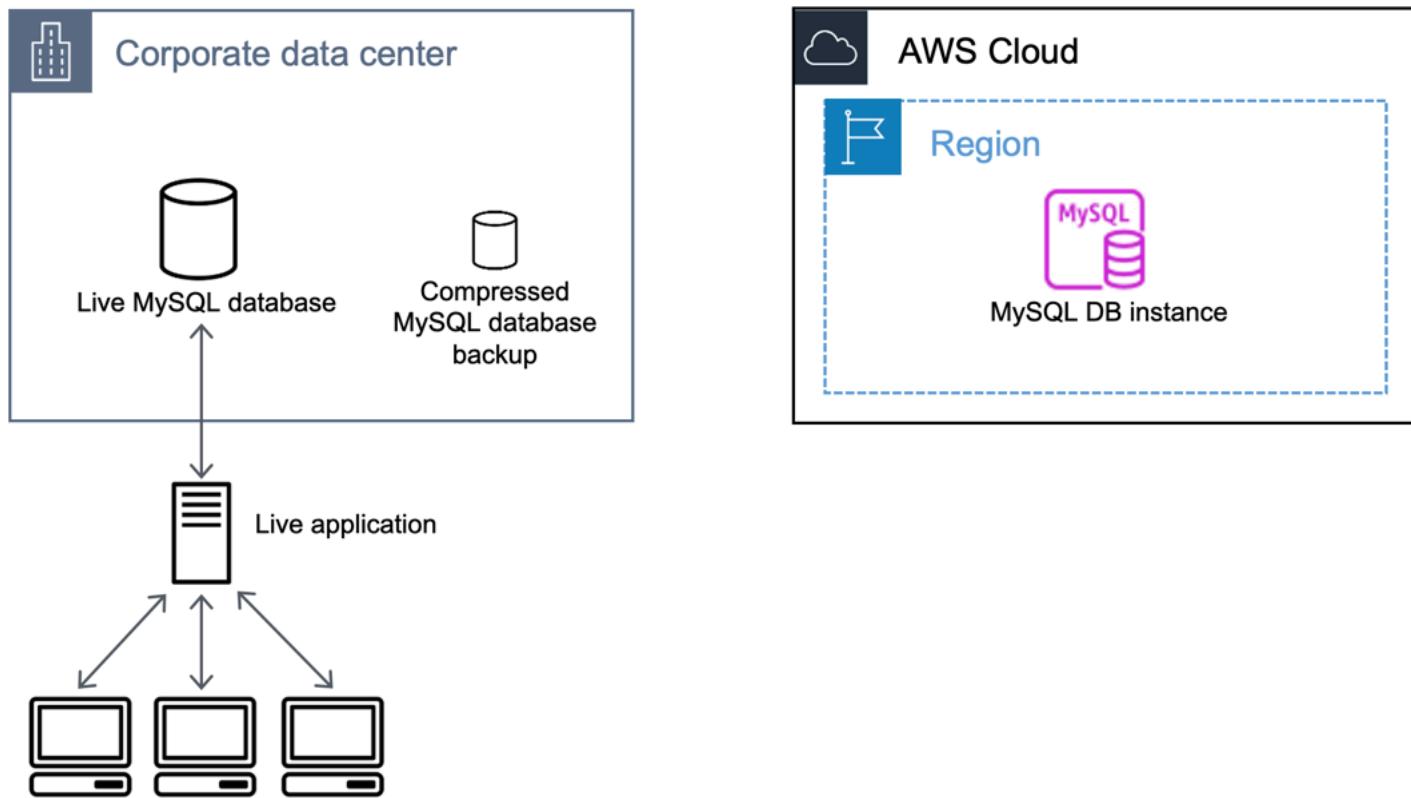
The following diagram shows importing an external MySQL database into a MySQL database on Amazon RDS.



Task 1: Create a copy of your existing database

The first step in the process of migrating a large amount of data to an RDS for MySQL database with minimal downtime is to create a copy of the source data.

The following diagram shows creating a backup of the MySQL database.



You can use the `mysqldump` utility to create a database backup in either SQL or delimited-text format. We recommend that you do a test run with each format in a non-production environment to see which method minimizes the amount of time that `mysqldump` runs.

We also recommend that you weigh `mysqldump` performance against the benefit offered by using the delimited-text format for loading. A backup using delimited-text format creates a tab-separated text file for each table being dumped. To reduce the amount of time required to import your database, you can load these files in parallel using the `LOAD DATA LOCAL INFILE` command. For more information, see [Step 5: Load the data](#) in the Importing data from any source procedure.

Before you start the backup operation, make sure to set the replication options on the MySQL database that you are copying to Amazon RDS. The replication options include turning on binary logging and setting a unique server ID. Setting these options causes your server to start logging database transactions and prepares it to be a source replication instance later in this process.

Make sure that you're aware of the following recommendations and considerations:

- Use the `--single-transaction` option with `mysqldump` because it dumps a consistent state of the database. To ensure a valid dump file, don't run data definition language (DDL) statements while `mysqldump` is running. You can schedule a maintenance window for these operations.
- Exclude the following schemas from the dump file:
 - `sys`
 - `performance_schema`
 - `information_schema`

The `mysqldump` utility excludes these schemas by default.

- If you need to migrate users and privileges, consider using a tool that generates the data control language (DCL) for recreating them, such as the [pt-show-grants](#) utility.

To set replication options

1. Edit the `my.cnf` file. This file is usually located under `/etc`.

```
sudo vi /etc/my.cnf
```

Add the `log_bin` and `server_id` options to the `[mysqld]` section. The `log_bin` option provides a file name identifier for binary log files. The `server_id` option provides a unique identifier for the server in source-replica relationships.

The following example shows the updated `[mysqld]` section of a `my.cnf` file:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

For more information, see [Setting the Replication Source Configuration](#) in the MySQL documentation.

2. For replication with a Multi-AZ DB cluster set the `ENFORCE_GTID_CONSISTENCY` and the `GTID_MODE` parameter to ON.

```
mysql> SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = ON;
```

```
mysql> SET @@GLOBAL.GTID_MODE = ON;
```

These settings aren't required for replication with a DB instance.

3. Restart the mysql service.

```
sudo service mysqld restart
```

To create a backup copy of your existing database

1. Create a backup of your data using the mysqldump utility, specifying either SQL or delimited-text format.

For MySQL 8.0.25 and lower versions, specify `--master-data=2` to create a backup file that can be used to start replication between servers. For MySQL 8.0.26 and higher versions, specify `--source-data=2` to create a backup file that can be used to start replication between servers. For more information, see the [mysqldump — A Database Backup Program](#) in the MySQL documentation.

To improve performance and ensure data integrity, use the `--order-by-primary` and `--single-transaction` options for mysqldump.

To avoid including the MySQL system database in the backup, don't use the `--all-databases` option with mysqldump. For more information, see [Creating a Data Snapshot Using mysqldump](#) in the MySQL documentation.

Use chmod, if necessary, to make sure that the directory where the backup file is being created is writeable.

 **Important**

On Windows, run the command window as an administrator.

- To produce SQL output, use the following command:

For Linux, macOS, or Unix:

```
sudo mysqldump \  
  --databases database_name \  
  --master-data=2 \  
  \
```

```
--single-transaction \
--order-by-primary \
-r backup.sql \
-u local_user \
-ppassword
```

Note

As a security best practice, specify credentials other than the prompts shown in the example.

For Windows:

```
mysqldump ^
--databases database_name ^
--master-data=2 ^
--single-transaction ^
--order-by-primary ^
-r backup.sql ^
-u local_user ^
-ppassword
```

Note

As a security best practice, specify credentials other than the prompts shown in the example.

- To produce delimited-text output, use the following command:

For Linux, macOS, or Unix:

```
sudo mysqldump \
--tab=target_directory \
--fields-terminated-by ',' \
--fields-enclosed-by '\"' \
--lines-terminated-by 0x0d0a \
database_name \
--master-data=2 \
--single-transaction \
```

```
--order-by-primary \
-ppassword
```

For Windows:

```
mysqldump ^
--tab=target_directory ^
--fields-terminated-by "," ^
--fields-enclosed-by "" ^
--lines-terminated-by 0x0d0a ^
database_name ^
--master-data=2 ^
--single-transaction ^
--order-by-primary ^
-ppassword
```

Note

As a security best practice, specify credentials other than the prompts shown in the example.

Make sure to create any stored procedures, triggers, functions, or events manually in your Amazon RDS database. If you have any of these objects in the database that you are copying, then exclude them when you run `mysqldump`. To do so, include the following arguments with your `mysqldump` command:

- `--routines=0`
- `--triggers=0`
- `--events=0`

For MySQL 8.0.22 and lower versions, when you run `mysqldump` and specify the delimited-text format, a `CHANGE MASTER TO` comment is returned. This comment contains the master log file name and position. For MySQL 8.0.23 and higher versions, when you run `mysqldump` using the delimited-text format, a `CHANGE REPLICATION SOURCE TO` comment is returned. This comment contains the source log file name and position. If the external instance is MySQL 8.0.23 and higher versions, note the values for `MASTER_LOG_FILE` and `MASTER_LOG_POS`. You need these values when setting up replication.

The following output is returned for MySQL 8.0.22 and lower versions:

```
-- Position to start replication or point-in-time recovery from  
--  
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031',  
MASTER_LOG_POS=107;
```

The following output is returned for MySQL 8.0.23 and higher versions:

```
-- Position to start replication or point-in-time recovery from  
--  
-- CHANGE SOURCE TO SOURCE_LOG_FILE='mysql-bin-changelog.000031',  
SOURCE_LOG_POS=107;
```

For MySQL 8.0.22 and lower versions, if you are using SQL format, you can get the master log file name and position in the `CHANGE MASTER TO` comment in the backup file. For MySQL 8.0.23 and higher versions, if you are using SQL format, you can get the source log file name and position in the `CHANGE REPLICATION SOURCE TO` comment in the backup file.

2. Compress the copied data to reduce the amount of network resources needed to copy your data to the Amazon RDS database. Note the size of the backup file. You need this information when determining how large an Amazon EC2 instance to create. When you are done, compress the backup file using GZIP or your preferred compression utility.

- To compress SQL output, use the following command:

```
gzip backup.sql
```

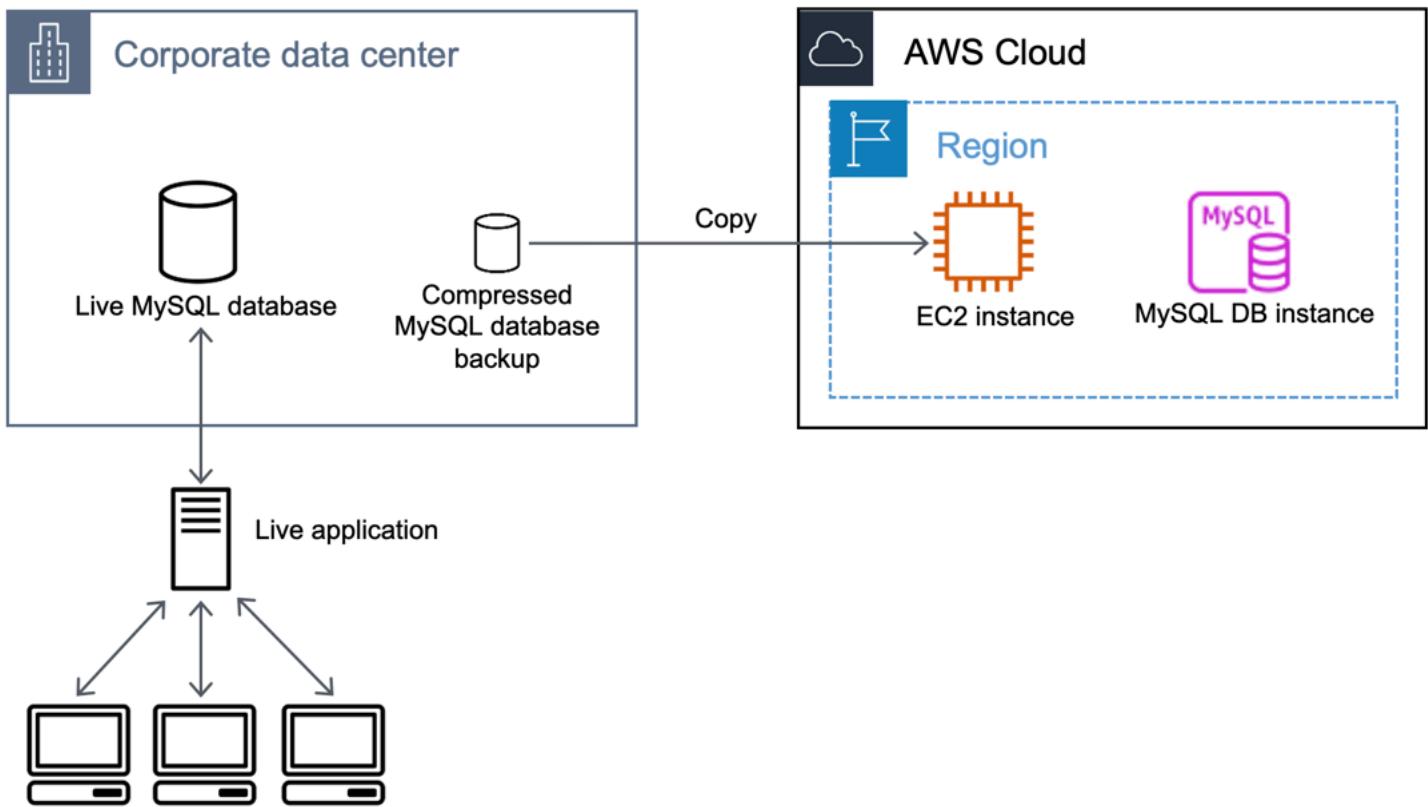
- To compress delimited-text output, use the following command:

```
tar -zcvf backup.tar.gz target_directory
```

Task 2: Create an Amazon EC2 instance and copy the compressed database

Copying your compressed database backup file to an Amazon EC2 instance takes fewer network resources than doing a direct copy of uncompressed data between database instances. After your data is in Amazon EC2, you can copy it from there directly to your MySQL database. For you to save on the cost of network resources, your Amazon EC2 instance must be in the same AWS Region as your Amazon RDS DB instance. Having the Amazon EC2 instance in the same AWS Region as your Amazon RDS database also reduces network latency during the import.

The following diagram shows copying the database backup to an Amazon EC2 instance.



To create an Amazon EC2 instance and copy your data

1. In the AWS Region where you plan to create the Amazon RDS database, create a virtual private cloud (VPC), a VPC security group, and a VPC subnet. Ensure that the inbound rules for your VPC security group allow the IP addresses required for your application to connect to AWS. You can specify a range of IP addresses—for example, 203.0.113.0/24—or another VPC security group. You can use the [Amazon VPC console](#) to create and manage VPCs, subnets, and security groups. For more information, see [Getting started with Amazon VPC](#) in the *Amazon Virtual Private Cloud User Guide*.
2. Open the [Amazon EC2 console](#) and choose the AWS Region to contain both your Amazon EC2 instance and your Amazon RDS database. Launch an Amazon EC2 instance using the VPC, subnet, and security group that you created in Step 1. Ensure that you select an instance type with enough storage for your database backup file when it is uncompressed. For details on Amazon EC2 instances, see [Getting started with Amazon EC2](#) in the *Amazon Elastic Compute Cloud User Guide*.
3. To connect to your Amazon RDS database from your Amazon EC2 instance, edit your VPC security group. Add an inbound rule specifying the private IP address of your EC2 instance.

You can find the private IP address on the **Details** tab of the **Instance** pane in the EC2 console window. To edit the VPC security group and add an inbound rule, choose **Security Groups** in the EC2 console navigation pane, choose your security group, and then add an inbound rule for MySQL or Aurora specifying the private IP address of your EC2 instance. To learn how to add an inbound rule to a VPC security group, see [Security group rules](#) in the *Amazon Virtual Private Cloud User Guide*.

4. Copy your compressed database backup file from your local system to your Amazon EC2 instance. Use chmod, if necessary, to make sure that you have write permission for the target directory of the Amazon EC2 instance. You can use scp or a Secure Shell (SSH) client to copy the file. The following command is an example scp command:

```
scp -r -i key_pair.pem backup.sql.gz ec2-user@EC2 DNS:/target_directory/backup.sql.gz
```

⚠ Important

When copying sensitive data, be sure to use a secure network transfer protocol.

5. Connect to your Amazon EC2 instance and install the latest updates and the MySQL client tools using the following commands:

```
sudo yum update -y  
sudo yum install mysql -y
```

For more information, see [Connect to your instance](#) for Linux instances in the *Amazon Elastic Compute Cloud User Guide*.

⚠ Important

This example installs the MySQL client on an Amazon Machine Image (AMI) for an Amazon Linux distribution. This example doesn't install the MySQL client on a different distribution, such as Ubuntu or Red Hat Enterprise Linux. For information about installing MySQL, see [Installing MySQL](#) in the MySQL documentation.

6. While connected to your Amazon EC2 instance, decompress your database backup file. The following commands are examples.

- To decompress SQL output, use the following command:

```
gzip backup.sql.gz -d
```

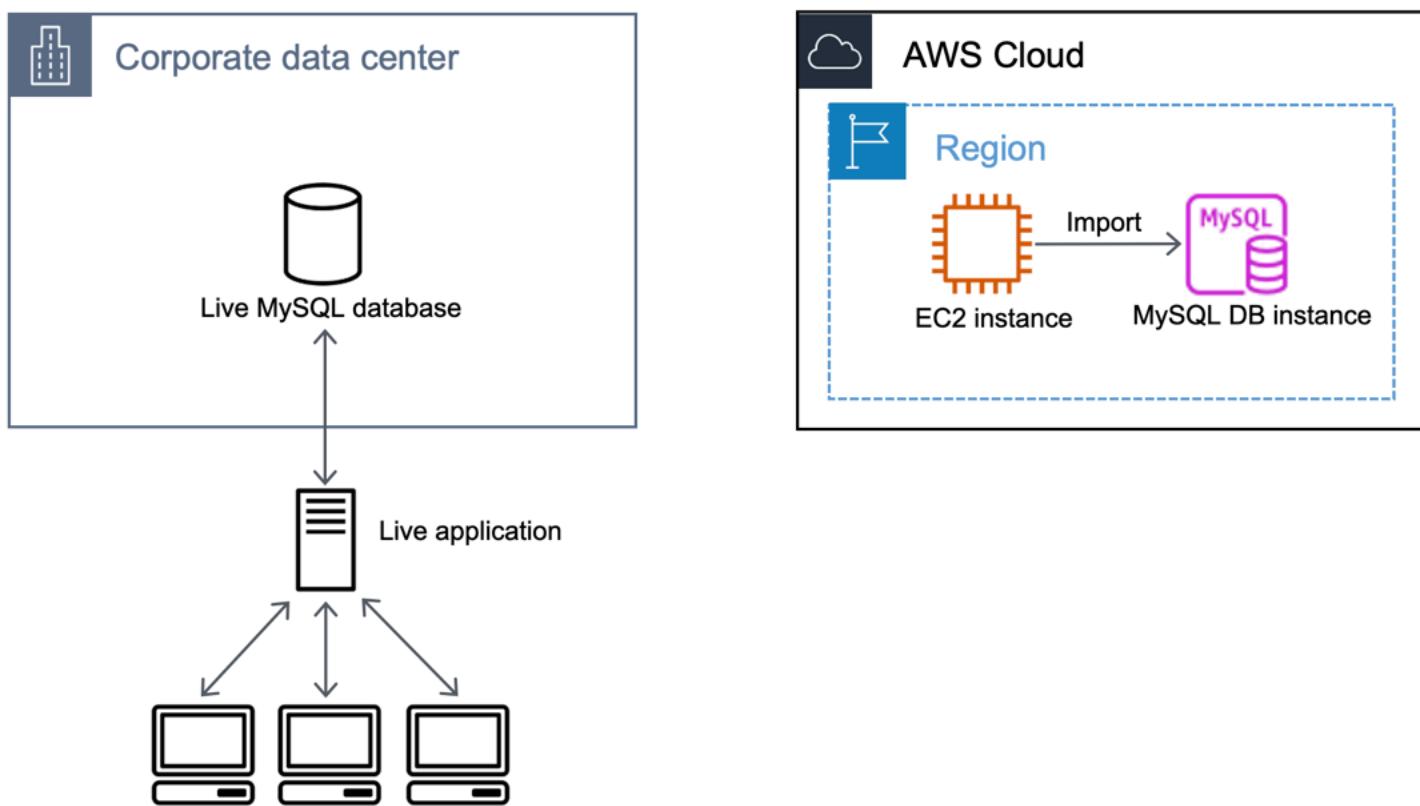
- To decompress delimited-text output, use the following command:

```
tar xzvf backup.tar.gz
```

Task 3: Create a MySQL database and import data from your Amazon EC2 instance

By creating an RDS for MySQL DB instance or an RDS for MySQL Multi-AZ DB cluster in the same AWS Region as your Amazon EC2 instance, you can import the database backup file from Amazon EC2 faster than over the internet.

The following diagram shows importing the backup from an Amazon EC2 instance into a MySQL database.



To create a MySQL database and import your data

1. Determine which DB instance class and what amount of storage space is required to support the expected workload for this Amazon RDS database. As part of this process, decide what is sufficient space and processing capacity for your data load procedures. Also, decide what is required to handle the production workload. You can estimate this based on the size and resources of the source MySQL database. For more information, see [DB instance classes](#).
2. Create a DB instance or Multi-AZ DB cluster in the AWS Region that contains your Amazon EC2 instance.

To create an RDS for MySQL Multi-AZ DB cluster, follow the instructions in [Creating a Multi-AZ DB cluster for Amazon RDS](#).

To create an RDS for MySQL DB instance, follow the instructions in [Creating an Amazon RDS DB instance](#) and use the following guidelines:

- Specify a DB engine version that is compatible with your source DB instance.
 - Specify the same virtual private cloud (VPC) and VPC security group as for your Amazon EC2 instance. This approach ensures that your Amazon EC2 instance and your Amazon RDS instance are visible to each other over the network. Make sure your DB instance is publicly accessible. To set up replication with your source database as described in a following section, your DB instance must be publicly accessible.
 - Don't configure multiple Availability Zones, backup retention, or read replicas until after you have imported the database backup. When that import is completed, you can configure Multi-AZ and backup retention for the production instance.
3. Review the default configuration options for the Amazon RDS database. If the default parameter group for the database doesn't have the configuration options that you want, find a different one that does or create a new parameter group. For more information about creating a parameter group, see [Parameter groups for Amazon RDS](#).
 4. Connect to the new Amazon RDS database as the master user. Create the users required to support the administrators, applications, and services that need to access the DB instance. The hostname for the Amazon RDS database is the **Endpoint** value for this DB instance without the port number, for example, mysampled.123456789012.us-west-2.rds.amazonaws.com. You can find the endpoint value in the database details in the Amazon RDS console.
 5. Connect to your Amazon EC2 instance. For more information, see [Connect to your instance](#) for Linux instances in the *Amazon Elastic Compute Cloud User Guide*.

6. Connect to your Amazon RDS database as a remote host from your Amazon EC2 instance using the mysql command. The following command is an example:

```
mysql -h host_name -P 3306 -u db_master_user -p
```

The *host_name* is the Amazon RDS database endpoint.

7. At the mysql prompt, run the source command and pass it the name of your database dump file. This command loads the data into the Amazon RDS DB instance.

- For SQL format, use the following command:

```
mysql> source backup.sql;
```

- For delimited-text format, first create the database, if it isn't the default database that you created when setting up the Amazon RDS database.

```
mysql> create database database_name;
mysql> use database_name;
```

Then create the tables.

```
mysql> source table1.sql
mysql> source table2.sql
etc...
```

Then import the data.

```
mysql> LOAD DATA LOCAL INFILE 'table1.txt' INTO TABLE table1 FIELDS TERMINATED BY
      ',' ENCLOSED BY '"' LINES TERMINATED BY '0x0d0a';
mysql> LOAD DATA LOCAL INFILE 'table2.txt' INTO TABLE table2 FIELDS TERMINATED BY
      ',' ENCLOSED BY '"' LINES TERMINATED BY '0x0d0a';
etc...
```

To improve performance, you can perform these operations in parallel from multiple connections so that all of your tables are created and then loaded at the same time.

Note

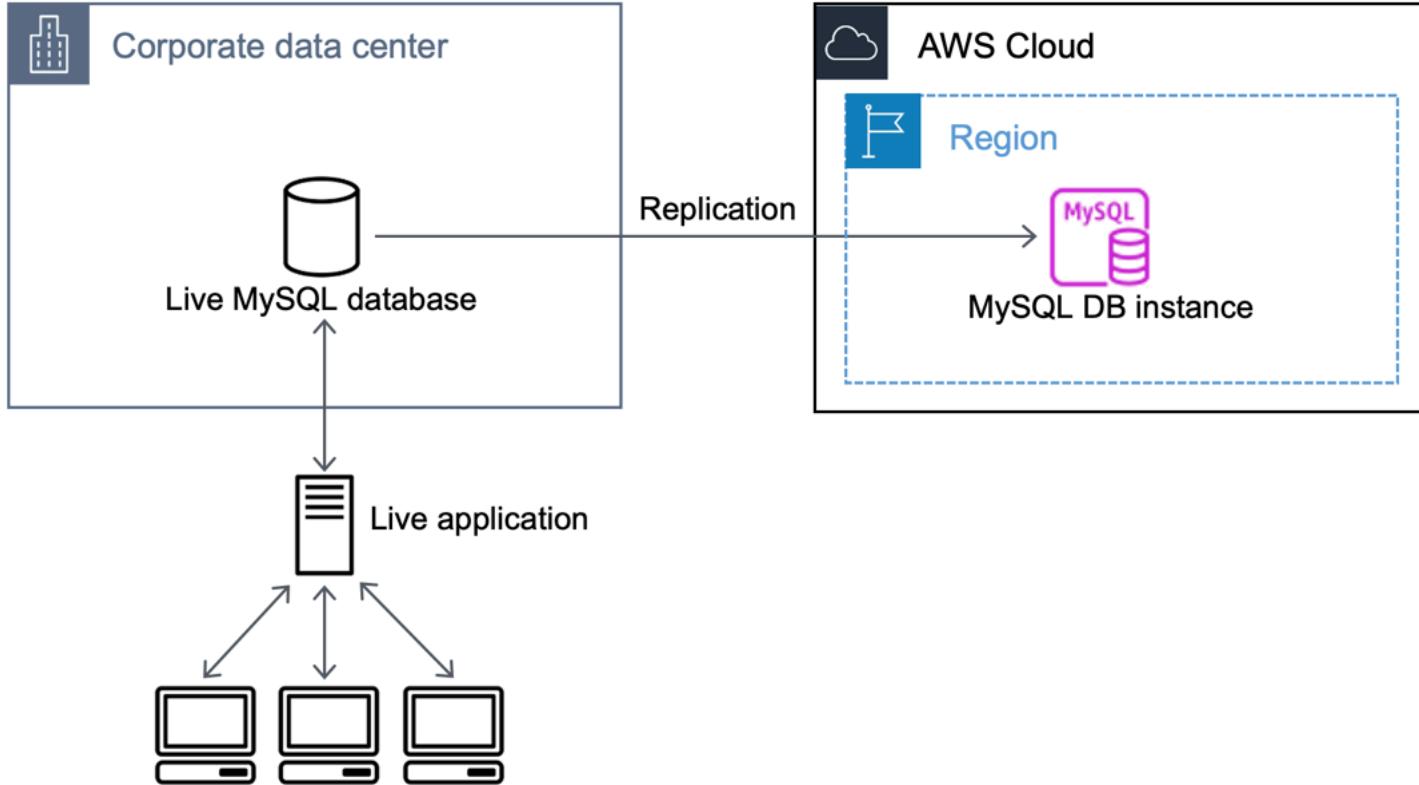
If you used any data-formatting options with `mysqldump` when you initially dumped the table, make sure to use the same options with `LOAD DATA LOCAL INFILE` to ensure proper interpretation of the data file contents.

- Run a simple `SELECT` query against one or two of the tables in the imported database to verify that the import was successful.

If you no longer need the Amazon EC2 instance used in this procedure, terminate the EC2 instance to reduce your AWS resource usage. To terminate an EC2 instance, see [Terminate an instance](#) in the *Amazon Elastic Compute Cloud User Guide*.

Task 4: Replicate data from your external database to your new Amazon RDS database

Your source database was likely updated during the time that it took to copy and transfer the data to the MySQL database. Thus, you can use replication to bring the copied database up-to-date with the source database.



The permissions required to start replication on an Amazon RDS database are restricted and aren't available to your Amazon RDS master user. Because of this, use the appropriate Amazon RDS stored procedure for your major engine version:

- [mysql_rds_set_external_master \(RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_external_source \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_set_external_master_gtid](#) to configure replication and [mysql.rds_start_replication](#) to start replication

To start replication

In Task 1, [when you set replication options](#), you turned on binary logging and set a unique server ID for your source database. Now, you can set up your Amazon RDS database as a replica with your live database as the source replication instance.

1. In the Amazon RDS console, add the IP address of the server that hosts the source database to the VPC security group for the Amazon RDS database. For more information on configuring a VPC security group, see [Configure security group rules](#) in the *Amazon Virtual Private Cloud User Guide*.

You might also need to configure your local network to permit connections from the IP address of your Amazon RDS database so that it can communicate with your source instance. To find the IP address of the Amazon RDS database, use the host command:

```
host host_name
```

The *host_name* is the DNS name from the Amazon RDS database endpoint, for example myinstance.123456789012.us-east-1.rds.amazonaws.com. You can find the endpoint value in the DB instance details in the Amazon RDS console.

2. Using the client of your choice, connect to the source instance and create a user to be used for replication. This account is used solely for replication and must be restricted to your domain to improve security. The following command is an example:

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

Note

Specify credentials other than the prompts shown here as a security best practice.

3. For the source instance, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. For example, to grant the REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the 'repl_user' user for your domain, issue the following command:

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

4. Make the Amazon RDS database the replica. Connect to the Amazon RDS database as the master user and identify the source database as the source replication instance by using the appropriate Amazon RDS stored procedure:

- [mysql_rds_set_external_master \(RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_external_source \(RDS for MySQL major versions 8.4 and higher\)](#)

If you have a SQL format backup file, use the master log file name and master log position that you determined in Step 4. If you used delimited-text format, use the name and position that you determined when creating the backup files. The following commands are examples:

MySQL 8.4 and higher versions

```
CALL mysql.rds_set_external_source ('myserver.mydomain.com', 3306,  
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

MySQL 8.0 and lower versions

```
CALL mysql.rds_set_external_master ('myserver.mydomain.com', 3306,  
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

Note

Specify credentials other than the prompts shown here as a security best practice.

5. On the Amazon RDS database, to start replication, run the following command that uses the [mysql.rds_start_replication](#) stored procedure:

```
CALL mysql.rds_start_replication;
```

6. On the Amazon RDS database, to determine when the replica is up to date with the source replication instance, run the [SHOW REPLICAS STATUS](#) command. The results of the SHOW REPLICAS STATUS command include the Seconds_Behind_Master field. When the Seconds_Behind_Master field returns 0, then the replica is up to date with the source replication instance.

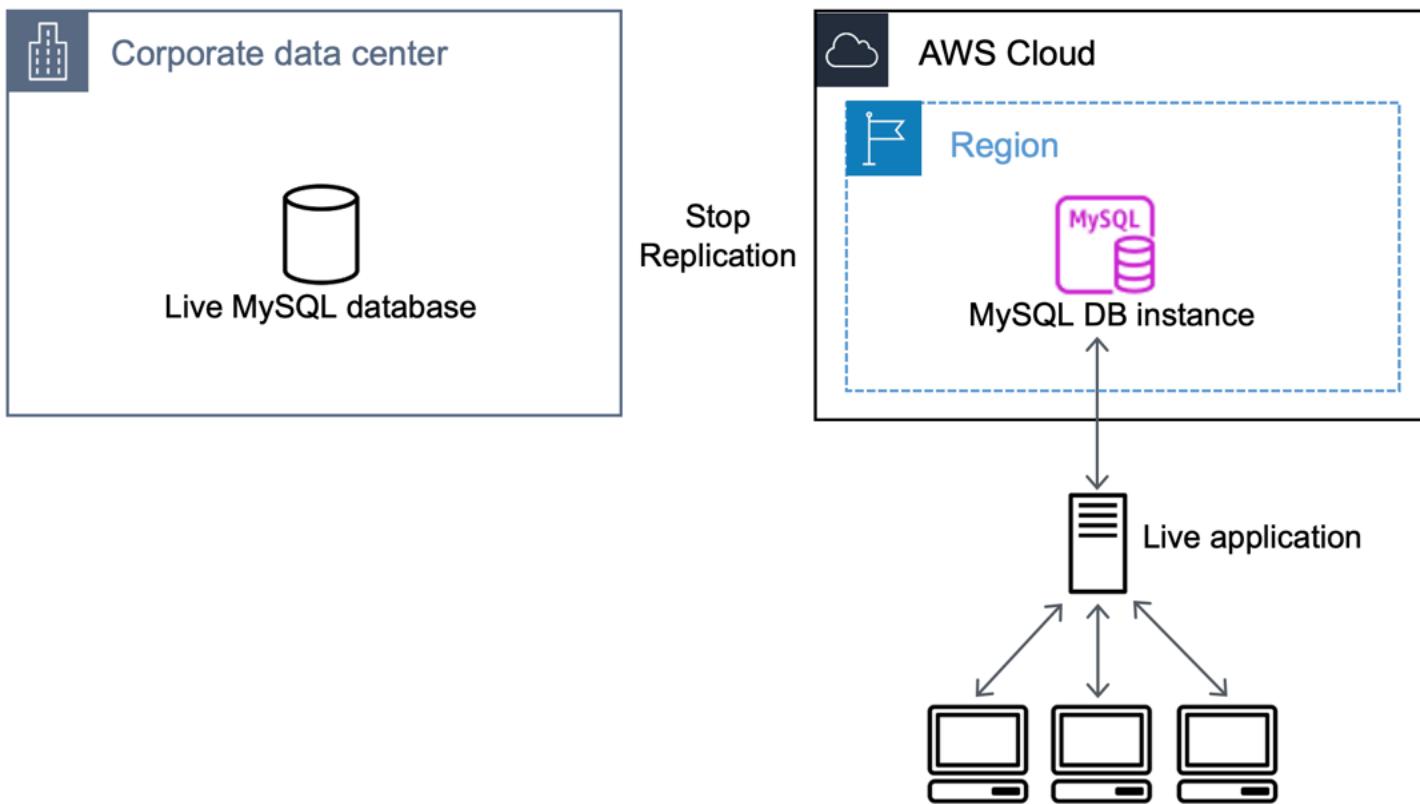
 **Note**

Previous versions of MySQL used SHOW SLAVE STATUS instead of SHOW REPLICAS STATUS. If you are using a MySQL version before 8.0.23, then use SHOW SLAVE STATUS.

7. After the Amazon RDS database is up to date, turn on automated backups so you can restore that database if needed. You can turn on or modify automated backups for your Amazon RDS database by using the [Amazon RDS console](#). For more information, see [Introduction to backups](#).

Task 5: Redirect your live application to your Amazon RDS instance

After the MySQL database is up to date with the source replication instance, you can now update your live application to use the Amazon RDS instance.



To redirect your live application to your MySQL database and stop replication

1. To add the VPC security group for the Amazon RDS database, add the IP address of the server that hosts the application. For more information on modifying a VPC security group, see [Configure security group rules](#) in the *Amazon Virtual Private Cloud User Guide*.
2. Verify that the Seconds_Behind_Master field in the [SHOW REPLICAS STATUS](#) command results is 0, which indicates that the replica is up to date with the source replication instance.

```
SHOW REPLICAS STATUS;
```

Note

Previous versions of MySQL used SHOW SLAVE STATUS instead of SHOW REPLICAS STATUS. If you are using a MySQL version before 8.0.23, then use SHOW SLAVE STATUS.

3. Close all connections to the source when their transactions complete.
4. Update your application to use the Amazon RDS database. This update typically involves changing the connection settings to identify the hostname and port of the Amazon RDS database, the user account and password to connect with, and the database to use.

5. Connect to the DB instance.

For a Multi-AZ DB cluster, connect to the writer DB instance.

6. Stop replication for the Amazon RDS instance by running the following command that uses the [mysql.rds_stop_replication](#) stored procedure:

```
CALL mysql.rds_stop_replication;
```

7. Reset the replication configuration so this instance is no longer identified as a replica by using the appropriate Amazon RDS stored procedure on your Amazon RDS database:

- [mysql_rds_reset_external_master \(RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_reset_external_source \(RDS for MySQL major versions 8.4 and higher\)](#)

MySQL 8.4 and higher versions

```
CALL mysql.rds_reset_external_source;
```

MySQL 8.0 and lower versions

```
CALL mysql.rds_reset_external_master;
```

8. Turn on additional Amazon RDS features such as Multi-AZ support and read replicas. For more information, see [Configuring and managing a Multi-AZ deployment for Amazon RDS](#) and [Working with DB instance read replicas](#).

Importing data from any source to an Amazon RDS for MySQL DB instance

With Amazon RDS, you can migrate existing MySQL data from any source to an RDS for MySQL DB instance. You can transfer data from on-premises databases, other cloud providers, or existing RDS for MySQL DB instances to your target RDS for MySQL DB instance. With this functionality, you can consolidate databases, implement disaster recovery solutions, or transition from self-managed databases. Common scenarios include moving from self-hosted MySQL servers to fully managed Amazon RDS DB instances, consolidating multiple MySQL databases into a single DB instance, or creating test environments with production data. The following sections provide step-by-step instructions for importing your MySQL data using methods such as `mysqldump`, backup files, or replication.

Step 1: Create flat files containing the data to be loaded

Use a common format, such as comma-separated values (CSV), to store the data to be loaded. Each table must have its own file—you can't combine data for multiple tables in the same file. Give each file the same name as the table it corresponds to. The file extension can be anything you like. For example, if the table name is sales, the file name could be sales.csv or sales.txt.

If possible, order the data by the primary key of the table being loaded. Doing this drastically improves load times and minimizes disk storage requirements.

The speed and efficiency of this procedure depends on keeping the size of the files small. If the uncompressed size of any individual file is larger than 1 GiB, split it into multiple files and load each one separately.

On Unix-like systems (including Linux), use the `split` command. For example, the following command splits the `sales.csv` file into multiple files of less than 1 GiB, splitting only at line breaks (`-C 1024m`). The names of the new files include ascending numerical suffixes. The following command produces files with names such as `sales.part_00` and `sales.part_01`.

```
split -C 1024m -d sales.csv sales.part_
```

Similar utilities are available for other operating systems.

You can store the flat files anywhere. However, when you load the data in [Step 5](#), you must invoke the `mysql` shell from the same location where the files exist, or use the absolute path for the files when you run `LOAD DATA LOCAL INFILE`.

Step 2: Stop any applications from accessing the target DB instance

Before starting a large load, stop all application activity from accessing the target DB instance that you plan to load to. We recommend this particularly if other sessions will be modifying the tables being loaded or tables that they reference. Doing this reduces the risk of constraint violations occurring during the load and improves load performance. It also makes it possible to restore the DB instance to the point just before the load without losing changes made by processes not involved in the load.

Of course, this might not be possible or practical. If you can't stop applications from accessing the DB instance before the load, take steps to ensure the availability and integrity of your data. The specific steps required vary greatly depending upon specific use cases and site requirements.

Step 3: Create a DB snapshot

If you plan to load data into a new DB instance that contains no data, you can skip this step. Otherwise, we recommend that you create DB snapshots of the target Amazon RDS DB instance both before and after the data load. Amazon RDS DB snapshots are complete backups of your DB instance that you can use to restore your DB instance to a known state. When you initiate a DB snapshot, I/O operations to your DB instance are momentarily suspended while your database is backed up.

Creating a DB snapshot immediately before the load makes it possible for you to restore the database to its state before the load, if you need to. A DB snapshot taken immediately after the load protects you from having to load the data again in case of a mishap. You can also use DB snapshots after the load to import data into new database instances.

The following example runs the AWS CLI [create-db-snapshot](#) command to create a DB snapshot of the AcmeRDS instance and give the DB snapshot the identifier "preload".

For Linux, macOS, or Unix:

```
aws rds create-db-snapshot \
--db-instance-identifier AcmeRDS \
--db-snapshot-identifier preload
```

For Windows:

```
aws rds create-db-snapshot ^
--db-instance-identifier AcmeRDS ^
--db-snapshot-identifier preload
```

You can also use the restore from DB snapshot functionality to create test DB instances for dry runs or to undo changes made during the load.

Keep in mind that restoring a database from a DB snapshot creates a new DB instance that, like all DB instances, has a unique identifier and endpoint. To restore the DB instance without changing the endpoint, first delete the DB instance so that you can reuse the endpoint.

For example, to create a DB instance for dry runs or other testing, you give the DB instance its own identifier. In the example, "AcmeRDS-2" is the identifier. The example connects to the DB instance using the endpoint associated with AcmeRDS-2. For more information, see [restore-db-instance-from-db-snapshot](#).

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \  
  --db-instance-identifier AcmeRDS-2 \  
  --db-snapshot-identifier preload
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^  
  --db-instance-identifier AcmeRDS-2 ^  
  --db-snapshot-identifier preload
```

To reuse the existing endpoint, first delete the DB instance and then give the restored database the same identifier. For more information, see [delete-db-instance](#).

The following example also takes a final DB snapshot of the DB instance before deleting it. This is optional but recommended.

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \  
  --db-instance-identifier AcmeRDS \  
  --final-db-snapshot-identifier AcmeRDS-Final  
  
aws rds restore-db-instance-from-db-snapshot \  
  --db-instance-identifier AcmeRDS \  
  --db-snapshot-identifier preload
```

For Windows:

```
aws rds delete-db-instance ^  
  --db-instance-identifier AcmeRDS ^  
  --final-db-snapshot-identifier AcmeRDS-Final  
  
aws rds restore-db-instance-from-db-snapshot ^  
  --db-instance-identifier AcmeRDS ^  
  --db-snapshot-identifier preload
```

Step 4 (Optional): Turn off Amazon RDS automated backups

Warning

Don't turn off automated backups if you need to perform point-in-time recovery.

Turning off automated backups is a performance optimization and isn't required for data loads. Turning off automated backups erases all existing backups. As a result, after you turn off automated backups, point-in-time recovery isn't possible. Manual DB snapshots aren't affected by turning off automated backups. All existing manual DB snapshots are still available for restore.

Turning off automated backups reduces load time by about 25 percent and reduces the amount of storage space required during the load. If you plan to load data into a new DB instance that contains no data, turning off backups is an easy way to speed up the load and avoid using the additional storage needed for backups. However, in some cases you might plan to load into a DB instance that already contains data. If so, weigh the benefits of turning off backups against the impact of losing the ability to perform point-in-time-recovery.

DB instances have automated backups turned on by default (with a one day retention period). To turn off automated backups, set the backup retention period to zero. After the load, you can turn backups back on by setting the backup retention period to a nonzero value. To turn on or turn off backups, Amazon RDS shuts the DB instance down and then restarts it to turn MySQL logging on or off.

Run the AWS CLI `modify-db-instance` command to set the backup retention to zero and apply the change immediately. Setting the retention period to zero requires a DB instance restart, so wait until the restart has completed before proceeding. For more information, see [modify-db-instance](#).

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier AcmeRDS \
--apply-immediately \
--backup-retention-period 0
```

For Windows:

```
aws rds modify-db-instance ^
```

```
--db-instance-identifier AcmeRDS ^  
--apply-immediately ^  
--backup-retention-period 0
```

You can check the status of your DB instance with the AWS CLI [describe-db-instances](#) command. The following example displays the DB instance status of the AcmeRDS DB instance:

```
aws rds describe-db-instances --db-instance-identifier AcmeRDS --query "*[].  
{DBInstanceStatus:DBInstanceState}"
```

When the DB instance status is available, you're ready to proceed to the next step.

Step 5: Load the data

To read rows from your flat files into the database tables, use the MySQL LOAD DATA LOCAL INFILE statement.

Note

You must invoke the mysql shell from the same location where your flat files exist, or use the absolute path for the files when you run LOAD DATA LOCAL INFILE.

The following example shows how to load data from a file named sales.txt into a table named Sales in the database:

```
mysql> LOAD DATA LOCAL INFILE 'sales.txt' INTO TABLE Sales FIELDS TERMINATED BY ''  
ENCLOSED BY '' ESCAPED BY '\\';  
Query OK, 1 row affected (0.01 sec)  
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

For more information about the LOAD DATA statement, see [LOAD DATA Statement](#) in the MySQL documentation.

Step 6: Turn back on Amazon RDS automated backups

If you turned off Amazon RDS automated backups in [Step 4](#), after the load is finished, turn automated backups on by setting the backup retention period back to its preload value. As noted in Step 4, Amazon RDS restarts the DB instance, so be prepared for a brief outage.

The following example runs the AWS CLI [modify-db-instance](#) command to turn on automated backups for the AcmeRDS DB instance and set the retention period to one day:

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier AcmeRDS \
--backup-retention-period 1 \
--apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier AcmeRDS ^
--backup-retention-period 1 ^
--apply-immediately
```

Working with MySQL replication in Amazon RDS

You usually use read replicas to configure replication between Amazon RDS DB instances. For general information about read replicas, see [Working with DB instance read replicas](#). For specific information about working with read replicas on Amazon RDS for MySQL, see [Working with MySQL read replicas](#).

You can use global transaction identifiers (GTIDs) for replication with RDS for MySQL. For more information, see [Using GTID-based replication](#).

You can also set up replication between an RDS for MySQL DB instance and a MariaDB or MySQL instance that is external to Amazon RDS. For information about configuring replication with an external source, see [Configuring binary log file position replication with an external source instance](#).

For any of these replication options, you can use either row-based replication, statement-based, or mixed replication. Row-based replication only replicates the changed rows that result from a SQL statement. Statement-based replication replicates the entire SQL statement. Mixed replication uses statement-based replication when possible, but switches to row-based replication when SQL statements that are unsafe for statement-based replication are run. In most cases, mixed replication is recommended. The binary log format of the DB instance determines whether replication is row-based, statement-based, or mixed. For information about setting the binary log format, see [Configuring RDS for MySQL binary logging for Single-AZ databases](#).

Note

You can configure replication to import databases from a MariaDB or MySQL instance that is external to Amazon RDS, or to export databases to such instances. For more information, see [Importing data to an Amazon RDS for MySQL database with reduced downtime](#) and [Exporting data from a MySQL DB instance by using replication](#).

After you restore your DB instance from a snapshot or perform a point-in-time recovery, you can view the last recovered binlog position from the source database in the RDS console. Under **Logs & events**, enter **binlog**. The binlog position appears under **System notes**.

Topics

- [Working with MySQL read replicas](#)

- [Using GTID-based replication](#)
- [Configuring binary log file position replication with an external source instance](#)
- [Configuring multi-source-replication for Amazon RDS for MySQL](#)

Working with MySQL read replicas

Following, you can find specific information about working with read replicas on RDS for MySQL. For general information about read replicas and instructions for using them, see [Working with DB instance read replicas](#).

For more information about MySQL read replicas, see the following topics.

- [Configuring replication filters with MySQL](#)
- [Configuring delayed replication with MySQL](#)
- [Updating read replicas with MySQL](#)
- [Working with Multi-AZ read replica deployments with MySQL](#)
- [Using cascading read replicas with RDS for MySQL](#)
- [Monitoring replication lag for MySQL read replicas](#)
- [Starting and stopping replication with MySQL read replicas](#)
- [Troubleshooting a MySQL read replica problem](#)

Configuring read replicas with MySQL

Before a MySQL DB instance can serve as a replication source, make sure to enable automatic backups on the source DB instance. To do this, set the backup retention period to a value other than 0. This requirement also applies to a read replica that is the source DB instance for another read replica. Automatic backups are supported for read replicas running any version of MySQL. You can configure replication based on binary log coordinates for a MySQL DB instance.

You can configure replication using global transaction identifiers (GTIDS) on the following versions:

- RDS for MySQL version 5.7.44 and higher 5.7 versions
- RDS for MySQL version 8.0.28 and higher 8.0 versions
- RDS for MySQL version 8.4.3 and higher 8.4 versions

For more information, see [Using GTID-based replication](#).

You can create up to 15 read replicas from one DB instance within the same Region. For replication to operate effectively, each read replica should have the same amount of compute and storage resources as the source DB instance. If you scale the source DB instance, also scale the read replicas.

RDS for MySQL supports cascading read replicas. To learn how to configure cascading read replicas, see [Using cascading read replicas with RDS for MySQL](#).

You can run multiple read replica create and delete actions at the same time that reference the same source DB instance. When you perform these actions, stay within the limit of 15 read replicas for each source instance.

A read replica of a MySQL DB instance can't use a lower DB engine version than its source DB instance.

Preparing MySQL DB instances that use MyISAM

If your MySQL DB instance uses a nontransactional engine such as MyISAM, you need to perform the following steps to successfully set up your read replica. These steps are required to make sure that the read replica has a consistent copy of your data. These steps are not required if all of your tables use a transactional engine such as InnoDB.

1. Stop all data manipulation language (DML) and data definition language (DDL) operations on non-transactional tables in the source DB instance and wait for them to complete. SELECT statements can continue running.
2. Flush and lock the tables in the source DB instance.
3. Create the read replica using one of the methods in the following sections.
4. Check the progress of the read replica creation using, for example, the `DescribeDBInstances` API operation. Once the read replica is available, unlock the tables of the source DB instance and resume normal database operations.

Configuring replication filters with MySQL

You can use replication filters to specify which databases and tables are replicated with a read replica. Replication filters can include databases and tables in replication or exclude them from replication.

The following are some use cases for replication filters:

- To reduce the size of a read replica. With replication filtering, you can exclude the databases and tables that aren't needed on the read replica.
- To exclude databases and tables from read replicas for security reasons.
- To replicate different databases and tables for specific use cases at different read replicas. For example, you might use specific read replicas for analytics or sharding.
- For a DB instance that has read replicas in different AWS Regions, to replicate different databases or tables in different AWS Regions.

 **Note**

You can also use replication filters to specify which databases and tables are replicated with a primary MySQL DB instance that is configured as a replica in an inbound replication topology. For more information about this configuration, see [Configuring binary log file position replication with an external source instance](#).

Topics

- [Setting replication filtering parameters for RDS for MySQL](#)
- [Replication filtering limitations for RDS for MySQL](#)
- [Replication filtering examples for RDS for MySQL](#)
- [Viewing the replication filters for a read replica](#)

Setting replication filtering parameters for RDS for MySQL

To configure replication filters, set the following replication filtering parameters on the read replica:

- `replicate-do-db` – Replicate changes to the specified databases. When you set this parameter for a read replica, only the databases specified in the parameter are replicated.
- `replicate-ignore-db` – Don't replicate changes to the specified databases. When the `replicate-do-db` parameter is set for a read replica, this parameter isn't evaluated.
- `replicate-do-table` – Replicate changes to the specified tables. When you set this parameter for a read replica, only the tables specified in the parameter are replicated. Also, when the `replicate-do-db` or `replicate-ignore-db` parameter is set, make sure to include the database that includes the specified tables in replication with the read replica.

- `replicate-ignore-table` – Don't replicate changes to the specified tables. When the `replicate-do-table` parameter is set for a read replica, this parameter isn't evaluated.
- `replicate-wild-do-table` – Replicate tables based on the specified database and table name patterns. The % and _ wildcard characters are supported. When the `replicate-do-db` or `replicate-ignore-db` parameter is set, make sure to include the database that includes the specified tables in replication with the read replica.
- `replicate-wild-ignore-table` – Don't replicate tables based on the specified database and table name patterns. The % and _ wildcard characters are supported. When the `replicate-do-table` or `replicate-wild-do-table` parameter is set for a read replica, this parameter isn't evaluated.

The parameters are evaluated in the order that they are listed. For more information about how these parameters work, see the MySQL documentation:

- For general information, see [Replica Server Options and Variables](#).
- For information about how database replication filtering parameters are evaluated, see [Evaluation of Database-Level Replication and Binary Logging Options](#).
- For information about how table replication filtering parameters are evaluated, see [Evaluation of Table-Level Replication Options](#).

By default, each of these parameters has an empty value. On each read replica, you can use these parameters to set, change, and delete replication filters. When you set one of these parameters, separate each filter from others with a comma.

You can use the % and _ wildcard characters in the `replicate-wild-do-table` and `replicate-wild-ignore-table` parameters. The % wildcard matches any number of characters, and the _ wildcard matches only one character.

The binary logging format of the source DB instance is important for replication because it determines the record of data changes. The setting of the `binlog_format` parameter determines whether the replication is row-based or statement-based. For more information, see [Configuring RDS for MySQL binary logging for Single-AZ databases](#).

Note

All data definition language (DDL) statements are replicated as statements, regardless of the binlog_format setting on the source DB instance.

Replication filtering limitations for RDS for MySQL

The following limitations apply to replication filtering for RDS for MySQL:

- Each replication filtering parameter has a 2,000-character limit.
- Commas aren't supported in replication filters for parameter values. In a list of parameters, commas can only be used as value separators. For example, ParameterValue='`a,b' isn't supported, but ParameterValue='a,b' is.
- The MySQL --binlog-do-db and --binlog-ignore-db options for binary log filtering aren't supported.
- Replication filtering doesn't support XA transactions.

For more information, see [Restrictions on XA Transactions](#) in the MySQL documentation.

Replication filtering examples for RDS for MySQL

To configure replication filtering for a read replica, modify the replication filtering parameters in the parameter group associated with the read replica.

Note

You can't modify a default parameter group. If the read replica is using a default parameter group, create a new parameter group and associate it with the read replica. For more information on DB parameter groups, see [Parameter groups for Amazon RDS](#).

You can set parameters in a parameter group using the AWS Management Console, AWS CLI, or RDS API. For information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#). When you set parameters in a parameter group, all of the DB instances associated with the parameter group use the parameter settings. If you set the replication filtering parameters in a parameter group, make sure that the parameter group is associated only with read replicas. Leave the replication filtering parameters empty for source DB instances.

The following examples set the parameters using the AWS CLI. These examples set `ApplyMethod` to `immediate` so that the parameter changes occur immediately after the CLI command completes. If you want a pending change to be applied after the read replica is rebooted, set `ApplyMethod` to `pending-reboot`.

The following examples set replication filters:

- [Including databases in replication](#)
- [Including tables in replication](#)
- [Including tables in replication with wildcard characters](#)
- [Excluding databases from replication](#)
- [Excluding tables from replication](#)
- [Excluding tables from replication using wildcard characters](#)

Example Including databases in replication

The following example includes the `mydb1` and `mydb2` databases in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myparametergroup ^
--parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example Including tables in replication

The following example includes the `table1` and `table2` tables in database `mydb1` in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myparametergroup ^
--parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example Including tables in replication using wildcard characters

The following example includes tables with names that begin with `order` and `return` in database `mydb` in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myparametergroup ^
--parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example Excluding databases from replication

The following example excludes the `mydb5` and `mydb6` databases from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myparametergroup ^
--parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

Example Excluding tables from replication

The following example excludes tables `table1` in database `mydb5` and `table2` in database `mydb6` from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myparametergroup ^
--parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example Excluding tables from replication using wildcard characters

The following example excludes tables with names that begin with `order` and `return` in database `mydb7` from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myparametergroup ^
--parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Viewing the replication filters for a read replica

You can view the replication filters for a read replica in the following ways:

- Check the settings of the replication filtering parameters in the parameter group associated with the read replica.

For instructions, see [Viewing parameter values for a DB parameter group in Amazon RDS](#).

- In a MySQL client, connect to the read replica and run the SHOW REPLICAS STATUS statement.

In the output, the following fields show the replication filters for the read replica:

- Replicate_Do_DB
- Replicate_Ignore_DB
- Replicate_Do_Table
- Replicate_Ignore_Table
- Replicate_Wild_Do_Table
- Replicate_Wild_Ignore_Table

For more information about these fields, see [Checking Replication Status](#) in the MySQL documentation.

Configuring delayed replication with MySQL

You can use delayed replication as a strategy for disaster recovery. With delayed replication, you specify the minimum amount of time, in seconds, to delay replication from the source to the

read replica. In the event of a disaster, such as a table deleted unintentionally, you complete the following steps to recover from the disaster quickly:

- Stop replication to the read replica before the change that caused the disaster is sent to it.

Use the [mysql.rds_stop_replication](#) stored procedure to stop replication.

- Start replication and specify that replication stops automatically at a log file location.

You specify a location just before the disaster using the [mysql.rds_start_replication_until](#) stored procedure.

- Promote the read replica to be the new source DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

Note

- On RDS for MySQL 8.4, delayed replication is supported for MySQL 8.4.3 and higher. On RDS for MySQL 8.0, delayed replication is supported for MySQL 8.0.28 and higher. On RDS for MySQL 5.7, delayed replication is supported for MySQL 5.7.44 and higher.
- Use stored procedures to configure delayed replication. You can't configure delayed replication with the AWS Management Console, the AWS CLI, or the Amazon RDS API.
- You can use replication based on global transaction identifiers (GTIDs) in a delayed replication configuration for the following versions:
 - RDS for MySQL version 5.7.44 and higher 5.7 versions
 - RDS for MySQL version 8.0.28 and higher 8.0 versions
 - RDS for MySQL version 8.4.3 and higher 8.4 versions

If you use GTID-based replication, use the [mysql.rds_start_replication_until_gtid](#) stored procedure instead of the [mysql.rds_start_replication_until](#) stored procedure. For more information about GTID-based replication, see [Using GTID-based replication](#).

Topics

- [Configuring delayed replication during read replica creation](#)
- [Modifying delayed replication for an existing read replica](#)
- [Setting a location to stop replication to a read replica](#)

- [Promoting a read replica](#)

Configuring delayed replication during read replica creation

To configure delayed replication for any future read replica created from a DB instance, run the [`mysql.rds_set_configuration`](#) stored procedure with the `target_delay` parameter.

To configure delayed replication during read replica creation

1. Using a MySQL client, connect to the MySQL DB instance to be the source for read replicas as the master user.
2. Run the [`mysql.rds_set_configuration`](#) stored procedure with the `target_delay` parameter.

For example, run the following stored procedure to specify that replication is delayed by at least one hour (3,600 seconds) for any read replica created from the current DB instance.

```
call mysql.rds_set_configuration('target_delay', 3600);
```

Note

After running this stored procedure, any read replica you create using the AWS CLI or Amazon RDS API is configured with replication delayed by the specified number of seconds.

Modifying delayed replication for an existing read replica

To modify delayed replication for an existing read replica, run the [`mysql.rds_set_source_delay`](#) stored procedure.

To modify delayed replication for an existing read replica

1. Using a MySQL client, connect to the read replica as the master user.
2. Use the [`mysql.rds_stop_replication`](#) stored procedure to stop replication.
3. Run the [`mysql.rds_set_source_delay`](#) stored procedure.

For example, run the following stored procedure to specify that replication to the read replica is delayed by at least one hour (3600 seconds).

```
call mysql.rds_set_source_delay(3600);
```

4. Use the [mysql.rds_start_replication](#) stored procedure to start replication.

Setting a location to stop replication to a read replica

After stopping replication to the read replica, you can start replication and then stop it at a specified binary log file location using the [mysql.rds_start_replication_until](#) stored procedure.

To start replication to a read replica and stop replication at a specific location

1. Using a MySQL client, connect to the source MySQL DB instance as the master user.
2. Run the [mysql.rds_start_replication_until](#) stored procedure.

The following example initiates replication and replicates changes until it reaches location 120 in the `mysql-bin-changelog.000777` binary log file. In a disaster recovery scenario, assume that location 120 is just before the disaster.

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

Replication stops automatically when the stop point is reached. The following RDS event is generated: Replication has been stopped since the replica reached the stop point specified by the `rds_start_replication_until` stored procedure.

Promoting a read replica

After replication is stopped, in a disaster recovery scenario, you can promote a read replica to be the new source DB instance. For information about promoting a read replica, see [Promoting a read replica to be a standalone DB instance](#).

Updating read replicas with MySQL

Read replicas are designed to support read queries, but you might need occasional updates. For example, you might need to add an index to optimize the specific types of queries accessing the replica.

Although you can enable updates by setting the `read_only` parameter to `0` in the DB parameter group for the read replica, we recommend that you don't do so because it can cause problems if the read replica becomes incompatible with the source DB instance. For maintenance operations, we recommend that you use blue/green deployments. For more information, see [Using Blue/Green Deployments for database updates](#).

If you disable read-only on a read replica, change the value of the `read_only` parameter back to `1` as soon as possible.

Working with Multi-AZ read replica deployments with MySQL

You can create a read replica from either single-AZ or Multi-AZ DB instance deployments. You use Multi-AZ deployments to improve the durability and availability of critical data, but you can't use the Multi-AZ secondary to serve read-only queries. Instead, you can create read replicas from high-traffic Multi-AZ DB instances to offload read-only queries. If the source instance of a Multi-AZ deployment fails over to the secondary, any associated read replicas automatically switch to use the secondary (now primary) as their replication source. For more information, see [Configuring and managing a Multi-AZ deployment for Amazon RDS](#).

You can create a read replica as a Multi-AZ DB instance. Amazon RDS creates a standby of your replica in another Availability Zone for failover support for the replica. Creating your read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance.

Using cascading read replicas with RDS for MySQL

RDS for MySQL supports cascading read replicas. With *cascading read replicas*, you can scale reads without adding overhead to your source RDS for MySQL DB instance.

With cascading read replicas, your RDS for MySQL DB instance sends data to the first read replica in the chain. That read replica then sends data to the second replica in the chain, and so on. The end result is that all read replicas in the chain have the changes from the RDS for MySQL DB instance, but without the overhead solely on the source DB instance.

You can create a series of up to three read replicas in a chain from a source RDS for MySQL DB instance. For example, suppose that you have an RDS for MySQL DB instance, `mysql-main`. You can do the following:

- Starting with `mysql-main`, create the first read replica in the chain, `read-replica-1`.
- Next, from `read-replica-1`, create the next read replica in the chain, `read-replica-2`.
- Finally, from `read-replica-2`, create the third read replica in the chain, `read-replica-3`.

You can't create another read replica beyond this third cascading read replica in the series for `mysql-main`. A complete series of instances from an RDS for MySQL source DB instance through to the end of a series of cascading read replicas can consist of at most four DB instances.

For cascading read replicas to work, each source RDS for MySQL DB instance must have automated backups turned on. To turn on automatic backups on a read replica, first create the read replica, and then modify the read replica to turn on automatic backups. For more information, see [Creating a read replica](#).

As with any read replica, you can promote a read replica that's part of a cascade. Promoting a read replica from within a chain of read replicas removes that replica from the chain. For example, suppose that you want to move some of the workload from your `mysql-main` DB instance to a new instance for use by the accounting department only. Assuming the chain of three read replicas from the example, you decide to promote `read-replica-2`. The chain is affected as follows:

- Promoting `read-replica-2` removes it from the replication chain.
 - It is now a full read/write DB instance.
 - It continues replicating to `read-replica-3`, just as it was doing before promotion.
- Your `mysql-main` continues replicating to `read-replica-1`.

For more information about promoting read replicas, see [Promoting a read replica to be a standalone DB instance](#).

Monitoring replication lag for MySQL read replicas

For MySQL read replicas, you can monitor replication lag in Amazon CloudWatch by viewing the Amazon RDS ReplicaLag metric. The ReplicaLag metric reports the value of the `Seconds_Behind_Master` field of the `SHOW REPLICA STATUS` command.

Common causes for replication lag for MySQL are the following:

- A network outage.
- Writing to tables that have different indexes on a read replica. If the `read_only` parameter is set to `0` on the read replica, replication can break if the read replica becomes incompatible with the source DB instance. After you've performed maintenance tasks on the read replica, we recommend that you set the `read_only` parameter back to `1`.
- Using a nontransactional storage engine such as MyISAM. Replication is only supported for the InnoDB storage engine on MySQL.

When the ReplicaLag metric reaches 0, the replica has caught up to the source DB instance. If the ReplicaLag metric returns -1, then replication is currently not active. ReplicaLag = -1 is equivalent to Seconds_Behind_Master = NULL.

Starting and stopping replication with MySQL read replicas

You can stop and restart the replication process on an Amazon RDS DB instance by calling the system stored procedures [mysql.rds_stop_replication](#) and [mysql.rds_start_replication](#). You can do this when replicating between two Amazon RDS instances for long-running operations such as creating large indexes. You also need to stop and start replication when importing or exporting databases. For more information, see [Importing data to an Amazon RDS for MySQL database with reduced downtime](#) and [Exporting data from a MySQL DB instance by using replication](#).

If replication is stopped for more than 30 consecutive days, either manually or due to a replication error, Amazon RDS terminates replication between the source DB instance and all read replicas. It does so to prevent increased storage requirements on the source DB instance and long failover times. The read replica DB instance is still available. However, replication can't be resumed because the binary logs required by the read replica are deleted from the source DB instance after replication is terminated. You can create a new read replica for the source DB instance to reestablish replication.

Troubleshooting a MySQL read replica problem

For MySQL DB instances, in some cases read replicas present replication errors or data inconsistencies (or both) between the read replica and its source DB instance. This problem occurs when some binary log (binlog) events or InnoDB redo logs aren't flushed during a failure of the read replica or the source DB instance. In these cases, manually delete and recreate the read replicas. You can reduce the chance of this happening by setting the following parameter values: sync_binlog=1 and innodb_flush_log_at_trx_commit=1. These settings might reduce performance, so test their impact before implementing the changes in a production environment.

Warning

In the parameter group associated with the source DB instance, we recommend keeping these parameter values: sync_binlog=1 and innodb_flush_log_at_trx_commit=1. These parameters are dynamic. If you don't want to use these settings, we recommend temporarily setting those values before executing any operation on the source DB instance that might cause it to restart. These operations include, but are not limited to, rebooting, rebooting with failover, upgrading the database version, and changing the DB instance

class or its storage. The same recommendation applies to creating new read replicas for the source DB instance.

Failure to follow this guidance increases the risk of read replicas presenting replication errors or data inconsistencies (or both) between the read replica and its source DB instance.

The replication technologies for MySQL are asynchronous. Because they are asynchronous, occasional BinLogDiskUsage increases on the source DB instance and ReplicaLag on the read replica are to be expected. For example, a high volume of write operations to the source DB instance can occur in parallel. In contrast, write operations to the read replica are serialized using a single I/O thread, which can lead to a lag between the source instance and read replica. For more information about read-only replicas in the MySQL documentation, see [Replication implementation details](#).

You can do several things to reduce the lag between updates to a source DB instance and the subsequent updates to the read replica, such as the following:

- Sizing a read replica to have a storage size and DB instance class comparable to the source DB instance.
- Ensuring that parameter settings in the DB parameter groups used by the source DB instance and the read replica are compatible. For more information and an example, see the discussion of the `max_allowed_packet` parameter later in this section.

Amazon RDS monitors the replication status of your read replicas and updates the `Replication State` field of the read replica instance to `Error` if replication stops for any reason. An example might be if DML queries run on your read replica conflict with the updates made on the source DB instance.

You can review the details of the associated error thrown by the MySQL engine by viewing the `Replication Error` field. Events that indicate the status of the read replica are also generated, including [RDS-EVENT-0045](#), [RDS-EVENT-0046](#), and [RDS-EVENT-0047](#). For more information about events and subscribing to events, see [Working with Amazon RDS event notification](#). If a MySQL error message is returned, review the error number in the [MySQL error message documentation](#).

One common issue that can cause replication errors is when the value for the `max_allowed_packet` parameter for a read replica is less than the `max_allowed_packet` parameter for the source DB instance. The `max_allowed_packet` parameter is a custom parameter that you can set in a DB parameter group. You use `max_allowed_packet` to

specify the maximum size of DML code that can be run on the database. In some cases, the `max_allowed_packet` value in the DB parameter group associated with a read replica is smaller than the `max_allowed_packet` value in the DB parameter group associated with the source DB instance. In these cases, the replication process can throw the error `Packet bigger than 'max_allowed_packet' bytes` and stop replication. To fix the error, have the source DB instance and read replica use DB parameter groups with the same `max_allowed_packet` parameter values.

Other common situations that can cause replication errors include the following:

- Writing to tables on a read replica. In some cases, you might create indexes on a read replica that are different from the indexes on the source DB instance. If you do, set the `read_only` parameter to 0 to create the indexes. If you write to tables on the read replica, it might break replication if the read replica becomes incompatible with the source DB instance. After you perform maintenance tasks on the read replica, we recommend that you set the `read_only` parameter back to 1.
- Using a non-transactional storage engine such as MyISAM. Read replicas require a transactional storage engine. Replication is only supported for the InnoDB storage engine on MySQL.
- Using unsafe nondeterministic queries such as `SYSDATE()`. For more information, see [Determination of safe and unsafe statements in binary logging](#).

If you decide that you can safely skip an error, you can follow the steps described in the section [Skipping the current replication error for RDS for MySQL](#). Otherwise, you can first delete the read replica. Then you create an instance using the same DB instance identifier so that the endpoint remains the same as that of your old read replica. If a replication error is fixed, the Replication State changes to *replicating*.

Using GTID-based replication

The following content explains how to use global transaction identifiers (GTIDs) with binary log (binlog) replication among Amazon RDS for MySQL DB instances.

If you use binlog replication and aren't familiar with GTID-based replication with MySQL, see [Replication with global transaction identifiers](#) in the MySQL documentation.

GTID-based replication is supported for the following versions:

- All RDS for MySQL 8.4 versions

- All RDS for MySQL 8.0 versions
- All RDS for MySQL 5.7 versions

All MySQL DB instances in a replication configuration must meet this version requirement.

Topics

- [Overview of global transaction identifiers \(GTIDs\)](#)
- [Parameters for GTID-based replication](#)
- [Enabling GTID-based replication for new read replicas for RDS for MySQL](#)
- [Enabling GTID-based replication for existing read replicas for RDS for MySQL](#)
- [Disabling GTID-based replication for a MySQL DB instance with read replicas](#)

Overview of global transaction identifiers (GTIDs)

Global transaction identifiers (GTIDs) are unique identifiers generated for committed MySQL transactions. You can use GTIDs to make binlog replication simpler and easier to troubleshoot.

MySQL uses two different types of transactions for binlog replication:

- *GTID transactions* – Transactions that are identified by a GTID.
- *Anonymous transactions* – Transactions that don't have a GTID assigned.

In a replication configuration, GTIDs are unique across all DB instances. GTIDs simplify replication configuration because when you use them, you don't have to refer to log file positions. GTIDs also make it easier to track replicated transactions and determine whether the source instance and replicas are consistent.

You can use GTID-based replication to replicate data with RDS for MySQL read replicas. You can configure GTID-based replication when you are creating new read replicas, or you can convert existing read replicas to use GTID-based replication.

You can also use GTID-based replication in a delayed replication configuration with RDS for MySQL. For more information, see [Configuring delayed replication with MySQL](#).

Parameters for GTID-based replication

Use the following parameters to configure GTID-based replication.

Parameter	Valid values	Description
gtid_mode	OFF, OFF_PERMISSIVE , ON_PERMISSIVE , ON	<p>OFF specifies that new transactions are anonymous transactions (that is, don't have GTIDs), and a transaction must be anonymous to be replicated.</p> <p>OFF_PERMISSIVE specifies that new transactions are anonymous transactions, but all transactions can be replicated.</p> <p>ON_PERMISSIVE specifies that new transactions are GTID transactions, but all transactions can be replicated.</p> <p>ON specifies that new transactions are GTID transactions, and a transaction must be a GTID transaction to be replicated.</p>
enforce_gtid_consistency	OFF, ON, WARN	<p>OFF allows transactions to violate GTID consistency.</p> <p>ON prevents transactions from violating GTID consistency.</p> <p>WARN allows transactions to violate GTID consistency but generates a warning when a violation occurs.</p>

Note

In the AWS Management Console, the gtid_mode parameter appears as gtid-mode.

For GTID-based replication, use these settings for the parameter group for your DB instance or read replica:

- ON and ON_PERMISSIVE apply only to outgoing replication from an RDS DB instance. Both of these values cause your RDS DB instance to use GTIDs for transactions that are replicated. ON requires that the target database also use GTID-based replication. ON_PERMISSIVE makes GTID-based replication optional on the target database.
- OFF_PERMISSIVE, if set, means that your RDS DB instances can accept incoming replication from a source database. They can do this regardless of whether the source database uses GTID-based replication.
- OFF, if set, means that your RDS DB instance only accepts incoming replication from source databases that don't use GTID-based replication.

For more information about parameter groups, see [Parameter groups for Amazon RDS](#).

Enabling GTID-based replication for new read replicas for RDS for MySQL

When GTID-based replication is enabled for an RDS for MySQL DB instance, GTID-based replication is configured automatically for read replicas of the DB instance.

To enable GTID-based replication for new read replicas

1. Make sure that the parameter group associated with the DB instance has the following parameter settings:
 - `gtid_mode` – ON or ON_PERMISSIVE
 - `enforce_gtid_consistency` – ON

For more information about setting configuration parameters using parameter groups, see [Parameter groups for Amazon RDS](#).

2. If you changed the parameter group of the DB instance, reboot the DB instance. For more information on how to do so, see [Rebooting a DB instance](#).
3. Create one or more read replicas of the DB instance. For more information on how to do so, see [Creating a read replica](#).

Amazon RDS attempts to establish GTID-based replication between the MySQL DB instance and the read replicas using the `MASTER_AUTO_POSITION`. If the attempt fails, Amazon RDS uses log file positions for replication with the read replicas. For more information about the `MASTER_AUTO_POSITION`, see [GTID auto-positioning](#) in the MySQL documentation.

Enabling GTID-based replication for existing read replicas for RDS for MySQL

For an existing MySQL DB instance with read replicas that doesn't use GTID-based replication, you can configure GTID-based replication between the DB instance and the read replicas.

To enable GTID-based replication for existing read replicas

1. If the DB instance or any read replica is using an 8.0 version of RDS for MySQL version lower than 8.0.26, upgrade the DB instance or read replica to 8.0.26 or a higher MySQL 8.0 version. All RDS for MySQL 8.4 versions and 5.7 versions support GTID-based replication.

For more information, see [Upgrades of the RDS for MySQL DB engine](#).

2. (Optional) Reset the GTID parameters and test the behavior of the DB instance and read replicas:
 - a. Make sure that the parameter group associated with the DB instance and each read replica has the `enforce_gtid_consistency` parameter set to `WARN`.

For more information about setting configuration parameters using parameter groups, see [Parameter groups for Amazon RDS](#).

- b. If you changed the parameter group of the DB instance, reboot the DB instance. If you changed the parameter group for a read replica, reboot the read replica.

For more information, see [Rebooting a DB instance](#).

- c. Run your DB instance and read replicas with your normal workload and monitor the log files.

If you see warnings about GTID-incompatible transactions, adjust your application so that it only uses GTID-compatible features. Make sure that the DB instance is not generating any warnings about GTID-incompatible transactions before proceeding to the next step.

3. Reset the GTID parameters for GTID-based replication that allows anonymous transactions until the read replicas have processed all of them.
 - a. Make sure that the parameter group associated with the DB instance and each read replica has the following parameter settings:
 - `gtid_mode` – `ON_PERMISSIVE`
 - `enforce_gtid_consistency` – `ON`

- b. If you changed the parameter group of the DB instance, reboot the DB instance. If you changed the parameter group for a read replica, reboot the read replica.
4. Wait for all of your anonymous transactions to be replicated. To check that these are replicated, do the following:
 - a. Run the following statement on your source DB instance.

MySQL 8.4

```
SHOW BINARY LOG STATUS;
```

MySQL 5.7 and 8.0

```
SHOW MASTER STATUS;
```

Note the values in the File and Position columns.

- b. On each read replica, use the file and position information from its source instance in the previous step to run the following query.

```
SELECT MASTER_POS_WAIT('file', position);
```

For example, if the file name is mysql-bin-changelog.000031 and the position is 107, run the following statement.

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

If the read replica is past the specified position, the query returns immediately. Otherwise, the function waits. Proceed to the next step when the query returns for all read replicas.

5. Reset the GTID parameters for GTID-based replication only.
 - a. Make sure that the parameter group associated with the DB instance and each read replica has the following parameter settings:
 - gtid_mode – ON
 - enforce_gtid_consistency – ON
 - b. Reboot the DB instance and each read replica.

6. On each read replica, run the following procedure.

MySQL 8.4 and higher major versions

```
CALL mysql.rds_set_source_auto_position(1);
```

MySQL 8.0 and lower major versions

```
CALL mysql.rds_set_master_auto_position(1);
```

Disabling GTID-based replication for a MySQL DB instance with read replicas

You can disable GTID-based replication for a MySQL DB instance with read replicas.

To disable GTID-based replication for a MySQL DB instance with read replicas

1. On each read replica, run the following procedure:

MySQL 8.4 and higher major versions

```
CALL mysql.rds_set_source_auto_position(0);
```

MySQL 8.0 and lower major versions

```
CALL mysql.rds_set_master_auto_position(0);
```

2. Reset the `gtid_mode` to `ON_PERMISSIVE`.

- a. Make sure that the parameter group associated with the MySQL DB instance and each read replica has `gtid_mode` set to `ON_PERMISSIVE`.

For more information about setting configuration parameters using parameter groups, see [Parameter groups for Amazon RDS](#).

- b. Reboot the MySQL DB instance and each read replica. For more information about rebooting, see [Rebooting a DB instance](#).

3. Reset the `gtid_mode` to `OFF_PERMISSIVE`.

- a. Make sure that the parameter group associated with the MySQL DB instance and each read replica has `gtid_mode` set to `OFF_PERMISSIVE`.

- b. Reboot the MySQL DB instance and each read replica.
4. Wait for all of the GTID transactions to be applied on all of the read replicas. To check that these are applied, do the following steps:
- a. On the MySQL DB instance, run the following command:

MySQL 8.4

```
SHOW BINARY LOG STATUS
```

MySQL 5.7 and 8.0

```
SHOW MASTER STATUS
```

Your output should be similar to the following output.

File	Position
mysql-bin-changelog.000031	107

Note the file and position in your output.

- b. On each read replica, use the file and position information from its source instance in the previous step to run the following query:

MySQL 8.4 and MySQL 8.0.26 and higher MySQL 8.0 versions

```
SELECT SOURCE_POS_WAIT('file', position);
```

MySQL 5.7

```
SELECT MASTER_POS_WAIT('file', position);
```

For example, if the file name is mysql-bin-changelog.000031 and the position is 107, run the following statement:

MySQL 8.4 and MySQL 8.0.26 and higher MySQL 8.0 versions

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

MySQL 5.7

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. Reset the GTID parameters to disable GTID-based replication.

- a. Make sure that the parameter group associated with the MySQL DB instance and each read replica has the following parameter settings:
 - `gtid_mode` – OFF
 - `enforce_gtid_consistency` – OFF
- b. Reboot the MySQL DB instance and each read replica.

Configuring binary log file position replication with an external source instance

You can set up replication between an RDS for MySQL or MariaDB DB instance and a MySQL or MariaDB instance that is external to Amazon RDS using binary log file replication.

Topics

- [Before you begin](#)
- [Configuring binary log file position replication with an external source instance](#)

Before you begin

You can configure replication using the binary log file position of replicated transactions.

The permissions required to start replication on an Amazon RDS DB instance are restricted and not available to your Amazon RDS master user. Because of this, make sure that you use the Amazon RDS [mysql.rds_set_external_master](#) (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) or [mysql.rds_set_external_source](#) (RDS for MySQL major versions 8.4 and higher), and [mysql.rds_start_replication](#) commands to set up replication between your live database and your Amazon RDS database.

To set the binary logging format for a MySQL or MariaDB database, update the `binlog_format` parameter. If your DB instance uses the default DB instance parameter group, create a new DB parameter group to modify the `binlog_format` parameter. In MariaDB and MySQL 8.0 and lower versions, `binlog_format` defaults to MIXED. However, you can also set `binlog_format` to ROW or STATEMENT if you need a specific binary log (binlog) format. Reboot your DB instance for the change to take effect. In MySQL 8.4 and higher versions, `binlog_format` defaults to ROW.

For information about setting the `binlog_format` parameter, see [Configuring RDS for MySQL binary logging for Single-AZ databases](#). For information about the implications of different MySQL replication types, see [Advantages and disadvantages of statement-based and row-based replication](#) in the MySQL documentation.

Configuring binary log file position replication with an external source instance

Follow these guidelines when you set up an external source instance and a replica on Amazon RDS:

- Monitor failover events for the Amazon RDS DB instance that is your replica. If a failover occurs, then the DB instance that is your replica might be recreated on a new host with a different network address. For information on how to monitor failover events, see [Working with Amazon RDS event notification](#).
- Maintain the binlogs on your source instance until you have verified that they have been applied to the replica. This maintenance makes sure that you can restore your source instance in the event of a failure.
- Turn on automated backups on your Amazon RDS DB instance. Turning on automated backups makes sure that you can restore your replica to a particular point in time if you need to re-synchronize your source instance and replica. For information on backups and point-in-time restore, see [Backing up, restoring, and exporting data](#).

To configure binary log file replication with an external source instance

1. Make the source MySQL or MariaDB instance read-only.

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. Run the `SHOW MASTER STATUS` command on the source MySQL or MariaDB instance to determine the binlog location.

You receive output similar to the following example.

File	Position
mysql-bin-changelog.000031	107

3. Copy the database from the external instance to the Amazon RDS DB instance using `mysqldump`. For very large databases, you might want to use the procedure in [Importing data to an Amazon RDS for MySQL database with reduced downtime](#).

For Linux, macOS, or Unix:

```
mysqldump --databases database_name \
--single-transaction \
--compress \
--order-by-primary \
-u local_user \
-plocal_password | mysql \
--host=hostname \
--port=3306 \
-u RDS_user_name \
-pRDS_password
```

For Windows:

```
mysqldump --databases database_name ^
--single-transaction ^
--compress ^
--order-by-primary ^
-u local_user ^
-plocal_password | mysql ^
--host=hostname ^
--port=3306 ^
-u RDS_user_name ^
-pRDS_password
```

 **Note**

Make sure that there isn't a space between the `-p` option and the entered password.

To specify the host name, user name, port, and password to connect to your Amazon RDS DB instance, use the `--host`, `--user` (`-u`), `--port` and `-p` options in the `mysql` command. The host name is the Domain Name Service (DNS) name from the Amazon RDS DB instance endpoint, for example `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the instance details in the AWS Management Console.

4. Make the source MySQL or MariaDB instance writeable again.

```
mysql> SET GLOBAL read_only = OFF;  
mysql> UNLOCK TABLES;
```

For more information on making backups for use with replication, see [the MySQL documentation](#).

5. In the AWS Management Console, add the IP address of the server that hosts the external database to the virtual private cloud (VPC) security group for the Amazon RDS DB instance. For more information on modifying a VPC security group, see [Security groups for your VPC](#) in the *Amazon Virtual Private Cloud User Guide*.

The IP address can change when the following conditions are met:

- You are using a public IP address for communication between the external source instance and the DB instance.
- The external source instance was stopped and restarted.

If these conditions are met, verify the IP address before adding it.

You might also need to configure your local network to permit connections from the IP address of your Amazon RDS DB instance. You do this so that your local network can communicate with your external MySQL or MariaDB instance. To find the IP address of the Amazon RDS DB instance, use the `host` command.

```
host db_instance_endpoint
```

The host name is the DNS name from the Amazon RDS DB instance endpoint.

- Using the client of your choice, connect to the external instance and create a user to use for replication. Use this account solely for replication and restrict it to your domain to improve security. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

 **Note**

Specify a password other than the prompt shown here as a security best practice.

- For the external instance, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. For example, to grant the REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the '*repl_user*' user for your domain, issue the following command.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

- Make the Amazon RDS DB instance the replica. To do so, first connect to the Amazon RDS DB instance as the master user. Then identify the external MySQL or MariaDB database as the source instance by using the [mysql.rds_set_external_source \(RDS for MySQL major versions 8.4 and higher\)](#) or [mysql.rds_set_external_master \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#) command. Use the master log file name and master log position that you determined in step 2. The following commands are examples.

MySQL 8.4

```
CALL mysql.rds_set_external_source ('mysourceserver.mydomain.com', 3306,  
          'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

MariaDB and MySQL 8.0 and 5.7

```
CALL mysql.rds_set_external_master ('mymasterserver.mydomain.com', 3306,  
          'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

 **Note**

On RDS for MySQL, you can choose to use delayed replication by running the [mysql.rds_set_external_source_with_delay \(RDS for MySQL major versions 8.4\)](#)

[and higher\) or mysql.rds_set_external_master_with_delay \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#) stored procedure instead. On RDS for MySQL, one reason to use delayed replication is to turn on disaster recovery with the [mysql.rds_start_replication_until](#) stored procedure. Currently, RDS for MariaDB supports delayed replication but doesn't support the `mysql.rds_start_replication_until` procedure.

9. On the Amazon RDS DB instance, issue the [mysql.rds_start_replication](#) command to start replication.

```
CALL mysql.rds_start_replication;
```

Configuring multi-source-replication for Amazon RDS for MySQL

With multi-source replication, you can set up an Amazon RDS for MySQL DB instance as a replica that receives binary log events from more than one RDS for MySQL source DB instance. Multi-source replication is supported for RDS for MySQL DB instances running the following engine versions:

- All MySQL 8.4 versions
- 8.0.35 and higher minor versions
- 5.7.44 and higher minor versions

For information about MySQL multi-source replication, see [MySQL Multi-Source Replication](#) in the MySQL documentation. The MySQL documentation contains detailed information about this feature, while this topic describes how to configure and manage the multi-source replication channels on your RDS for MySQL DB instances.

Use cases for multi-source replication

The following cases are good candidates for using multi-source replication on RDS for MySQL:

- Applications that need to merge or combine multiple shards on separate DB instances into a single shard.
- Applications that need to generate reports from data consolidated from multiple sources.

- Requirements to create consolidated long-term backups of data that's distributed among multiple RDS for MySQL DB instances.

Prerequisites for multi-source replication

Before you configure multi-source replication, complete the following prerequisites.

- Make sure that each source RDS for MySQL DB instance has automatic backups enabled. Enabling automatic backups enables binary logging. To learn how to enable automatic backups, see [the section called "Enabling automated backups"](#).
- To avoid replication errors, we recommended that you block write operations to the source DB instances. You can do so by setting the `read-only` parameter to `ON` in a custom parameter group attached to the RDS for MySQL source DB instance. You can use the AWS Management Console or the AWS CLI to create a new custom parameter group or to modify an existing one. For more information, see [the section called "Creating a DB parameter group"](#) and [the section called "Modifying parameters in a DB parameter group"](#).
- For each source DB instance, add the IP address of the instance to the Amazon virtual private cloud (VPC) security group for the multi-source DB instance. To identify the IP address of a source DB instance, you can run the command `dig RDS Endpoint`. Run the command from an Amazon EC2 instance in the same VPC as the destination multi-source DB instance.
- For each source DB instance, use a client to connect to the DB instance and create a database user with the required privileges for replication, as in the following example.

```
CREATE USER 'repl_user' IDENTIFIED BY 'password';  
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user';
```

Note

Starting with MySQL 8.4, the `REPLICATION SLAVE` privilege has been deprecated and replaced with `REPLICATION REPLICA`. For MySQL 8.4 and later versions, use the following syntax instead:

```
CREATE USER 'repl_user' IDENTIFIED BY 'password';  
GRANT REPLICATION CLIENT, REPLICATION REPLICA ON *.* TO 'repl_user';
```

Configuring multi-source replication channels on RDS for MySQL DB instances

Configuring multi-source replication channels is similar to configuring single source replication. For multi-source replication, you first turn on binary logging on the source instance. Then, you import data from the sources to the multi-source replica. Then, you start replication from each source by using the binary log coordinates or by using GTID auto-positioning.

To configure an RDS for MySQL DB instance as a multi-source replica of two or more RDS for MySQL DB instances, perform the following steps.

Topics

- [Step 1: Import data from the source DB instances to the multi-source replica](#)
- [Step 2: Start replication from the source DB instances to the multi-source replica](#)

Step 1: Import data from the source DB instances to the multi-source replica

Perform the following steps on each source DB instance.

Before you import the data from a source to the multi-source replica, determine the current binary log file and position by running the `SHOW MASTER STATUS` command. Take note of these details for use in the next step. In this example output, the file is `mysql-bin-changelog.000031` and the position is `107`.

Note

Starting with MySQL 8.4, the `SHOW MASTER STATUS` command has been deprecated and replaced with `SHOW BINARY LOG STATUS`. For MySQL 8.4 and later versions, use `SHOW BINARY LOG STATUS` instead.

File	Position
<code>mysql-bin-changelog.000031</code>	107

Now copy the database from the source DB instance to the multi-source replica by using `mysqldump`, as in the following example.

```
mysqldump --databases database_name \
```

```
--single-transaction \
--compress \
--order-by-primary \
-u RDS_user_name \
-p RDS_password \
--host=RDS Endpoint | mysql \
--host=RDS Endpoint \
--port=3306 \
-u RDS_user_name \
-p RDS_password
```

After copying the database, you can set the read-only parameter to OFF on the source DB instance.

Step 2: Start replication from the source DB instances to the multi-source replica

For each source DB instance, use the administrative user credentials to connect to the instance, and run the following two stored procedures. These stored procedures configure replication on a channel and start replication. This example uses the binlog file name and position from the example output in the previous step.

```
CALL mysql.rds_set_external_source_for_channel('mysourcehost.example.com', 3306,
'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1, 'channel_1');
CALL mysql.rds_start_replication_for_channel('channel_1');
```

For more information about using these stored procedures and others to set up and manage your replication channels, see [the section called “Managing multi-source replication”](#).

Using filters with multi-source replication

You can use replication filters to specify which databases and tables are replicated with in multi-source replica. Replication filters can include databases and tables in replication or exclude them from replication. For more information on replication filters, see [the section called “Configuring replication filters”](#).

With multi-source replication, you can configure replication filters globally or at the channel level. Channel-level filtering is available only with supported DB instances running version 8.0 or version 8.4. The following examples show how to configure filters globally or at the channel level.

Note the following requirements and behavior with filtering in multi-source replication:

- Back quotes (` `) around the channel names are required.

- If you change replication filters in the parameter group, the multi-source replica's sql_thread for all channels with updates are restarted to apply the changes dynamically. If an update involves a global filter, then all replication channels in the running state are restarted.
- All global filters are applied before any channel-specific filters.
- If a filter is applied globally and at the channel level, then only the channel-level filter is applied. For example, if the filters are replicate_ignore_db="db1, `channel_22` :db2", then replicate_ignore_db set to db1 is applied to all channels except for channel_22, and only channel_22 ignores changes from db2.

Example 1: Setting a global filter

In the following example, the temp_data database is excluded from replication in every channel.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-ignore-
db,ParameterValue='temp_data',ApplyMethod=immediate"
```

Example 2: Setting a channel-level filter

In the following example, changes from the sample22 database are only included in channel channel_22. Similarly, changes from the sample99 database are only included in channel channel_99.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-do-db,ParameterValue='\`channel_22\` :sample22,
\`channel_99\` :sample99',ApplyMethod=immediate"
```

Monitoring multi-source replication channels

You can monitor individual channels in a multi-source replica by using the following methods:

- To monitor the status of all channels or a specific channel, connect to the multi-source replica and run the SHOW REPLICA STATUS or SHOW REPLICA STATUS FOR CHANNEL

'`channel_name`' command. For more information, see [Checking Replication Status](#) in the MySQL documentation.

- To receive notification when a replication channel is started, stopped, or removed, use RDS event notification. For more information, see [the section called “Working with Amazon RDS event notification”](#).
- To monitor the lag for a specific channel, check the `ReplicationChannelLag` metric for it. Data points for this metric have a period of 60 seconds (1 minute) are available for 15 days. To locate the replication channel lag for a channel, use the instance identifier and the replication channel name. To receive notification when this lag exceeds a particular threshold, you can set up a CloudWatch alarm. For more information, see [the section called “Monitoring RDS with CloudWatch”](#).

Considerations and best practices for multi-source replication

Before you use multi-source replication on RDS for MySQL, review the following considerations and best practices:

- Make sure that a DB instance configured as a multi-source replica has sufficient resources such as throughput, memory, CPU, and IOPS to handle the workload from multiple source instances.
- Regularly monitor resource utilization on your multi-source replica and adjust the storage or instance configuration to handle the workload without straining resources.
- You can configure multi-threaded replication on a multi-source replica by setting the system variable `replica_parallel_workers` to a value greater than 0. In this case, the number of threads allocated to each channel is the value of this variable, plus one coordinator thread to manage the applier threads.
- Configure replication filters appropriately to avoid conflicts. To replicate an entire database to another database on a replica, you can use the `--replicate-rewrite-db` option. For example, you can replicate all tables in database A to database B on a replica instance. This approach can be helpful when all source instances are using the same schema naming convention. For information about the `--replicate-rewrite-db` option, see [Replica Server Options and Variables](#) in the MySQL documentation.
- To avoid replication errors, avoid writing to the replica. We recommend that you enable the `read_only` parameter on multi-source replicas to block write operations. Doing so helps to eliminate replication issues caused by conflicting write operations.

- To increase the performance of read operations such as sorts and high-load joins that are executed on the multi-source replica, consider using RDS Optimized Reads. This feature can help with queries that depend on large temporary tables or sort files. For more information, see [the section called “Improving query performance with RDS Optimized Reads”](#).
- To minimize replication lag and improve the performance of a multi-source replica, consider enabling optimized writes. For more information, see [the section called “Improving write performance with RDS Optimized Writes for MySQL”](#).
- Perform management operations (such as changing configuration) on one channel at a time, and avoid performing changes to multiple channels from multiple connections. These practices can lead to conflicts in replication operations. For example, simultaneously executing `rds_skip_repl_error_for_channel` and `rds_start_replication_for_channel` procedures from multiple connections can cause skipping of events on a different channel than intended.
- You can enable backups on a multi-source replication instance and export data from that instance to an Amazon S3 bucket to store it for long-term purposes. However, it's important to also configure backups with appropriate retention on the individual source instances. For information about exporting snapshot data to Amazon S3, see [the section called “Exporting DB snapshot data to Amazon S3”](#).
- To distribute the read workload on a multi-source replica, you can create read replicas from a multi-source replica. You can locate these read replicas in different AWS Regions based on your application's requirements. For more information about read replicas, see [the section called “MySQL read replicas”](#).

Limitations for multi-source replication on RDS for MySQL

The following limitations apply to multi-source replication on RDS for MySQL:

- Currently, RDS for MySQL supports configuring a maximum of 15 channels for a multi-source replica.
- A read replica instance can't be configured as a multi-source replica.
- To configure multi-source replication on RDS for MySQL running engine version 5.7, Performance Schema must be enabled on the replica instance. Enabling Performance Schema is optional on RDS for MySQL running engine version 8.0 or 8.4.

- For RDS for MySQL running engine version 5.7, replication filters apply to all replication channels. For RDS for MySQL running engine version 8.0 or 8.4, you can configure filters that apply to all replication channels or to individual channels.
- Restoring an RDS snapshot or performing a Point-in-time-Restore (PITR) doesn't restore multi-source replica channel configurations.
- When you create a read replica of a multi-source replica, it only replicates data from the multi-source instance. It doesn't restore any channel configuration.
- MySQL doesn't support setting up a different number of parallel workers for each channel. Every channel gets the same number of parallel workers based on the `replica_parallel_workers` value.

The following additional limitations apply if your multi-source replication target is a Multi-AZ DB cluster:

- A channel must be configured for a source RDS for MySQL instance before any writes to that instance occur.
- Each source RDS for MySQL instance must have GTID-based replication enabled.
- A failover event on the DB cluster removes the multi-source replication configuration. Restoring that configuration requires repeating the configuration steps.

Configuring active-active clusters for RDS for MySQL

An active-active cluster in Amazon RDS is a database configuration where multiple nodes actively handle read and write operations, distributing the workload across instances to improve availability and scalability. Each node in the cluster is synchronized to maintain data consistency, enabling high availability and faster failover in case of node failure.

You can set up an active-active cluster for RDS for MySQL by using the MySQL Group Replication plugin. The Group Replication plugin is supported for RDS for MySQL DB instances running the following versions:

- All MySQL 8.4 versions
- MySQL 8.0.35 and higher minor versions

For information about MySQL Group Replication, see [Group Replication](#) in the MySQL documentation. The MySQL documentation contains detailed information about this feature, while this topic describes how to configure and manage the plugin on your RDS for MySQL DB instances.

 **Note**

For the sake of brevity, all mentions of "active-active" cluster in this topic refer to active-active clusters using the MySQL Group Replication plugin.

Use cases for active-active clusters

The following cases are good candidates for using active-active clusters:

- Applications that need all of the DB instances in the cluster to support write operations. The Group Replication plugin keeps the data consistent on each DB instance in the active-active cluster. For more information about how this works, see [Group Replication](#) in the MySQL documentation.
- Applications that require continuous availability of the database. With an active-active cluster, the data is retained on all of the DB instances in the cluster. If one DB instance fails, the application can reroute traffic to another DB instance in the cluster.
- Applications that might need to split read and write operations among different DB instances in the cluster for load balancing purposes. With an active-active cluster, your applications can

send read traffic to specific DB instances and write traffic to others. You can also switch which DB instances to send reads or writes to at any time.

Topics

- [Limitations and considerations for active-active clusters](#)
- [Preparing for a cross-VPC active-active cluster](#)
- [Required parameter settings for active-active clusters](#)
- [Converting an existing DB instance to an active-active cluster](#)
- [Setting up an active-active cluster with new DB instances](#)
- [Adding a DB instance to an active-active cluster](#)
- [Monitoring active-active clusters](#)
- [Stopping Group Replication on a DB instance in an active-active cluster](#)
- [Renaming a DB instance in an active-active cluster](#)
- [Removing a DB instance from an active-active cluster](#)

Limitations and considerations for active-active clusters

Active-active clusters in Amazon RDS offer enhanced availability and scalability by distributing workloads across multiple instances. However, there are important limitations and considerations to keep in mind when using this architecture.

The following sections outline key factors such as replication delays, conflict resolution, resource allocation, and failover behavior. Understanding these considerations can help ensure optimal performance and reliability in active-active cluster deployments.

Topics

- [Limitations for RDS for MySQL active-active clusters](#)
- [Considerations and best practices for RDS for MySQL active-active clusters](#)

Limitations for RDS for MySQL active-active clusters

The following limitations apply to active-active clusters for RDS for MySQL:

- The master user name can't be `rdsgrprepladmin` for DB instances in an active-active cluster. This user name is reserved for Group Replication connections.

- For DB instances with read replicas in active-active clusters, a prolonged replication status other than Replicating can cause log files to exceed storage limits. For information about the status of read replicas, see [Monitoring read replication](#).
- Blue/green deployments aren't supported for DB instances in an active-active cluster. For more information, see [Using Amazon RDS Blue/Green Deployments for database updates](#).
- Kerberos authentication isn't supported for DB instances in an active-active cluster. For more information, see [Using Kerberos authentication for Amazon RDS for MySQL](#).
- The DB instances in a Multi-AZ DB cluster can't be added to an active-active cluster. However, the DB instances in a Multi-AZ DB instance deployment can be added to an active-active cluster. For more information, see [Configuring and managing a Multi-AZ deployment for Amazon RDS](#).
- Tables that don't have a primary key aren't replicated in an active-active cluster because writes are rejected by the Group Replication plugin.
- Non-InnoDB tables aren't replicated in an active-active cluster.
- Active-active clusters don't support concurrent DML and DDL statements on different DB instances in the cluster.
- You can't configure an active-active cluster to use single-primary mode for the group's replication mode. For this configuration, we recommend using a Multi-AZ DB cluster instead. For more information, see [Multi-AZ DB cluster deployments for Amazon RDS](#).
- Multi-source replication isn't supported for DB instances in an active-active cluster.
- A cross-Region active-active cluster can't enforce certificate authority (CA) verification for Group Replication connections.

Considerations and best practices for RDS for MySQL active-active clusters

Before you use RDS for MySQL active-active clusters, review the following considerations and best practices:

- Active-active clusters can't have more than nine DB instances.
- With the Group Replication plugin, you can control the transaction consistency guarantees of the active-active cluster. For more information, see [Transaction Consistency Guarantees](#) in the MySQL documentation.
- Conflicts are possible when different DB instances update the same row in an active-active cluster. For information about conflicts and conflict resolution, see [Group Replication](#) in the MySQL documentation.

- For fault tolerance, include at least three DB instances in your active-active cluster. It is possible to configure an active-active cluster with only one or two DB instances, but the cluster won't be fault tolerant. For information about fault tolerance, see [Fault-tolerance](#) in the MySQL documentation.
- When a DB instance joins an existing active-active cluster and is running the same engine version as the lowest engine version in the cluster, the DB instance joins in read-write mode.
- When a DB instance joins an existing active-active cluster and is running a higher engine version than the lowest engine version in the cluster, the DB instance must remain in read-only mode.
- If you enable Group Replication for a DB instance by setting its `rds.group_replication_enabled` parameter to 1 in the DB parameter group, but replication hasn't started or has failed to start, the DB instance is placed in super-read-only mode to prevent data inconsistencies. For information about super-read-only mode, see the [MySQL documentation](#).
- You can upgrade a DB instance in an active-active cluster, but the DB instance is read-only until all of the other DB instances in the active-active cluster are upgraded to same engine version or a higher engine version. When you upgrade a DB instance, the DB instance automatically joins the same active-active cluster when the upgrade completes. To avoid an unintended switch to read-only mode for a DB instance, disable automatic minor version upgrades for it. For information about upgrading a MySQL DB instance, see [Upgrades of the RDS for MySQL DB engine](#).
- You can add a DB instance in a Multi-AZ DB instance deployment to an existing active-active cluster. You can also convert a Single-AZ DB instance in an active-active cluster to a Multi-AZ DB instance deployment. If a primary DB instance in a Multi-AZ deployment fails, that primary instance fails over to the standby instance. The new primary DB instance automatically joins the same cluster after failover completes. For more information about Multi-AZ DB instance deployments, see [Multi-AZ DB instance deployments for Amazon RDS](#).
- We recommend that the DB instances in an active-active cluster have different time ranges for their maintenance windows. This practice avoids multiple DB instances in the cluster going offline for maintenance at the same time. For more information, see [Amazon RDS maintenance window](#).
- Active-active clusters can use SSL for connections between DB instances. To configure SSL connections, set the [group_replication_recovery_use_ssl](#) and [group_replication_ssl_mode](#) parameters. The values for these parameters must match for all DB instances in the active-active cluster.

Currently, active-active clusters don't support certificate authority (CA) verification for connections between AWS Regions. So, the [group_replication_ssl_mode](#) parameter must be set to DISABLED (the default) or REQUIRED for cross-Region clusters.

- An RDS for MySQL active-active cluster runs in multi-primary mode. The default value of the [group_replication_enforce_update_everywhere_checks](#) is ON and the parameter is static. When this parameter is set to ON, applications can't insert into a table that has cascading foreign key constraints.
- An RDS for MySQL active-active cluster uses the MySQL communication stack for connection security instead of XCOM. For more information, see [Communication Stack for Connection Security Management](#) in the MySQL documentation.
- When a DB parameter group is associated with a DB instance in an active-active cluster, we recommend only associating this DB parameter group with other DB instances that are in the cluster.
- Active-active clusters only support RDS for MySQL DB instances. These DB instances must be running supported versions of the DB engine.
- When a DB instance in an active-active cluster has an unexpected failure, RDS starts recovery of the DB instance automatically. If the DB instance doesn't recover, we recommend replacing it with a new DB instance by performing a point-in-time recovery with a healthy DB instance in the cluster. For instructions, see [Adding a DB instance to an active-active cluster using point-in-time recovery](#).
- You can delete a DB instance in an active-active cluster without affecting the other DB instances in the cluster. For information about deleting a DB instance, see [Deleting a DB instance](#).
- When a DB instance unintentionally leaves an active-active cluster, by default the [group_replication_exit_state_action](#) parameter changes to OFFLINE_MODE. In this state, the DB instance is inaccessible and you must reboot the DB instance to bring it back online and to rejoin the cluster. You can change this behavior by modifying the [group_replication_exit_state_action](#) parameter in a custom parameter group. By setting the parameter to READ_ONLY, when the DB instance unintentionally leaves a cluster, it enters a super read-only state rather than going offline.

Preparing for a cross-VPC active-active cluster

You can configure an active-active cluster with Amazon RDS for MySQL DB instances in more than one VPC. The VPCs can be in the same AWS Region or different AWS Regions.

Note

Sending traffic between multiple AWS Regions might incur additional costs. For more information, see [Overview of Data Transfer Costs for Common Architectures](#).

If you are configuring an active-active cluster in a single VPC, you can skip these steps and move on to [Setting up an active-active cluster with new DB instances](#).

To prepare for an active-active cluster with DB instances in more than one VPC

1. Make sure the IPv4 address ranges in the CIDR blocks meet the following requirements:

- The IPv4 address ranges in the CIDR blocks of the VPCs can't overlap.
- All of the IPv4 address ranges in the CIDR blocks either must be lower than $128.0.0.0/\text{subnet_mask}$ or higher than $128.0.0.0/\text{subnet_mask}$.

The following ranges illustrate these requirements:

- $10.1.0.0/16$ in one VPC and $10.2.0.0/16$ in the other VPC is supported.
- $172.1.0.0/16$ in one VPC and $172.2.0.0/16$ in the other VPC is supported.
- $10.1.0.0/16$ in one VPC and $10.1.0.0/16$ in the other VPC is *not* supported because the ranges overlap.
- $10.1.0.0/16$ in one VPC and $172.1.0.0/16$ in the other VPC is *not* supported because one is below $128.0.0.0/\text{subnet_mask}$ and the other is above $128.0.0.0/\text{subnet_mask}$.

For information about CIDR blocks, see [VPC CIDR blocks](#) in the *Amazon VPC User Guide*.

2. In each VPC, make sure DNS resolution and DNS hostnames are both enabled.

For instructions, see [View and update DNS attributes for your VPC](#) in the *Amazon VPC User Guide*.

3. Configure the VPCs so that you can route traffic between them in one of the following ways:

- Create a VPC peering connection between the VPCs.

For instructions, see [Create a VPC peering connection](#) in the *Amazon VPC Peering Guide*.

In each VPC, make sure there are inbound rules for your security groups that reference

security groups in the peered VPC. Doing so allows traffic to flow to and from instances that are associated with the referenced security group in the peered VPC. For instructions, see [Update your security groups to reference peer security groups](#) in the *Amazon VPC Peering Guide*.

- Create a transit gateway between the VPCs.

For instructions, see [Getting started with transit gateways](#) in *Amazon VPC Transit Gateways*. In each VPC, make sure there are inbound rules for your security groups that allow traffic from the other VPC, such as inbound rules that specify the CIDR of the other VPC. Doing so allows traffic to flow to and from instances that are associated with the referenced security group in the active-active cluster. For more information, see [Control traffic to your AWS resources using security groups](#) in the *Amazon VPC User Guide*.

Required parameter settings for active-active clusters

Configuring parameters for active-active clusters in Amazon RDS for MySQL is essential for maintaining consistent performance and operational stability. This table details the key parameters that control replication, conflict resolution, and workload distribution. Correct configuration ensures efficient synchronization between nodes, minimizes replication lag, and optimizes resource utilization in distributed or high-traffic environments.

Parameter	Description	Required setting
binlog_format	Sets the binary logging format. The default value for RDS for MySQL 8.0 versions and lower is MIXED. The default value for RDS for MySQL 8.4 versions is ROW. For more information, see the MySQL documentation .	ROW
enforce_gtid_consistency	Enforces GTID consistency for statement execution. The default value for RDS for MySQL is OFF. For more	ON

Parameter	Description	Required setting
	information, see the MySQL documentation .	
<code>group_replication_group_name</code>	Sets the Group Replication name to a UUID. The UUID format is 11111111-2222-3333-4444-55555555 . You can generate a MySQL UUID by connecting to a MySQL DB instance and running <code>SELECT UUID()</code> . The value must be the same for all of the DB instances in the active-active cluster. For more information, see the MySQL documentation .	A MySQL UUID
<code>gtid-mode</code>	Controls GTID-based logging. The default value for RDS for MySQL is OFF_PERMISSIVE . For more information, see the MySQL documentation .	ON

Parameter	Description	Required setting
<code>rds.custom_dns_resolution</code>	<p>Specifies whether to allow DNS resolution from the Amazon DNS server in your VPC. DNS resolution must be enabled when Group Replication is enabled with the <code>rds.group_replication_enabled</code> parameter. DNS resolution can't be enabled when Group Replication is disabled with the <code>rds.group_replication_enabled</code> parameter. For more information, see Amazon DNS server in the <i>Amazon VPC User Guide</i>.</p>	1
<code>rds.group_replication_enabled</code>	<p>Specifies whether Group Replication is enabled for a DB instance. Group Replication must be enabled on a DB instance in an active-active cluster.</p>	1
<code>replica_preserve_commit_order</code> (RDS for MySQL 8.4 and higher versions) or <code>slave_preserve_commit_order</code> (RDS for MySQL 8.0 versions)	<p>Controls the order that transactions are committed on a replica. The default value for RDS for MySQL is ON. For more information, see the MySQL documentation.</p>	ON

Converting an existing DB instance to an active-active cluster

The DB engine version of the DB instance you want to migrate to an active-active cluster must be one of the following versions:

- All MySQL 8.4 versions
- MySQL 8.0.35 and higher minor versions

If you need to upgrade the engine version, see [Upgrades of the RDS for MySQL DB engine](#).

If you are setting up an active-active cluster with DB instances in more than one VPC, make sure you complete the prerequisites in [Preparing for a cross-VPC active-active cluster](#).

Complete the following steps to migrate an existing DB instance to an active-active cluster for RDS for MySQL.

Topics

- [Step 1: Set the active-active cluster parameters in one or more custom parameter groups](#)
- [Step 2: Associate the DB instance with a DB parameter group that has the required Group Replication parameters set](#)
- [Step 3: Create the active-active cluster](#)
- [Step 4: Create additional RDS for MySQL DB instances for the active-active cluster](#)
- [Step 5: Initialize the group on the DB instance you are converting](#)
- [Step 6: Start replication on the other DB instances in the active-active cluster](#)
- [Step 7: \(Recommended\) Check the status of the active-active cluster](#)

Step 1: Set the active-active cluster parameters in one or more custom parameter groups

The RDS for MySQL DB instances in an active-active cluster must be associated with a custom parameter group that has the correct setting for required parameters. For information about the parameters and the required setting for each one, see [Required parameter settings for active-active clusters](#).

You can set these parameters in new parameter groups or in existing parameter groups. However, to avoid accidentally affecting DB instances that aren't part of the active-active cluster, we strongly

recommend that you create a new custom parameter group. The DB instances in an active-active cluster can be associated with the same DB parameter group or with different DB parameter groups.

You can use the AWS Management Console or the AWS CLI to create a new custom parameter group. For more information, see [Creating a DB parameter group in Amazon RDS](#). The following example runs the [create-db-parameter-group](#) AWS CLI command to create a custom DB parameter group named *myactivepg* for RDS for MySQL 8.0:

For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
--db-parameter-group-name myactivepg \
--db-parameter-group-family mysql8.0 \
--description "Parameter group for active-active clusters"
```

For Windows:

```
aws rds create-db-parameter-group ^
--db-parameter-group-name myactivepg ^
--db-parameter-group-family mysql8.0 ^
--description "Parameter group for active-active clusters"
```

You can also use the AWS Management Console or the AWS CLI to set the parameters in the custom parameter group. For more information, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

The following example runs the [modify-db-parameter-group](#) AWS CLI command to set the parameters for RDS for MySQL 8.0. To use this example with RDS for MySQL 8.4, change `slave_preserve_commit_order` to `replica_preserve_commit_order`.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myactivepg \
--parameters
"ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-
reboot" \
"ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-
reboot" \
```

```
"ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-reboot" \
    "ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-reboot" \
"ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" \
"ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate" \
"ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-555555555555',ApplyMethod=pending-reboot"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myactivepg ^
--parameters
"ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-reboot" ^
"ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-reboot" ^
"ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-reboot" ^
    "ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-reboot" ^
"ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" ^
"ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate" ^
"ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-555555555555',ApplyMethod=pending-reboot"
```

Step 2: Associate the DB instance with a DB parameter group that has the required Group Replication parameters set

Associate the DB instance with a parameter group you created or modified in the previous step. For instructions, see [Associating a DB parameter group with a DB instance in Amazon RDS](#).

Reboot the DB instance for the new parameter settings to take effect. For instructions, see [Rebooting a DB instance](#).

Step 3: Create the active-active cluster

In the DB parameter group associated with the DB instance, set the `group_replication_group_seeds` parameter to the endpoint of the DB instance you are converting.

You can use the AWS Management Console or the AWS CLI to set the parameter. You don't need to reboot the DB instance after setting this parameter. For more information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

The following example runs the [modify-db-parameter-group](#) AWS CLI command to set the parameters:

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myactivepg \
--parameters
"ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myactivepg ^
--parameters
"ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

Step 4: Create additional RDS for MySQL DB instances for the active-active cluster

To create additional DB instances for the active-active cluster, perform point-in-time recovery on the DB instance you are converting. For instructions, see [Adding a DB instance to an active-active cluster using point-in-time recovery](#).

An active-active cluster can have up to nine DB instances. Perform point-in-time recovery on the DB instance until you have the number of DB instances you want for the cluster. When you perform point-in-recovery, make sure you associate the DB instance you are adding with a DB parameter group that has `rds.group_replication_enabled` set to 1. Otherwise, Group Replication won't start on the newly added DB instance.

Step 5: Initialize the group on the DB instance you are converting

Initialize the group and start replication:

1. Connect to that DB instance you are converting in a SQL client. For more information about connecting to an RDS for MySQL DB instance, see [Connecting to your MySQL DB instance](#).
2. In the SQL client, run the following stored procedures and replace `group_replication_user_password` with the password for the `rdsgrpadmin` user. The `rdsgrpadmin` user is reserved for Group Replication connections in an active-active cluster. The password for this user must be the same on all of the DB instances in an active-active cluster.

```
call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog
call mysql.rds_group_replication_create_user('group_replication_user_password');
call
mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
call mysql.rds_group_replication_start(1);
```

This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on the DB instance. You can adjust this value to meet your requirements.

This example specifies 1 in the `mysql.rds_group_replication_start` stored procedure to initialize a new group with the current DB instance.

For more information about the stored procedures called in the example, see [Managing active-active clusters](#).

Step 6: Start replication on the other DB instances in the active-active cluster

For each of the DB instances in the active-active cluster, use a SQL client to connect to the instance, and run the following stored procedures. Replace `group_replication_user_password` with the password for the `rdsgrpadmin` user.

```
call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog
call mysql.rds_group_replication_create_user('group_replication_user_password');
call
mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
call mysql.rds_group_replication_start(0);
```

This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on each DB instance. You can adjust this value to meet your requirements.

This example specifies `0` in the `mysql.rds_group_replication_start` stored procedure to join the current DB instance to an existing group.

 **Tip**

Make sure you run these stored procedures on all of the other DB instances in the active-active cluster.

Step 7: (Recommended) Check the status of the active-active cluster

To make sure each member of the cluster is configured correctly, check the status of the cluster by connecting to a DB instance in the active-active cluster, and running the following SQL command:

```
SELECT * FROM performance_schema.replication_group_members;
```

Your output should show `ONLINE` for the `MEMBER_STATE` of each DB instance, as in the following sample output:

```
+-----+-----+
+-----+-----+-----+-----+
+-----+-----+
```

```
| CHANNEL_NAME           | MEMBER_ID          | MEMBER_HOST      |
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION | MEMBER_COMMUNICATION_STACK
|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
| group_replication_applier | 9854d4a2-5d7f-11ee-b8ec-0ec88c43c251 | ip-10-15-3-137 |
| 3306 | ONLINE     | PRIMARY    | 8.0.35        | MySQL
| group_replication_applier | 9e2e9c28-5d7f-11ee-8039-0e5d58f05fef | ip-10-15-3-225 |
| 3306 | ONLINE     | PRIMARY    | 8.0.35        | MySQL
| group_replication_applier | a6ba332d-5d7f-11ee-a025-0a5c6971197d | ip-10-15-1-83 |
| 3306 | ONLINE     | PRIMARY    | 8.0.35        | MySQL
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
3 rows in set (0.00 sec)
```

For information about the possible MEMBER_STATE values, see [Group Replication Server States](#) in the MySQL documentation.

Setting up an active-active cluster with new DB instances

Complete the following steps to set up an active-active cluster using new Amazon RDS for MySQL DB instances.

If you are setting up an active-active cluster with DB instances in more than one VPC, make sure you complete the prerequisites in [Preparing for a cross-VPC active-active cluster](#).

Topics

- [Step 1: Set the active-active cluster parameters in one or more custom parameter groups](#)
- [Step 2: Create new RDS for MySQL DB instances for the active-active cluster](#)
- [Step 3: Specify the DB instances in the active-active cluster](#)
- [Step 4: Initialize the group on a DB instance and start replication](#)
- [Step 5: Start replication on the other DB instances in the active-active cluster](#)
- [Step 6: \(Recommended\) Check the status of the active-active cluster](#)
- [Step 7: \(Optional\) Import data into a DB instance in the active-active cluster](#)

Step 1: Set the active-active cluster parameters in one or more custom parameter groups

The RDS for MySQL DB instances in an active-active cluster must be associated with a custom parameter group that has the correct setting for required parameters. For information about the parameters and the required setting for each one, see [Required parameter settings for active-active clusters](#).

You can set these parameters in new parameter groups or in existing parameter groups. However, to avoid accidentally affecting DB instances that aren't part of the active-active cluster, we strongly recommend that you create a new custom parameter group. The DB instances in an active-active cluster can be associated with the same DB parameter group or with different DB parameter groups.

You can use the AWS Management Console or the AWS CLI to create a new custom parameter group. For more information, see [Creating a DB parameter group in Amazon RDS](#). The following example runs the [create-db-parameter-group](#) AWS CLI command to create a custom DB parameter group named *myactivepg* for RDS for MySQL 8.0:

For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
--db-parameter-group-name myactivepg \
--db-parameter-group-family mysql8.0 \
--description "Parameter group for active-active clusters"
```

For Windows:

```
aws rds create-db-parameter-group ^
--db-parameter-group-name myactivepg ^
--db-parameter-group-family mysql8.0 ^
--description "Parameter group for active-active clusters"
```

You can also use the AWS Management Console or the AWS CLI to set the parameters in the custom parameter group. For more information, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

The following example runs the [modify-db-parameter-group](#) AWS CLI command to set the parameters for RDS for MySQL 8.0. To use this example with RDS for MySQL 8.4, change `slave_preserve_commit_order` to `replica_preserve_commit_order`.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myactivepg \
--parameters
"ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-
reboot" \
"ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-
reboot" \
"ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-
reboot" \
"ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-
reboot" \
"ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" \
"ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate" \
"ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-555555555555
reboot"
```

For Windows:

```
aws rds modify-db-parameter-group ^
--db-parameter-group-name myactivepg ^
--parameters
"ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-
reboot" ^
"ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-
reboot" ^
"ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-
reboot" ^
"ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-
reboot" ^
"ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" ^
```

```
"ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate"  
^  
  
"ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-555555555555'  
reboot"
```

Step 2: Create new RDS for MySQL DB instances for the active-active cluster

Active-active clusters are supported for the following versions of RDS for MySQL DB instances:

- All MySQL version 8.4 versions
- MySQL version 8.0.35 and higher minor versions

You can create up to nine new DB instances for the cluster.

You can use the AWS Management Console or the AWS CLI to create new DB instances. For more information about creating a DB instance, see [Creating an Amazon RDS DB instance](#). When you create the DB instance, associate it with a DB parameter group that you created or modified in the previous step.

Step 3: Specify the DB instances in the active-active cluster

In the DB parameter group associated with each DB instance, set the `group_replication_group_seeds` parameter to the endpoints of the DB instances you want to include in the cluster.

You can use the AWS Management Console or the AWS CLI to set the parameter. You don't need to reboot the DB instance after setting this parameter. For more information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

The following example runs the [modify-db-parameter-group](#) AWS CLI command to set the parameters:

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \  
  --db-parameter-group-name myactivepg \  
  --parameters  
  "ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-
```

```
east-1.rds.amazonaws.com:3306,myactivedb2.123456789012.us-  
east-1.rds.amazonaws.com:3306,myactivedb3.123456789012.us-  
east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^  
--db-parameter-group-name myactivepg ^  
--parameters  
"ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-  
east-1.rds.amazonaws.com:3306,myactivedb2.123456789012.us-  
east-1.rds.amazonaws.com:3306,myactivedb3.123456789012.us-  
east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

 **Tip**

Make sure you set the `group_replication_group_seeds` parameter in each DB parameter group that is associated with a DB instance in the active-active cluster.

Step 4: Initialize the group on a DB instance and start replication

You can choose any new DB to initialize the group and start replication. To do so, complete the following steps:

1. Choose a DB instance in the active-active cluster, and connect to that DB instance in a SQL client. For more information about connecting to an RDS for MySQL DB instance, see [Connecting to your MySQL DB instance](#).
2. In the SQL client, run the following stored procedures and replace `group_replication_user_password` with the password for the `rdsgrpadmin` user. The `rdsgrpadmin` user is reserved for Group Replication connections in an active-active cluster. The password for this user must be the same on all of the DB instances in an active-active cluster.

```
call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog  
call mysql.rds_group_replication_create_user('group_replication_user_password');  
call  
mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');  
call mysql.rds_group_replication_start(1);
```

This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on the DB instance. You can adjust this value to meet your requirements.

This example specifies 1 in the `mysql.rds_group_replication_start` stored procedure to initialize a new group with the current DB instance.

For more information about the stored procedures called in the example, see [Managing active-active clusters](#).

Step 5: Start replication on the other DB instances in the active-active cluster

For each of the DB instances in the active-active cluster, use a SQL client to connect to the instance, and run the following stored procedures. Replace `group_replication_user_password` with the password for the `rdsgrpadmin` user.

```
call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog
call mysql.rds_group_replication_create_user('group_replication_user_password');
call
mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
call mysql.rds_group_replication_start(0);
```

This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on each DB instance. You can adjust this value to meet your requirements.

This example specifies 0 in the `mysql.rds_group_replication_start` stored procedure to join the current DB instance to an existing group.

Tip

Make sure you run these stored procedures on all of the other DB instances in the active-active cluster.

Step 6: (Recommended) Check the status of the active-active cluster

To make sure each member of the cluster is configured correctly, check the status of the cluster by connecting to a DB instance in the active-active cluster, and running the following SQL command:

```
SELECT * FROM performance_schema.replication_group_members;
```

Your output should show ONLINE for the MEMBER_STATE of each DB instance, as in the following sample output:

```
+-----+-----+
+-----+-----+-----+-----+
+-----+
| CHANNEL_NAME          | MEMBER_ID           | MEMBER_HOST   |
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION | MEMBER_COMMUNICATION_STACK
|           |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
| group_replication_applier | 9854d4a2-5d7f-11ee-b8ec-0ec88c43c251 | ip-10-15-3-137 | | |
| 3306 | ONLINE      | PRIMARY     | 8.0.35       | MySQL        |
| group_replication_applier | 9e2e9c28-5d7f-11ee-8039-0e5d58f05fef | ip-10-15-3-225 |
| 3306 | ONLINE      | PRIMARY     | 8.0.35       | MySQL        |
| group_replication_applier | a6ba332d-5d7f-11ee-a025-0a5c6971197d | ip-10-15-1-83  |
| 3306 | ONLINE      | PRIMARY     | 8.0.35       | MySQL        |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
3 rows in set (0.00 sec)
```

For information about the possible MEMBER_STATE values, see [Group Replication Server States](#) in the MySQL documentation.

Step 7: (Optional) Import data into a DB instance in the active-active cluster

You can import data from a MySQL database into a DB instance in the active-active cluster. After the data is imported, Group Replication replicates it to the other DB instances in the cluster.

For information about importing data, see [Importing data to an Amazon RDS for MySQL database with reduced downtime](#).

Adding a DB instance to an active-active cluster

You can add a DB instance to an Amazon RDS for MySQL active-active cluster by restoring a DB snapshot or by restoring a DB instance to a point in time. An active-active cluster can include up to nine DB instances.

When you recover a DB instance to a point in time, it usually includes more recent transactions than a DB instance that was restored from a DB snapshot. When the DB instance has more recent transactions, fewer transactions need to be applied when you start replication. So, using point-in-time recovery to add a DB instance to a cluster is usually faster than restoring from a DB snapshot.

Topics

- [Adding a DB instance to an active-active cluster using point-in-time recovery](#)
- [Adding a DB instance to an active-active cluster using a DB snapshot](#)

Adding a DB instance to an active-active cluster using point-in-time recovery

You can add a DB instance to an active-active cluster by performing point-in-time recovery on a DB instance in the cluster.

For information about recovering a DB instance to a point in time in a different AWS Region, see [Replicating automated backups to another AWS Region](#).

To add a DB instance to an active-active cluster using point-in-time recovery

1. Create a new DB instance by performing point-in-time recovery on a DB instance in the active-active cluster.

You can perform point-in-time recovery on any DB instance in the cluster to create the new DB instance. For instructions, see [Restoring a DB instance to a specified time for Amazon RDS](#).

Important

During point-in-time-recovery, associate the new DB instance with a DB parameter group that has the active-active cluster parameters set. Otherwise, Group Replication won't start on the new DB instance. For information about the parameters and the required setting for each one, see [Required parameter settings for active-active clusters](#).

Tip

If you take a snapshot of the DB instance before you start point-in-time recovery, you might be able to reduce the amount of time required to apply transactions on the new DB instance.

2. Add the DB instance to the `group_replication_group_seeds` parameter in each DB parameter group associated with a DB instance in the active-active cluster, including the DB parameter group that you associated with the new DB instance.

For more information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

3. In a SQL client, connect to the new DB instance, and call the `mysql.rds_group_replication_set_recovery_channel` stored procedure. Replace `group_replication_user_password` with the password for the `rdsgrprepladmin` user.

```
call mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
```

4. Using the SQL client, call the `mysql.rds_group_replication_start` stored procedure to start replication:

```
call mysql.rds_group_replication_start(0);
```

Adding a DB instance to an active-active cluster using a DB snapshot

You can add a DB instance to an active-active cluster by creating a DB snapshot of a DB instance in the cluster and then restoring the DB snapshot.

For information about copying a snapshot to a different AWS Region, see [the section called “Cross-Region copying”](#).

To add a DB instance to an active-active cluster using a DB snapshot

1. Create a DB snapshot of a DB instance in the active-active cluster.

You can create a DB snapshot of any DB instance in the cluster. For instructions, see [Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS](#).

2. Restore a DB instance from the DB snapshot.

During the snapshot restore operation, associate the new DB instance with a DB parameter group that has the active-active cluster parameters set. For information about the parameters and the required setting for each one, see [Required parameter settings for active-active clusters](#).

For information about restoring a DB instance from a DB snapshot, see [Restoring to a DB instance](#).

3. Add the DB instance to the `group_replication_group_seeds` parameter in each DB parameter group associated with a DB instance in the active-active cluster, including the DB parameter group that you associated with the new DB instance.

For more information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

4. In a SQL client, connect to the new DB instance, and call the `mysql.rds_group_replication_set_recovery_channel` stored procedure. Replace `group_replication_user_password` with the password for the `rdsgrprepladmin` user.

```
call
mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
```

5. Using the SQL client, call the `mysql.rds_group_replication_start` stored procedure to start replication:

```
call mysql.rds_group_replication_start(0);
```

Monitoring active-active clusters

Monitoring active-active clusters in Amazon RDS for MySQL is crucial for tracking performance, replication integrity, and node synchronization. You can monitor your active-active cluster by connecting to a DB instance in the cluster, and running the following SQL command:

```
SELECT * FROM performance_schema.replication_group_members;
```

Your output should show ONLINE for the MEMBER_STATE of each DB instance, as in the following sample output:

```
+-----+-----+-----+-----+-----+
| CHANNEL_NAME      | MEMBER_ID          | MEMBER_HOST   |
| MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION | MEMBER_COMMUNICATION_STACK
|               |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
| group_replication_applier | 9854d4a2-5d7f-11ee-b8ec-0ec88c43c251 | ip-10-15-3-137 | | |
|           3306 | ONLINE       | PRIMARY     | 8.0.35        | MySQL          |
| group_replication_applier | 9e2e9c28-5d7f-11ee-8039-0e5d58f05fef | ip-10-15-3-225 |
|           3306 | ONLINE       | PRIMARY     | 8.0.35        | MySQL          |
| group_replication_applier | a6ba332d-5d7f-11ee-a025-0a5c6971197d | ip-10-15-1-83  |
|           3306 | ONLINE       | PRIMARY     | 8.0.35        | MySQL          |
+-----+
+-----+-----+-----+-----+
+-----+
3 rows in set (0.00 sec)
```

For information about the possible MEMBER_STATE values, see [Group Replication Server States](#) in the MySQL documentation.

Stopping Group Replication on a DB instance in an active-active cluster

You can stop Group Replication on a DB instance in an active-active cluster. When you stop Group Replication, the DB instance is placed in super-read-only mode until replication is restarted or that DB instance is removed from the active-active cluster. For information about super-read-only mode, see the [MySQL documentation](#).

To stop Group Replication temporarily for an active-active cluster

1. Connect to a DB instance in the active-active cluster using a SQL client.

For more information about connecting to an RDS for MySQL DB instance, see [Connecting to your MySQL DB instance](#).

2. In the SQL client, call the [mysql.rds_group_replication_stop](#) stored procedure:

```
call mysql.rds_group_replication_stop();
```

Renaming a DB instance in an active-active cluster

You can change the name of a DB instance in an active-active cluster. To rename more than one DB instance in an active-active cluster, do so one DB instance at a time. So, rename one DB instance and rejoin it to the cluster before you rename the next DB instance.

To rename a DB instance in an active-active cluster

1. Connect to the DB instance in a SQL client, and call the [mysql.rds_group_replication_stop](#) stored procedure:

```
call mysql.rds_group_replication_stop();
```

2. Rename the DB instance by following the instructions in [Renaming a DB instance](#).
3. Modify the `group_replication_group_seeds` parameter in each DB parameter group associated with a DB instance in the active-active cluster.

In the parameter setting, replace the old DB instance endpoint with the new DB instance endpoint. For more information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

4. Connect to the DB instance in a SQL client, and call the [mysql.rds_group_replication_start](#) stored procedure:

```
call mysql.rds_group_replication_start(0);
```

Removing a DB instance from an active-active cluster

When you remove a DB instance from an active-active cluster, it reverts to a standalone DB instance.

To remove a DB instance from an active-active cluster

1. Connect to the DB instance in a SQL client, and call the [mysql.rds_group_replication_stop](#) stored procedure:

```
call mysql.rds_group_replication_stop();
```

2. Modify the `group_replication_group_seeds` parameter for the DB instances that will remain in the active-active cluster.

In the `group_replication_group_seeds` parameter, delete the DB instance that you are removing from the active-active cluster. For more information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

3. Modify the parameters of the DB instance you are removing from the active-active cluster so that it is no longer part of the cluster.

You can either associate the DB instance with a different parameter group, or modify the parameters in the DB parameter group associated with the DB instance. The parameters to modify include `group_replication_group_name`, `rds.group_replication_enabled`, and `group_replication_group_seeds`. For more information about active-active cluster parameters, see [Required parameter settings for active-active clusters](#).

If you modify the parameters in a DB parameter group, make sure the DB parameter group isn't associated with other DB instances in the active-active cluster.

4. Reboot the DB instance you removed from the active-active cluster for the new parameter settings to take effect.

For instructions, see [Rebooting a DB instance](#).

Exporting data from a MySQL DB instance by using replication

To export data from an RDS for MySQL DB instance to a MySQL instance running external to Amazon RDS, you can use replication. In this scenario, the MySQL DB instance is the *source MySQL DB instance*, and the MySQL instance running external to Amazon RDS is the *external MySQL database*.

The external MySQL database can run either on-premises in your data center, or on an Amazon EC2 instance. The external MySQL database must run the same version as the source MySQL DB instance, or a later version.

Replication to an external MySQL database is only supported during the time it takes to export a database from the source MySQL DB instance. The replication should be terminated when the data has been exported and applications can start accessing the external MySQL instance.

The following list shows the steps to take. Each step is discussed in more detail in later sections.

1. Prepare an external MySQL DB instance.
2. Prepare the source MySQL DB instance for replication.
3. Use the mysqldump utility to transfer the database from the source MySQL DB instance to the external MySQL database.
4. Start replication to the external MySQL database.
5. After the export completes, stop replication.

Prepare an external MySQL database

Perform the following steps to prepare the external MySQL database.

To prepare the external MySQL database

1. Install the external MySQL database.
2. Connect to the external MySQL database as the master user. Then create the users required to support the administrators, applications, and services that access the database.
3. Follow the directions in the MySQL documentation to prepare the external MySQL database as a replica. For more information, see [Setting the Replica Configuration](#) in the MySQL documentation.

4. Configure an egress rule for the external MySQL database to operate as a read replica during the export. The egress rule allows the external MySQL database to connect to the source MySQL DB instance during replication. Specify an egress rule that allows Transmission Control Protocol (TCP) connections to the port and IP address of the source MySQL DB instance.

Specify the appropriate egress rules for your environment:

- If the external MySQL database is running in an Amazon EC2 instance in a virtual private cloud (VPC) based on the Amazon VPC service, specify the egress rules in a VPC security group. For more information, see [Controlling access with security groups](#).
 - If the external MySQL database is installed on-premises, specify the egress rules in a firewall.
5. If the external MySQL database is running in a VPC, configure rules for the VPC access control list (ACL) rules in addition to the security group egress rule:
 - Configure an ACL ingress rule allowing TCP traffic to ports 1024–65535 from the IP address of the source MySQL DB instance.
 - Configure an ACL egress rule allowing outbound TCP traffic to the port and IP address of the source MySQL DB instance.

For more information about Amazon VPC network ACLs, see [Network ACLs in Amazon VPC User Guide](#).

6. (Optional) Set the `max_allowed_packet` parameter to the maximum size to avoid replication errors. We recommend this setting.

Prepare the source MySQL DB instance

Perform the following steps to prepare the source MySQL DB instance as the replication source.

To prepare the source MySQL DB instance

1. Ensure that your client computer has enough disk space available to save the binary logs while setting up replication.
2. Connect to the source MySQL DB instance, and create a replication account by following the directions in [Creating a User for Replication](#) in the MySQL documentation.
3. Configure ingress rules on the system running the source MySQL DB instance to allow the external MySQL database to connect during replication. Specify an ingress rule that allows

TCP connections to the port used by the source MySQL DB instance from the IP address of the external MySQL database.

4. Specify the egress rules:

- If the source MySQL DB instance is running in a VPC, specify the ingress rules in a VPC security group. For more information, see [Controlling access with security groups](#).
5. If source MySQL DB instance is running in a VPC, configure VPC ACL rules in addition to the security group ingress rule:
- Configure an ACL ingress rule to allow TCP connections to the port used by the Amazon RDS instance from the IP address of the external MySQL database.
 - Configure an ACL egress rule to allow TCP connections from ports 1024–65535 to the IP address of the external MySQL database.

For more information about Amazon VPC network ACLs, see [Network ACLs](#) in the *Amazon VPC User Guide*.

6. Ensure that the backup retention period is set long enough that no binary logs are purged during the export. If any of the logs are purged before the export has completed, you must restart replication from the beginning. For more information about setting the backup retention period, see [Introduction to backups](#).
7. Use the `mysql.rds_set_configuration` stored procedure to set the binary log retention period long enough that the binary logs aren't purged during the export. For more information, see [Accessing MySQL binary logs](#).
8. Create an Amazon RDS read replica from the source MySQL DB instance to further ensure that the binary logs of the source MySQL DB instance are not purged. For more information, see [Creating a read replica](#).
9. After the Amazon RDS read replica has been created, call the `mysql.rds_stop_replication` stored procedure to stop the replication process. The source MySQL DB instance no longer purges its binary log files, so they are available for the replication process.
10. (Optional) Set both the `max_allowed_packet` parameter and the `slave_max_allowed_packet` parameter to the maximum size to avoid replication errors. The maximum size for both parameters is 1 GB. We recommend this setting for both parameters. For information about setting parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Copy the database

Perform the following steps to copy the database.

To copy the database

1. Connect to the RDS read replica of the source MySQL DB instance, and run the MySQL SHOW REPLICA STATUS\G statement. Note the values for the following:

- Master_Host
- Master_Port
- Master_Log_File
- Exec_Master_Log_Pos

Note

Previous versions of MySQL used SHOW SLAVE STATUS instead of SHOW REPLICA STATUS. If you are using a MySQL version before 8.0.23, then use SHOW SLAVE STATUS.

2. Use the mysqldump utility to create a snapshot, which copies the data from Amazon RDS to your local client computer. Ensure that your client computer has enough space to hold the mysqldump files from the databases to be replicated. This process can take several hours for very large databases. Follow the directions in [Creating a Data Snapshot Using mysqldump](#) in the MySQL documentation.

The following example runs mysqldump on a client and writes the dump to a file.

For Linux, macOS, or Unix:

```
mysqldump -h source_MySQL_DB_instance_endpoint \
-u user \
-ppassword \
--port=3306 \
--single-transaction \
--routines \
--triggers \
--databases database1 database2 > path/rds-dump.sql
```

For Windows:

```
mysqldump -h source_MySQL_DB_instance_endpoint ^
-u user ^
-ppassword ^
--port=3306 ^
--single-transaction ^
--routines ^
--triggers ^
--databases database database2 > path\rds-dump.sql
```

You can load the backup file into the external MySQL database. For more information, see [Reloading SQL-Format Backups](#) in the MySQL documentation. You can run another utility to load the data into the external MySQL database.

Complete the export

Perform the following steps to complete the export.

To complete the export

1. Use the MySQL CHANGE MASTER statement to configure the external MySQL database. Specify the ID and password of the user granted REPLICATION SLAVE permissions. Specify the Master_Host, Master_Port, Relay_Master_Log_File, and Exec_Master_Log_Pos values that you got from the MySQL SHOW REPLICA STATUS\G statement that you ran on the RDS read replica. For more information, see [CHANGE MASTER TO Statement](#) in the MySQL documentation.

 **Note**

Previous versions of MySQL used SHOW SLAVE STATUS instead of SHOW REPLICA STATUS. If you are using a MySQL version before 8.0.23, then use SHOW SLAVE STATUS.

2. Use the MySQL START REPLICA command to initiate replication from the source MySQL DB instance to the external MySQL database.

Doing this starts replication from the source MySQL DB instance and exports all source changes that have occurred after you stopped replication from the Amazon RDS read replica.

 **Note**

Previous versions of MySQL used START SLAVE instead of START REPLICA. If you are using a MySQL version before 8.0.23, then use START SLAVE.

3. Run the MySQL SHOW REPLICAS STATUS\G command on the external MySQL database to verify that it is operating as a read replica. For more information about interpreting the results, see [SHOW SLAVE | REPLICAS STATUS Statement](#) in the MySQL documentation.
4. After replication on the external MySQL database has caught up with the source MySQL DB instance, use the MySQL STOP REPLICA command to stop replication from the source MySQL DB instance.

 **Note**

Previous versions of MySQL used STOP SLAVE instead of STOP REPLICA. If you are using a MySQL version before 8.0.23, then use STOP SLAVE.

5. On the Amazon RDS read replica, call the mysql.rds_start_replication stored procedure. Doing this allows Amazon RDS to start purging the binary log files from the source MySQL DB instance.

Options for MySQL DB instances

Following, you can find a description of options, or additional features, that are available for Amazon RDS instances running the MySQL DB engine. To enable these options, you can add them to a custom option group, and then associate the option group with your DB instance. For more information about working with option groups, see [Working with option groups](#).

Amazon RDS supports the following options for MySQL:

Option	Option ID	Engine versions
MariaDB Audit Plugin support for MySQL	MARIADB_AUDIT_PLUGIN	All MySQL 8.4 versions
		MySQL 8.0.28 and higher 8.0 versions
		All MySQL 5.7 versions
MySQL memcached support	MEMCACHED	All MySQL 5.7 and 8.0 versions

MariaDB Audit Plugin support for MySQL

Amazon RDS offers an audit plugin for MySQL database instances based on the open source MariaDB Audit Plugin. For more information, see the [Audit Plugin for MySQL Server GitHub repository](#).

Note

The audit plugin for MySQL is based on the MariaDB Audit Plugin. Throughout this article, we refer to it as MariaDB Audit Plugin.

The MariaDB Audit Plugin records database activity, including users logging on to the database and queries run against the database. The record of database activity is stored in a log file.

Audit Plugin option settings

Amazon RDS supports the following settings for the MariaDB Audit Plugin option.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_FILE_PATH	/rdsdbdata/log/audit/	/rdsdbdata/log/audit/	The location of the log file. The log file contains the record of the activity specified in SERVER_AUDIT_EVENTS . For more information, see Viewing and listing database log files and MySQL database log files .
SERVER_AUDIT_FILE_ROTATE_SIZE	1–1000000	1000000	The size in bytes that when reached, causes the file to rotate. For more information, see Overview of RDS for MySQL database logs .
SERVER_AUDIT_FILE_ROTATIONS	0–100	9	The number of log rotations to save when server_audit_output_type=file . If set to 0, then the log file never rotates. For more information, see Overview of RDS for MySQL database logs .

Option setting	Valid values	Default value	Description
			MySQL database logs and Downloading a database log file .
SERVER_AUDIT_EVENTS	CONNECT, QUERY, QUERY_DDL , QUERY_DML , QUERY_DML_NO_SELECT , QUERY_DCL	CONNECT, QUERY	<p>The types of activity to record in the log. Installing the MariaDB Audit Plugin is itself logged.</p> <ul style="list-style-type: none"> • CONNECT: Log successful and unsuccessful connections to the database, and disconnections from the database. • QUERY: Log the text of all queries run against the database. • QUERY_DDL : Similar to the QUERY event, but returns only data definition language (DDL) queries (CREATE, ALTER, and so on). • QUERY_DML : Similar to the QUERY event, but returns only data manipulation language (DML) queries (INSERT, UPDATE, and so on, and also SELECT). • QUERY_DML_NO_SELECT : Similar to the QUERY_DML event, but doesn't log SELECT queries. • QUERY_DCL : Similar to the QUERY event, but returns only data control language (DCL) queries (GRANT, REVOKE, and so on). <p>For MySQL, TABLE is not supported.</p>

Option setting	Valid values	Default value	Description
SERVER_AUDIT_INCL_USERS	Multiple comma-separated values	None	Include only activity from the specified users. By default, activity is recorded for all users. SERVER_AUDIT_INCL_USERS and SERVER_AUDIT_EXCL_USERS are mutually exclusive. If you add values to SERVER_AUDIT_INCL_USERS, make sure no values are added to SERVER_AUDIT_EXCL_USERS.
SERVER_AUDIT_EXCL_USERS	Multiple comma-separated values	None	<p>Exclude activity from the specified users. By default, activity is recorded for all users. SERVER_AUDIT_INCL_USERS and SERVER_AUDIT_EXCL_USERS are mutually exclusive. If you add values to SERVER_AUDIT_EXCL_USERS, make sure no values are added to SERVER_AUDIT_INCL_USERS.</p> <p>The rdsadmin user queries the database every second to check the health of the database. Depending on your other settings, this activity can possibly cause the size of your log file to grow very large, very quickly. If you don't need to record this activity, add the rdsadmin user to the SERVER_AUDIT_EXCL_USERS list.</p> <div data-bbox="861 1474 900 1516" style="border-radius: 50%; width: 1em; height: 1em; background-color: #0070C0; color: white; display: flex; align-items: center; justify-content: center; font-size: 0.8em; margin-right: 0.2em;"></div> Note <p>CONNECT activity is always recorded for all users, even if the user is specified for this option setting.</p>

Option setting	Valid values	Default value	Description
SERVER_AUDIT_LOGGING	ON	ON	Logging is active. The only valid value is ON. Amazon RDS does not support deactivating logging. If you want to deactivate logging, remove the MariaDB Audit Plugin. For more information, see Removing the MariaDB Audit Plugin .
SERVER_AUDIT_QUERY_LOG_LIMIT	0–2147483647	1024	The limit on the length of the query string in a record.

Adding the MariaDB Audit Plugin

The general process for adding the MariaDB Audit Plugin to a DB instance is the following:

- Create a new option group, or copy or modify an existing option group
- Add the option to the option group
- Associate the option group with the DB instance

After you add the MariaDB Audit Plugin, you don't need to restart your DB instance. As soon as the option group is active, auditing begins immediately.

Important

Adding the MariaDB Audit Plugin to a DB instance might cause an outage. We recommend adding the MariaDB Audit Plugin during a maintenance window or during a time of low database workload.

To add the MariaDB Audit Plugin

1. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step.

Otherwise, create a custom DB option group. Choose **mysql** for **Engine**, and choose **5.7, 8.0, or 8.4** for **Major engine version**. For more information, see [Creating an option group](#).

2. Add the **MARIADB_AUDIT_PLUGIN** option to the option group, and configure the option settings. For more information about adding options, see [Adding an option to an option group](#). For more information about each setting, see [Audit Plugin option settings](#).
3. Apply the option group to a new or existing DB instance.
 - For a new DB instance, you apply the option group when you launch the instance. For more information, see [Creating an Amazon RDS DB instance](#).
 - For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. For more information, see [Modifying an Amazon RDS DB instance](#).

Audit log format

Log files are represented as comma-separated variable (CSV) files in UTF-8 format.

Tip

Log file entries are not in sequential order. To order the entries, use the timestamp value. To see the latest events, you might have to review all log files. For more flexibility in sorting and searching the log data, turn on the setting to upload the audit logs to CloudWatch and view them using the CloudWatch interface. To view audit data with more types of fields and with output in JSON format, you can also use the Database Activity Streams feature. For more information, see [Monitoring Amazon RDS with Database Activity Streams](#).

The audit log files include the following comma-delimited information in rows, in the specified order:

Field	Description
timestamp	The YYYYMMDD followed by the HH:MI:SS (24-hour clock) for the logged event.
serverhost	The name of the instance that the event is logged for.

Field	Description
username	The connected user name of the user.
host	The host that the user connected from.
connectionid	The connection ID number for the logged operation.
queryid	The query ID number, which can be used for finding the relational table events and related queries. For TABLE events, multiple lines are added.
operation	The recorded action type. Possible values are: CONNECT, QUERY, READ, WRITE, CREATE, ALTER, RENAME, and DROP.
database	The active database, as set by the USE command.
object	For QUERY events, this value indicates the query that the database performed. For TABLE events, it indicates the table name.
retcode	The return code of the logged operation.
connection_type	<p>The security state of the connection to the server. Possible values are:</p> <ul style="list-style-type: none"> • Undefined • TCP/IP • Socket • Named pipe • SSL/TLS • Shared memory

Viewing and downloading the MariaDB Audit Plugin log

After you enable the MariaDB Audit Plugin, you access the results in the log files the same way you access any other text-based log files. The audit log files are located at `/rdsdbdata/log/audit/`. For information about viewing the log file in the console, see [Viewing and listing database log files](#). For information about downloading the log file, see [Downloading a database log file](#).

Modifying MariaDB Audit Plugin settings

After you enable the MariaDB Audit Plugin, you can modify the settings. For more information about how to modify option settings, see [Modifying an option setting](#). For more information about each setting, see [Audit Plugin option settings](#).

Removing the MariaDB Audit Plugin

Amazon RDS doesn't support turning off logging in the MariaDB Audit Plugin. However, you can remove the plugin from a DB instance. When you remove the MariaDB Audit Plugin, the DB instance is restarted automatically to stop auditing.

To remove the MariaDB Audit Plugin from a DB instance, do one of the following:

- Remove the MariaDB Audit Plugin option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see [Removing an option from an option group](#)
- Modify the DB instance and specify a different option group that doesn't include the plugin. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see [Modifying an Amazon RDS DB instance](#).

MySQL memcached support

Amazon RDS supports using the memcached interface to InnoDB tables that was introduced in MySQL 5.6. The memcached API enables applications to use InnoDB tables in a manner similar to NoSQL key-value data stores.

Note

The memcached interface is no longer available in MySQL 8.4. When you upgrade your DB instances to MySQL 8.4, you must disable memcached in existing option groups.

The memcached interface is a simple, key-based cache. Applications use memcached to insert, manipulate, and retrieve key-value data pairs from the cache. MySQL 5.6 introduced a plugin that implements a daemon service that exposes data from InnoDB tables through the memcached protocol. For more information about the MySQL memcached plugin, see [InnoDB integration with memcached](#).

To enable memcached support for an RDS for MySQL DB instance

1. Determine the security group to use for controlling access to the memcached interface. If the set of applications already using the SQL interface are the same set that will access the memcached interface, you can use the existing VPC security group used by the SQL interface. If a different set of applications will access the memcached interface, define a new VPC or DB security group. For more information about managing security groups, see [Controlling access with security groups](#)
2. Create a custom DB option group, selecting MySQL as the engine type and version. For more information about creating an option group, see [Creating an option group](#).
3. Add the MEMCACHED option to the option group. Specify the port that the memcached interface will use, and the security group to use in controlling access to the interface. For more information about adding options, see [Adding an option to an option group](#).
4. Modify the option settings to configure the memcached parameters, if necessary. For more information about how to modify option settings, see [Modifying an option setting](#).
5. Apply the option group to an instance. Amazon RDS enables memcached support for that instance when the option group is applied:

- You enable memcached support for a new instance by specifying the custom option group when you launch the instance. For more information about launching a MySQL instance, see [Creating an Amazon RDS DB instance](#).
 - You enable memcached support for an existing instance by specifying the custom option group when you modify the instance. For more information about modifying a DB instance, see [Modifying an Amazon RDS DB instance](#).
6. Specify which columns in your MySQL tables can be accessed through the memcached interface. The memcached plug-in creates a catalog table named `containers` in a dedicated database named `innodb_memcache`. You insert a row into the `containers` table to map an InnoDB table for access through memcached. You specify a column in the InnoDB table that is used to store the memcached key values, and one or more columns that are used to store the data values associated with the key. You also specify a name that a memcached application uses to refer to that set of columns. For details on inserting rows in the `containers` table, see [InnoDB memcached plugin internals](#). For an example of mapping an InnoDB table and accessing it through memcached, see [Writing applications for the InnoDB memcached plugin](#).
7. If the applications accessing the memcached interface are on different computers or EC2 instances than the applications using the SQL interface, add the connection information for those computers to the VPC security group associated with the MySQL instance. For more information about managing security groups, see [Controlling access with security groups](#).

You turn off the memcached support for an instance by modifying the instance and specifying the default option group for your MySQL version. For more information about modifying a DB instance, see [Modifying an Amazon RDS DB instance](#).

MySQL memcached security considerations

The memcached protocol does not support user authentication. For more information about MySQL memcached security considerations, see [Security Considerations for the InnoDB memcached Plugin](#) in the MySQL documentation.

You can take the following actions to help increase the security of the memcached interface:

- Specify a different port than the default of 11211 when adding the `MEMCACHED` option to the option group.

- Ensure that you associate the memcached interface with a VPC security group that limits access to known, trusted client addresses and EC2 instances. For more information about managing security groups, see [Controlling access with security groups](#).

MySQL memcached connection information

To access the memcached interface, an application must specify both the DNS name of the Amazon RDS instance and the memcached port number. For example, if an instance has a DNS name of my-cache-instance.cg034hpkmjt.region.rds.amazonaws.com and the memcached interface is using port 11212, the connection information specified in PHP would be:

```
<?php  
  
$cache = new Memcache;  
$cache->connect('my-cache-instance.cg034hpkmjt.region.rds.amazonaws.com', 11212);  
?>
```

To find the DNS name and memcached port of a MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the top right corner of the AWS Management Console, select the region that contains the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose the MySQL DB instance name to display its details.
5. In the **Connect** section, note the value of the **Endpoint** field. The DNS name is the same as the endpoint. Also, note that the port in the **Connect** section is not used to access the memcached interface.
6. In the **Details** section, note the name listed in the **Option Group** field.
7. In the navigation pane, choose **Option groups**.
8. Choose the name of the option group used by the MySQL DB instance to show the option group details. In the **Options** section, note the value of the **Port** setting for the **MEMCACHED** option.

MySQL memcached option settings

Amazon RDS exposes the MySQL memcached parameters as option settings in the Amazon RDS MEMCACHED option.

MySQL memcached parameters

- DAEMON_MEMCACHED_R_BATCH_SIZE – an integer that specifies how many memcached read operations (get) to perform before doing a COMMIT to start a new transaction. The allowed values are 1 to 4294967295; the default is 1. The option does not take effect until the instance is restarted.
- DAEMON_MEMCACHED_W_BATCH_SIZE – an integer that specifies how many memcached write operations, such as add, set, or incr, to perform before doing a COMMIT to start a new transaction. The allowed values are 1 to 4294967295; the default is 1. The option does not take effect until the instance is restarted.
- INNODB_API_BK_COMMIT_INTERVAL – an integer that specifies how often to auto-commit idle connections that use the InnoDB memcached interface. The allowed values are 1 to 1073741824; the default is 5. The option takes effect immediately, without requiring that you restart the instance.
- INNODB_API_DISABLE_ROWLOCK – a Boolean that disables (1 (true)) or enables (0 (false)) the use of row locks when using the InnoDB memcached interface. The default is 0 (false). The option does not take effect until the instance is restarted.
- INNODB_API_ENABLE_MDL – a Boolean that when set to 0 (false) locks the table used by the InnoDB memcached plugin, so that it cannot be dropped or altered by DDL through the SQL interface. The default is 0 (false). The option does not take effect until the instance is restarted.
- INNODB_API_TRX_LEVEL – an integer that specifies the transaction isolation level for queries processed by the memcached interface. The allowed values are 0 to 3. The default is 0. The option does not take effect until the instance is restarted.

Amazon RDS configures these MySQL memcached parameters, and they cannot be modified: DAEMON_MEMCACHED_LIB_NAME, DAEMON_MEMCACHED_LIB_PATH, and INNODB_API_ENABLE_BINLOG. The parameters that MySQL administrators set by using daemon_memcached_options are available as individual MEMCACHED option settings in Amazon RDS.

MySQL daemon_memcached_options parameters

- **BINDING_PROTOCOL** – a string that specifies the binding protocol to use. The allowed values are auto, ascii, or binary. The default is auto, which means the server automatically negotiates the protocol with the client. The option does not take effect until the instance is restarted.
- **BACKLOG_QUEUE_LIMIT** – an integer that specifies how many network connections can be waiting to be processed by memcached. Increasing this limit may reduce errors received by a client that is not able to connect to the memcached instance, but does not improve the performance of the server. The allowed values are 1 to 2048; the default is 1024. The option does not take effect until the instance is restarted.
- **CAS_DISABLED** – a Boolean that enables (1 (true)) or disables (0 (false)) the use of compare and swap (CAS), which reduces the per-item size by 8 bytes. The default is 0 (false). The option does not take effect until the instance is restarted.
- **CHUNK_SIZE** – an integer that specifies the minimum chunk size, in bytes, to allocate for the smallest item's key, value, and flags. The allowed values are 1 to 48. The default is 48 and you can significantly improve memory efficiency with a lower value. The option does not take effect until the instance is restarted.
- **CHUNK_SIZE_GROWTH_FACTOR** – a float that controls the size of new chunks. The size of a new chunk is the size of the previous chunk times CHUNK_SIZE_GROWTH_FACTOR. The allowed values are 1 to 2; the default is 1.25. The option does not take effect until the instance is restarted.
- **ERROR_ON_MEMORY_EXHAUSTED** – a Boolean that when set to 1 (true) specifies that memcached will return an error rather than evicting items when there is no more memory to store items. If set to 0 (false), memcached will evict items if there is no more memory. The default is 0 (false). The option does not take effect until the instance is restarted.
- **MAX_SIMULTANEOUS_CONNECTIONS** – an integer that specifies the maximum number of concurrent connections. Setting this value to anything under 10 prevents MySQL from starting. The allowed values are 10 to 1024; the default is 1024. The option does not take effect until the instance is restarted.
- **VERBOSITY** – a string that specifies the level of information logged in the MySQL error log by the memcached service. The default is v. The option does not take effect until the instance is restarted. The allowed values are:
 - v – Logs errors and warnings while running the main event loop.
 - vv – In addition to the information logged by v, also logs each client command and the response.

- **vvv** – In addition to the information logged by **vv**, also logs internal state transitions.

Amazon RDS configures these MySQL DAEMON_MEMCACHED_OPTIONS parameters, they cannot be modified: DAEMON_PROCESS, LARGE_MEMORY_PAGES, MAXIMUM_CORE_FILE_LIMIT, MAX_ITEM_SIZE, LOCK_DOWN_PAGE_MEMORY, MASK, IDFILE, REQUESTS_PER_EVENT, SOCKET, and USER.

Parameters for MySQL

By default, a MySQL DB instance uses a DB parameter group that is specific to a MySQL database. This parameter group contains parameters for the MySQL database engine. For information about working with parameter groups and setting parameters, see [Parameter groups for Amazon RDS](#).

RDS for MySQL parameters are set to the default values of the storage engine that you have selected. For more information about MySQL parameters, see the [MySQL documentation](#). For more information about MySQL storage engines, see [Supported storage engines for RDS for MySQL](#).

You can view the parameters available for a specific RDS for MySQL version using the RDS console or the AWS CLI. For information about viewing the parameters in a MySQL parameter group in the RDS console, see [Viewing parameter values for a DB parameter group in Amazon RDS](#).

Using the AWS CLI, you can view the parameters for an RDS for MySQL version by running the [describe-engine-default-parameters](#) command. Specify one of the following values for the --db-parameter-group-family option:

- mysql8.4
- mysql8.0
- mysql5.7

For example, to view the parameters for RDS for MySQL version 8.0, run the following command.

```
aws rds describe-engine-default-parameters --db-parameter-group-family mysql8.0
```

Your output looks similar to the following.

```
{  
    "EngineDefaults": {  
        "Parameters": [  
            {  
                "ParameterName": "activate_all_roles_on_login",  
                "ParameterValue": "0",  
                "Description": "Automatically set all granted roles as active after the user has authenticated successfully.",  
                "Source": "engine-default",  
                "ApplyType": "dynamic",  
                "DataType": "boolean",  
                "IsModifiable": true,  
                "MinimumValue": null,  
                "MaximumValue": null,  
                "AllowedValues": null  
            }  
        ]  
    }  
}
```

```
        "AllowedValues": "0,1",
        "IsModifiable": true
    },
    {
        "ParameterName": "allow-suspicious-udfs",
        "Description": "Controls whether user-defined functions that have only
an xxx symbol for the main function can be loaded",
        "Source": "engine-default",
        "ApplyType": "static",
        "DataType": "boolean",
        "AllowedValues": "0,1",
        "IsModifiable": false
    },
    {
        "ParameterName": "auto_generate_certs",
        "Description": "Controls whether the server autogenerated SSL key and
certificate files in the data directory, if they do not already exist.",
        "Source": "engine-default",
        "ApplyType": "static",
        "DataType": "boolean",
        "AllowedValues": "0,1",
        "IsModifiable": false
    },
    ...
}
```

To list only the modifiable parameters for RDS for MySQL version 8.0, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mysql8.0 \
--query 'EngineDefaults.Parameters[?IsModifiable==`true`]'
```

For Windows:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mysql8.0 ^
--query "EngineDefaults.Parameters[?IsModifiable==`true`]"
```

Common DBA tasks for MySQL DB instances

In the following content, you can find descriptions of the Amazon RDS-specific implementations of some common DBA tasks for DB instances running the MySQL database engine. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. Also, it restricts access to certain system procedures and tables that require advanced privileges.

For information about working with MySQL log files on Amazon RDS, see [MySQL database log files](#).

Understanding predefined users

Amazon RDS automatically creates several predefined users with new RDS for MySQL DB instances. Predefined users and their privileges can't be changed. You can't drop, rename, or modify privileges for these predefined users. Attempting to do so results in an error.

- **rdsadmin** – A user that's created to handle many of the management tasks that the administrator with superuser privileges would perform on a standalone MySQL database. This user is used internally by RDS for MySQL for many management tasks.
- **rdsrepladmin** – A user that's used internally by Amazon RDS to support replication activities on RDS for MySQL DB instances and clusters.

For information about other common DBA tasks, see the following topics.

Topics

- [Role-based privilege model for RDS for MySQL](#)
- [Dynamic privileges for RDS for MySQL](#)
- [Ending a session or query for RDS for MySQL](#)
- [Skipping the current replication error for RDS for MySQL](#)
- [Working with InnoDB tablespaces to improve crash recovery times for RDS for MySQL](#)
- [Managing the Global Status History for RDS for MySQL](#)
- [Configuring buffer pool size and redo log capacity in MySQL 8.4](#)

Role-based privilege model for RDS for MySQL

Starting with RDS for MySQL version 8.0.36, you can't modify the tables in the `mysql` database directly. In particular, you can't create database users by performing data manipulation language

(DML) operations on the grant tables. Instead, you use MySQL account-management statements such as CREATE USER, GRANT, and REVOKE to grant role-based privileges to users. You also can't create other kinds of objects such as stored procedures in the mysql database. You can still query the mysql tables. If you use binary log replication, changes made directly to the mysql tables on the source DB instance aren't replicated to the target cluster.

In some cases, your application might use shortcuts to create users or other objects by inserting into the mysql tables. If so, change your application code to use the corresponding statements such as CREATE USER.

To export metadata for database users during the migration from an external MySQL database, use one of the following methods:

- Use MySQL Shell's instance dump utility with a filter to exclude users, roles, and grants. The following example shows you the command syntax to use. Make sure that outputUrl is empty.

```
mysqlsh user@host -- util.dumpInstance(outputUrl, {excludeSchemas: ['mysql'], users: true})
```

For more information, see [Instance Dump Utility, Schema Dump Utility, and Table Dump Utility](#) in the MySQL Reference Manual.

- Use the mysqlpump client utility. This example includes all tables except for tables in the mysql system database. It also includes CREATE USER and GRANT statements to reproduce all MySQL users in the migrated database.

```
mysqlpump --exclude-databases=mysql --users
```

The mysqlpump client utility is no longer available with MySQL 8.4. Instead, use mysqldump.

To simplify managing permissions for many users or applications, you can use the CREATE ROLE statement to create a role that has a set of permissions. Then you can use the GRANT and SET ROLE statements and the current_role function to assign roles to users or applications, switch the current role, and check which roles are in effect. For more information on the role-based permission system in MySQL 8.0, see [Using Roles](#) in the MySQL Reference Manual.

⚠ Important

We strongly recommend that you do not use the master user directly in your applications. Instead, adhere to the best practice of using a database user created with the minimal privileges required for your application.

Starting with version 8.0.36, RDS for MySQL includes a special role that has all of the following privileges. This role is named `rds_superuser_role`. The primary administrative user for each DB instance already has this role granted. The `rds_superuser_role` role includes the following privileges for all database objects:

- ALTER
- APPLICATION_PASSWORD_ADMIN
- ALTER ROUTINE
- CREATE
- CREATE ROLE
- CREATE ROUTINE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- DROP ROLE
- EVENT
- EXECUTE
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT

- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT
- SHOW DATABASES
- SHOW VIEW
- TRIGGER
- UPDATE
- XA_RECOVER_ADMIN

The role definition also includes WITH GRANT OPTION so that an administrative user can grant that role to other users. In particular, the administrator must grant any privileges needed to perform binary log replication with the MySQL cluster as the target.

 **Tip**

To see the full details of the permissions, use the following statement.

```
SHOW GRANTS FOR rds_superuser_role@'%';
```

When you grant access by using roles in RDS for MySQL version 8.0.36 and higher, you also activate the role by using the SET ROLE *role_name* or SET ROLE ALL statement. The following example shows how. Substitute the appropriate role name for CUSTOM_ROLE.

```
# Grant role to user
mysql> GRANT CUSTOM_ROLE TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the CUSTOM_ROLE role has not
# been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE()          |
+-----+
| `rds_superuser_role`@`%` |
+-----+
```

```
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| `CUSTOM_ROLE`@`%`, `rds_superuser_role`@`%` |
+-----+
```

Dynamic privileges for RDS for MySQL

Dynamic privileges are MySQL privileges that you can explicitly grant by using the GRANT statement. Depending on your version of RDS for MySQL, RDS allows you to grant only specific dynamic privileges. RDS disallows some of these privileges because they can interfere with the specific database operations, such as replication and backup.

The following table shows which of these privileges you can grant for different MySQL versions. If you are upgrading from a MySQL version lower than 8.0.36 to version 8.0.36 or higher, you might have to update your application code if granting a particular privilege is no longer allowed.

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
<u>ALLOW_NON_EXISTENT_DEFINER</u>	Not available	Not available	Disallowed
<u>APPLICATION_PASSWORD_ADMIN</u>	Allowed	Allowed	Allowed
<u>AUDIT_ABORT_EXEMPT</u>	Allowed	Disallowed	Disallowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
AUDIT_ADMIN	Disallowed	Disallowed	Disallowed
AUTHENTICATION_POLICY_ADMIN	Allowed	Disallowed	Disallowed
BACKUP_ADMIN	Allowed	Disallowed	Disallowed
BINLOG_ADMIN	Allowed	Disallowed	Disallowed
BINLOG_ENCRYPTION_ADMIN	Disallowed	Disallowed	Disallowed
CLONE_ADMIN	Disallowed	Disallowed	Disallowed
CONNECTION_ADMIN	Allowed	Disallowed	Disallowed
ENCRYPTION_KEY_ADMIN	Disallowed	Disallowed	Disallowed
FIREWALL_ADMIN	Disallowed	Disallowed	Disallowed
FIREWALL_EXEMPT	Allowed	Disallowed	Disallowed
FIREWALL_USER	Disallowed	Disallowed	Disallowed
FLUSH_OPTIMIZER_COSTS	Allowed	Allowed	Allowed
FLUSH_PRIVILEGES	Not available	Not available	Allowed
FLUSH_STATUS	Allowed	Allowed	Allowed
FLUSH_TABLES	Allowed	Allowed	Allowed
FLUSH_USE_RESOURCES	Allowed	Allowed	Allowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
<u>GROUP_REPLICATION_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>GROUP_REPLICATION_STREAM</u>	Disallowed	Disallowed	Disallowed
<u>INNODB_READONLY_LOG_ARCHIVE</u>	Disallowed	Disallowed	Disallowed
<u>INNODB_READONLY_LOG_ENABLE</u>	Disallowed	Disallowed	Disallowed
<u>MASKING_DICTIONARIES_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>NDB_STORED_USER</u>	Disallowed	Disallowed	Disallowed
<u>OPTIMIZE_LOCAL_TABLE</u>	Not available	Not available	Disallowed
<u>PASSWORDLESS_USER_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>PERSIST_RO_VARIABLES_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>REPLICATION_APPLIER</u>	Allowed	Disallowed	Disallowed
<u>REPLICATION_SLAVE_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>RESOURCE_GROUP_ADMIN</u>	Allowed	Disallowed	Disallowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
<u>RESOURCE_GROUP_USER</u>	Allowed	Disallowed	Disallowed
<u>ROLE_ADMIN</u>	Allowed	Allowed	Allowed
<u>SENSITIVE_VARIABLES_OBSERVER</u>	Allowed	Allowed	Allowed
<u>SERVICE_CONNECTION_ADMIN</u>	Allowed	Disallowed	Disallowed
<u>SESSION_VARIABLES_ADMIN</u>	Allowed	Allowed	Allowed
<u>SET_ANY_DEFINER</u>	Not available	Not available	Allowed
<u>SET_USER_ID</u>	Allowed	Allowed	Not available
<u>SHOW_ROUTINE</u>	Allowed	Allowed	Allowed
<u>SKIP_QUERY_REWRITE</u>	Disallowed	Disallowed	Disallowed
<u>SYSTEM_USER</u>	Disallowed	Disallowed	Disallowed
<u>SYSTEM_VARIABLES_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>TABLE_ENCRYPTION_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>TELEMETRY_LOG_ADMIN</u>	Allowed	Disallowed	Disallowed
<u>TP_CONNECTION_ADMIN</u>	Disallowed	Disallowed	Disallowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
<u>TRANSACTION_GTID_TAG</u>	Not available	Not available	Disallowed
<u>VERSION_TOKEN_ADMIN</u>	Disallowed	Disallowed	Disallowed
<u>XA_RECOVER_ADMIN</u>	Allowed	Allowed	Allowed

Ending a session or query for RDS for MySQL

You can end user sessions or queries on DB instances by using the `rds_kill` and `rds_kill_query` commands. First connect to your MySQL DB instance, then issue the appropriate command as shown following. For more information, see [Connecting to your MySQL DB instance](#).

```
CALL mysql.rds_kill(thread-ID)
CALL mysql.rds_kill_query(thread-ID)
```

For example, to end the session that is running on thread 99, you would type the following:

```
CALL mysql.rds_kill(99);
```

To end the query that is running on thread 99, you would type the following:

```
CALL mysql.rds_kill_query(99);
```

Skipping the current replication error for RDS for MySQL

You can skip an error on your read replica if the error is causing your read replica to stop responding and the error doesn't affect the integrity of your data.

Note

First verify that the error in question can be safely skipped. In a MySQL utility, connect to the read replica and run the following MySQL command.

```
SHOW REPLICA STATUS\G
```

For information about the values returned, see [the MySQL documentation](#).

Previous versions of MySQL used SHOW SLAVE STATUS instead of SHOW REPLICA STATUS. If you are using a MySQL version before 8.0.23, then use SHOW SLAVE STATUS.

You can skip an error on your read replica in the following ways.

Topics

- [Calling the mysql.rds_skip_repl_error procedure](#)
- [Setting the slave_skip_errors parameter](#)

Calling the mysql.rds_skip_repl_error procedure

Amazon RDS provides a stored procedure that you can call to skip an error on your read replicas. First connect to your read replica, then issue the appropriate commands as shown following. For more information, see [Connecting to your MySQL DB instance](#).

To skip the error, issue the following command.

```
CALL mysql.rds_skip_repl_error;
```

This command has no effect if you run it on the source DB instance, or on a read replica that hasn't encountered a replication error.

For more information, such as the versions of MySQL that support `mysql.rds_skip_repl_error`, see [mysql.rds_skip_repl_error](#).

Important

If you attempt to call `mysql.rds_skip_repl_error` and encounter the following error: ERROR 1305 (42000): PROCEDURE `mysql.rds_skip_repl_error` does not exist, then upgrade your MySQL DB instance to the latest minor version or one of the minimum minor versions listed in [mysql.rds_skip_repl_error](#).

Setting the slave_skip_errors parameter

To skip one or more errors, you can set the `slave_skip_errors` static parameter on the read replica. You can set this parameter to skip one or more specific replication error codes. Currently, you can set this parameter only for RDS for MySQL 5.7 DB instances. After you change the setting for this parameter, make sure to reboot your DB instance for the new setting to take effect. For information about setting this parameter, see the [MySQL documentation](#).

We recommend setting this parameter in a separate DB parameter group. You can associate this DB parameter group only with the read replicas that need to skip errors. Following this best practice reduces the potential impact on other DB instances and read replicas.

 **Important**

Setting a nondefault value for this parameter can lead to replication inconsistency. Only set this parameter to a nondefault value if you have exhausted other options to resolve the problem and you are sure of the potential impact on your read replica's data.

Working with InnoDB tablespaces to improve crash recovery times for RDS for MySQL

Every table in MySQL consists of a table definition, data, and indexes. The MySQL storage engine InnoDB stores table data and indexes in a *tablespace*. InnoDB creates a global shared tablespace that contains a data dictionary and other relevant metadata, and it can contain table data and indexes. InnoDB can also create separate tablespaces for each table and partition. These separate tablespaces are stored in files with a .ibd extension and the header of each tablespace contains a number that uniquely identifies it.

Amazon RDS provides a parameter in a MySQL parameter group called `innodb_file_per_table`. This parameter controls whether InnoDB adds new table data and indexes to the shared tablespace (by setting the parameter value to 0) or to individual tablespaces (by setting the parameter value to 1). Amazon RDS sets the default value for `innodb_file_per_table` parameter to 1, which allows you to drop individual InnoDB tables and reclaim storage used by those tables for the DB instance. In most use cases, setting the `innodb_file_per_table` parameter to 1 is the recommended setting.

You should set the `innodb_file_per_table` parameter to 0 when you have a large number of tables, such as over 1000 tables when you use standard (magnetic) or general purpose SSD storage

or over 10,000 tables when you use Provisioned IOPS storage. When you set this parameter to 0, individual tablespaces are not created and this can improve the time it takes for database crash recovery.

MySQL processes each metadata file, which includes tablespaces, during the crash recovery cycle. The time it takes MySQL to process the metadata information in the shared tablespace is negligible compared to the time it takes to process thousands of tablespace files when there are multiple tablespaces. Because the tablespace number is stored within the header of each file, the aggregate time to read all the tablespace files can take up to several hours. For example, a million InnoDB tablespaces on standard storage can take from five to eight hours to process during a crash recovery cycle. In some cases, InnoDB can determine that it needs additional cleanup after a crash recovery cycle so it will begin another crash recovery cycle, which will extend the recovery time. Keep in mind that a crash recovery cycle also entails rolling-back transactions, fixing broken pages, and other operations in addition to the processing of tablespace information.

Since the `innodb_file_per_table` parameter resides in a parameter group, you can change the parameter value by editing the parameter group used by your DB instance without having to reboot the DB instance. After the setting is changed, for example, from 1 (create individual tables) to 0 (use shared tablespace), new InnoDB tables will be added to the shared tablespace while existing tables continue to have individual tablespaces. To move an InnoDB table to the shared tablespace, you must use the `ALTER TABLE` command.

Migrating multiple tablespaces to the shared tablespace

You can move an InnoDB table's metadata from its own tablespace to the shared tablespace, which will rebuild the table metadata according to the `innodb_file_per_table` parameter setting. First connect to your MySQL DB instance, then issue the appropriate commands as shown following. For more information, see [Connecting to your MySQL DB instance](#).

```
ALTER TABLE table_name ENGINE = InnoDB, ALGORITHM=COPY;
```

For example, the following query returns an `ALTER TABLE` statement for every InnoDB table that is not in the shared tablespace.

For MySQL 5.7 DB instances:

```
SELECT CONCAT('ALTER TABLE `',
REPLACE(LEFT(NAME , INSTR((NAME), '/') - 1), '`', '''), `'', `,
REPLACE(SUBSTR(NAME FROM INSTR(NAME, '/') + 1), '`', '''), `'' ENGINE=InnoDB,
ALGORITHM=COPY;') AS Query
```

```
FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES  
WHERE SPACE <> 0 AND LEFT(NAME, INSTR((NAME), '/') - 1) NOT IN ('mysql','');
```

For MySQL 8.4 and 8.0 DB instances:

```
SELECT CONCAT('ALTER TABLE `',  
REPLACE(LEFT(NAME , INSTR((NAME), '/') - 1), '``', '```'), '`.`',  
REPLACE(SUBSTR(NAME FROM INSTR(NAME, '/') + 1), '``', '```'), '` ENGINE=InnoDB,  
ALGORITHM=COPY;') AS Query  
FROM INFORMATION_SCHEMA.INNODB_TABLES  
WHERE SPACE <> 0 AND LEFT(NAME, INSTR((NAME), '/') - 1) NOT IN ('mysql','');
```

Rebuilding a MySQL table to move the table's metadata to the shared tablespace requires additional storage space temporarily to rebuild the table, so the DB instance must have storage space available. During rebuilding, the table is locked and inaccessible to queries. For small tables or tables not frequently accessed, this might not be an issue. For large tables or tables frequently accessed in a heavily concurrent environment, you can rebuild tables on a read replica.

You can create a read replica and migrate table metadata to the shared tablespace on the read replica. While the ALTER TABLE statement blocks access on the read replica, the source DB instance is not affected. The source DB instance will continue to generate its binary logs while the read replica lags during the table rebuilding process. Because the rebuilding requires additional storage space and the replay log file can become large, you should create a read replica with storage allocated that is larger than the source DB instance.

To create a read replica and rebuild InnoDB tables to use the shared tablespace, take the following steps:

1. Make sure that backup retention is enabled on the source DB instance so that binary logging is enabled.
2. Use the AWS Management Console or AWS CLI to create a read replica for the source DB instance. Because the creation of a read replica involves many of the same processes as crash recovery, the creation process can take some time if there is a large number of InnoDB tablespaces. Allocate more storage space on the read replica than is currently used on the source DB instance.
3. When the read replica has been created, create a parameter group with the parameter settings `read_only = 0` and `innodb_file_per_table = 0`. Then associate the parameter group with the read replica.
4. Issue the following SQL statement for all tables that you want migrated on the replica:

```
ALTER TABLE name ENGINE = InnoDB
```

5. When all of your ALTER TABLE statements have completed on the read replica, verify that the read replica is connected to the source DB instance and that the two instances are in sync.
6. Use the console or CLI to promote the read replica to be the instance. Make sure that the parameter group used for the new standalone DB instance has the `innodb_file_per_table` parameter set to 0. Change the name of the new standalone DB instance, and point any applications to the new standalone DB instance.

Managing the Global Status History for RDS for MySQL

Tip

To analyze database performance, you can also use Performance Insights on Amazon RDS. For more information, see [Monitoring DB load with Performance Insights on Amazon RDS](#).

MySQL maintains many status variables that provide information about its operation. Their value can help you detect locking or memory issues on a DB instance. The values of these status variables are cumulative since last time the DB instance was started. You can reset most status variables to 0 by using the FLUSH STATUS command.

To allow for monitoring of these values over time, Amazon RDS provides a set of procedures that will snapshot the values of these status variables over time and write them to a table, along with any changes since the last snapshot. This infrastructure, called Global Status History (GoSH), is installed on all MySQL DB instances starting with versions 5.5.23. GoSH is disabled by default.

To enable GoSH, you first enable the event scheduler from a DB parameter group by setting the parameter `event_scheduler` to ON. For MySQL DB instances running MySQL 5.7, also set the parameter `show_compatibility_56` to 1. For information about creating and modifying a DB parameter group, see [Parameter groups for Amazon RDS](#). For information about the side effects of enabling this parameter, see [show_compatibility_56](#) in the *MySQL 5.7 Reference Manual*.

You can then use the procedures in the following table to enable and configure GoSH. First connect to your MySQL DB instance, then issue the appropriate commands as shown following. For more information, see [Connecting to your MySQL DB instance](#). For each procedure, run the following command and replace `procedure-name`:

```
CALL procedure-name;
```

The following table lists all of the procedures that you can use for *procedure-name* in the previous command.

Procedure	Description
<code>mysql.rds_enable_gsh_collector</code>	Enables GoSH to take default snapshots at intervals specified by <code>rds_set_gsh_collector</code> .
<code>mysql.rds_set_gsh_collector</code>	Specifies the interval, in minutes, between snapshots. Default value is 5.
<code>mysql.rds_disable_gsh_collector</code>	Disables snapshots.
<code>mysql.rds_collect_global_status_history</code>	Takes a snapshot on demand.
<code>mysql.rds_enable_gsh_rotation</code>	Enables rotation of the contents of the <code>mysql.rds_global_status_history</code> table to <code>mysql.rds_global_status_history_old</code> at intervals specified by <code>rds_set_gsh_rotation</code> .
<code>mysql.rds_set_gsh_rotation</code>	Specifies the interval, in days, between table rotations. Default value is 7.
<code>mysql.rds_disable_gsh_rotation</code>	Disables table rotation.
<code>mysql.rds_rotate_global_status_history</code>	Rotates the contents of the <code>mysql.rds_global_status_history</code> table to <code>mysql.rds_global_status_history_old</code> on demand.

When GoSH is running, you can query the tables that it writes to. For example, to query the hit ratio of the Innodb buffer pool, you would issue the following query:

```
select a.collection_end, a.collection_start, (( a.variable_Delta-b.variable_delta)/  
a.variable_delta)*100 as "HitRatio"  
  from mysql.rds_global_status_history as a join mysql.rds_global_status_history as b  
on a.collection_end = b.collection_end  
 where a. variable_name = 'Innodb_buffer_pool_read_requests' and b.variable_name =  
'Innodb_buffer_pool_reads'
```

Configuring buffer pool size and redo log capacity in MySQL 8.4

In MySQL 8.4, Amazon RDS enables the `innodb_dedicated_server` parameter by default. With the `innodb_dedicated_server` parameter, the database engine calculates the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters. For information about how these parameters are calculated, see [Configuring InnoDB Buffer Pool Size](#) and [Redo Log](#) in the MySQL documentation.

With `innodb_dedicated_server` enabled, the `innodb_buffer_pool_size` parameter is calculated based on the DB instance class memory. The following table shows the detected server memory and the corresponding buffer pool size.

Detected server memory	Buffer pool size
< 1 GB	Default value of 128 MB
1 GB to 4 GB	<i>Detected server memory</i> * 0.5
> 4 GB	<i>Detected server memory</i> * 0.75

The `innodb_redo_log_capacity` parameter automatically scales with the instance class to (number of vCPUs / 2) GB up to a maximum of 16 GB. Larger instance classes have a larger redo log capacity, which can improve performance and resilience for write-intensive workloads.

Before upgrading from MySQL 8.0 to MySQL 8.4, be sure to increase your storage space to accommodate a potential increase in the size of the redo logs that might occur after the upgrade completes. For more information, see [Increasing DB instance storage capacity](#).

If you don't want the `innodb_dedicated_server` parameter to calculate the values for the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters, you can override these values by setting specific values for them in a custom parameter group.

Alternatively, you can disable the `innodb_dedicated_server` parameter and set values for the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters in a custom parameter group. For more information, see [Default and custom parameter groups](#).

If you disable the `innodb_dedicated_server` parameter by setting it to `0` and don't set values for the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters, then Amazon RDS sets the latter two parameters to 128 MB and 100 MB, respectively. These defaults result in poor performance on larger instance classes.

Local time zone for MySQL DB instances

By default, the time zone for a MySQL DB instance is Universal Time Coordinated (UTC). You can set the time zone for your DB instance to the local time zone for your application instead.

To set the local time zone for a DB instance, set the `time_zone` parameter in the parameter group for your DB instance to one of the supported values listed later in this section. When you set the `time_zone` parameter for a parameter group, all DB instances and read replicas that are using that parameter group change to use the new local time zone. For information on setting parameters in a parameter group, see [Parameter groups for Amazon RDS](#).

After you set the local time zone, all new connections to the database reflect the change. If you have any open connections to your database when you change the local time zone, you won't see the local time zone update until after you close the connection and open a new connection.

You can set a different local time zone for a DB instance and one or more of its read replicas. To do this, use a different parameter group for the DB instance and the replica or replicas and set the `time_zone` parameter in each parameter group to a different local time zone.

If you are replicating across AWS Regions, then the source DB instance and the read replica use different parameter groups (parameter groups are unique to an AWS Region). To use the same local time zone for each instance, you must set the `time_zone` parameter in the instance's and read replica's parameter groups.

When you restore a DB instance from a DB snapshot, the local time zone is set to UTC. You can update the time zone to your local time zone after the restore is complete. If you restore a DB instance to a point in time, then the local time zone for the restored DB instance is the time zone setting from the parameter group of the restored DB instance.

The Internet Assigned Numbers Authority (IANA) publishes new time zones at <https://www.iana.org/time-zones> several times a year. Every time RDS releases a new minor maintenance release of MySQL, it ships with the latest time zone data at the time of the release. When you use the latest RDS for MySQL versions, you have recent time zone data from RDS. To ensure that your DB instance has recent time zone data, we recommend upgrading to a higher DB engine version. Alternatively, you can modify the time zone tables in MariaDB DB instances manually. To do so, you can use SQL commands or run the [mysql_tzinfo_to_sql tool](#) in a SQL client. After updating the time zone data manually, reboot your DB instance so that the changes take effect. RDS doesn't modify or reset the time zone data of running DB instances. New time zone data is installed only when you perform a database engine version upgrade.

You can set your local time zone to one of the following values.

Zone	Time zone
Africa	Africa/Cairo, Africa/Casablanca, Africa/Harare, Africa/Monrovia, Africa/Nairobi, Africa/Tripoli, Africa/Windhoek
America	America/Araguaina, America/Asuncion, America/Bogota, America/Buenos_Aires, America/Caracas, America/Chihuahua, America/Cuiaba, America/Denver, America/Fortaleza, America/Guatemala, America/Halifax, America/Manaus, America/Matamoros, America/Monterrey, America/Montevideo, America/Phoenix, America/Santiago, America/Tijuana
Asia	Asia/Amman, Asia/Ashgabat, Asia/Baghdad, Asia/Baku, Asia/Bangkok, Asia/Beirut, Asia/Calcutta, Asia/Damascus, Asia/Dhaka, Asia/Irkutsk, Asia/Jerusalem, Asia/Kabul, Asia/Karachi, Asia/Kathmandu, Asia/Krasnoyarsk, Asia/Magadan, Asia/Muscat, Asia/Novosibirsk, Asia/Riyadh, Asia/Seoul, Asia/Shanghai, Asia/Singapore, Asia/Taipei, Asia/Tehran, Asia/Tokyo, Asia/Ulaanbaatar, Asia/Vladivostok, Asia/Yakutsk, Asia/Yerevan
Atlantic	Atlantic/Azores
Australia	Australia/Adelaide, Australia/Brisbane, Australia/Darwin, Australia/Hobart, Australia/Perth, Australia/Sydney
Brazil	Brazil/DeNoronha, Brazil/East
Canada	Canada/Newfoundland, Canada/Saskatchewan, Canada/Yukon
Europe	Europe/Amsterdam, Europe/Athens, Europe/Dublin, Europe/Helsinki, Europe/Istanbul, Europe/Kaliningrad, Europe/Moscow, Europe/Paris, Europe/Prague, Europe/Sarajevo
Pacific	Pacific/Auckland, Pacific/Fiji, Pacific/Guam, Pacific/Honolulu, Pacific/Samoa
US	US/Alaska, US/Central, US/East-Indiana, US/Eastern, US/Pacific
UTC	UTC

Known issues and limitations for Amazon RDS for MySQL

Known issues and limitations for working with Amazon RDS for MySQL are as follows.

Topics

- [InnoDB reserved word](#)
- [Storage-full behavior for Amazon RDS for MySQL](#)
- [Inconsistent InnoDB buffer pool size](#)
- [Index merge optimization returns incorrect results](#)
- [MySQL parameter exceptions for Amazon RDS DB instances](#)
- [MySQL file size limits in Amazon RDS](#)
- [MySQL Keyring Plugin not supported](#)
- [Custom ports](#)
- [MySQL stored procedure limitations](#)
- [GTID-based replication with an external source instance](#)
- [MySQL default authentication plugin](#)
- [Overriding innodb_buffer_pool_size](#)
- [Upgrading from MySQL 5.7 to MySQL 8.4](#)
- [InnoDB page compression](#)

InnoDB reserved word

InnoDB is a reserved word for RDS for MySQL. You can't use this name for a MySQL database.

Storage-full behavior for Amazon RDS for MySQL

When storage becomes full for a MySQL DB instance, there can be metadata inconsistencies, dictionary mismatches, and orphan tables. To prevent these issues, Amazon RDS automatically stops a DB instance that reaches the storage-full state.

A MySQL DB instance reaches the storage-full state in the following cases:

- The DB instance has less than 20,000 MiB of storage, and available storage reaches 200 MiB or less.

- The DB instance has more than 102,400 MiB of storage, and available storage reaches 1024 MiB or less.
- The DB instance has between 20,000 MiB and 102,400 MiB of storage, and has less than 1% of storage available.

After Amazon RDS stops a DB instance automatically because it reached the `storage-full` state, you can still modify it. To restart the DB instance, complete at least one of the following:

- Modify the DB instance to enable storage autoscaling.

For more information about storage autoscaling, see [Managing capacity automatically with Amazon RDS storage autoscaling](#).

- Modify the DB instance to increase its storage capacity.

For more information about increasing storage capacity, see [Increasing DB instance storage capacity](#).

After you make one of these changes, the DB instance is restarted automatically. For information about modifying a DB instance, see [Modifying an Amazon RDS DB instance](#).

Inconsistent InnoDB buffer pool size

For MySQL 5.7, there is currently a bug in the way that the InnoDB buffer pool size is managed. MySQL 5.7 might adjust the value of the `innodb_buffer_pool_size` parameter to a large value that can result in the InnoDB buffer pool growing too large and using up too much memory. This effect can cause the MySQL database engine to stop running or can prevent it from starting. This issue is more common for DB instance classes that have less memory available.

To resolve this issue, set the value of the `innodb_buffer_pool_size` parameter to a multiple of the product of the `innodb_buffer_pool_instances` parameter value and the `innodb_buffer_pool_chunk_size` parameter value. For example, you might set the `innodb_buffer_pool_size` parameter value to a multiple of eight times the product of the `innodb_buffer_pool_instances` and `innodb_buffer_pool_chunk_size` parameter values, as shown in the following example.

```
innodb_buffer_pool_chunk_size = 536870912  
innodb_buffer_pool_instances = 4  
innodb_buffer_pool_size = (536870912 * 4) * 8 = 17179869184
```

For details on this MySQL 5.7 bug, see <https://bugs.mysql.com/bug.php?id=79379> in the MySQL documentation.

Index merge optimization returns incorrect results

Queries that use index merge optimization might return incorrect results because of a bug in the MySQL query optimizer that was introduced in MySQL 5.5.37. When you issue a query against a table with multiple indexes, the optimizer scans ranges of rows based on the multiple indexes, but does not merge the results together correctly. For more information on the query optimizer bug, see <http://bugs.mysql.com/bug.php?id=72745> and <http://bugs.mysql.com/bug.php?id=68194> in the MySQL bug database.

For example, consider a query on a table with two indexes where the search arguments reference the indexed columns.

```
SELECT * FROM table1  
WHERE indexed_col1 = 'value1' AND indexed_col2 = 'value2';
```

In this case, the search engine will search both indexes. However, because of the bug, the merged results are incorrect.

To resolve this issue, you can do one of the following:

- Set the `optimizer_switch` parameter to `index_merge=off` in the DB parameter group for your MySQL DB instance. For information on setting DB parameter group parameters, see [Parameter groups for Amazon RDS](#).
- Upgrade your MySQL DB instance to MySQL version 5.7 or 8.0. For more information, see [Upgrades of the RDS for MySQL DB engine](#).
- If you cannot upgrade your instance or change the `optimizer_switch` parameter, you can work around the bug by explicitly identifying an index for the query, for example:

```
SELECT * FROM table1  
USE INDEX covering_index  
WHERE indexed_col1 = 'value1' AND indexed_col2 = 'value2';
```

For more information, see [Index merge optimization](#) in the MySQL documentation.

MySQL parameter exceptions for Amazon RDS DB instances

Some MySQL parameters require special considerations when used with an Amazon RDS DB instance.

lower_case_table_names

Because Amazon RDS uses a case-sensitive file system, setting the value of the `lower_case_table_names` server parameter to 2 (names stored as given but compared in lowercase) is not supported. The following are the supported values for Amazon RDS for MySQL DB instances:

- 0 (names stored as given and comparisons are case-sensitive) is supported for all RDS for MySQL versions.
- 1 (names stored in lowercase and comparisons are not case-sensitive) is supported for RDS for MySQL version 5.7, version 8.0.28 and higher 8.0 versions, and version 8.4.

Set the `lower_case_table_names` parameter in a custom DB parameter group before creating a DB instance. Then, specify the custom DB parameter group when you create the DB instance.

When a parameter group is associated with a MySQL DB instance with a version lower than 8.0, we recommend that you avoid changing the `lower_case_table_names` parameter in the parameter group. Changing it could cause inconsistencies with point-in-time recovery backups and read replica DB instances.

When a parameter group is associated with a version 8.0 or 8.4 MySQL DB instance, you can't modify the `lower_case_table_names` parameter in the parameter group.

Read replicas should always use the same `lower_case_table_names` parameter value as the source DB instance.

long_query_time

You can set the `long_query_time` parameter to a floating point value so that you can log slow queries to the MySQL slow query log with microsecond resolution. You can set a value such as 0.1 seconds, which would be 100 milliseconds, to help when debugging slow transactions that take less than one second.

MySQL file size limits in Amazon RDS

For MySQL versions 8.0 and higher DB instances, the maximum file size is 16 TiB. When using file-per-table tablespaces, the maximum file size limits the size of an InnoDB table to 16 TiB. InnoDB file-per-table tablespaces (with tables each in their own tablespace) is set by default for MySQL DB instances. For more information, see [InnoDB Limits](#) in the MySQL documentation.

Note

Some existing DB instances have a lower limit. For example, MySQL DB instances created before April 2014 have a file and table size limit of 2 TB. This 2 TB file size limit also applies to DB instances or read replicas created from DB snapshots taken before April 2014, regardless of when the DB instance was created.

There are advantages and disadvantages to using InnoDB file-per-table tablespaces, depending on your application. To determine the best approach for your application, see [File-per-table tablespaces](#) in the MySQL documentation.

We don't recommend allowing tables to grow to the maximum file size. In general, a better practice is to partition data into smaller tables, which can improve performance and recovery times.

One option that you can use for breaking up a large table into smaller tables is partitioning. Partitioning distributes portions of your large table into separate files based on rules that you specify. For example, if you store transactions by date, you can create partitioning rules that distribute older transactions into separate files using partitioning. Then periodically, you can archive the historical transaction data that doesn't need to be readily available to your application. For more information, see [Partitioning](#) in the MySQL documentation.

Because there is no single system table or view that provides the size of all the tables and the InnoDB system tablespace, you must query multiple tables to determine the size of the tablespaces.

To determine the size of the InnoDB system tablespace and the data dictionary tablespace

- Use the following SQL command to determine if any of your tablespaces are too large and are candidates for partitioning.

Note

The data dictionary tablespace is specific to MySQL 8.0 and higher versions.

```
select FILE_NAME, TABLESPACE_NAME, ROUND(((TOTAL_EXTENTS*EXTENT_SIZE)/1024/1024/1024), 2) as "File Size (GB)" from information_schema.FILES where tablespace_name in ('mysql','innodb_system');
```

To determine the size of InnoDB user tables outside of the InnoDB system tablespace (for MySQL 5.7 versions)

- Use the following SQL command to determine if any of your tables are too large and are candidates for partitioning.

```
SELECT SPACE,NAME,ROUND((ALLOCATED_SIZE/1024/1024/1024), 2)  
as "Tablespace Size (GB)"  
FROM information_schema.INNODB_SYS_TABLESPACES ORDER BY 3 DESC;
```

To determine the size of InnoDB user tables outside of the InnoDB system tablespace (for MySQL 8.0 and higher versions)

- Use the following SQL command to determine if any of your tables are too large and are candidates for partitioning.

```
SELECT SPACE,NAME,ROUND((ALLOCATED_SIZE/1024/1024/1024), 2)  
as "Tablespace Size (GB)"  
FROM information_schema.INNODB_TABLESPACES ORDER BY 3 DESC;
```

To determine the size of non-InnoDB user tables

- Use the following SQL command to determine if any of your non-InnoDB user tables are too large.

```
SELECT TABLE_SCHEMA, TABLE_NAME, round(((DATA_LENGTH + INDEX_LENGTH+DATA_FREE)/ 1024 / 1024/ 1024), 2) As "Approximate size (GB)" FROM information_schema.TABLES
```

```
WHERE TABLE_SCHEMA NOT IN ('mysql', 'information_schema', 'performance_schema')
and ENGINE<>'InnoDB';
```

To enable InnoDB file-per-table tablespaces

- Set the *innodb_file_per_table* parameter to 1 in the parameter group for the DB instance.

To disable InnoDB file-per-table tablespaces

- Set the *innodb_file_per_table* parameter to 0 in the parameter group for the DB instance.

For information on updating a parameter group, see [Parameter groups for Amazon RDS](#).

When you have enabled or disabled InnoDB file-per-table tablespaces, you can issue an ALTER TABLE command to move a table from the global tablespace to its own tablespace, or from its own tablespace to the global tablespace as shown in the following example:

```
ALTER TABLE table_name TABLESPACE=innodb_file_per_table;
```

MySQL Keyring Plugin not supported

Currently, Amazon RDS for MySQL doesn't support the MySQL keyring_aws Amazon Web Services Keyring Plugin.

Custom ports

Amazon RDS blocks connections to custom port 33060 for the MySQL engine. Choose a different port for your MySQL engine.

MySQL stored procedure limitations

The [mysql.rds_kill](#) and [mysql.rds_kill_query](#) stored procedures can't terminate sessions or queries owned by MySQL users with usernames longer than 16 characters on the following RDS for MySQL versions:

- 8.0.32 and lower 8 versions
- 5.7.41 and lower 5.7 versions

GTID-based replication with an external source instance

Amazon RDS supports replication based on global transaction identifiers (GTIDs) from an external MySQL instance into an Amazon RDS for MySQL DB instance that requires setting GTID_PURGED during configuration. However, only RDS for MySQL 8.0.37 and higher versions support this functionality.

MySQL default authentication plugin

RDS for MySQL version 8.0.34 and higher 8.0 versions use the mysql_native_password plugin. You can't change the default_authentication_plugin setting.

RDS for MySQL version 8.4 and higher versions use the caching_sha2_password plugin as the default authentication plugin. You can change the default authentication plugin for MySQL 8.4. The mysql_native_password plugin still works with MySQL 8.4, but support of this plugin ends with MySQL 8.4. To change the default authentication plugin, create a custom parameter group and modify the value of the authentication_policy parameter. For more information, see [the section called "Default and custom parameter groups"](#).

Overriding innodb_buffer_pool_size

With micro or small DB instance classes, the default value for the innodb_buffer_pool_size parameter might differ from the value returned by running the following command:

```
mysql> SELECT @@innodb_buffer_pool_size;
```

This difference can occur when Amazon RDS needs to override the default value as part of managing the DB instance classes. If necessary, you can override the default value and set it to a value that your DB instance class supports. To determine a valid value, add the memory usage and the total memory available on your DB instance. For more information, see [Amazon RDS instance types](#).

If your DB instance has only 4 GB of memory, you can't set innodb_buffer_pool_size to 8 GB but you might be able to set it to 3 GB, depending on how much memory you allocated for other parameters.

If the value that you input is too large, Amazon RDS lowers the value to the following limits:

- Micro DB instance classes: 256 MB

- db.t4g.micro DB instance classes: 128 MB

Upgrading from MySQL 5.7 to MySQL 8.4

You can't upgrade directly from MySQL 5.7 to MySQL 8.4. You must first upgrade from MySQL 5.7 to MySQL 8.0, and then upgrade from MySQL 8.0 to MySQL 8.4. For more information, see [Major version upgrades for RDS for MySQL](#).

InnoDB page compression

InnoDB page compression doesn't work with Amazon RDS DB instances that have a file system block size of 16k because the file system block size must be smaller than the InnoDB page size. Starting in February 2024, all newly created DB instances have a file system block size of 16k, which increases throughput and decreases IOPS consumption during page flushes.

RDS for MySQL stored procedure reference

These topics describe system stored procedures that are available for Amazon RDS instances running the MySQL DB engine. The master user must run these procedures.

Topics

- [Collecting and maintaining the Global Status History](#)
- [Configuring, starting, and stopping binary log \(binlog\) replication](#)
- [Ending a session or query](#)
- [Managing active-active clusters](#)
- [Managing multi-source replication](#)
- [Replicating transactions using GTIDs](#)
- [Rotating the query logs](#)
- [Setting and showing binary log configuration](#)
- [Warming the InnoDB cache](#)

Collecting and maintaining the Global Status History

Amazon RDS provides a set of procedures that take snapshots of the values of status variables over time and write them to a table, along with any changes since the last snapshot. This infrastructure is called Global Status History. For more information, see [Managing the Global Status History for RDS for MySQL](#).

The following stored procedures manage how the Global Status History is collected and maintained.

Topics

- [mysql.rds_collect_global_status_history](#)
- [mysql.rds_disable_gsh_collector](#)
- [mysql.rds_disable_gsh_rotation](#)
- [mysql.rds_enable_gsh_collector](#)
- [mysql.rds_enable_gsh_rotation](#)
- [mysql.rds_rotate_global_status_history](#)
- [mysql.rds_set_gsh_collector](#)
- [mysql.rds_set_gsh_rotation](#)

mysql.rds_collect_global_status_history

Takes a snapshot on demand for the Global Status History.

Syntax

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

Turns off snapshots taken by the Global Status History.

Syntax

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

Turns off rotation of the mysql.global_status_history table.

Syntax

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

Turns on the Global Status History to take default snapshots at intervals specified by rds_set_gsh_collector.

Syntax

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

Turns on rotation of the contents of the mysql.global_status_history table to mysql.global_status_history_old at intervals specified by rds_set_gsh_rotation.

Syntax

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

Rotates the contents of the mysql.global_status_history table to mysql.global_status_history_old on demand.

Syntax

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

Specifies the interval, in minutes, between snapshots taken by the Global Status History.

Syntax

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

Parameters

intervalPeriod

The interval, in minutes, between snapshots. Default value is 5.

mysql.rds_set_gsh_rotation

Specifies the interval, in days, between rotations of the mysql.global_status_history table.

Syntax

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

Parameters

intervalPeriod

The interval, in days, between table rotations. Default value is 7.

Configuring, starting, and stopping binary log (binlog) replication

The following stored procedures control how transactions are replicated from an external database into RDS for MySQL, or from RDS for MySQL to an external database.

When using these stored procedures to manage replication with a replication user configured with `caching_sha2_password`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4. For more information, see [Encrypting with SSL/TLS](#).

For information about configuring, using, and managing read replicas, see [the section called "MySQL read replicas"](#).

Topics

- [mysql.rds_next_master_log \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_next_source_log \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_reset_external_master \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_reset_external_source \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_set_external_master \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_external_source \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_set_external_master_with_auto_position \(RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_external_source_with_auto_position \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_set_external_master_with_delay \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_external_source_with_delay \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_set_external_source_gtid_purged](#)
- [mysql.rds_set_master_auto_position \(RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_source_auto_position \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_set_source_delay](#)
- [mysql.rds_skip_repl_error](#)

- [mysql.rds_start_replication](#)
- [mysql.rds_start_replication_until](#)
- [mysql.rds_stop_replication](#)

mysql.rds_next_master_log (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Changes the source database instance log position to the start of the next binary log on the source database instance. Use this procedure only if you are receiving replication I/O error 1236 on a read replica.

Syntax

```
CALL mysql.rds_next_master_log(  
    curr_master_log  
)
```

Parameters

curr_master_log

The index of the current master log file. For example, if the current file is named mysql-bin-changelog.012345, then the index is 12345. To determine the current master log file name, run the `SHOW REPLICA STATUS` command and view the `Master_Log_File` field.

Usage notes

The master user must run the `mysql.rds_next_master_log` procedure.

Warning

Call `mysql.rds_next_master_log` only if replication fails after a failover of a Multi-AZ DB instance that is the replication source, and the `Last_IO_Error` field of `SHOW REPLICA STATUS` reports I/O error 1236.

Calling `mysql.rds_next_master_log` can result in data loss in the read replica if transactions in the source instance were not written to the binary log on disk before the failover event occurred. You can reduce the chance of this happening by setting the source

instance parameters sync_binlog and innodb_support_xa to 1, even though this might reduce performance. For more information, see [Troubleshooting a MySQL read replica problem](#).

Examples

Assume replication fails on an RDS for MySQL read replica. Running SHOW REPLICAS STATUS\G on the read replica returns the following result:

```
***** 1. row *****
Replica_IO_State:
    Source_Host: myhost.XXXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
    Source_User: MasterUser
    Source_Port: 3306
    Connect_Retry: 10
    Source_Log_File: mysql-bin-changelog.012345
    Read_Source_Log_Pos: 1219393
    Relay_Log_File: relaylog.012340
    Relay_Log_Pos: 30223388
    Relay_Source_Log_File: mysql-bin-changelog.012345
    Replica_IO_Running: No
    Replica_SQL_Running: Yes
    Replicate_Do_DB:
    Replicate_Ignore_DB:
    Replicate_Do_Table:
    Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
        Last_Error:
        Skip_Counter: 0
        Exec_Source_Log_Pos: 30223232
        Relay_Log_Space: 5248928866
        Until_Condition: None
        Until_Log_File:
        Until_Log_Pos: 0
    Source_SSL_Allowed: No
    Source_SSL_CA_File:
    Source_SSL_CA_Path:
        Source_SSL_Cert:
    Source_SSL_Cipher:
        Source_SSL_Key:
```

```
Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
    Last_IO_Errorno: 1236
        Last_IO_Error: Got fatal error 1236 from master when reading data from
binary log: 'Client requested master to start replication from impossible position;
the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
'/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
    Last_SQL_Errorno: 0
    Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Source_Server_Id: 67285976
```

The `Last_IO_Errorno` field shows that the instance is receiving I/O error 1236. The `Master_Log_File` field shows that the file name is `mysql-bin-changelog.012345`, which means that the log file index is 12345. To resolve the error, you can call `mysql.rds_next_master_log` with the following parameter:

```
CALL mysql.rds_next_master_log(12345);
```

mysql.rds_next_source_log (RDS for MySQL major versions 8.4 and higher)

Changes the source database instance log position to the start of the next binary log on the source database instance. Use this procedure only if you are receiving replication I/O error 1236 on a read replica.

Syntax

```
CALL mysql.rds_next_source_log(
    curr_source_log
);
```

Parameters

curr_source_log

The index of the current source log file. For example, if the current file is named `mysql-bin-changelog.012345`, then the index is 12345. To determine the current source log file name, run the `SHOW REPLICA STATUS` command and view the `Source_Log_File` field.

Usage notes

The administrative user must run the `mysql.rds_next_source_log` procedure.

Warning

Call `mysql.rds_next_source_log` only if replication fails after a failover of a Multi-AZ DB instance that is the replication source, and the `Last_IO_Error` field of `SHOW REPLICAS STATUS` reports I/O error 1236.

Calling `mysql.rds_next_source_log` can result in data loss in the read replica if transactions in the source instance were not written to the binary log on disk before the failover event occurred. You can reduce the chance of this happening by setting the source instance parameters `sync_binlog` and `innodb_support_xa` to 1, even though this might reduce performance. For more information, see [Troubleshooting a MySQL read replica problem](#).

Examples

Assume replication fails on an RDS for MySQL read replica. Running `SHOW REPLICAS STATUS\G` on the read replica returns the following result:

```
***** 1. row *****
Replica_IO_State:
  Source_Host: myhost.XXXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
  Source_User: MasterUser
  Source_Port: 3306
  Connect_Retry: 10
  Source_Log_File: mysql-bin-changelog.012345
  Read_Source_Log_Pos: 1219393
  Relay_Log_File: relaylog.012340
  Relay_Log_Pos: 30223388
Relay_Source_Log_File: mysql-bin-changelog.012345
  Replica_IO_Running: No
  Replica_SQL_Running: Yes
  Replicate_Do_DB:
  Replicate_Ignore_DB:
  Replicate_Do_Table:
  Replicate_Ignore_Table:
  Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
```

```
        Last_Error:
        Skip_Counter: 0
    Exec_Source_Log_Pos: 30223232
        Relay_Log_Space: 5248928866
        Until_Condition: None
        Until_Log_File:
        Until_Log_Pos: 0
    Source_SSL_Allowed: No
    Source_SSL_CA_File:
    Source_SSL_CA_Path:
        Source_SSL_Cert:
    Source_SSL_Cipher:
        Source_SSL_Key:
    Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
        Last_IO_Errno: 1236
        Last_IO_Error: Got fatal error 1236 from source when reading data from
binary log: 'Client requested source to start replication from impossible position;
the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
'/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
        Last_SQL_Errno: 0
        Last_SQL_Error:
Replicate_Ignore_Server_Ids:
    Source_Server_Id: 67285976
```

The `Last_IO_Errno` field shows that the instance is receiving I/O error 1236. The `Source_Log_File` field shows that the file name is `mysql-bin-changelog.012345`, which means that the log file index is 12345. To resolve the error, you can call `mysql.rds_next_source_log` with the following parameter:

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_reset_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Reconfigures an RDS for MySQL DB instance to no longer be a read replica of an instance of MySQL running external to Amazon RDS.

Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_reset_external_master;
```

Usage notes

The master user must run the `mysql.rds_reset_external_master` procedure. This procedure must be run on the MySQL DB instance to be removed as a read replica of a MySQL instance running external to Amazon RDS.

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#).

For more information about using replication to import data from an instance of MySQL running external to Amazon RDS, see [Configuring binary log file position replication with an external source instance](#).

`mysql.rds_reset_external_source` (RDS for MySQL major versions 8.4 and higher)

Reconfigures an RDS for MySQL DB instance to no longer be a read replica of an instance of MySQL running external to Amazon RDS.

Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_reset_external_source;
```

Usage notes

The administrative user must run the `mysql.rds_reset_external_source` procedure. This procedure must be run on the MySQL DB instance to be removed as a read replica of a MySQL instance running external to Amazon RDS.

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS.

For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#). For more information about using replication to import data from an instance of MySQL running external to Amazon RDS, see [Configuring binary log file position replication with an external source instance](#).

`mysql.rds_set_external_master` (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Note

You can use the [mysql.rds_set_external_master_with_delay \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#) stored procedure to configure an external source database instance and delayed replication.

Syntax

```
CALL mysql.rds_set_external_master (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , mysql_binary_log_file_name
    , mysql_binary_log_file_location
    , ssl_encryption
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source database instance that contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication starts reading the replication information.

You can determine the binlog file name and location by running `SHOW MASTER STATUS` on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

 **Note**

The `MASTER_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

Usage notes

The master user must run the `mysql.rds_set_external_master` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_master`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify `replication_user_name` and

replication_user_password values that indicate a replication user that has REPLICATION CLIENT and REPLICATION SLAVE permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

MySQL 5.7

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

MySQL 8.0

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY 'password';
```

Note

Specify a password other than the prompt shown here as a security best practice.

2. On the external instance of MySQL, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. The following example grants REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the 'repl_user' user for your domain.

MySQL 5.7

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY 'password';
```

MySQL 8.0

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

To use encrypted replication, configure source database instance to use SSL connections.

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#).

After calling `mysql.rds_set_external_master` to configure an Amazon RDS DB instance as a read replica, you can call [`mysql.rds_start_replication`](#) on the read replica to start the replication process. You can call [`mysql.rds_reset_external_master` \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#) to remove the read replica configuration.

When `mysql.rds_set_external_master` is called, Amazon RDS records the time, user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

`mysql.rds_set_external_source` (RDS for MySQL major versions 8.4 and higher)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

⚠ Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_source (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , mysql_binary_log_file_name
    , mysql_binary_log_file_location
    , ssl_encryption
);

```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in *replication_user_name*.

mysql_binary_log_file_name

The name of the binary log on the source database instance that contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication starts reading the replication information.

You can determine the binlog file name and location by running `SHOW MASTER STATUS` on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

Note

The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

Usage notes

The administrative user must run the `mysql.rds_set_external_source` procedure. This procedure must be run on the RDS for MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_source`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify `replication_user_name` and `replication_user_password` values that indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

Note

Specify a password other than the prompt shown here as a security best practice.

2. On the external instance of MySQL, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. The following example grants REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the 'repl_user' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

To use encrypted replication, configure source database instance to use SSL connections. Also, import the certificate authority certificate, client certificate, and client key into the DB instance or DB cluster using the [mysql.rds_import_binlog_ssl_material](#) procedure.

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#).

After calling `mysql.rds_set_external_source` to configure an RDS for MySQL DB instance as a read replica, you can call [mysql.rds_start_replication](#) on the read replica to start the replication process. You can call [mysql.rds_reset_external_source \(RDS for MySQL major versions 8.4 and higher\)](#) to remove the read replica configuration.

When `mysql.rds_set_external_source` is called, Amazon RDS records the time, user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

Examples

When run on an RDS for MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

```
call mysql.rds_set_external_source(
    'Externaldb.some.com',
    3306,
    'repl_user',
    'password',
    'mysql-bin-changelog.0777',
    120,
    1);
```

mysql.rds_set_external_master_with_auto_position (RDS for MySQL major versions 8.0 and lower)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. This procedure also configures delayed replication and replication based on global transaction identifiers (GTIDs).

⚠ Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_master_with_auto_position (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , ssl_encryption
    , delay
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

 **Note**

The `MASTER_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86,400 seconds).

Usage notes

The master user must run the `mysql.rds_set_external_master_with_auto_position` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

This procedure is supported for all RDS for MySQL 5.7 versions, and RDS for MySQL 8.0.26 and higher 8.0 versions.

Before you run `mysql.rds_set_external_master_with_auto_position`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the '`repl_user`' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY 'SomePassW0rd'
```

For more information, see [Configuring binary log file position replication with an external source instance](#).

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored

procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#).

Before you call `mysql.rds_set_external_master_with_auto_position`, make sure to call [the section called “mysql.rds_set_external_source_gtid_purged”](#) to set the `gtid_purged` system variable with a specified GTID range from an external source.

After calling `mysql.rds_set_external_master_with_auto_position` to configure an Amazon RDS DB instance as a read replica, you can call [mysql.rds_start_replication](#) on the read replica to start the replication process. You can call [mysql.rds_reset_external_master](#) ([RDS for MariaDB](#) and [RDS for MySQL major versions 8.0 and lower](#)) to remove the read replica configuration.

When you call `mysql.rds_set_external_master_with_auto_position`, Amazon RDS records the time, the user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the [mysql.rds_start_replication_until](#) or [mysql.rds_start_replication_until_gtid](#) stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_master_with_auto_position` procedure. After the `mysql.rds_start_replication_until_gtid` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

To use the `mysql.rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the [mysql.rds_skip_transaction_with_gtid](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication](#).

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_master_with_auto_position(
    'Externaldb.some.com',
    3306,
    'repl_user',
    'SomePassW0rd',
    1,
    3600);
```

mysql.rds_set_external_source_with_auto_position (RDS for MySQL major versions 8.4 and higher)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. This procedure also configures delayed replication and replication based on global transaction identifiers (GTIDs).

Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_source_with_auto_position (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , ssl_encryption
    , delay
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

Note

The SOURCE_SSL_VERIFY_SERVER_CERT option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86,400 seconds).

Usage notes

The administrative user must run the

`mysql.rds_set_external_source_with_auto_position` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_source_with_auto_position`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify

values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has REPLICATION CLIENT and REPLICATION SLAVE permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external instance of MySQL, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. The following example grants REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the '`repl_user`' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY 'SomePassW0rd'
```

For more information, see [Configuring binary log file position replication with an external source instance](#).

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#).

Before you call `mysql.rds_set_external_source_with_auto_position`, make sure to call [the section called “mysql.rds_set_external_source_gtid_purged”](#) to set the `gtid_purged` system variable with a specified GTID range from an external source.

After calling `mysql.rds_set_external_source_with_auto_position` to configure an Amazon RDS DB instance as a read replica, you can call [mysql.rds_start_replication](#) on the read

replica to start the replication process. You can call [mysql.rds_reset_external_source \(RDS for MySQL major versions 8.4 and higher\)](#) to remove the read replica configuration.

When you call `mysql.rds_set_external_source_with_auto_position`, Amazon RDS records the time, the user, and an action of set `master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the [mysql.rds_start_replication_until](#) or [mysql.rds_start_replication_until_gtid](#) stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_source_with_auto_position` procedure. After the `mysql.rds_start_replication_until_gtid` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

To use the `mysql.rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the [mysql.rds_skip_transaction_with_gtid](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication](#).

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_source_with_auto_position(
    'Externaldb.some.com',
    3306,
    'repl_user',
    'SomePassW0rd',
    1,
    3600);
```

mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS and configures delayed replication.

Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_master_with_delay(  
    host_name  
    , host_port  
    , replication_user_name  
    , replication_user_password  
    , mysql_binary_log_file_name  
    , mysql_binary_log_file_location  
    , ssl_encryption  
    , delay  
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS that will become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes SSH port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source database instance contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication will start reading the replication information.

You can determine the binlog file name and location by running `SHOW MASTER STATUS` on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

Note

The `MASTER_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86400 seconds).

Usage notes

The master user must run the `mysql.rds_set_external_master_with_delay` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_master_with_delay`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the '`repl_user`' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY 'SomePassW0rd'
```

For more information, see [Configuring binary log file position replication with an external source instance](#).

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to

Amazon RDS. For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#).

After calling `mysql.rds_set_external_master_with_delay` to configure an Amazon RDS DB instance as a read replica, you can call [mysql.rds_start_replication](#) on the read replica to start the replication process. You can call [mysql.rds_reset_external_master \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#) to remove the read replica configuration.

When you call `mysql.rds_set_external_master_with_delay`, Amazon RDS records the time, the user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the [mysql.rds_start_replication_until](#) or [mysql.rds_start_replication_until_gtid](#) stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_master_with_delay` procedure. After the `mysql.rds_start_replication_until` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

To use the `mysql.rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the [mysql.rds_skip_transaction_with_gtid](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication](#).

The `mysql.rds_set_external_master_with_delay` procedure is available in these versions of RDS for MySQL:

- MySQL 8.0.26 and higher 8.0 versions
- All 5.7 versions

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_master_with_delay(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'SomePassW0rd',
  'mysql-bin-changelog.000777',
  120,
  1,
  3600);
```

mysql.rds_set_external_source_with_delay (RDS for MySQL major versions 8.4 and higher)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS and configures delayed replication.

⚠ Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_source_with_delay (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
  , delay
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS that will become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes SSH port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source database instance contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication will start reading the replication information.

You can determine the binlog file name and location by running `SHOW MASTER STATUS` on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

Note

The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86400 seconds).

Usage notes

The administrative user must run the `mysql.rds_set_external_source_with_delay` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_source_with_delay`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has REPLICATION CLIENT and REPLICATION SLAVE permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external instance of MySQL, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. The following example grants REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the '`repl_user`' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY 'SomePassW0rd'
```

For more information, see [Configuring binary log file position replication with an external source instance](#).

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see [Working with DB instance read replicas](#).

After calling `mysql.rds_set_external_source_with_delay` to configure an Amazon RDS DB instance as a read replica, you can call [`mysql.rds_start_replication`](#) on the read replica to start the replication process. You can call [`mysql.rds_reset_external_source` \(RDS for MySQL major versions 8.4 and higher\)](#) to remove the read replica configuration.

When you call `mysql.rds_set_external_source_with_delay`, Amazon RDS records the time, the user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the [`mysql.rds_start_replication_until`](#) or [`mysql.rds_start_replication_until_gtid`](#) stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_source_with_delay` procedure. After the `mysql.rds_start_replication_until` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

To use the `mysql.rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the [`mysql.rds_skip_transaction_with_gtid`](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication](#).

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source

database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_source_with_delay(  
    'Externaldb.some.com',  
    3306,  
    'repl_user',  
    'SomePassW0rd',  
    'mysql-bin-changelog.000777',  
    120,  
    1,  
    3600);
```

mysql.rds_set_external_source_gtid_purged

Sets the [gtid_purged](#) system variable with a specified GTID range from an external source. The gtid_purged value is required for configuring GTID-based replication to resume the replication using auto positioning.

⚠ Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_source_gtid_purged(  
    server_uuid  
    , start_pos  
    , end_pos  
);
```

Parameters

server_uuid

The universally unique identifier (UUID) of the external server from which the GTID range is being imported.

start_pos

The starting position of the GTID range to be set.

end_pos

The ending position of the GTID range to be set.

Usage notes

The `mysql.rds_set_external_source_gtid_purged` procedure is only available with MySQL 8.0.37 and higher 8.0 versions.

Call `mysql.rds_set_external_source_gtid_purged` before you call

[`mysql.rds_set_external_master_with_auto_position` \(RDS for MySQL major versions 8.0 and lower\)](#),

[`mysql.rds_set_external_source_with_auto_position` \(RDS for MySQL major versions 8.4 and higher\)](#),

or [`mysql.rds_set_external_source_with_auto_position_for_channel`](#).

Before you call `mysql.rds_set_external_source_gtid_purged`, make sure to stop all active replication channels for the database. To check the status of a channel, use the `SHOW REPLICAS STATUS` MySQL statement. To stop replication on a channel, call [the section called "mysql.rds_stop_replication_for_channel"](#).

The GTID range that you specify must be a superset of the existing `GTID_PURGED` value. This stored procedure checks the following values before it sets the `GTID_PURGED` value:

- The `server_uuid` is valid.
- The value of `start_pos` is greater than 0 and less than the value of `end_pos`.
- The value of `end_pos` is greater than or equal to the value of `start_pos`.

If the GTID set on your external server contains multiple ranges of values, consider calling the procedure multiple times with different GTID set values.

When you call `mysql.rds_set_external_source_gtid_purged`, Amazon RDS records the time, the user, and an action of `set_gtid_purged` in the `mysql.rds_history` table.

If you don't set the `gtid_purged` value appropriately for the backup that you use for replication, this can result in missing or duplicated transactions during the replication process. Perform the following steps to set the correct `gtid_purged` value.

To set the gtid_purged value on the replica

1. Determine the point in time or the specific backup file to use as the starting point for replication. This could be a logical backup (a mysqldump file) or a physical backup (an Amazon RDS snapshot).
2. Determine the gtid_executed value. This value represents the set of all GTIDs that were committed on the server. To retrieve this value, on the source instance, do one of the following:
 - Run the SQL statement `SELECT @@GLOBAL.GTID_EXECUTED;` at the time the backup was taken.
 - If any related options are included in the respective backup utility, extract the value from the backup file. For more information, see the [set-gtid-purged](#) option in the MySQL documentation.
3. Determine the gtid_purged value to use for the call to `mysql.rds_set_external_source_gtid_purged`. The gtid_purged value should include all the GTIDs that were executed on the source instance and are no longer needed for replication. Therefore, the gtid_purged value should be a subset of the gtid_executed value that you retrieved in the previous step.

To determine the gtid_purged value, identify the GTIDs that aren't included in the backup and are no longer needed for replication. You can do so by analyzing the binary logs or by using a tool such as mysqlbinlog to find the GTIDs that were purged from the binary logs.

Alternatively, if you have a consistent backup that includes all of the binary logs up to the backup point, you can set the gtid_purged value to be the same as the gtid_executed value at the backup point.

4. After you determine the appropriate gtid_purged value that's consistent with your backup, call the `mysql.rds_set_external_source_gtid_purged` stored procedure on your RDS for MySQL DB instance to set the value.

Examples

When run on a MySQL DB instance, the following example sets the GTID range from an external MySQL server with the UUID 12345678-abcd-1234-efgh-123456789abc, a starting position of 1, and an ending position of 100. The resulting GTID value is set to +12345678-abcd-1234-efgh-123456789abc:1-100.

```
CALL mysql.rds_set_external_source_gtid_purged('12345678-abcd-1234-efgh-123456789abc',  
    1, 100);
```

mysql.rds_set_master_auto_position (RDS for MySQL major versions 8.0 and lower)

Sets the replication mode to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_master_auto_position (  
    auto_position_mode  
);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- 0 – Use the replication method based on binary log file position. The default is 0.
- 1 – Use the GTID-based replication method.

Usage notes

The master user must run the `mysql.rds_set_master_auto_position` procedure.

This procedure is supported for all RDS for MySQL 5.7 versions and RDS for MySQL 8.0.26 and higher 8.0 versions.

mysql.rds_set_source_auto_position (RDS for MySQL major versions 8.4 and higher)

Sets the replication mode to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- 0 – Use the replication method based on binary log file position. The default is 0.
- 1 – Use the GTID-based replication method.

Usage notes

The administrative user must run the `mysql.rds_set_source_auto_position` procedure.

`mysql.rds_set_source_delay`

Sets the minimum number of seconds to delay replication from source database instance to the current read replica. Use this procedure when you are connected to a read replica to delay replication from its source database instance.

Syntax

```
CALL mysql.rds_set_source_delay(  
    delay  
);
```

Parameters

delay

The minimum number of seconds to delay replication from the source database instance.

The limit for this parameter is one day (86400 seconds).

Usage notes

The master user must run the `mysql.rds_set_source_delay` procedure.

For disaster recovery, you can use this procedure with the [mysql.rds_start_replication_until](#) stored procedure or the [mysql.rds_start_replication_until_gtid](#) stored procedure.

To roll forward changes to a delayed read replica to the time just before a

disaster, you can run the `mysql.rds_set_source_delay` procedure. After the `mysql.rds_start_replication_until` or `mysql.rds_start_replication_until_gtid` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

To use the `mysql.rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the [`mysql.rds_skip_transaction_with_gtid`](#) stored procedure. For more information on GTID-based replication, see [Using GTID-based replication](#).

The `mysql.rds_set_source_delay` procedure is available in these versions of RDS for MySQL:

- All RDS for MySQL 8.4 versions
- MySQL 8.0.26 and higher 8.0 versions
- All 5.7 versions

Examples

To delay replication from source database instance to the current read replica for at least one hour (3,600 seconds), you can call `mysql.rds_set_source_delay` with the following parameter:

```
CALL mysql.rds_set_source_delay(3600);
```

`mysql.rds_skip_repl_error`

Skips and deletes a replication error on a MySQL DB read replica.

Syntax

```
CALL mysql.rds_skip_repl_error;
```

Usage notes

The master user must run the `mysql.rds_skip_repl_error` procedure on a read replica. For more information about this procedure, see [Calling the `mysql.rds_skip_repl_error` procedure](#).

To determine if there are errors, run the MySQL `SHOW REPLICAS STATUS\G` command. If a replication error isn't critical, you can run `mysql.rds_skip_repl_error` to skip the error. If

there are multiple errors, `mysql.rds_skip_repl_error` deletes the first error, then warns that others are present. You can then use `SHOW REPLICA STATUS\G` to determine the correct course of action for the next error. For information about the values returned, see [SHOW REPLICA STATUS statement](#) in the MySQL documentation.

For more information about addressing replication errors with Amazon RDS, see [Troubleshooting a MySQL read replica problem](#).

Replication stopped error

When you call the `mysql.rds_skip_repl_error` procedure, you might receive an error message stating that the replica is down or disabled.

This error message appears if you run the procedure on the primary instance instead of the read replica. You must run this procedure on the read replica for the procedure to work.

This error message might also appear if you run the procedure on the read replica, but replication can't be restarted successfully.

If you need to skip a large number of errors, the replication lag can increase beyond the default retention period for binary log (binlog) files. In this case, you might encounter a fatal error because of binlog files being purged before they have been replayed on the read replica. This purge causes replication to stop, and you can no longer call the `mysql.rds_skip_repl_error` command to skip replication errors.

You can mitigate this issue by increasing the number of hours that binlog files are retained on your source database instance. After you have increased the binlog retention time, you can restart replication and call the `mysql.rds_skip_repl_error` command as needed.

To set the binlog retention time, use the [mysql.rds_set_configuration](#) procedure and specify a configuration parameter of '`binlog retention hours`' along with the number of hours to retain binlog files on the DB cluster. The following example sets the retention period for binlog files to 48 hours.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

Initiates replication from an RDS for MySQL DB instance.

Note

You can use the [mysql.rds_start_replication_until](#) or [mysql.rds_start_replication_until_gtid](#) stored procedure to initiate replication from an RDS for MySQL DB instance and stop replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication;
```

Usage notes

The master user must run the `mysql.rds_start_replication` procedure.

To import data from an instance of MySQL external to Amazon RDS, call `mysql.rds_start_replication` on the read replica to start the replication process after you call [mysql.rds_set_external_master \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#) or [mysql.rds_set_external_source \(RDS for MySQL major versions 8.4 and higher\)](#) to build the replication configuration. For more information, see [Restoring a backup into an Amazon RDS for MySQL DB instance](#).

To export data to an instance of MySQL external to Amazon RDS, call `mysql.rds_start_replication` and `mysql.rds_stop_replication` on the read replica to control some replication actions, such as purging binary logs. For more information, see [Exporting data from a MySQL DB instance by using replication](#).

You can also call `mysql.rds_start_replication` on the read replica to restart any replication process that you previously stopped by calling `mysql.rds_stop_replication`. For more information, see [Working with DB instance read replicas](#).

mysql.rds_start_replication_until

Initiates replication from an RDS for MySQL DB instance and stops replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication_until (
  replication_log_file
  , replication_stop_point
);
```

Parameters

replication_log_file

The name of the binary log on the source database instance that contains the replication information.

replication_stop_point

The location in the `replication_log_file` binary log at which replication will stop.

Usage notes

The master user must run the `mysql.rds_start_replication_until` procedure.

The `mysql.rds_start_replication_until` procedure is available in these versions of RDS for MySQL:

- All RDS for MySQL 8.4 versions
- MySQL 8.0.26 and higher 8.0 versions
- All 5.7 versions

You can use this procedure with delayed replication for disaster recovery. If you have delayed replication configured, you can use this procedure to roll forward changes to a delayed read replica to the time just before a disaster. After this procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

You can configure delayed replication using the following stored procedures:

- [mysql.rds_set_configuration](#)
- [mysql.rds_set_external_master_with_delay \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_external_source_with_delay \(RDS for MySQL major versions 8.4 and higher\)](#)

- [mysql.rds_set_source_delay](#)

The file name specified for the `replication_log_file` parameter must match the source database instance binlog file name.

When the `replication_stop_point` parameter specifies a stop location that is in the past, replication is stopped immediately.

Examples

The following example initiates replication and replicates changes until it reaches location 120 in the `mysql-bin-changelog.000777` binary log file.

```
call mysql.rds_start_replication_until(
    'mysql-bin-changelog.000777',
    120);
```

mysql.rds_stop_replication

Stops replication from a MySQL DB instance.

Syntax

```
CALL mysql.rds_stop_replication;
```

Usage notes

The master user must run the `mysql.rds_stop_replication` procedure.

If you are configuring replication to import data from an instance of MySQL running external to Amazon RDS, you call `mysql.rds_stop_replication` on the read replica to stop the replication process after the import has completed. For more information, see [Restoring a backup into an Amazon RDS for MySQL DB instance](#).

If you are configuring replication to export data to an instance of MySQL external to Amazon RDS, you call `mysql.rds_start_replication` and `mysql.rds_stop_replication` on the read replica to control some replication actions, such as purging binary logs. For more information, see [Exporting data from a MySQL DB instance by using replication](#).

You can also use `mysql.rds_stop_replication` to stop replication between two Amazon RDS DB instances. You typically stop replication to perform a long running operation on the read replica, such as creating a large index on the read replica. You can restart any replication process that you stopped by calling [`mysql.rds_start_replication`](#) on the read replica. For more information, see [Working with DB instance read replicas](#).

Ending a session or query

The following stored procedures end a session or query.

Topics

- [mysql.rds_kill](#)
- [mysql.rds_kill_query](#)

mysql.rds_kill

Ends a connection to the MySQL server.

Syntax

```
CALL mysql.rds_kill(processID);
```

Parameters

processID

The identity of the connection thread to be ended.

Usage notes

Each connection to the MySQL server runs in a separate thread. To end a connection, use the `mysql.rds_kill` procedure and pass in the thread ID of that connection. To obtain the thread ID, use the MySQL [SHOW PROCESSLIST](#) command.

For information about limitations, see [MySQL stored procedure limitations](#).

Examples

The following example ends a connection with a thread ID of 4243:

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

Ends a query running against the MySQL server.

Syntax

```
CALL mysql.rds_kill_query(processID);
```

Parameters

processID

The identity of the process or thread that is running the query to be ended.

Usage notes

To stop a query running against the MySQL server, use the `mysql_rds_kill_query` procedure and pass in the connection ID of the thread that is running the query. The procedure then terminates the connection.

To obtain the ID, query the MySQL [INFORMATION_SCHEMA PROCESSLIST table](#) or use the MySQL [SHOW PROCESSLIST](#) command. The value in the ID column from `SHOW PROCESSLIST` or `SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST` is the *processID*.

For information about limitations, see [MySQL stored procedure limitations](#).

Examples

The following example stops a query with a query thread ID of 230040:

```
CALL mysql.rds_kill_query(230040);
```

Managing active-active clusters

The following stored procedures set up and manage RDS for MySQL active-active clusters. For more information, see [the section called “Configuring active-active clusters”](#).

These stored procedures are only available with RDS for MySQL DB instances running the following versions:

- All MySQL 8.4 versions
- MySQL 8.0.35 and higher minor versions

Topics

- [mysql.rds_group_replication_advance_gtid](#)
- [mysql.rds_group_replication_create_user](#)
- [mysql.rds_group_replication_set_recovery_channel](#)
- [mysql.rds_group_replication_start](#)
- [mysql.rds_group_replication_stop](#)

mysql.rds_group_replication_advance_gtid

Creates placeholder GTIDs on the current DB instance.

Syntax

```
CALL mysql.rds_group_replication_advance_gtid(  
  begin_id  
  , end_id  
  , server_uuid  
);
```

Parameters

begin_id

The start transaction ID to be created.

end_id

The end transaction ID to be created.

begin_id

The `group_replication_group_name` for the transaction to be created. The `group_replication_group_name` is specified as a UUID in the DB parameter group associated with the DB instance.

Usage notes

In an active-active cluster, for a DB instance to join a group, all GTID transactions executed on the new DB instance must exist on the other members in the cluster. In unusual cases, a new DB instance might have more transactions when transactions are executed before joining the instance to group. In this case, you can't remove any existing transactions, but you can use this procedure to create the corresponding placeholder GTIDs on the other DB instances in the group. Before doing so, verify that the transactions *don't affect the replicated data*.

When you call this procedure, GTID transactions of `server_uuid:begin_id-end_id` are created with empty content. To avoid replication issues, don't use this procedure under any other conditions.

Important

Avoid calling this procedure when the active-active cluster is functioning normally. Don't call this procedure unless you understand the possible consequences of the transactions you are creating. Calling this procedure might result in inconsistent data.

Example

The following example creates placeholder GTIDs on current DB instance.:

```
CALL mysql.rds_group_replication_advance_gtid(5, 6,
'11111111-2222-3333-4444-555555555555');
```

`mysql.rds_group_replication_create_user`

Creates the replication user `rdsgrpreadadmin` for group replication on the DB instance.

Syntax

```
CALL mysql.rds_group_replication_create_user(
```

```
replication_user_password  
);
```

Parameters

replication_user_password

The password of the replication user `rdsgrpadmin`.

Usage notes

- The password of the replication user `rdsgrpadmin` must be the same on all of the DB instances in an active-active cluster.
- The `rdsgrpadmin` user name is reserved for group replication connections. No other user, including the master user, can have this user name.

Example

The following example creates the replication user `rdsgrpadmin` for group replication on the DB instance:

```
CALL mysql.rds_group_replication_create_user('password');
```

mysql.rds_group_replication_set_recovery_channel

Sets the `group_replication_recovery` channel for an active-active cluster. The procedure uses the reserved `rdsgrpadmin` user to configure the channel.

Syntax

```
CALL mysql.rds_group_replication_set_recovery_channel(  
replication_user_password);
```

Parameters

replication_user_password

The password of the replication user `rdsgrpadmin`.

Usage notes

The password of the replication user `rdsgrpreadadmin` must be the same on all of the DB instances in an active-active cluster. A call to the `mysql.rds_group_replication_create_user` specifies the password.

Example

The following example sets the `group_replication_recovery` channel for an active-active cluster:

```
CALL mysql.rds_group_replication_set_recovery_channel('password');
```

mysql.rds_group_replication_start

Starts group replication on the current DB instance.

Syntax

```
CALL mysql.rds_group_replication_start(  
  bootstrap  
)
```

Parameters

bootstrap

A value that specifies whether to initialize a new group or join an existing group. `1` initializes a new group with the current DB instance. `0` joins the current DB instance to an existing group by connecting to the endpoints defined in `group_replication_group_seeds` parameter in the DB parameter group associated with the DB instance.

Example

The following example initializes a new group with the current DB instance:

```
CALL mysql.rds_group_replication_start(1);
```

mysql.rds_group_replication_stop

Stops group replication on the current DB instance.

Syntax

```
CALL mysql.rds_group_replication_stop();
```

Usage notes

When you stop replication on a DB instance, it doesn't affect any other DB instance in the active-active cluster.

Managing multi-source replication

The following stored procedures set up and manage replication channels on a RDS for MySQL multi-source replica. For more information, see [the section called “Configuring multi-source replication”](#).

These stored procedures are only available with RDS for MySQL DB instances running the following engine versions:

- All 8.4 versions
- 8.0.35 and higher minor versions
- 5.7.44 and higher minor versions

When using stored procedures to manage replication with a replication user configured with `caching_sha2_password`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4.

 **Note**

Although this documentation refers to source DB instances as RDS for MySQL DB instances, these procedures also work for MySQL instances running external to Amazon RDS.

Topics

- [mysql.rds_next_source_log_for_channel](#)
- [mysql.rds_reset_external_source_for_channel](#)
- [mysql.rds_set_external_source_for_channel](#)
- [mysql.rds_set_external_source_with_auto_position_for_channel](#)
- [mysql.rds_set_external_source_with_delay_for_channel](#)
- [mysql.rds_set_source_auto_position_for_channel](#)
- [mysql.rds_set_source_delay_for_channel](#)
- [mysql.rds_skip_repl_error_for_channel](#)
- [mysql.rds_start_replication_for_channel](#)
- [mysql.rds_start_replication_until_for_channel](#)
- [mysql.rds_start_replication_until_gtid_for_channel](#)

- [mysql.rds_stop_replication_for_channel](#)

mysql.rds_next_source_log_for_channel

Changes the source DB instance log position to the start of the next binary log on the source DB instance for the channel. Use this procedure only if you are receiving replication I/O error 1236 on a multi-source replica.

Syntax

```
CALL mysql.rds_next_source_log_for_channel(  
    curr_master_log,  
    channel_name  
);
```

Parameters

curr_master_log

The index of the current source log file. For example, if the current file is named mysql-bin-changelog.012345, then the index is 12345. To determine the current source log file name, run the SHOW REPLICA STATUS FOR CHANNEL '*channel_name*' command and view the Source_Log_File field.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_next_source_log_for_channel` procedure. If there is an IO_Thread error, for example, you can use this procedure to skip all the events in the current binary log file and resume the replication from the next binary log file for the channel specified in `channel_name`.

Example

Assume replication fails on a channel on a multi-source replica. Running SHOW REPLICAS STATUS FOR CHANNEL 'channel_1'\G on the multi-source replica returns the following result:

```
mysql> SHOW REPLICAS STATUS FOR CHANNEL 'channel_1'\G
***** 1. row *****
    Replica_IO_State: Waiting for source to send event
        Source_Host: myhost.XXXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
        Source_User: ReplicationUser
        Source_Port: 3306
        Connect_Retry: 60
        Source_Log_File: mysql-bin-changelog.012345
        Read_Source_Log_Pos: 1219393
        Relay_Log_File: replica-relay-bin.000003
        Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
        Replica_IO_Running: No
        Replica_SQL_Running: Yes
        Replicate_Do_DB: .
        .
        .
        Last_IO_Errno: 1236
        Last_IO_Error: Got fatal error 1236 from master when reading data from
binary log: 'Client requested master to start replication from impossible position;
the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
'/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
        Last_SQL_Errno: 0
        Last_SQL_Error:
        .
        .
        Channel_name: channel_1
        .

-- Some fields are omitted in this example output
```

The `Last_IO_Errno` field shows that the instance is receiving I/O error 1236. The `Source_Log_File` field shows that the file name is `mysql-bin-changelog.012345`, which means that the log file index is 12345. To resolve the error, you can call `mysql.rds_next_source_log_for_channel` with the following parameters:

```
CALL mysql.rds_next_source_log_for_channel(12345, 'channel_1');
```

mysql.rds_reset_external_source_for_channel

Stops the replication process on the specified channel, and removes the channel and associated configurations from the multi-source replica.

Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_reset_external_source_for_channel (channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_reset_external_source_for_channel` procedure. This procedure deletes all relay logs that belong to the channel being removed.

mysql.rds_set_external_source_for_channel

Configures a replication channel on an RDS for MySQL DB instance to replicate the data from another RDS for MySQL DB instance.

⚠ Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

ℹ Note

You can use the [the section called “mysql.rds_set_external_source_with_delay_for_channel”](#) stored procedure instead to configure this channel with delayed replication.

Syntax

```
CALL mysql.rds_set_external_source_for_channel (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , mysql_binary_log_file_name
    , mysql_binary_log_file_location
    , ssl_encryption
    , channel_name
);
```

Parameters

host_name

The host name or IP address of the RDS for MySQL source DB instance.

host_port

The port used by the RDS for MySQL source DB instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the RDS for MySQL source DB instance. We recommend that you provide an account that is used solely for replication with the source DB instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source DB instance that contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication starts reading the replication information.

You can determine the binlog file name and location by running `SHOW BINARY LOG STATUS` on the source DB instance.

Note

Previous versions of MySQL used `SHOW MASTER STATUS` instead of `SHOW BINARY LOG STATUS`. If you are using a MySQL version before 8.4, then use `SHOW MASTER STATUS`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

Note

The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

channel_name

The name of the replication channel. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_external_source_for_channel` procedure. This procedure must be run on the target RDS for MySQL DB instance on which you're creating the replication channel.

Before you run `mysql.rds_set_external_source_for_channel`, configure a replication user on the source DB instance with the privileges required for the multi-source replica. To connect the multi-source replica to the source DB instance, you must specify `replication_user_name` and `replication_user_password` values of a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the source DB instance.

To configure a replication user on the source DB instance

1. Using the MySQL client of your choice, connect to the source DB instance and create a user account to be used for replication. The following is an example.

 **Important**

As a security best practice, specify a password other than the placeholder value shown in the following examples.

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

2. On the source DB instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com';
```

To use encrypted replication, configure the source DB instance to use SSL connections.

After calling `mysql.rds_set_external_source_for_channel` to configure this replication channel, you can call [`mysql.rds_start_replication_for_channel`](#) on the replica to start the replication process on the channel. You can call [the section called "mysql.rds_reset_external_source_for_channel"](#) to stop replication on the channel and remove the channel configuration from the replica.

When you call `mysql.rds_set_external_source_for_channel`, Amazon RDS records the time, user, and an action of `set channel source` in the `mysql.rds_history` table without channel-specific details, and in the `mysql.rds_replication_status` table, with the channel name. This information is recorded only for internal usage and monitoring purposes. To record the complete procedure call for auditing purpose, consider enabling audit logs or general logs, based on the specific requirements of your application.

Examples

When run on a RDS for MySQL DB instance, the following example configures a replication channel named `channel_1` on this DB instance to replicate data from the source specified by host `sourcedb.example.com` and port `3306`.

```
call mysql.rds_set_external_source_for_channel(
    'sourcedb.example.com',
    3306,
    'repl_user',
    'password',
    'mysql-bin-changelog.0777',
    120,
    0,
    'channel_1');
```

`mysql.rds_set_external_source_with_auto_position_for_channel`

Configures a replication channel on an RDS for MySQL DB instance with an optional replication delay. The replication is based on global transaction identifiers (GTIDs).

Important

To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to `1`. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_source_with_auto_position_for_channel (
    host_name
    , host_port
```

```
, replication_user_name  
, replication_user_password  
, ssl_encryption  
, delay  
, channel_name  
);
```

Parameters

host_name

The host name or IP address of the RDS for MySQL source DB instance.

host_port

The port used by the RDS for MySQL source DB instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the RDS for MySQL source DB instance. We recommend that you provide an account that is used solely for replication with the source DB instance.

replication_user_password

The password of the user ID specified in *replication_user_name*.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

Note

The SOURCE_SSL_VERIFY_SERVER_CERT option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source DB instance.

The limit for this parameter is one day (86,400 seconds).

channel_name

The name of the replication channel. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the

`mysql.rds_set_external_source_with_auto_position_for_channel` procedure. This procedure must be run on the target RDS for MySQL DB instance on which you're creating the replication channel.

Before you run `rds_set_external_source_with_auto_position_for_channel`, configure a replication user on the source DB instance with the privileges required for the multi-source replica. To connect the multi-source replica to the source DB instance, you must specify `replication_user_name` and `replication_user_password` values of a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the source DB instance.

To configure a replication user on the source DB instance

1. Using the MySQL client of your choice, connect to the source DB instance and create a user account to be used for replication. The following is an example.

Important

As a security best practice, specify a password other than the placeholder value shown in the following examples.

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

2. On the source DB instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com';
```

To use encrypted replication, configure the source DB instance to use SSL connections.

Before you call `mysql.rds_set_external_source_with_auto_position_for_channel`, make sure to call [the section called “mysql.rds_set_external_source_gtid_purged”](#) to set the `gtid_purged` system variable with a specified GTID range from an external source.

After calling `mysql.rds_set_external_source_with_auto_position_for_channel` to configure an Amazon RDS DB instance as a read replica on a specific channel, you can call [the section called “mysql.rds_start_replication_for_channel”](#) on the read replica to start the replication process on that channel.

After calling `mysql.rds_set_external_source_with_auto_position_for_channel` to configure this replication channel, you can call [mysql.rds_start_replication_for_channel](#) on the replica to start the replication process on the channel. You can call [the section called “mysql.rds_reset_external_source_for_channel”](#) to stop replication on the channel and remove the channel configuration from the replica.

Examples

When run on a RDS for MySQL DB instance, the following example configures a replication channel named `channel_1_1` on this DB instance to replicate data from the source specified by host `sourcedb.example.com` and port `3306`. It sets the minimum replication delay to one hour (3,600 seconds). This means that a change from the source RDS for MySQL DB instance isn't applied on the multi-source replica for at least one hour.

```
call mysql.rds_set_external_source_with_auto_position_for_channel(
  'sourcedb.example.com',
  3306,
  'repl_user',
  'password',
  1,
  3600,
  'channel_1');
```

mysql.rds_set_external_source_with_delay_for_channel

Configures a replication channel on an RDS for MySQL DB instance with a specified replication delay.

⚠ Important

To run this procedure, autocommit must be enabled. To enable it, set the autocommit parameter to 1. For information about modifying parameters, see [Modifying parameters in a DB parameter group in Amazon RDS](#).

Syntax

```
CALL mysql.rds_set_external_source_with_delay_for_channel (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , mysql_binary_log_file_name
    , mysql_binary_log_file_location
    , ssl_encryption
    , delay
    , channel_name
);
```

Parameters

host_name

The host name or IP address of the RDS for MySQL source DB instance.

host_port

The port used by the RDS for MySQL source DB instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the RDS for MySQL source DB instance. We recommend that you provide an account that is used solely for replication with the source DB instance.

replication_user_password

The password of the user ID specified in *replication_user_name*.

mysql_binary_log_file_name

The name of the binary log on the source DB instance contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication will start reading the replication information.

You can determine the binlog file name and location by running `SHOW BINARY LOG STATUS` on the source database instance.

 **Note**

Previous versions of MySQL used `SHOW MASTER STATUS` instead of `SHOW BINARY LOG STATUS`. If you are using a MySQL version before 8.4, then use `SHOW MASTER STATUS`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

 **Note**

The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source DB instance.

The limit for this parameter is one day (86400 seconds).

channel_name

The name of the replication channel. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_external_source_with_delay_for_channel` procedure. This procedure must be run on the target RDS for MySQL DB instance on which you're creating the replication channel.

Before you run `mysql.rds_set_external_source_with_delay_for_channel`, configure a replication user on the source DB instance with the privileges required for the multi-source replica. To connect the multi-source replica to the source DB instance, you must specify `replication_user_name` and `replication_user_password` values of a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the source DB instance.

To configure a replication user on the source DB instance

1. Using the MySQL client of your choice, connect to the source DB instance and create a user account to be used for replication. The following is an example.

 **Important**

As a security best practice, specify a password other than the placeholder value shown in the following examples.

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

2. On the source DB instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com';
```

To use encrypted replication, configure the source DB instance to use SSL connections.

After calling `mysql.rds_set_external_source_with_delay_for_channel` to configure this replication channel, you can call [`mysql.rds_start_replication_for_channel`](#) on the replica to start the replication process on the channel. You can call [the section called "mysql.rds_reset_external_source_for_channel"](#) to stop replication on the channel and remove the channel configuration from the replica.

When you call `mysql.rds_set_external_source_with_delay_for_channel`, Amazon RDS records the time, user, and an action of `set_channel_source` in the `mysql.rds_history` table without channel-specific details, and in the `mysql.rds_replication_status` table, with the channel name. This information is recorded only for internal usage and monitoring purposes. To record the complete procedure call for auditing purpose, consider enabling audit logs or general logs, based on the specific requirements of your application.

Examples

When run on a RDS for MySQL DB instance, the following example configures a replication channel named `channel_1` on this DB instance to replicate data from the source specified by host `sourcedb.example.com` and port `3306`. It sets the minimum replication delay to one hour (3,600 seconds). This means that a change from the source RDS for MySQL DB instance isn't applied on the multi-source replica for at least one hour.

```
call mysql.rds_set_external_source_with_delay_for_channel(
  'sourcedb.example.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.000777',
  120,
  1,
  3600,
  'channel_1');
```

`mysql.rds_set_source_auto_position_for_channel`

Sets the replication mode for the specified channel to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_source_auto_position_for_channel (
  auto_position_mode
  , channel_name
);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- 0 – Use the replication method based on binary log file position. The default is 0.
- 1 – Use the GTID-based replication method.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_source_auto_position_for_channel` procedure. This procedure restarts replication on the specified channel to apply the specified auto position mode.

Examples

The following example sets the auto position mode for `channel_1` to use the GTID-based replication method.

```
call mysql.rds_set_source_auto_position_for_channel(1,'channel_1');
```

mysql.rds_set_source_delay_for_channel

Sets the minimum number of seconds to delay replication from the source database instance to the multi-source replica for the specified channel.

Syntax

```
CALL mysql.rds_set_source_delay_for_channel(delay, channel_name);
```

Parameters

delay

The minimum number of seconds to delay replication from the source DB instance.

The limit for this parameter is one day (86400 seconds).

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_source_delay_for_channel` procedure. To use this procedure, first call `mysql.rds_stop_replication_for_channel` to stop the replication. Then, call this procedure to set the replication delay value. When the delay is set, call `mysql.rds_start_replication_for_channel` to restart the replication.

Examples

The following example sets the delay for replication from the source database instance on `channel_1` of the multi-source replica for at least one hour (3,600 seconds).

```
CALL mysql.rds_set_source_delay_for_channel(3600, 'channel_1');
```

mysql.rds_skip_repl_error_for_channel

Skips a binary log event and deletes a replication error on a MySQL DB multi-source replica for the specified channel.

Syntax

```
CALL mysql.rds_skip_repl_error_for_channel(channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_skip_repl_error_for_channel` procedure on a read replica. You can use this procedure in a similar way `mysql.rds_skip_repl_error` is used to skip an error on a read replica. For more information, see [Calling the mysql.rds_skip_repl_error procedure](#).

Note

To skip errors in GTID-based replication, we recommend that you use the procedure [the section called "mysql.rds_skip_transaction_with_gtid"](#) instead.

To determine if there are errors, run the MySQL `SHOW REPLICAS STATUS FOR CHANNEL 'channel_name' \G` command. If a replication error isn't critical, you can run `mysql.rds_skip_repl_error_for_channel` to skip the error. If there are multiple errors, `mysql.rds_skip_repl_error_for_channel` deletes the first error on the specified replication channel, then warns that others are present. You can then use `SHOW REPLICAS STATUS FOR CHANNEL 'channel_name' \G` to determine the correct course of action for the next error. For information about the values returned, see [SHOW REPLICAS STATUS statement](#) in the MySQL documentation.

mysql.rds_start_replication_for_channel

Initiates replication from an RDS for MySQL DB instance to a multi-source replica on the specified channel.

Note

You can use the [mysql.rds_start_replication_until_for_channel](#) or [mysql.rds_start_replication_until_gtid_for_channel](#) stored procedure to initiate replication from an RDS for MySQL DB instance and stop replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication_for_channel(channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_start_replication_for_channel` procedure. After you import the data from the source RDS for MySQL DB instance, run this command on the multi-source replica to start replication on the specified channel.

Examples

The following example starts replication on `channel_1` of the multi-source replica.

```
CALL mysql.rds_start_replication_for_channel('channel_1');
```

mysql.rds_start_replication_until_for_channel

Initiates replication from an RDS for MySQL DB instance on the specified channel and stops replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication_until_for_channel (
  replication_log_file
  , replication_stop_point
  , channel_name
);
```

Parameters

replication_log_file

The name of the binary log on the source DB instance contains the replication information.

replication_stop_point

The location in the `replication_log_file` binary log at which replication will stop.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_start_replication_until_for_channel` procedure. With this procedure, replication starts and then stops when the specified binlog file position is reached. This procedure stops both the SQL_THREAD and IO_THREAD.

The file name specified for the `replication_log_file` parameter must match the source DB instance binlog file name.

When the `replication_stop_point` parameter specifies a stop location that's in the past, replication is stopped immediately.

Examples

The following example initiates replication on `channel_1`, and replicates changes until it reaches location 120 in the `mysql-bin-changelog.000777` binary log file.

```
call mysql.rds_start_replication_until_for_channel(
  'mysql-bin-changelog.000777',
  120,
  'channel_1'
);
```

mysql.rds_start_replication_until_gtid_for_channel

Initiates replication on the specified channel from an RDS for MySQL DB instance and stops replication at the specified global transaction identifier (GTID).

Syntax

```
CALL mysql.rds_start_replication_until_gtid_for_channel(gtid, channel_name);
```

Parameters

gtid

The GTID after which to stop replication.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_start_replication_until_gtid_for_channel` procedure. The procedure starts replication on the specified channel and applies all changes up to the specified GTID value. Then, it stops replication on the channel.

When the `gtid` parameter specifies a transaction that has already been run by the replica, replication is stopped immediately.

Before you run this procedure, you must disable multi-threaded replication by setting the value of `replica_parallel_workers` or `slave_parallel_workers` to `0`.

Examples

The following example initiates replication on `channel_1`, and replicates changes until it reaches GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
call mysql.rds_start_replication_until_gtid_for_channel('3E11FA47-71CA-11E1-9E33-C80AA9429562:23','channel_1');
```

mysql.rds_stop_replication_for_channel

Stops replication from a MySQL DB instance on the specified channel.

Syntax

```
CALL mysql.rds_stop_replication_for_channel(channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_stop_replication_for_channel` procedure.

Examples

The following example stops replication on `channel_1` of the multi-source replica.

```
CALL mysql.rds_stop_replication_for_channel('channel_1');
```

Replicating transactions using GTIDs

The following stored procedures control how transactions are replicated using global transaction identifiers (GTIDs) with RDS for MySQL. For more information about replication based on GTIDs with RDS for MySQL, see [Using GTID-based replication](#).

When using stored procedures to manage replication with a replication user configured with `caching_sha2_password`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4.

Topics

- [mysql.rds_skip_transaction_with_gtid](#)
- [mysql.rds_start_replication_until_gtid](#)

mysql.rds_skip_transaction_with_gtid

Skips replication of a transaction with the specified global transaction identifier (GTID) on a MySQL DB instance.

You can use this procedure for disaster recovery when a specific GTID transaction is known to cause a problem. Use this stored procedure to skip the problematic transaction. Examples of problematic transactions include transactions that disable replication, delete important data, or cause the DB instance to become unavailable.

Syntax

```
CALL mysql.rds_skip_transaction_with_gtid (
  gtid_to_skip
);
```

Parameters

gtid_to_skip

The GTID of the replication transaction to skip.

Usage notes

The master user must run the `mysql.rds_skip_transaction_with_gtid` procedure.

This procedure is supported for all RDS for MySQL 5.7 versions, all RDS for MySQL 8.0 versions, and all RDS for MySQL 8.4 versions.

Examples

The following example skips replication of the transaction with the GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23.

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_start_replication_until_gtid

Initiates replication from an RDS for MySQL DB instance and stops replication immediately after the specified global transaction identifier (GTID).

Syntax

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

Parameters

gtid

The GTID after which replication is to stop.

Usage notes

The master user must run the `mysql.rds_start_replication_until_gtid` procedure.

This procedure is supported for all RDS for MySQL 5.7 versions, all RDS for MySQL 8.0 versions, and all RDS for MySQL 8.4 versions.

You can use this procedure with delayed replication for disaster recovery. If you have delayed replication configured, you can use this procedure to roll forward changes to a delayed read replica to the time just before a disaster. After this procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

You can configure delayed replication using the following stored procedures:

- [mysql.rds_set_configuration](#)
- [mysql.rds_set_external_master_with_delay \(RDS for MariaDB and RDS for MySQL major versions 8.0 and lower\)](#)
- [mysql.rds_set_external_source_with_delay \(RDS for MySQL major versions 8.4 and higher\)](#)
- [mysql.rds_set_source_delay](#)

When the `gtid` parameter specifies a transaction that has already been run by the replica, replication is stopped immediately.

Examples

The following example initiates replication and replicates changes until it reaches GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

Rotating the query logs

The following stored procedures rotate MySQL logs to backup tables. For more information, see [MySQL database log files](#).

Topics

- [mysql.rds_rotate_general_log](#)
- [mysql.rds_rotate_slow_log](#)

mysql.rds_rotate_general_log

Rotates the mysql.general_log table to a backup table.

Syntax

```
CALL mysql.rds_rotate_general_log;
```

Usage notes

You can rotate the mysql.general_log table to a backup table by calling the mysql.rds_rotate_general_log procedure. When log tables are rotated, the current log table is copied to a backup log table and the entries in the current log table are removed. If a backup log table already exists, then it is deleted before the current log table is copied to the backup. You can query the backup log table if needed. The backup log table for the mysql.general_log table is named mysql.general_log_backup.

You can run this procedure only when the log_output parameter is set to TABLE.

mysql.rds_rotate_slow_log

Rotates the mysql.slow_log table to a backup table.

Syntax

```
CALL mysql.rds_rotate_slow_log;
```

Usage notes

You can rotate the `mysql.slow_log` table to a backup table by calling the `mysql.rds_rotate_slow_log` procedure. When log tables are rotated, the current log table is copied to a backup log table and the entries in the current log table are removed. If a backup log table already exists, then it is deleted before the current log table is copied to the backup.

You can query the backup log table if needed. The backup log table for the `mysql.slow_log` table is named `mysql.slow_log_backup`.

Setting and showing binary log configuration

The following stored procedures set and show configuration parameters, such as for binary log file retention.

Topics

- [mysql.rds_set_configuration](#)
- [mysql.rds_show_configuration](#)

mysql.rds_set_configuration

Specifies the number of hours to retain binary logs or the number of seconds to delay replication.

Syntax

```
CALL mysql.rds_set_configuration(name, value);
```

Parameters

name

The name of the configuration parameter to set.

value

The value of the configuration parameter.

Usage notes

The mysql.rds_set_configuration procedure supports the following configuration parameters:

- [binlog retention hours](#)
- [source delay](#)
- [target delay](#)

The configuration parameters are stored permanently and survive any DB instance reboot or failover.

binlog retention hours

The `binlog retention hours` parameter is used to specify the number of hours to retain binary log files. Amazon RDS normally purges a binary log as soon as possible, but the binary log might still be required for replication with a MySQL database external to RDS.

The default value of `binlog retention hours` is `NULL`. For RDS for MySQL, `NULL` means binary logs aren't retained (0 hours).

To specify the number of hours to retain binary logs on a DB instance, use the `mysql.rds_set_configuration` stored procedure and specify a period with enough time for replication to occur, as shown in the following example.

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



Note

You can't use the value `0` for `binlog retention hours`.

For MySQL DB instances, the maximum `binlog retention hours` value is 168 (7 days).

After you set the retention period, monitor storage usage for the DB instance to make sure that the retained binary logs don't take up too much storage.

For Multi-AZ DB cluster deployments, you can only configure binary log retention from the writer DB instance, and the setting is propagated to all reader DB instances asynchronously. If binary logs on the DB cluster exceed half of the total local storage space, Amazon RDS automatically moves stale logs to the EBS volume. However, the newest logs remain in local storage, so they're subject to be lost if there's a failure that requires a host replacement, or if you scale the database up or down.

source delay

Use the `source delay` parameter in a read replica to specify the number of seconds to delay replication from the read replica to its source DB instance. Amazon RDS normally replicates changes as soon as possible, but you might want some environments to delay replication. For example, when replication is delayed, you can roll forward a delayed read replica to the time just before a disaster. If a table is dropped accidentally, you can use delayed replication to quickly recover it. The default value of `target delay` is `0` (don't delay replication).

When you use this parameter, it runs [mysql.rds_set_source_delay](#) and applies CHANGE primary TO MASTER_DELAY = input value. If successful, the procedure saves the source delay parameter to the mysql.rds_configuration table.

To specify the number of seconds for Amazon RDS to delay replication to a source DB instance, use the mysql.rds_set_configuration stored procedure and specify the number of seconds to delay replication. In the following example, the replication is delayed by at least one hour (3,600 seconds).

```
call mysql.rds_set_configuration('source delay', 3600);
```

The procedure then runs mysql.rds_set_source_delay(3600).

The limit for the source delay parameter is one day (86400 seconds).

target delay

Use the target delay parameter to specify the number of seconds to delay replication between a DB instance and any future RDS-managed read replicas created from this instance. This parameter is ignored for non-RDS-managed read replicas. Amazon RDS normally replicates changes as soon as possible, but you might want some environments to delay replication. For example, when replication is delayed, you can roll forward a delayed read replica to the time just before a disaster. If a table is dropped accidentally, you can use delayed replication to recover it quickly. The default value of target delay is 0 (don't delay replication).

For disaster recovery, you can use this configuration parameter with the [mysql.rds_start_replication_until](#) stored procedure or the [mysql.rds_start_replication_until_gtid](#) stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the mysql.rds_set_configuration procedure with this parameter set. After the mysql.rds_start_replication_until or mysql.rds_start_replication_until_gtid procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in [Promoting a read replica to be a standalone DB instance](#).

To use the mysql.rds_start_replication_until_gtid procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the [mysql.rds_skip_transaction_with_gtid](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication](#).

To specify the number of seconds for Amazon RDS to delay replication to a read replica, use the mysql.rds_set_configuration stored procedure and specify the number of seconds to delay

replication. The following example specifies that replication is delayed by at least one hour (3,600 seconds).

```
call mysql.rds_set_configuration('target delay', 3600);
```

The limit for the target delay parameter is one day (86400 seconds).

mysql.rds_show_configuration

The number of hours that binary logs are retained.

Syntax

```
CALL mysql.rds_show_configuration;
```

Usage notes

To verify the number of hours that Amazon RDS retains binary logs, use the mysql.rds_show_configuration stored procedure.

Examples

The following example displays the retention period:

```
call mysql.rds_show_configuration;
      name          value      description
      binlog retention hours    24      binlog retention hours specifies
the duration in hours before binary logs are automatically deleted.
```

Warming the InnoDB cache

The following stored procedures save, load, or cancel loading the InnoDB buffer pool on RDS for MySQL DB instances. For more information, see [InnoDB cache warming for MySQL on Amazon RDS](#).

Topics

- [mysql.rds_innodb_buffer_pool_dump_now](#)
- [mysql.rds_innodb_buffer_pool_load_abort](#)
- [mysql.rds_innodb_buffer_pool_load_now](#)

mysql.rds_innodb_buffer_pool_dump_now

Dumps the current state of the buffer pool to disk.

Syntax

```
CALL mysql.rds_innodb_buffer_pool_dump_now();
```

Usage notes

The master user must run the mysql.rds_innodb_buffer_pool_dump_now procedure.

mysql.rds_innodb_buffer_pool_load_abort

Cancels a load of the saved buffer pool state while in progress.

Syntax

```
CALL mysql.rds_innodb_buffer_pool_load_abort();
```

Usage notes

The master user must run the mysql.rds_innodb_buffer_pool_load_abort procedure.

mysql.rds_innodb_buffer_pool_load_now

Loads the saved state of the buffer pool from disk.

Syntax

```
CALL mysql.rds_innodb_buffer_pool_load_now();
```

Usage notes

The master user must run the `mysql.rds_innodb_buffer_pool_load_now` procedure.