Amazon RDS for MariaDB

Amazon RDS supports several versions of MariaDB for DB instances. For complete information about the supported versions, see MariaDB on Amazon RDS versions.

To create a MariaDB DB instance, use the Amazon RDS management tools or interfaces. You can then use the Amazon RDS tools to perform management actions for the DB instance. These include actions such as the following:

- Reconfiguring or resizing the DB instance
- Authorizing connections to the DB instance
- Creating and restoring from backups or snapshots
- Creating Multi-AZ secondaries
- Creating read replicas
- Monitoring the performance of your DB instance

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

MariaDB is available in all of the AWS Regions. For more information about AWS Regions, see Regions, Availability Zones, and Local Zones.

You can use Amazon RDS for MariaDB databases to build HIPAA-compliant applications. You can store healthcare-related information, including protected health information (PHI), under a Business Associate Agreement (BAA) with AWS. For more information, see HIPAA compliance. AWS Services in Scope have been fully assessed by a third-party auditor and result in a certification, attestation of compliance, or Authority to Operate (ATO). For more information, see AWS services in scope by compliance program.

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 MariaDB inside a virtual private cloud (VPC) based on Amazon VPC. You can also add features to your MariaDB DB instance by enabling various options. Amazon RDS supports Multi-AZ deployments for MariaDB 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

MariaDB feature support on Amazon RDS

RDS for MariaDB supports most of the features and capabilities of MariaDB. 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 **`MariaDB 2023`**.

> **Note**
>
> The following lists are not exhaustive.

For more information about MariaDB feature support on Amazon RDS, see the following topics.

Topics

- Supported storage engines for MariaDB on Amazon RDS
- Cache warming for MariaDB on Amazon RDS
- MariaDB features not supported by Amazon RDS

MariaDB feature support on Amazon RDS for MariaDB major versions

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

Topics

- MariaDB 11.8 support on Amazon RDS
- MariaDB 11.4 support on Amazon RDS
- MariaDB 10.11 support on Amazon RDS
- MariaDB 10.6 support on Amazon RDS
- MariaDB 10.5 support on Amazon RDS
- MariaDB 10.4 support on Amazon RDS

For information about supported minor versions of Amazon RDS for MariaDB, see MariaDB on Amazon RDS versions.

MariaDB 11.8 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 11.8 or higher.

> **Note**
>
> In MariaDB 11.8, the default value for `require_secure_transport` is now 1, requiring secure SSL/TLS connections. Set to `0` if non-secure connections are needed.

- **New default value for parameter** – The default value of `require_secure_transport` parameter changed from `0` to 1, enforcing secure transport connections by default. For more information, see Requiring SSL/TLS for all connections to a MariaDB DB instance on Amazon RDS.
- **Vector support** – You can use the MariaDB Vector to store and search AI-generated vectors directly in MariaDB. This feature introduces the following system variables:
  - The variable `mhnsw_default_distance` specifies the default distance metric for MHNSW vector indexing.
  - The variable `mhnsw_default_m` defines the default value for the `M` parameter in MHNSW vector indexing.
  - The variable `mhnsw_ef_search` defines the minimal number of result candidates for vector index searches.
  - The variable `mhnsw_max_cache_size` sets the upper limit for one MHNSW vector index cache.
- **Temporary file size limits** – You can now limit the size of created disk temporary files and tables using two system variables available in the RDS Maria DB 11.8 parameter group:
  - The variable `max_tmp_session_space_usage` limits the temporary space allowance per user.
  - The variable `max_tmp_total_space_usage` limits the temporary space allowance for all users.
- **Temporary tablespace management** – The temporary tablespace stores temporary tables and grows as data is added. When temporary tables are dropped, the space is not automatically reclaimed. You can use the mysql.rds_execute_operation procedure to shrink the temporary tablespace and reclaim disk space.

For a list of all MariaDB 11.8 features and their documentation, see Changes and improvements in MariaDB 11.8 and Release notes - MariaDB 11.8 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 11.4 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 11.4 or higher.

- **Crypographic library** – RDS for MariaDB replaced OpenSSL with AWS Libcrypto (AWS-LC), which is FIPS 140-3 certified.
- **Simple Password Check plugin** – You can use the MariaDB Simple Password Check Plugin to check whether a password contains at least a specific number of characters of a specific type. For more information, see the section called "Password validation plugins".
- **Cracklib Password Check plugin** – You can use the MariaDB Cracklib Password Check Plugin to check the strength of new passwords. For more information, see the section called "Password validation plugins".
- **InnoDB enhancements** – These enhancements include the following items:
  - The change buffer was removed. For more information, see InnoDB Change Buffering.
  - InnoDB Defragmentation was removed. For more information, see InnoDB Defragmentation.
- **New privilege** – The admin user now also has the `SHOW CREATE ROUTINE` privilege. This privilege permits the grantee to view the `SHOW CREATE` definition statement of a routine that's owned by another user. For more information, see Database Privileges.
- **Replication improvement** – MariaDB version 11.4 DB instances support binlog indexing. You can create a GTID index for each binlog file. These indexes improve the performance of replication by reducing the time it takes to locate a GTID. For more information, see Binlog Indexing.
- **Deprecated or removed parameters** – The following parameters have been deprecated or removed for MariaDB version 11.4 DB instances:
  - `engine_condition_pushdown` is removed from optimizer_switch
  - innodb_change_buffer_max_size
  - innodb_defragment
  - `TLSv1.0` and `TLSv1.1` are removed from tls_version
- **New default values for a parameter** – The default value of the innodb_undo_tablespaces parameter changed from `0` to `3`.

- **New valid values for parameters** – The following parameters have new valid values for MariaDB version 11.4 DB instances:
  - The valid values for the binlog_row_image parameter now include FULL_NODUP.
  - The valid values for the OLD_MODE parameter now include NO_NULL_COLLATION_IDS.
- **New parameters** – The following parameters are new for MariaDB version 11.4 DB instances:
  - The transaction_isolation parameter replaces the tx_isolation parameter.
  - The transaction_read_only parameter replaces the tx_read_only parameter.
  - The block_encryption_mode parameter defines the default block encryption mode for the AES_ENCRYPT() and AES_DECRYPT() functions.
  - The character_set_collations defines overrides for character set default collations.
  - The binlog_gtid_index, binlog_gtid_index_page_size, and binlog_gtid_index_span_min define the properties of the binlog GTID index. For more information, see Binlog Indexing.

For a list of all MariaDB 11.4 features and their documentation, see Changes and improvements in MariaDB 11.4 and Release notes - MariaDB 11.4 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.11 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.11 or higher.

- **Password Reuse Check plugin** – You can use the MariaDB Password Reuse Check plugin to prevent users from reusing passwords and to set the retention period of passwords. For more information, see Password Reuse Check Plugin.
- **GRANT TO PUBLIC authorization** – You can grant privileges to all users who have access to your server. For more information, see GRANT TO PUBLIC.
- **Separation of SUPER and READ ONLY ADMIN privileges** – You can remove READ ONLY ADMIN privileges from all users, even users that previously had SUPER privileges.
- **Security** – You can now set option `--ssl` as the default for your MariaDB client. MariaDB no longer silently disables SSL if the configuration is incorrect.
- **SQL commands and functions** – You can now use the `SHOW ANALYZE FORMAT=JSON` command and the functions `ROW_NUMBER`, `SFORMAT`, and `RANDOM_BYTES`. `SFORMAT` allows string formatting and is enabled by default. You can convert partition to table and table to partition

in a single command. There are also several improvements around `JSON_*()` functions. `DES_ENCRYPT` and `DES_DECRYPT` functions were deprecated for version 10.10 and higher. For more information, see SFORMAT.

- **InnoDB enhancements** – These enhancements include the following items:
  - Performance improvements in the redo log to reduce write amplification and to improve concurrency.
  - The ability for you to change the undo tablespace without reinitializing the data directory. This enhancement reduces control plane overhead. It requires restarting but it doesn't require reinitialization after changing undo tablespace.
  - Support for `CHECK TABLE ... EXTENDED` and for descending indexes internally.
  - Improvements to bulk insert.
- **Binlog changes** – These changes include the following items:
  - Logging `ALTER` in two phases to decrease replication latency. The `binlog_alter_two_phase` parameter is disabled by default, but can be enabled through parameter groups.
  - Logging `explicit_defaults_for_timestamp`.
  - No longer logging `INCIDENT_EVENT` if the transaction can be safely rolled back.
- **Replication improvement**s – MariaDB version 10.11 DB instances use GTID replication by default if the master supports it. Also, `Seconds_Behind_Master` is more precise.
- **Clients** – You can use new command-line options for `mysqlbinglog` and `mariadb-dump`. You can use `mariadb-dump` to dump and restore historical data.
- **System versioning** – You can modify history. MariaDB automatically creates new partitions.
- **Atomic DDL** – `CREATE OR REPLACE` is now atomic. Either the statement succeeds or it's completely reversed.
- **Redo log write** – Redo log writes asynchronously.
- **Stored functions** – Stored functions now support the same `IN`, `OUT`, and `INOUT` parameters as in stored procedures.
- **Deprecated or removed parameters** – The following parameters have been deprecated or removed for MariaDB version 10.11 DB instances:
  - innodb_change_buffering
  - innodb_disallow_writes
  - innodb_log_write_ahead_size

  - innodb_prefix_index_cluster_optimization
  - keep_files_on_create
  - old
- **Dynamic parameters** – The following parameters are now dynamic for MariaDB version 10.11 DB instances:
  - innodb_log_file_size
  - innodb_write_io_threads
  - innodb_read_io_threads
- **New default values for parameters** – The following parameters have new default values for MariaDB version 10.11 DB instances:
  - The default value of the explicit_defaults_for_timestamp parameter changed from `OFF` to `ON`.
  - The default value of the optimizer_prune_level parameter changed from 1 to 2.
- **New valid values for parameters** – The following parameters have new valid values for MariaDB version 10.11 DB instances:
  - The valid values for the old parameter were merged into those for the old_mode parameter.
  - The valid values for the histogram_type parameter now include `JSON_HB`.
  - The valid value range for the innodb_log_buffer_size parameter is now `262144` to `4294967295` (256KB to 4096MB).
  - The valid value range for the innodb_log_file_size parameter is now `4194304` to `512GB` (4MB to 512GB).
  - The valid values for the optimizer_prune_level parameter now include 2.
- **New parameters** – The following parameters are new for MariaDB version 10.11 DB instances:
  - The binlog_alter_two_phase parameter can improve replication performance.
  - The log_slow_min_examined_row_limit parameter can improve performance.
  - The log_slow_query parameter and the log_slow_query_file parameter are aliases for `slow_query_log` and `slow_query_log_file`, respectively.
  - optimizer_extra_pruning_depth
  - system_versioning_insert_history

For a list of all MariaDB 10.11 features and their documentation, see Changes and improvements in MariaDB 10.11 and Release notes - MariaDB 10.11 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.6 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.6 or higher:

- **MyRocks storage engine** – You can use the MyRocks storage engine with RDS for MariaDB to optimize storage consumption of your write-intensive, high-performance web applications. For more information, see Supported storage engines for MariaDB on Amazon RDS and MyRocks.
- **AWS Identity and Access Management (IAM) DB authentication** – You can use IAM DB authentication for better security and central management of connections to your MariaDB DB instances. For more information, see IAM database authentication for MariaDB, MySQL, and PostgreSQL.
- **Upgrade options** – You can now upgrade to RDS for MariaDB version 10.6 from any prior major release (10.3, 10.4, 10.5). You can also restore a snapshot of an existing MySQL 5.6 or 5.7 DB instance to a MariaDB 10.6 instance. For more information, see Upgrades of the MariaDB DB engine.
- **Delayed replication** – You can now set a configurable time period for which a read replica lags behind the source database. In a standard MariaDB replication configuration, there is minimal replication delay between the source and the replica. With delayed replication, you can set an intentional delay as a strategy for disaster recovery. For more information, see Configuring delayed replication with MariaDB.
- **Oracle PL/SQL compatibility** – By using RDS for MariaDB version 10.6, you can more easily migrate your legacy Oracle applications to Amazon RDS. For more information, see SQL_MODE=ORACLE.
- **Atomic DDL** – Your dynamic data language (DDL) statements can be relatively crash-safe with RDS for MariaDB version 10.6. `CREATE TABLE`, `ALTER TABLE`, `RENAME TABLE`, `DROP TABLE`, `DROP DATABASE` and related DDL statements are now atomic. Either the statement succeeds, or it's completely reversed. For more information, see Atomic DDL.
- **Other enhancements** – These enhancements include a `JSON_TABLE` function for transforming JSON data to relational format within SQL, and faster empty table data load with Innodb. They also include new `sys_schema` for analysis and troubleshooting, optimizer enhancement for ignoring unused indexes, and performance improvements. For more information, see JSON_TABLE.

- **New default values for parameters** – The following parameters have new default values for MariaDB version 10.6 DB instances:
  - The default value for the following parameters has changed from `utf8` to `utf8mb3`:
    - character_set_client
    - character_set_connection
    - character_set_results
    - character_set_system

    Although the default values have changed for these parameters, there is no functional change. For more information, see Supported Character Sets and Collations in the MariaDB documentation.
  - The default value of the collation_connection parameter has changed from `utf8_general_ci` to `utf8mb3_general_ci`. Although the default value has changed for this parameter, there is no functional change.
  - The default value of the old_mode parameter has changed from unset to `UTF8_IS_UTF8MB3`. Although the default value has changed for this parameter, there is no functional change.

For a list of all MariaDB 10.6 features and their documentation, see Changes and improvements in MariaDB 10.6 and Release notes - MariaDB 10.6 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.5 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.5 or later:

- **InnoDB enhancements** – MariaDB version 10.5 includes InnoDB enhancements. For more information, see InnoDB: Performance Improvements etc. in the MariaDB documentation.
- **Performance schema updates** – MariaDB version 10.5 includes performance schema updates. For more information, see Performance Schema Updates to Match MySQL 5.7 Instrumentation and Tables in the MariaDB documentation.
- **One file in the InnoDB redo log** – In versions of MariaDB before version 10.5, the value of the `innodb_log_files_in_group` parameter was set to 2. In MariaDB version 10.5, the value of this parameter is set to 1.

If you are upgrading from a prior version to MariaDB version 10.5, and you don't modify the parameters, the `innodb_log_file_size` parameter value is unchanged. However, it applies to one log file instead of two. The result is that your upgraded MariaDB version 10.5 DB instance uses half of the redo log size that it was using before the upgrade. This change can have a noticeable performance impact. To address this issue, you can double the value of the `innodb_log_file_size` parameter. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

- **SHOW SLAVE STATUS command not supported** – In versions of MariaDB before version 10.5, the `SHOW SLAVE STATUS` command required the `REPLICATION SLAVE` privilege. In MariaDB version 10.5, the equivalent `SHOW REPLICA STATUS` command requires the `REPLICATION REPLICA ADMIN` privilege. This new privilege isn't granted to the RDS master user.

  Instead of using the `SHOW REPLICA STATUS` command, run the new `mysql.rds_replica_status` stored procedure to return similar information. For more information, see mysql.rds_replica_status.

- **SHOW RELAYLOG EVENTS command not supported** – In versions of MariaDB before version 10.5, the `SHOW RELAYLOG EVENTS` command required the `REPLICATION SLAVE` privilege. In MariaDB version 10.5, this command requires the `REPLICATION REPLICA ADMIN` privilege. This new privilege isn't granted to the RDS master user.

- **New default values for parameters** – The following parameters have new default values for MariaDB version 10.5 DB instances:

  - The default value of the max_connections parameter has changed to `LEAST({DBInstanceClassMemory/25165760},12000)`. For information about the `LEAST` parameter function, see DB parameter functions.
  - The default value of the innodb_adaptive_hash_index parameter has changed to `OFF` (`0`).
  - The default value of the innodb_checksum_algorithm parameter has changed to `full_crc32`.
  - The default value of the innodb_log_file_size parameter has changed to 2 GB.

For a list of all MariaDB 10.5 features and their documentation, see Changes and improvements in MariaDB 10.5 and Release notes - MariaDB 10.5 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.4 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.4 or later:

- **User account security enhancements** – Password expiration and account locking improvements
- **Optimizer enhancements** – Optimizer trace feature
- **InnoDB enhancements** – Instant DROP COLUMN support and instant VARCHAR extension for `ROW_FORMAT=DYNAMIC` and `ROW_FORMAT=COMPACT`
- **New parameters** – Including tcp_nodedelay, tls_version, and gtid_cleanup_batch_size

For a list of all MariaDB 10.4 features and their documentation, see Changes and improvements in MariaDB 10.4 and Release notes - MariaDB 10.4 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

Supported storage engines for MariaDB on Amazon RDS

RDS for MariaDB supports the following storage engines.

Topics

- The InnoDB storage engine
- The MyRocks storage engine

Other storage engines aren't currently supported by RDS for MariaDB.

The InnoDB storage engine

Although MariaDB supports multiple storage engines with varying capabilities, not all of them are optimized for recovery and data durability. InnoDB is the recommended storage engine for MariaDB DB instances on Amazon RDS. Amazon RDS features such as point-in-time restore and snapshot restore require a recoverable storage engine and are supported only for the recommended storage engine for the MariaDB version.

For more information, see InnoDB.

The MyRocks storage engine

The MyRocks storage engine is available in RDS for MariaDB version 10.6 and higher. Before using the MyRocks storage engine in a production database, we recommend that you perform thorough benchmarking and testing to verify any potential benefits over InnoDB for your use case.

The default parameter group for MariaDB version 10.6 includes MyRocks parameters. For more information, see Parameters for MariaDB and Parameter groups for Amazon RDS.

To create a table that uses the MyRocks storage engine, specify `ENGINE=RocksDB` in the `CREATE TABLE` statement. The following example creates a table that uses the MyRocks storage engine.

```
CREATE TABLE test (a INT NOT NULL, b CHAR(10)) ENGINE=RocksDB;
```

We strongly recommend that you don't run transactions that span both InnoDB and MyRocks tables. MariaDB doesn't guarantee ACID (atomicity, consistency, isolation, durability) for transactions across storage engines. Although it is possible to have both InnoDB and MyRocks tables in a DB instance, we don't recommend this approach except during a migration from one storage engine to the other. When both InnoDB and MyRocks tables exist in a DB instance, each storage engine has its own buffer pool, which might cause performance to degrade.

MyRocks doesn't support `SERIALIZABLE` isolation or gap locks. So, generally you can't use MyRocks with statement-based replication. For more information, see MyRocks and Replication.

Currently, you can modify only the following MyRocks parameters:

- `rocksdb_block_cache_size`
- `rocksdb_bulk_load`
- `rocksdb_bulk_load_size`
- `rocksdb_deadlock_detect`
- `rocksdb_deadlock_detect_depth`
- `rocksdb_max_latest_deadlocks`

The MyRocks storage engine and the InnoDB storage engine can compete for memory based on the settings for the `rocksdb_block_cache_size` and `innodb_buffer_pool_size` parameters. In some cases, you might only intend to use the MyRocks storage engine on a particular DB instance. If so, we recommend setting the `innodb_buffer_pool_size minimal` parameter to a minimal value and setting the `rocksdb_block_cache_size` as high as possible.

You can access MyRocks log files by using the `DescribeDBLogFiles` and `DownloadDBLogFilePortion` operations.

For more information about MyRocks, see MyRocks on the MariaDB website.

Cache warming for MariaDB on Amazon RDS

InnoDB cache warming can provide performance gains for your MariaDB 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 approach 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. For more information on cache warming, see Dumping and restoring the buffer pool in the MariaDB documentation.

Cache warming is enabled by default on MariaDB 10.3 and higher DB instances.
To enable it, 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 affects all MariaDB DB instances that use that parameter group. To enable cache warming for specific MariaDB DB instances, you might need to create a new parameter group for those DB instances. For information on parameter groups, see Parameter groups for Amazon RDS.

Cache warming primarily provides a performance benefit for DB instances that use standard storage. If you use PIOPS storage, you don't commonly see a significant performance benefit.

Important

If your MariaDB DB instance doesn't shut down normally, such as during a failover, then the buffer pool state isn't saved to disk. In this case, MariaDB 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 before the restart. To ensure that you have a recent state of the buffer pool available to warm the cache on startup, we recommend that you periodically dump the buffer pool "on demand." You can dump or load the buffer pool on demand.
You can create an event to dump the buffer pool automatically and at 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, see Events in the MariaDB documentation.

Dumping and loading the buffer pool on demand

You can save and load the cache on demand using the following stored procedures:

- 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.

MariaDB features not supported by Amazon RDS

The following MariaDB features are not supported on Amazon RDS:

- S3 storage engine
- Authentication plugin – GSSAPI
- Authentication plugin – Unix Socket
- AWS Key Management encryption plugin
- Delayed replication for MariaDB versions lower than 10.6
- Native MariaDB encryption at rest for InnoDB and Aria

  You can enable encryption at rest for a MariaDB DB instance by following the instructions in Encrypting Amazon RDS resources.
- HandlerSocket
- JSON table type for MariaDB versions lower than 10.6
- MariaDB ColumnStore
- MariaDB Galera Cluster
- Multisource replication

- MyRocks storage engine for MariaDB versions lower than 10.6
- Password validation plugin, `simple_password_check`, and `cracklib_password_check` for MariaDB versions lower than 11.4
- Spider storage engine
- Sphinx storage engine
- TokuDB storage engine
- Storage engine-specific object attributes, as described in Engine-defined new Table/Field/Index attributes in the MariaDB documentation
- Table and tablespace encryption
- Hashicorp Key Management plugin
- Running two upgrades in parallel

To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances, and it 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.

MariaDB on Amazon RDS versions

For MariaDB, 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 10.5 to 10.6. A version change is considered minor if only the minor version number changes, for example going from version 10.6.14 to 10.6.16.

Topics

Supported MariaDB minor versions on Amazon RDS

Amazon RDS currently supports the following minor versions of MariaDB.

> **Note**
>
> Dates with only a month and a year are approximate and are updated with an exact date when it's known.

The following table shows the minor versions of MariaDB 11.8 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
11.8.3	06 August 2025	25 August 2025	September 2026

The following table shows the minor versions of MariaDB 11.4 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
11.4.8	06 August 2025	13 August 2025	September 2026
11.4.7	22 May 2025	4 June 2025	September 2026
11.4.5	4 February 2025	24 February 2025	March 2026
11.4.4	1 November 2024	20 December 2024	March 2026
11.4.3	8 August 2024	15 October 2024	March 2026

The following table shows the minor versions of MariaDB 10.11 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.11.14	06 August 2025	13 August 2025	September 2026
10.11.13	22 May 2025	4 June 2025	September 2026
10.11.11	4 February 2025	24 February 2025	March 2026
10.11.10	1 November 2024	20 December 2024	March 2026
10.11.9	8 August 2024	4 September 2024	March 2026
10.11.8	16 May 2024	14 June 2024	October 2025

The following table shows the minor versions of MariaDB 10.6 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.6.23	6 August 2025	13 August 2025	August 2026

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.6.22	6 May 2025	20 May 2025	July 2026
10.6.21	4 February 2025	24 February 2025	March 2026
10.6.20	1 November 2024	20 December 2024	March 2026
10.6.19	8 August 2024	4 September 2024	March 2026
10.6.18	16 May 2024	14 June 2024	October 2025

The following table shows the minor versions of MariaDB 10.5 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.5.29	6 May 2025	20 May 2025	February 2026
10.5.28	4 February 2025	24 February 2025	February 2026
10.5.27	1 November 2024	20 December 2024	February 2026
10.5.26	8 August 2024	4 September 2024	October 2025
10.5.25	16 May 2024	14 June 2024	October 2025

The following table shows the minor versions of MariaDB 10.4 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.4.34	16 May 2024	14 June 2024	28 February 2025
10.4.33	7 February 2024	26 February 2024	28 February 2025

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.4.32	13 November 2023	12 December 2023	28 February 2025
10.4.31	14 August 2023	7 September 2023	28 February 2025
10.4.30	7 June 2023	22 June 2023	28 February 2025
10.4.29	10 May 2023	15 June 2023	28 February 2025

You can specify any currently supported MariaDB version when creating a new DB instance. You can specify the major version (such as MariaDB 10.5), 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, use the `describe-db-engine-versions` AWS CLI command.

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

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

The default MariaDB 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 mariadb
 --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 MariaDB minor engine version for the 10.5 major version and the US West (Oregon) AWS Region (us-west-2):

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

MariaDB minor versions on Amazon RDS

Minor versions

- MariaDB version 11.8.3
- MariaDB version 11.4.8
- MariaDB version 11.4.7
- MariaDB version 11.4.5
- MariaDB version 11.4.4
- MariaDB version 10.11.14
- MariaDB version 10.11.13
- MariaDB version 10.11.11
- MariaDB version 10.11.10
- MariaDB version 10.6.23
- MariaDB version 10.6.22
- MariaDB version 10.6.21
- MariaDB version 10.6.20
- MariaDB version 10.5.29
- MariaDB version 10.5.28
- MariaDB version 10.5.27

MariaDB version 11.8.3

MariaDB version 11.8.3 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- **New default value for parameter** – The default value of `require_secure_transport` parameter changed from `0` to `1`, enforcing secure transport connections by default. For more

information, see Requiring SSL/TLS for all connections to a MariaDB DB instance on Amazon RDS.

MariaDB version 11.4.8

MariaDB version 11.4.8 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 11.4.7

MariaDB version 11.4.7 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

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

MariaDB version 11.4.5

MariaDB version 11.4.5 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 11.4.4

MariaDB version 11.4.4 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

MariaDB version 10.11.14

MariaDB version 10.11.14 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 10.11.13

MariaDB version 10.11.13 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

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

MariaDB version 10.11.11

MariaDB version 10.11.11 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 10.11.10

MariaDB version 10.11.10 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

MariaDB version 10.6.23

MariaDB version 10.6.23 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 10.6.22

MariaDB version 10.6.22 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

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

MariaDB version 10.6.21

MariaDB version 10.6.21 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 10.6.20

MariaDB version 10.6.20 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

MariaDB version 10.5.29

MariaDB version 10.5.29 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

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

MariaDB version 10.5.28

MariaDB version 10.5.28 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 10.5.27

MariaDB version 10.5.27 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

Supported MariaDB major versions on Amazon RDS

RDS for MariaDB major versions remain available at least until community end of life for the corresponding community version. You can use the following dates to plan your testing and upgrade cycles. If Amazon extends support for an RDS for MariaDB version for longer than originally stated, we plan to update this table to reflect the later date.

> **Note**
>
> Dates with only a month and a year are approximate and are updated with an exact date when it's known.
> You can also view information about support dates for major engine versions by running the describe-db-major-engine-versions AWS CLI command or by using the DescribeDBMajorEngineVersions RDS API operation.

MariaDB major version	Community release date	RDS release date	Community end of life date	RDS end of standard support date
MariaDB 11.8	6 August 2025	25 August 2025	June 2030	June 2030
MariaDB 11.4	8 August 2024	15 October 2024	May 2029	May 2029
MariaDB 10.11	16 February 2023	21 August 2023	16 February 2028	February 2028
MariaDB 10.6	6 July 2021	3 February 2022	6 July 2026	July 2026
MariaDB 10.5	24 June 2020	21 January 2021	24 June 2025	February 2026
MariaDB 10.4	18 June 2019	6 April 2020	18 June 2024	February 2025

Working with the Database Preview environment

RDS for MariaDB DB instances in the Database Preview environment are functionally similar to other RDS for MariaDB 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 MariaDB 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

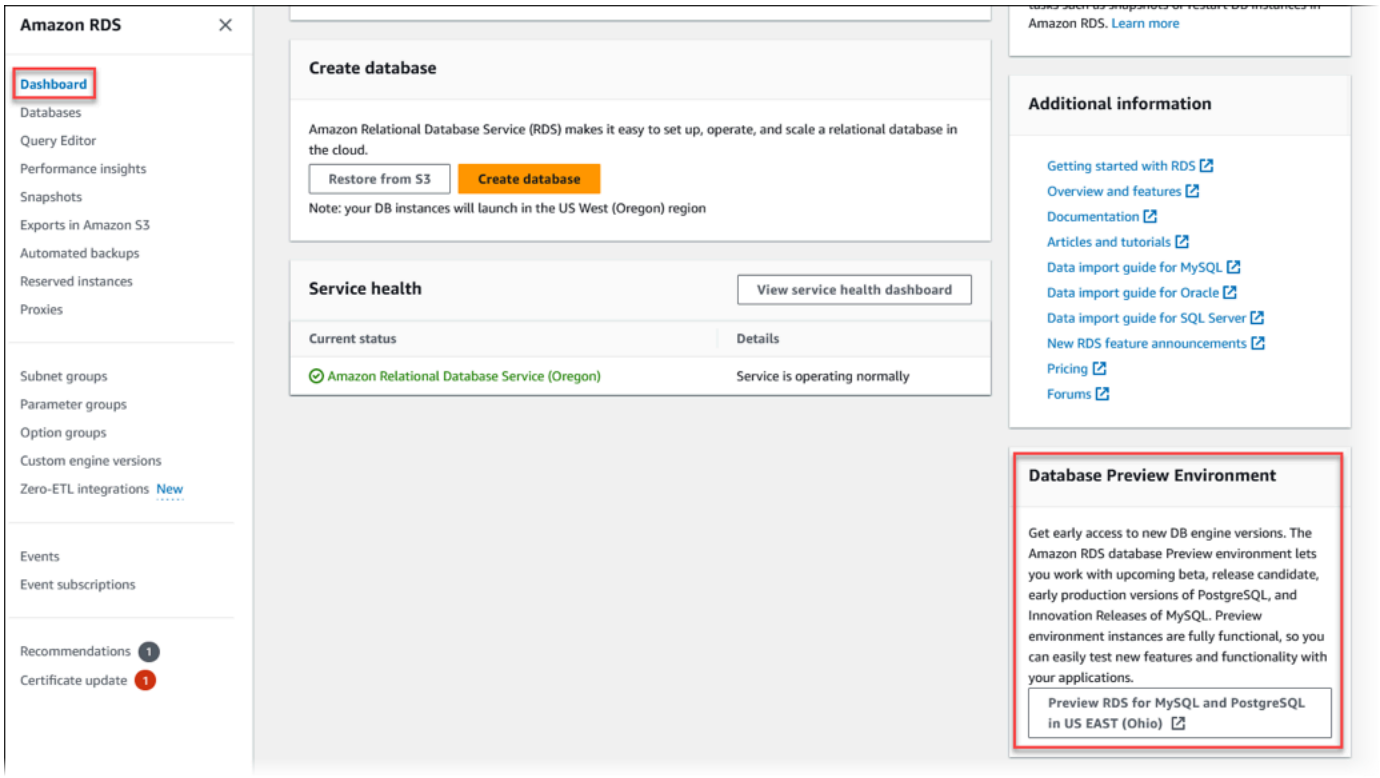
Creating a new DB instance in the Database Preview environment

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

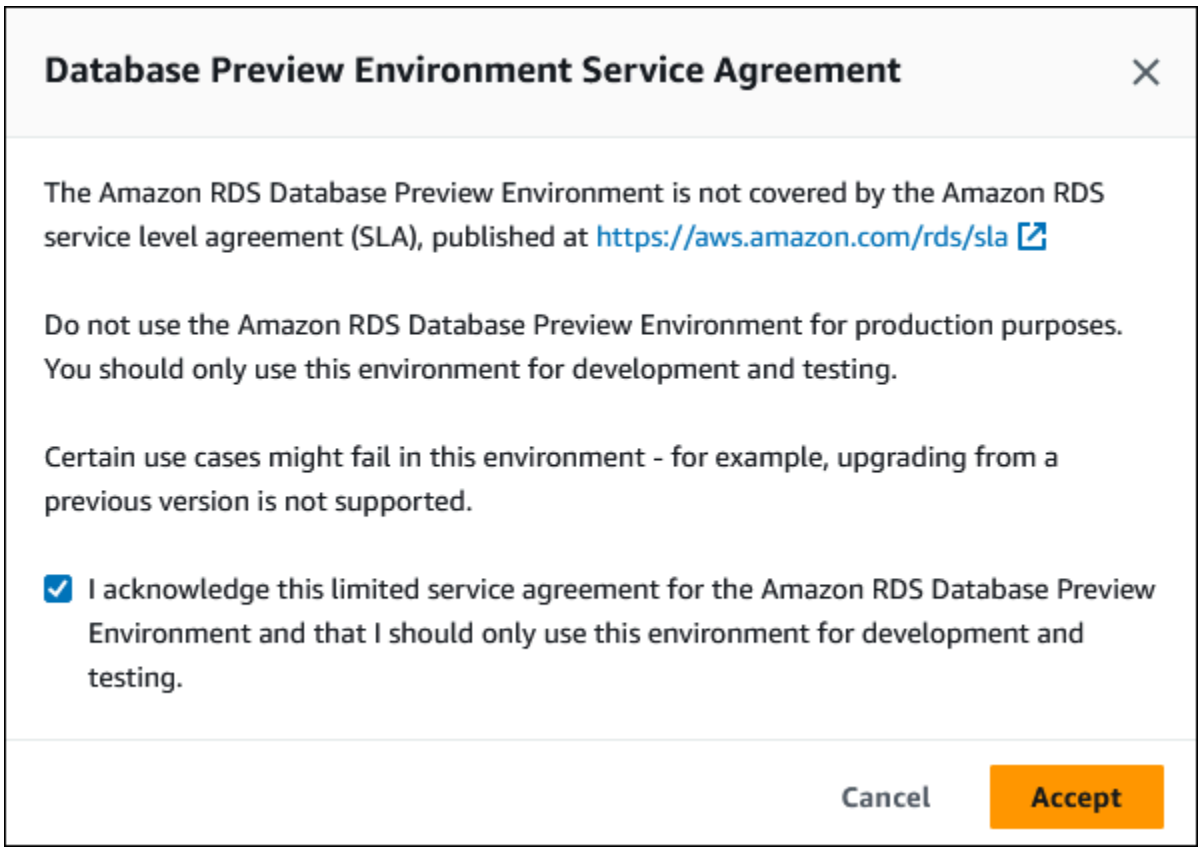
Console

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.



4. To create the RDS for MariaDB 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 MariaDB 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 MariaDB 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.

MariaDB version 11.7 in the Database Preview environment

MariaDB version 11.7 is now available in the Amazon RDS Database Preview environment. MariaDB version 11.7 contains several improvements that are described in Changes and improvements in MariaDB 11.7. This version also includes support for the vector data type, indexing, and search. For more information, see Vector Overview in the MariaDB documentation.

You can use the Database Preview environment to test your workloads against this release before it is available in all AWS Regions for production workloads. For information on the Database Preview environment, see the section called “The Database Preview environment”. To access the Preview Environment from the console, select rds-preview/.

Deprecated versions for Amazon RDS for MariaDB

Amazon RDS for MariaDB versions 10.0, 10.1, 10.2, and 10.3 are deprecated.

For information about the Amazon RDS deprecation policy for MariaDB, see Amazon RDS FAQs.

Connecting to your MariaDB DB instance

After Amazon RDS provisions your DB instance, you can use any standard MariaDB client application or utility to connect to the instance. In the connection string, you specify the Domain Name System (DNS) address from the DB instance endpoint as the host parameter. You also specify the port number from the DB instance endpoint as the port parameter.

You can connect to an Amazon RDS for MariaDB DB instance by using tools like the MySQL command-line client. For more information on using the MySQL command-line client, see mysql command-line client in the MariaDB documentation. One GUI-based application that you can use to connect is Heidi. For more information, see the Download HeidiSQL page. For information about installing MySQL (including the MySQL command-line client), see Installing and upgrading MySQL.

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
```

To connect to a DB instance from outside of a virtual private cloud (VPC) based on Amazon VPC, the DB instance must be publicly accessible. Also, 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 SSL encryption on connections to a MariaDB DB instance. For information, see SSL/TLS support for MariaDB DB instances on Amazon RDS.

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

Topics

- Finding the connection information for a MariaDB DB instance
- Connecting from the MySQL command-line client (unencrypted) for RDS for MariaDB
- Connecting to RDS for MariaDB with the AWS JDBC Driver and AWS Python Driver;
- Troubleshooting connections to your MariaDB DB instance

Finding the connection information for a MariaDB 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 MariaDB DB engine. For example, you might use the MySQL command-line client or MySQL Workbench.

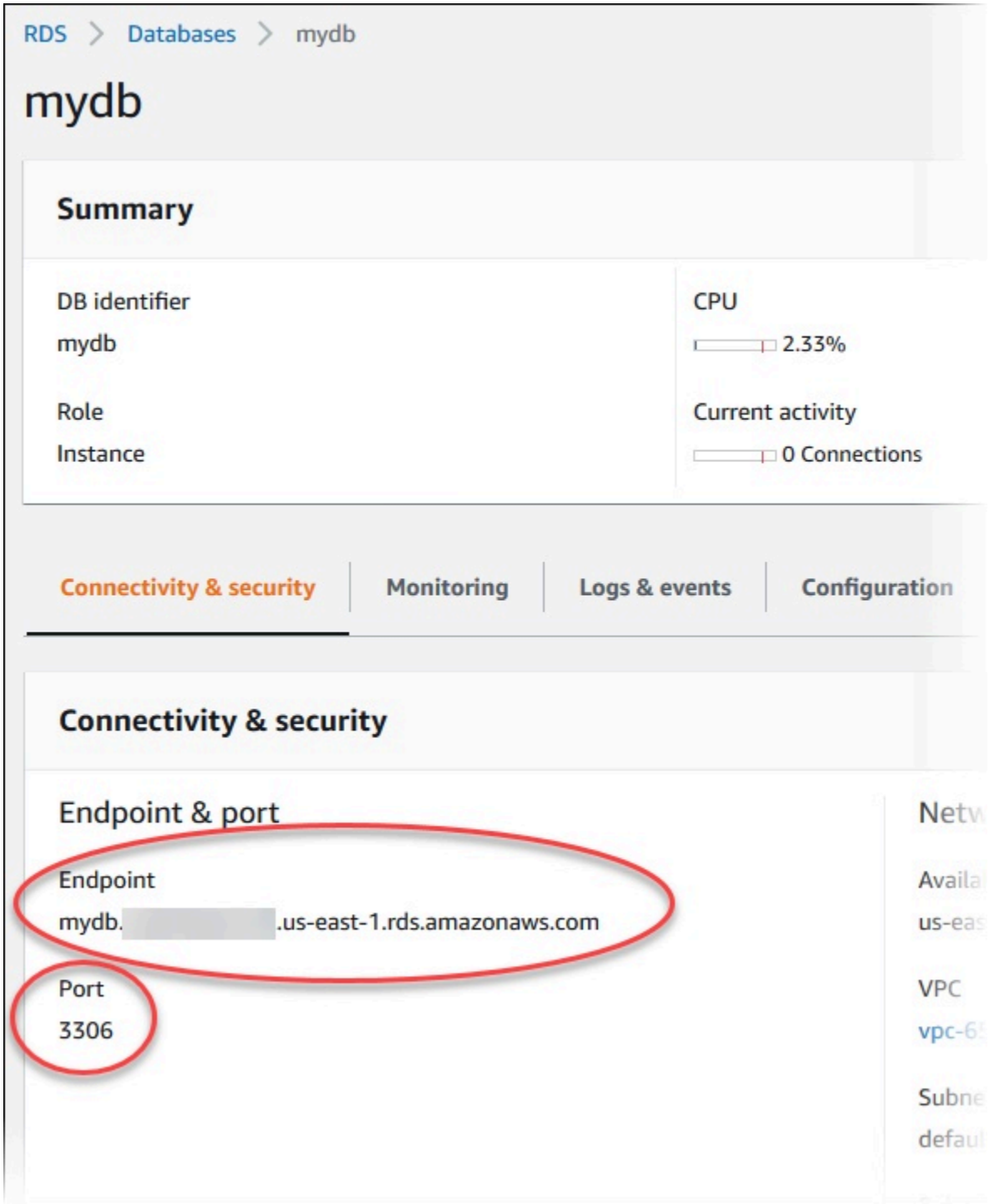
To find the connection information for a DB instance, you can use the AWS Management Console, the AWS Command Line Interface (AWS CLI) describe-db-instances command, or the Amazon RDS API DescribeDBInstances operation to list its details.

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 MariaDB 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.



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 MariaDB 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=mariadb" \
  --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

For Windows:

```
aws rds describe-db-instances ^
  --filters "Name=engine,Values=mariadb" ^
  --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.

Connecting from the MySQL command-line client (unencrypted) for RDS for MariaDB

> **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 MariaDB 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 a command prompt on a client computer. Doing this connects you to a database on a MariaDB DB instance. Substitute the DNS name (endpoint) for your DB instance for *<endpoint>* and the master user name that you used for *<mymasteruser>*. Provide the master password that you used when prompted for a password.

```
mysql -h <endpoint> -P 3306 -u <mymasteruser> -p
```

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

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.6.10-MariaDB-log Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Connecting to RDS for MariaDB with the AWS JDBC Driver and AWS Python Driver;

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

Topics

Connecting to RDS for MariaDB 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 MariaDB 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.

Troubleshooting connections to your MariaDB DB instance

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

- The DB instance was created using a security group that doesn't authorize connections from the device or Amazon EC2 instance where the MariaDB 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 MariaDB DB instance connections

You can manage the security of your MariaDB DB instances.

Topics

MariaDB security on Amazon RDS

Security for MariaDB 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 Socket 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.
- Once a connection has been opened to a MariaDB DB instance, authentication of the login and permissions are applied the same way as in a stand-alone instance of MariaDB. Commands such as `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE`, and `SET PASSWORD` work just as they do in stand-alone databases, as does directly modifying database schema tables.

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

- `alter`
- `alter routine`
- `create`
- `create routine`
- `create temporary tables`

- `create user`
- `create view`
- `delete`
- `drop`
- `event`
- `execute`
- `grant option`
- `index`
- `insert`
- `lock tables`
- `process`
- `references`
- `reload`

  This privilege is limited on MariaDB DB instances. It doesn't grant access to the `FLUSH LOGS` or `FLUSH TABLES WITH READ LOCK` operations.
- `replication client`
- `replication slave`
- `select`
- `show create routine`

  This privilege is only on MariaDB DB instances running version 11.4 and higher.
- `show databases`
- `show view`
- `trigger`
- `update`

For more information about these privileges, see User account management in the MariaDB documentation.

> **Note**
>
> Although you can delete the master user on a DB instance, we don't recommend doing so. To recreate the master user, use the `ModifyDBInstance` API or the `modify-db-instance` AWS CLI 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 for, or change privileges for the `rdsadmin` account results in an error.

To allow management of the DB instance, the standard `kill` and `kill_query` commands have been restricted. The Amazon RDS commands `mysql.rds_kill`, `mysql.rds_kill_query`, and `mysql.rds_kill_query_id` are provided for use in MariaDB and also MySQL so that you can end user sessions or queries on DB instances.

Using the password validation plugins for RDS for MariaDB

Starting with RDS for MariaDB version 11.4, you can use the following password validation plugins to enhance the security of your database connections:

- simple_password_check – checks whether a password contains at least a specific number of characters of a specific type.
- cracklib_password_check – checks whether a password appears in a dictionary file of the CrackLib library.

To enable these plugins, set the value of the parameter `simple_password_check` or `cracklib_password_check` to `FORCE_PLUS_PERMANENT` in the DB parameter group associated with the DB instance. When this value is set, the plugin can't be uninstalled by using the `UNINSTALL PLUGIN` statement at runtime.

To disable these plugins, set the value of the parameter `simple_password_check` or `cracklib_password_check` to `OFF` in the DB parameter group associated with the DB instance. When this value is set, the plugin validation rules no longer apply for new passwords.

For information about setting the values of parameters in parameter groups, see the section called "Modifying parameters in a DB parameter group".

Encrypting client connections with SSL/TLS to MariaDB 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 MariaDB DB instances. Using SSL/TLS, you can encrypt a connection between your application client and your MariaDB DB instance. SSL/TLS support is available in all AWS Regions.

With Amazon RDS, you can secure data in transit by encrypting client connections to MariaDB DB instances with SSL/TLS, requiring SSL/TLS for all connections to a MariaDB 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 MariaDB DB instances on Amazon RDS.

Topics

SSL/TLS support for MariaDB 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 MariaDB instances using new SSL/TLS certificates.

Amazon RDS for MariaDB supports Transport Layer Security (TLS) versions 1.3, 1.2, 1.1, and 1.0. TLS support depends on the MariaDB minor version. The following table shows the TLS support for MariaDB minor versions.

TLS version	MariaDB 11.8	MariaDB 11.4	MariaDB 10.11	MariaDB 10.6	MariaDB 10.5	MariaDB 10.4
TLS 1.3	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions
TLS 1.2	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions
TLS 1.1	Not supported	Not supported	Not supported	10.6.16 and lower	10.5.23 and lower	10.4.32 and lower
TLS 1.0	Not supported	Not supported	Not supported	10.6.16 and lower	10.5.23 and lower	10.4.32 and lower

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

You can require SSL/TLS encryption for specified user account connections to your MariaDB 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 MariaDB, see Securing Connections for Client and Server in the MariaDB documentation.

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

Use the `require_secure_transport` parameter to require that all user connections to your MariaDB DB instance use SSL/TLS. For versions 11.4 and earlier, the `require_secure_transport` parameter is set to `OFF` by default. For 11.8 and later versions, the default value is set to `ON`, enforcing SSL/TLS for connections to your DB instance. You can change the `require_secure_transport` parameter to `OFF` if non-secure connections are needed.

> **Note**
>
> The `require_secure_transport` parameter is only supported for MariaDB version 10.5 and higher.

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:

```
ERROR 1045 (28000): Access denied for user 'USER'@'localhost' (using password: YES |
 NO)
```

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 MariaDB documentation.

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

The `mysql` client program parameters are slightly different if you are using the MySQL 5.7 version, the MySQL 8.0 version, or the MariaDB version.

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 MariaDB client:

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

   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
```

3. Enter the master user password when prompted.

You should see output similar to the following.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.6.10-MariaDB-log Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Updating applications to connect to MariaDB 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 your applications require certificate verification to connect to your DB instances.

> **Note**
>
> Some applications are configured to connect to MariaDB 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 still connects successfully.

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 MariaDB DB instances, see SSL/TLS support for MariaDB DB instances on Amazon RDS.

Topics

Determining whether a client requires certificate verification in order 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 server CA 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.
> Specify a password other than the prompt shown here as a security best practice.

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 5.7 or MySQL 8.0 Client, 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
```

When using the MySQL 5.6 Client, an SSL connection requires verification against the server CA certificate if you specify the `--ssl-verify-server-cert` option, as in the following example.

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

Updating your application trust store

For information about updating the trust store for MySQL applications, see Using TLS/SSL with MariaDB Connector/J in the MariaDB 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 MariaDB Connector/J 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_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

> **Note**
>
> Specify passwords other than the prompts 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 using JDBC.

```
private static final String DB_USER = "admin";
```

```
        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void main(String[] args) throws Exception {
        Class.forName("org.mariadb.jdbc.Driver");

        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

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


        Connection connection = DriverManager.getConnection("jdbc:mysql://ssl-mariadb-
public.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306?useSSL=true",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        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.

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

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

Topics

Overview of RDS Optimized Reads

When you use an RDS for MariaDB DB instance that has RDS Optimized Reads turned on, your DB instance achieves faster query performance through the use of an instance store. An *instance store* provides temporary block-level storage for your DB instance. 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 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 MariaDB 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 contain only the 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 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.
- Use the default value of `MIXED` for the `binlog_format`. Depending on the size of the transactions, setting `binlog_format` to `ROW` can result in large binlog cache files 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.

Using RDS Optimized Reads

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

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

- Create an RDS for MariaDB DB instance using one of these DB instance classes. For more information, see Creating an Amazon RDS DB instance.
- Modify an existing RDS for MariaDB DB instance 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 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 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 RDS for MariaDB versions:
  - All available minor versions of 11.4 and higher major releases
  - 10.11.4 and higher 10.11 versions
  - 10.6.7 and higher 10.6 versions
  - 10.5.16 and higher 10.5 versions
  - 10.4.25 and higher 10.4 versions

  For information about RDS for MariaDB versions, see MariaDB 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 MariaDB, 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 Amazon RDS Optimized Writes for MariaDB

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

Topics

Overview of RDS Optimized Writes

When you turn on RDS Optimized Writes, your RDS for MariaDB 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 MariaDB, provide the *ACID properties* of atomicity, consistency, isolation, and durability for reliable database transactions. To help provide these properties, MariaDB 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 MariaDB 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 MariaDB doublewrite buffer, see InnoDB Doublewrite Buffer in the MariaDB documentation.

With RDS Optimized Writes turned on, RDS for MariaDB 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 MariaDB 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 MariaDB databases. Access this parameter in the DB parameter groups of RDS for MariaDB for the following versions:

- All available minor versions of 11.8 and higher major releases
- 11.4.3 and higher 11.4 versions
- 10.11.4 and higher 10.11 versions
- 10.6.10 and higher 10.6 versions

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 migrate an RDS for MariaDB 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 MariaDB doublewrite buffer.

To determine whether an RDS for MariaDB 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 MariaDB 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 the following RDS for MariaDB versions:

- All available minor versions of 11.8 and higher major releases
- 11.4.3 and higher 11.4 versions
- 10.11.4 and higher 10.11 versions
- 10.6.10 and higher 10.6 versions

For information about RDS for MariaDB versions, see MariaDB on Amazon RDS versions.

- RDS Optimized Writes is supported for RDS for MariaDB 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”.

- 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.

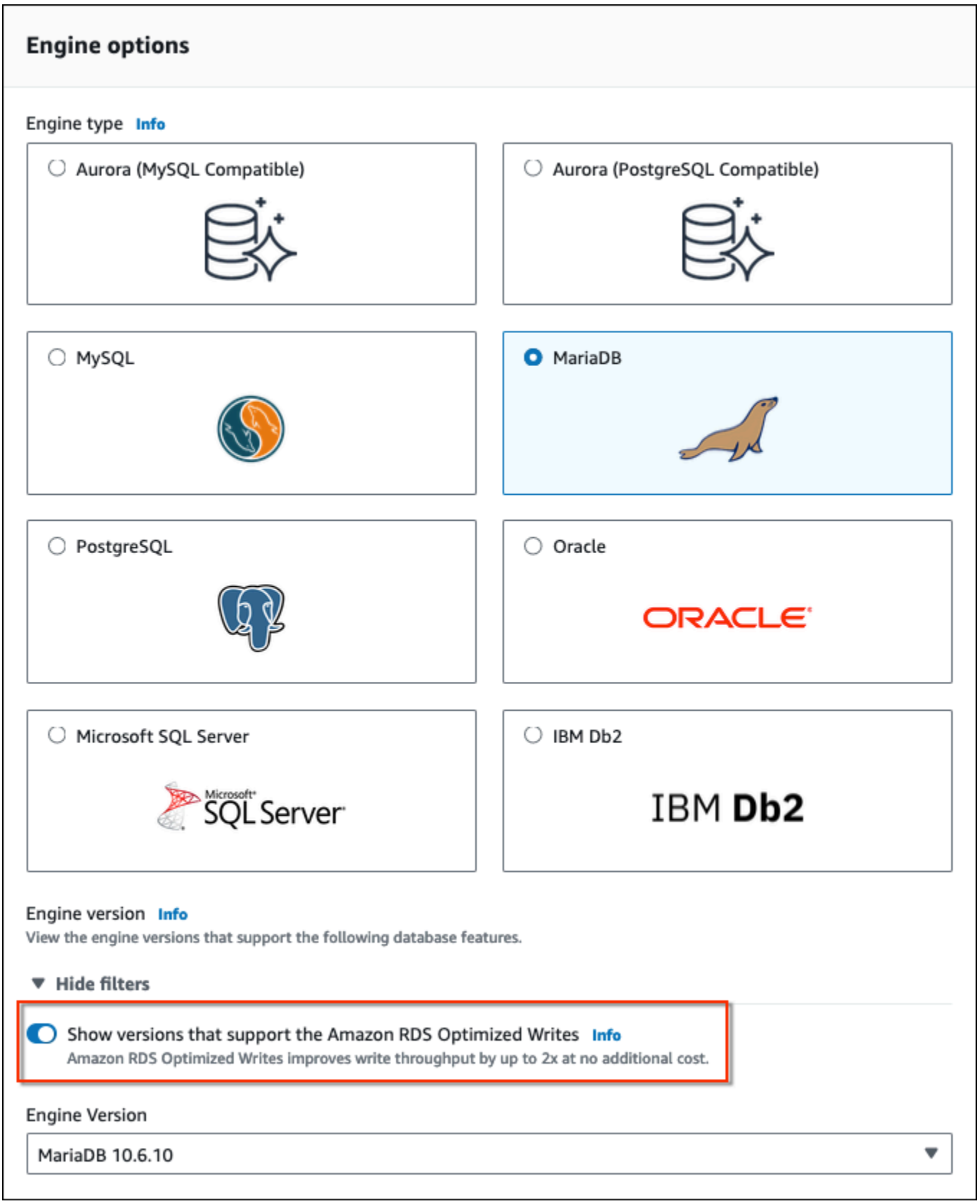
If you want to use a DB engine version and DB instance class that support RDS Optimized Writes, but you don't want to use this feature, then specify a custom parameter group when you create the database. In this parameter group, set the `rds.optimized_writes` parameter to `OFF`. If you want the database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

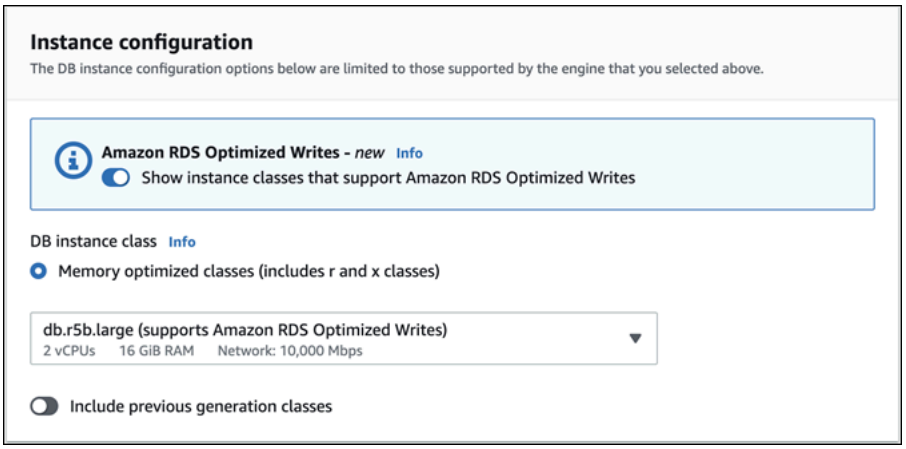
Console

When you use the RDS console to create an RDS for MariaDB 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 MariaDB 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
ORACLE
Microsoft SQL Server
Microsoft SQL Server
IBM Db2
IBM Db2
Engine version Info
View the engine versions that support the following database features.
Hide filters
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
MariaDB 10.6.10

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



After you make these selections, you can choose other settings that meet your requirements and finish creating the RDS for MariaDB 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 mariadb \
    --engine-version 10.6.10 \
    --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 mariadb ^
    --engine-version 10.6.10 ^
    --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 MariaDB 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 March 7, 2023, 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 the section called “Using with a new database”.
- 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 MariaDB 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 MariaDB 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 MariaDB DB instances: major version upgrades and minor 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. However, before you perform a major version upgrade, we recommend that you follow the instructions in Major version upgrades for RDS for MariaDB.

In contrast, *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 your DB instance is automatically upgraded after Amazon RDS tests and approves the new version. For information about performing an upgrade, see Upgrading a DB instance engine version.

If your MariaDB DB instance is using read replicas, you must upgrade all of the read replicas before upgrading the source instance. If your DB instance is in a Multi-AZ deployment, both the writer and standby replicas are upgraded. Your DB instance might not be available until the upgrade is complete.

For more information about MariaDB supported versions and version management, see MariaDB on Amazon RDS versions.

Database engine upgrades require downtime. The duration of the downtime varies based on the size of your DB instance.

> **Tip**
>
> You can minimize the downtime required for DB instance upgrade by using a blue/green deployment. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

Topics

- Considerations for MariaDB upgrades

- Finding valid upgrade targets
- MariaDB version numbers
- RDS version numbers in RDS for MariaDB
- Major version upgrades for RDS for MariaDB
- Upgrading a MariaDB DB instance
- Automatic minor version upgrades for RDS for MariaDB
- Using a read replica to reduce downtime when upgrading an RDS for MariaDB database
- Monitoring RDS for MariaDB DB engine upgrades with events

Considerations for MariaDB 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 is using read replication, you must upgrade all of the Read Replicas before upgrading the source instance.

If your DB instance is in a Multi-AZ deployment, both the primary and standby DB instances are upgraded. The primary and standby DB instances are upgraded at the same time and you will experience an outage until the upgrade is complete. The time for the outage varies based on your database engine, engine version, and the size of your DB 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 mariadb \
  --engine-version version_number \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

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

For example, to identify the valid upgrade targets for a MariaDB version 10.5.17 DB instance, run the following AWS CLI command:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
  --engine mariadb \
  --engine-version 10.5.17 \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
  --engine mariadb ^
  --engine-version 10.5.17 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

MariaDB version numbers

The version numbering sequence for the RDS for MariaDB database engine is either in the form of *major.minor.patch.YYYYMMDD* or *major.minor.patch*, for example, 10.11.5.R2.20231201 or 10.4.30. The format used depends on the MariaDB engine version.

major

The major version number is both the integer and the first fractional part of the version number, for example, 10.11. A major version upgrade increases the major part of the version number. For example, an upgrade from *10.5*.20 to 10.6.12 is a major version upgrade, where *10.5* and *10.6* are the major version numbers.

minor

The minor version number is the third part of the version number, for example, the 5 in 10.11.5.

patch

The patch is the fourth part of the version number, for example, the R2 in 10.11.5.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 10.11.5.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 MariaDB version 10.11.

10.11 minor version	Naming scheme
≥5	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.11.5.R2.20231201.

10.11 minor version	Naming scheme
	Existing DB instances might use *major.minor.patch*, for example, 10.11.5.R2, until your next major or minor version upgrade.
< 5	Existing DB instances use *major.minor.patch*, for example, 10.11.4.R2.

The following table explains the naming scheme for RDS for MariaDB version 10.6.

10.6 minor version	Naming scheme
≥ 14	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.6.14.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 10.6.14.R2, until your next major or minor version upgrade.
< 14	Existing DB instances use *major.minor.patch*, for example, 10.6.13.R2.

The following table explains the naming scheme for RDS for MariaDB version 10.5.

10.5 minor version	Naming scheme
≥ 21	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.5.21.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 10.5.21.R2, until your next major or minor version upgrade.
< 21	Existing DB instances use *major.minor.patch*, for example, 10.5.20.R2.

The following table explains the naming scheme for RDS for MariaDB version 10.4.

10.4 minor version	Naming scheme
≥ 30	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.4.30.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 10.4.30.R2, until your next major or minor version upgrade.
< 30	Existing DB instances use *major.minor.patch*, for example, 10.4.29.R2.

RDS version numbers in RDS for MariaDB

RDS version numbers use either the *major*.*minor*.*patch* or the *major*.*minor*.*patch*.*YYYYMMDD* naming scheme. An RDS patch version includes important bug fixes added to a minor version after its release. An RDS date version (*YYMMDD*) is a security patch. A security patch doesn't include any fixes that might change the engine's behavior.

To identify the Amazon RDS version number of your database, you must first create the `rds_tools` extension by using the following command:

```
CREATE EXTENSION rds_tools;
```

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

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

For example, querying an RDS for MariaDB 10.6.14 database returns the following output:

```
+---------------------+
| mysql.rds_version() |
+---------------------+
| 10.6.14.R2.20231201  |
+---------------------+
```

```
1 row in set (0.01 sec)
```

Major version upgrades for RDS for MariaDB

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.

> **Note**
>
> In MariaDB 11.8, the default value for `require_secure_transport` is now 1, requiring secure SSL/TLS connections. Set to `0` if non-secure connections are needed.

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

- Any MariaDB version to MariaDB 11.8
- Any MariaDB version to MariaDB 11.4
- Any MariaDB version to MariaDB 10.11
- Any MariaDB version to MariaDB 10.6
- MariaDB 10.4 to MariaDB 10.5

If you are using a custom parameter group, and you perform a major version upgrade, you must specify either a default parameter group for the new DB engine version or create your own custom parameter group for the new DB engine version. Associating the new parameter group with the DB instance requires a customer-initiated database reboot after the upgrade completes. The instance's parameter group status will show `pending-reboot` if the instance needs to be rebooted to apply the parameter group changes. An instance's parameter group status can be viewed in the AWS Management Console or by running a "describe" call such as `describe-db-instances`.

Upgrading a MariaDB DB instance

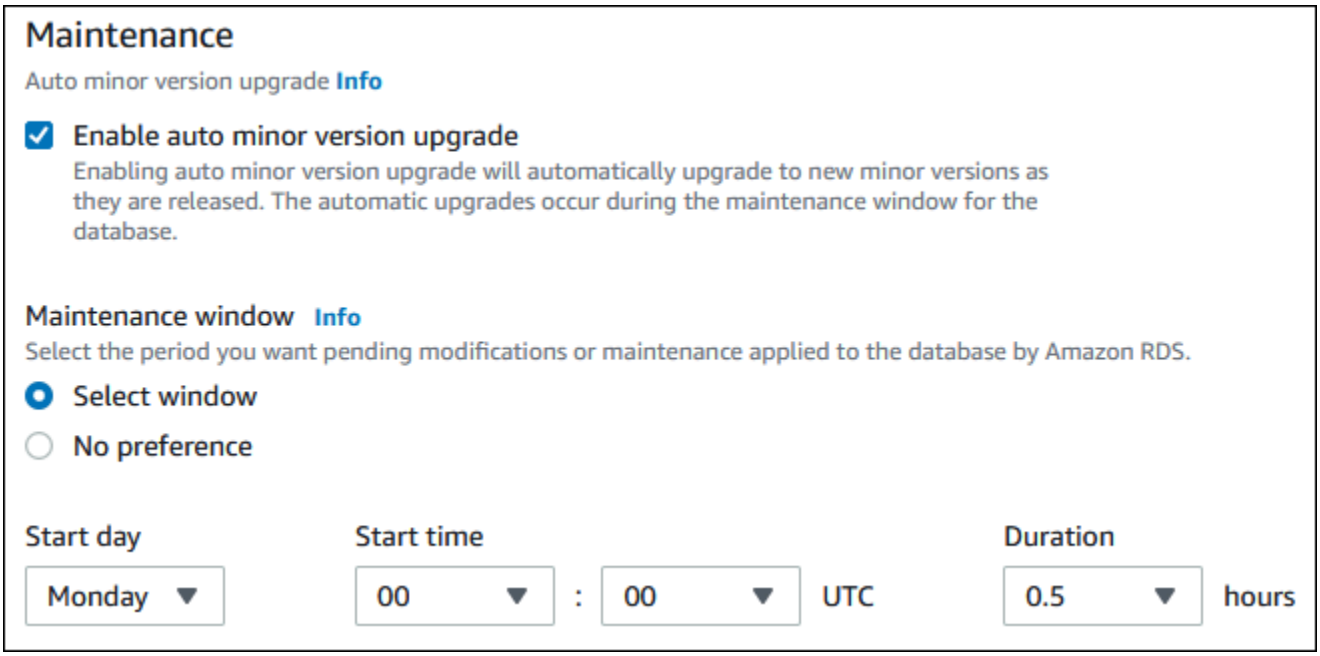
For information about manually or automatically upgrading a MariaDB DB instance, see Upgrading a DB instance engine version.

Automatic minor version upgrades for RDS for MariaDB

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.



For more information about these settings, see Settings for DB instances.

For some RDS for MariaDB 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 MariaDB community version
- Overall fleet stability since the minor version was released

> **Note**
>
> Support for using TLS version 1.0 and 1.1 was removed starting with specific minor versions of MariaDB. For information about supported MariaDB minor versions, see the section called "SSL/TLS support for MariaDB".

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

For Linux, macOS, or Unix:

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

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mariadb ^
--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 MariaDB minor version 10.5.16 in the US East (Ohio) AWS Region (us-east-2).

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mariadb \
--engine-version 10.5.16 \
--region us-east-2 \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
```

```
--output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mariadb ^
--engine-version 10.5.16 ^
--region us-east-2 ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output table
```

Your output is similar to the following.

```
---------------------------------
|    DescribeDBEngineVersions    |
+--------------+-----------------+
|  AutoUpgrade |  EngineVersion  |
+--------------+-----------------+
|  True        |  10.5.17        |
|  False       |  10.5.18        |
|  False       |  10.5.19        |
|  False       |  10.6.5         |
|  False       |  10.6.7         |
|  False       |  10.6.8         |
|  False       |  10.6.10        |
|  False       |  10.6.11        |
|  False       |  10.6.12        |
+--------------+-----------------+
```

In this example, the `AutoUpgrade` value is `True` for MariaDB version 10.5.17. So, the automatic minor upgrade target is MariaDB version 10.5.17, which is highlighted in the output.

A MariaDB 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 MariaDB database

In most cases, a blue/green deployment is the best option to reduce downtime when upgrading a MariaDB 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 MariaDB 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 MariaDB version 10.5 to MariaDB version 10.6. You can use the same general steps for upgrades to other major versions.

To upgrade a MariaDB 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 MariaDB 10.5 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 MariaDB 10.5 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 as a Single-AZ deployment 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 now.

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 MariaDB 10.6.

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 MariaDB 10.6 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 MariaDB 10.5 DB instance. To verify, connect to the read replica and run the `SHOW REPLICA STATUS` command. If the `Seconds_Behind_Master` field is `0`, then replication is up-to-date.

> **Note**
>
> Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version before 10.6, 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 MariaDB 10.5 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 MariaDB 10.6. 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 MariaDB 10.6 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 MariaDB 10.6 read replica a standalone DB instance.

   Important

   When you promote your MariaDB 10.6 read replica to a standalone DB instance, it is no longer a replica of your MariaDB 10.5 DB instance. We recommend that you promote your MariaDB 10.6 read replica during a maintenance window when your source MariaDB 10.5 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 MariaDB 10.6 DB instance to ensure that no write operations are lost. In addition, we recommend that, before promoting your MariaDB 10.6 read replica, you perform all necessary data definition language (DDL) operations on your MariaDB

10.6 read replica. An example is creating indexes. This approach avoids negative effects on the performance of the MariaDB 10.6 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 MariaDB database. At this point, you can direct your applications to the new MariaDB 10.6 DB instance.

Monitoring RDS for MariaDB DB engine upgrades with events

When you upgrade the engine version of an RDS for MariaDB 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 MariaDB DB snapshot engine version

With Amazon RDS, you can create a storage volume DB snapshot of your MariaDB 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 MariaDB, you can upgrade to all available engine versions. You can upgrade encrypted or unencrypted DB snapshots.

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

```
aws rds describe-db-engine-versions --engine mariadb --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 MariaDB.

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 MariaDB 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 MariaDB

The following table shows which engine versions you can upgrade to from an unsupported engine version for RDS for MariaDB 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
10.0.17, 10.0.24, 10.0.28, 10.0.31, 10.0.32, 10.0.34, 10.0.35	10.0.38, 10.1.48, 10.6.19, 10.6.20, 10.11.9, 10.11.10, 11.4.3, 11.4.4, 11.8.3
10.1.14, 10.1.16, 10.1.19, 10.1.23, 10.1.26, 10.1.31, 10.1.34	10.1.48, 10.2.44, 10.6.19, 10.6.20, 10.11.9, 10.11.10, 11.4.3, 11.4.4, 11.8.3
10.2.11, 10.2.12, 10.2.15, 10.2.21, 10.2.32, 10.2.37, 10.2.39, 10.2.40, 10.2.41, 10.2.43	10.2.44, 10.3.39, 10.6.19, 10.6.20, 10.6.21, 10.11.9, 10.11.10, 10.11.11, 11.4.3, 11.4.4, 11.8.3
10.3.8, 10.3.13, 10.3.20, 10.3.23, 10.3.28, 10.3.31, 10.3.32	10.3.39, 10.4.34, 10.6.19, 10.6.20 10.6.21, 10.11.9, 10.11.10, 10.11.11, 11.4.3, 11.4.4, 11.8.3
10.4.8	10.4.34, 10.5.16, 10.5.17, 10.5.18, 10.5.20, 10.5.21, 10.5.22, 10.5.23, 10.5.24, 10.5.25, 10.5.26, 10.5.27, 10.5.28, 10.6.8, 10.6.10, 10.6.11, 10.6.13, 10.6.14, 10.6.15, 10.6.16, 10.6.18, 10.6.19, 10.6.20, 10.6.21, 10.11.4, 10.6.17, 10.11.5, 10.11.6, 10.11.7, 10.11.8, 10.11.9, 10.11.10, 10.11.11

Importing data into an Amazon RDS for MariaDB DB instance

You can use several different techniques to import data into an RDS for MariaDB 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 MariaDB DB instance:

> **Note**
>
> Amazon RDS doesn't support `mariadb-backup` or importing from Amazon S3 for MariaDB.

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
Existing MariaDB database on premises or on Amazon EC2	Any	Ongoing	Minimal	Configure replication with an existing MariaDB database as the replication source. You can configure replication into a MariaDB DB instance using MariaDB global transaction identifiers (GTIDs) when the external instance is MariaDB version 10.0.24 or higher, or using binary log coordinates for MariaDB instances on earlier versions than 10.0.24. MariaDB GTIDs are implemented differently than	Configuri ng binary log file position replicati on with an external source instance

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
				MySQL GTIDs, which aren't supported by Amazon RDS.	Importing data to an Amazon RDS for MariaDB DB instance with reduced downtime

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
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-com patible database as a target for AWS DMS in the *AWS Database Migration Service User Guide*
Existing MariaDB 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	Applicati on downtim	Technique	More informati on
Existing MariaDB database	Small	One time	Some	Copy the data directly to your MariaDB DB instance using a command-line utility.	Importing data from an external MariaDB database to an Amazon RDS for MariaDB DB instance
Data not stored in an existing database	Medium	One time	Some	Create flat files and import them using MariaDB `LOAD DATA LOCAL INFILE` statements.	Importing data from any source to an Amazon RDS for MariaDB 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 errors 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 restore-db-instance-to-point-in-time command.

Importing data considerations for MariaDB

The following content contains technical information related to loading data into MariaDB. This content is aimed at users who are familiar with the MariaDB 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 MariaDB 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, MariaDB extracts the data from the binary log into a flat file. MariaDB 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 MariaDB 10.11 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 MariaDB 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

MariaDB 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 mariadb-dump files. If a failure occurs while loading mariadb-dump 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—without 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 MariaDB databases. Unlike other database systems, MariaDB 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. `mariadb-dump` 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

Importing data from an external MariaDB database to an Amazon RDS for MariaDB DB instance

You can import data from an existing MariaDB database to an RDS for MariaDB DB instance. You do so by copying the database with mysqldump or mariadb-dump, and piping the database directly into the RDS for MariaDB DB instance. The `mysqldump` or `mariadb-dump` command line utility is commonly used to make backups and transfer data from one MariaDB server to another. It's included with MariaDB client software.

Starting with MariaDB 11.0.1, you must use `mariadb-dump` instead of `mysqldump`.

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. For MariaDB 11.0.1 and higher versions, replace `mysqldump` with `mariadb-dump`.

```
mysqldump -u local_user \
    --databases database_name \
    --single-transaction \
    --compress \
    --order-by-primary  \
    --routines=0 \
    --triggers=0 \
    --events=0 \
    -plocal_password | mariadb -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` and `mariadb-dump` 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 MariaDB database that you identify with the `--databases` parameter.

- `--databases` *database_name* – Use to specify the name of the database on the local MariaDB 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` or `mariadb-dump` 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 MariaDB DB instance that you identify with the `--host` parameter.
- `--port` *port_number* – Use to specify the port for your MariaDB 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` or `mariadb-dump`. To do so, include the following parameters with your `mysqldump` or `mariadb-dump` command:

- `--routines=0`
- `--triggers=0`
- `--events=0`

Example

The following example copies the `world` sample database on the local host to an RDS for MariaDB DB instance. Replace values with your own information.

For Linux, macOS, or Unix:

```
sudo mariadb-dump -u local_user \
    --databases world \
    --single-transaction \
    --compress \
    --order-by-primary  \
    --routines=0 \
    --triggers=0 \
    --events=0 \
    -plocal_password | mariadb -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.

```
mariadb-dump -u local_user ^
    --databases world ^
    --single-transaction ^
    --compress ^
    --order-by-primary  ^
    --routines=0 ^
    --triggers=0 ^
```

```
    --events=0 ^
    -plocal_password | mariadb -u RDS_user ^
        --port=3306 ^
        --host=my_instance.123456789012.us-east-1.rds.amazonaws.com ^
        -pRDS_password
```

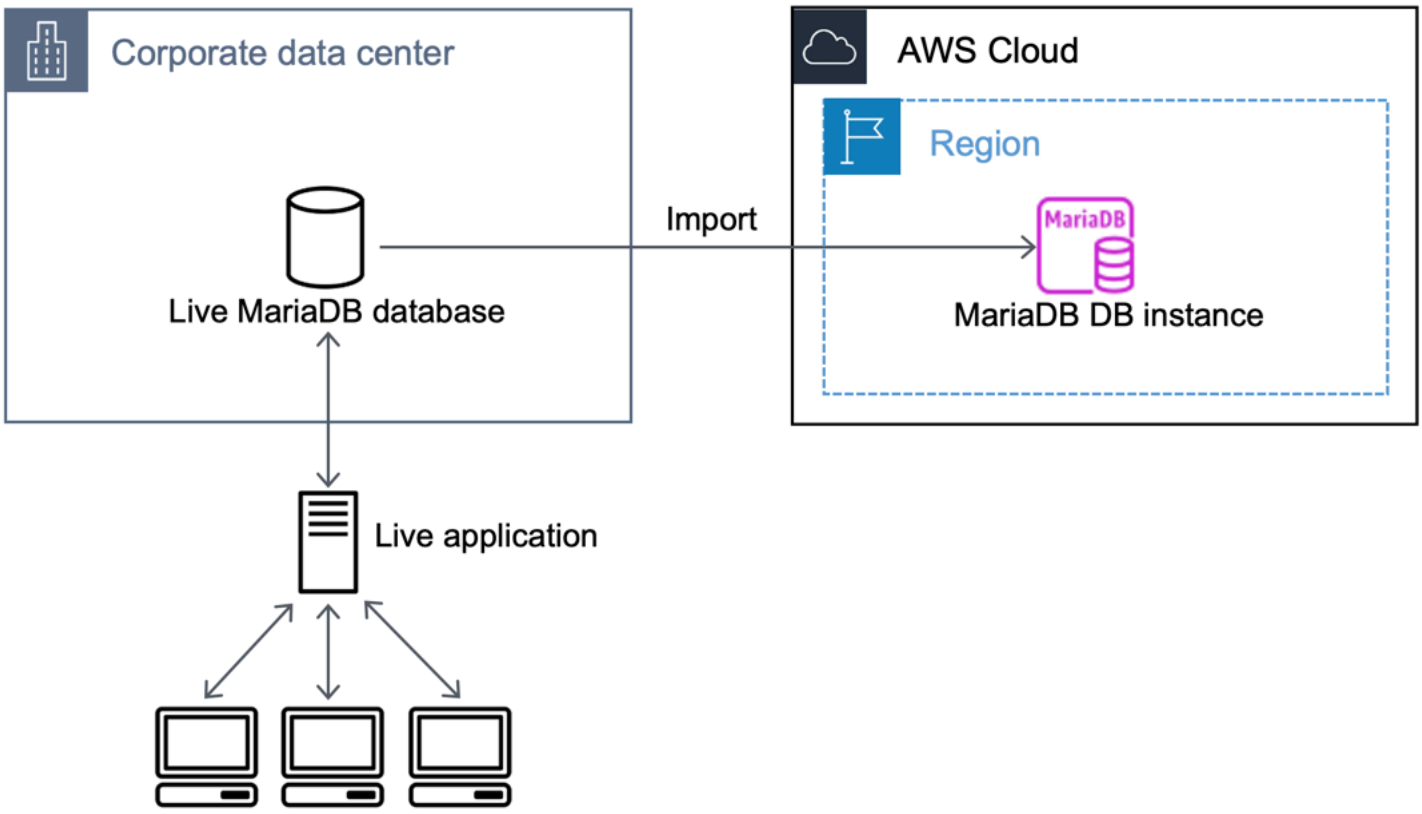
> **Note**
>
> As a security best practice, specify credentials other than the prompts shown in the example.

Importing data to an Amazon RDS for MariaDB DB instance with reduced downtime

In some cases, you might need to import data from an external MariaDB database that supports a live application to an RDS for MariaDB DB instance. 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. If the external instance is MariaDB 10.0.24 or higher and the target instance is RDS for MariaDB, configure MariaDB replication based on global transaction identifiers (GTIDs). Otherwise, configure replication based on binary log coordinates. We recommend GTID-based replication if your external database supports it because GTID-based replication is a more reliable method. For more information, see Global transaction ID in the MariaDB documentation.

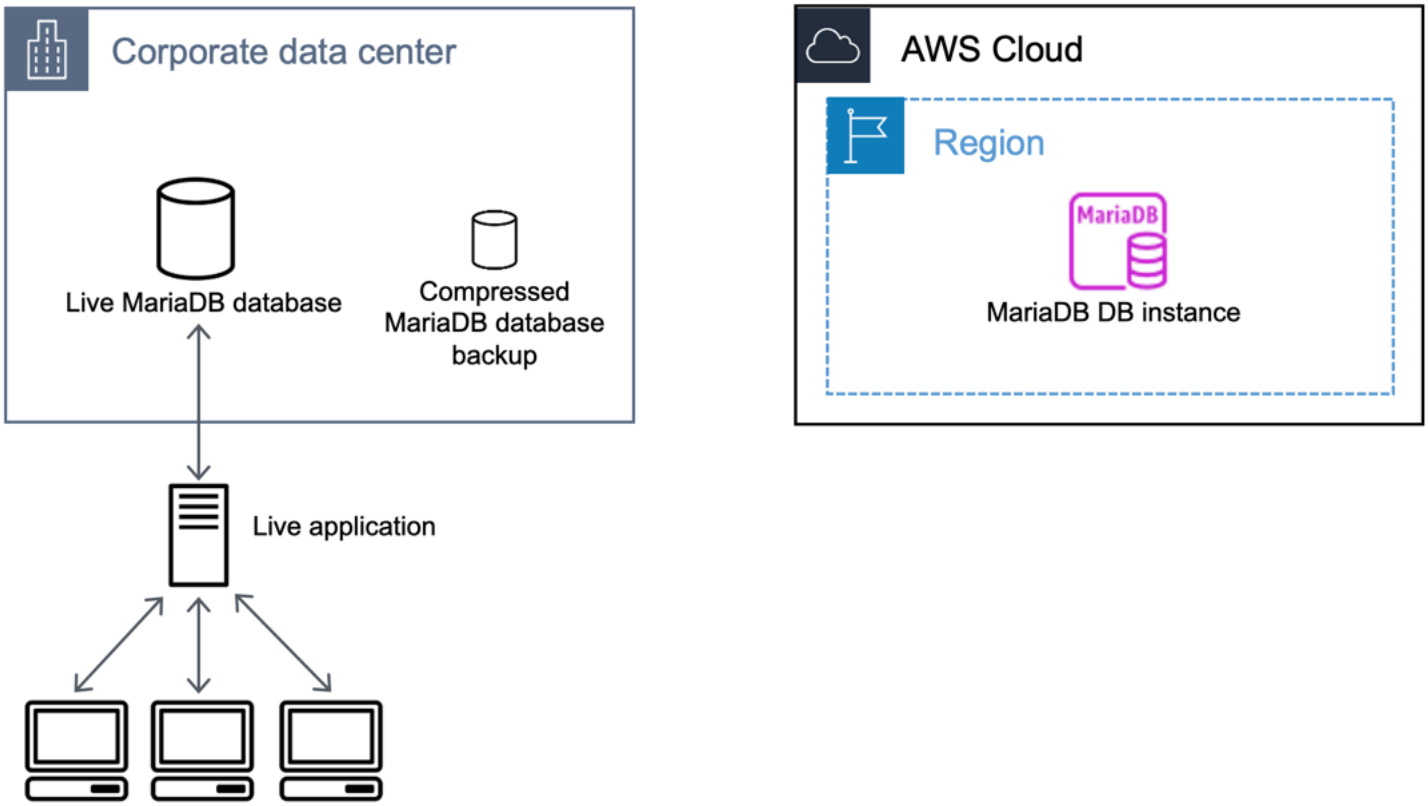
The following diagram shows importing an external MariaDB database into a MariaDB database on Amazon RDS.



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 MariaDB database with minimal downtime is to create a copy of the source data.

The following diagram shows creating a backup of the MariaDB database.



You can use the `mysqldump` or `mariadb-dump` utility to create a database backup in either SQL or delimited-text format. In MariaDB 10.5, the client is called mariadb-dump. Starting with MariaDB 11.0.1, you must use `mariadb-dump` instead of `mysqldump`. 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` or `mariadb-dump` runs.

We also recommend that you weigh `mysqldump` or `mariadb-dump` 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 MariaDB 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` or `mariadb-dump` 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` or `mariadb-dump` 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` and `mariadb-dump` utilities exclude 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 `[mariadb]` section of a `my.cnf` file:

   ```
   [mariadb]
   log-bin
   server-id=1
   log-basename=master1
   binlog-format=mixed
   ```

   For more information, see Setting the Replication Source Configuration in the MariaDB documentation.
2. For replication with a Multi-AZ DB cluster, enable `gtid_strict_mode`. For more information, see gtid_strict_mode in the MariaDB documentation.

   Enabling `gtid_strict_mode` isn't required for replication with a DB instance.

3. Restart the `mariadb` service.

```
sudo service mariadb restart
```

To create a backup copy of your existing database

1. Create a backup of your data using the `mysqldump` or `mariadb-dump` utility, specifying either SQL or delimited-text format.

   To improve performance and ensure data integrity, use the `--order-by-primary` and `--single-transaction` options for `mysqldump` and `mariadb-dump`.

   To avoid including the MySQL system database in the backup, don't use the `--all-databases` option with `mysqldump` or `mariadb-dump`. 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 MariaDB 10.11 versions and lower, replace `mariadb-dump` with `mysqldump`.

     For Linux, macOS, or Unix:

     ```
     sudo mariadb-dump \
         --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:

```
mariadb-dump ^
    --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 MariaDB 11.01 and higher versions, replace `mysqldump` with `mariadb-dump`.

  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` or `mariadb-dump`. To do so, include the following arguments with your `mysqldump` or `mariadb-dump` command:

- `--routines=0`
- `--triggers=0`
- `--events=0`

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. If the external instance is a MariaDB 10.0.23 or lower version, 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 MariaDB 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;
```

2. If the external instance you are using is MariaDB version 10.0.24 or higher, use GTID-based replication. Run `SHOW MASTER STATUS` on the external MariaDB instance to get the binary log file name and position, and then convert them to a GTID by running `BINLOG_GTID_POS` on the external MariaDB instance.

   ```
   SELECT BINLOG_GTID_POS('binary_log_file_name', binary_log_file_position);
   ```

   Note the GTID returned. You need the GTID to configure replication.

3. 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
     ```

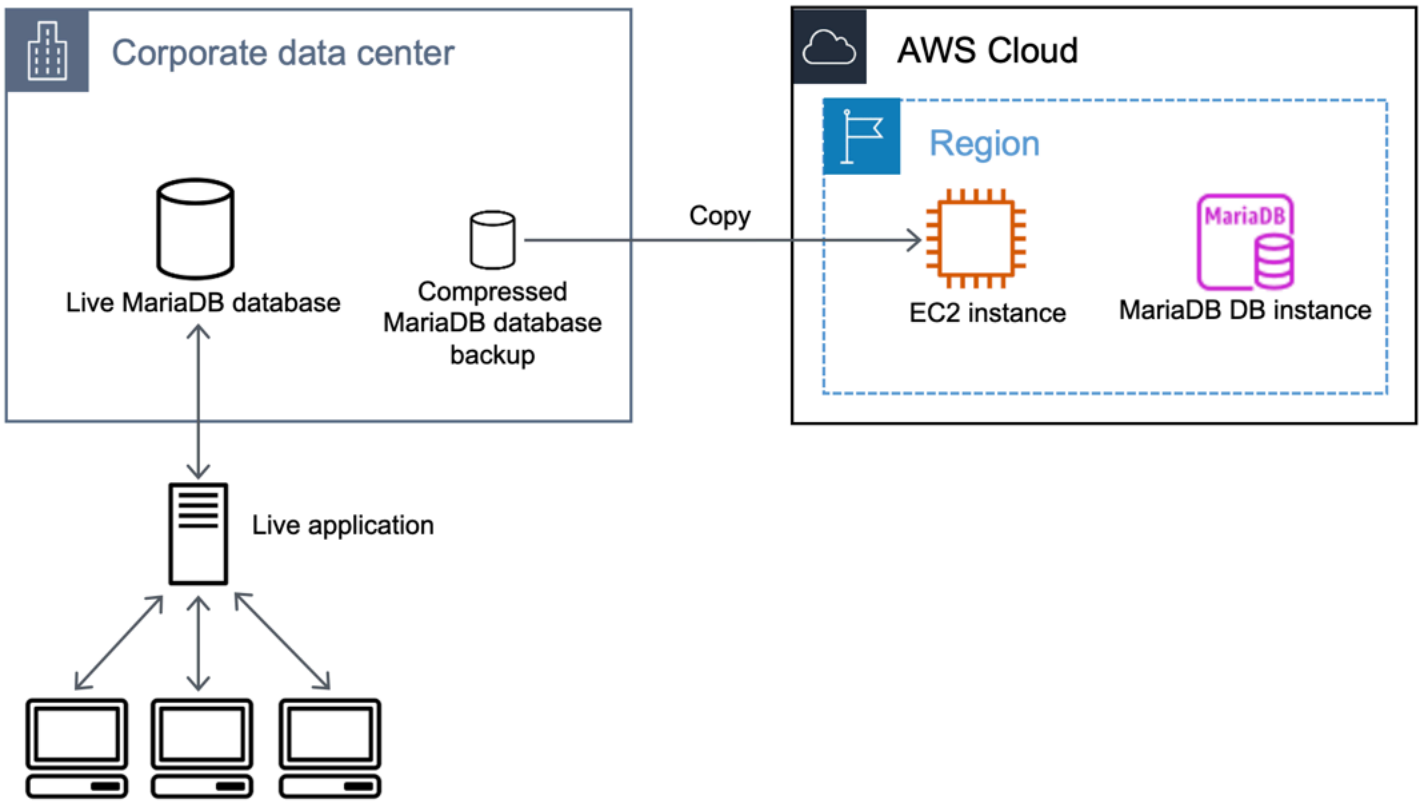
   - To compress delimited-text output, use the following command:

     ```
     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 MariaDB 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 MariaDB client tools using the following commands:

```
sudo yum update -y
sudo yum install mariadb1011-client-utils -y
```

For more information, see Connect to your instance for Linux instances in the *Amazon Elastic Compute Cloud User Guide* and MariaDB Connectors in the MariaDB 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
```

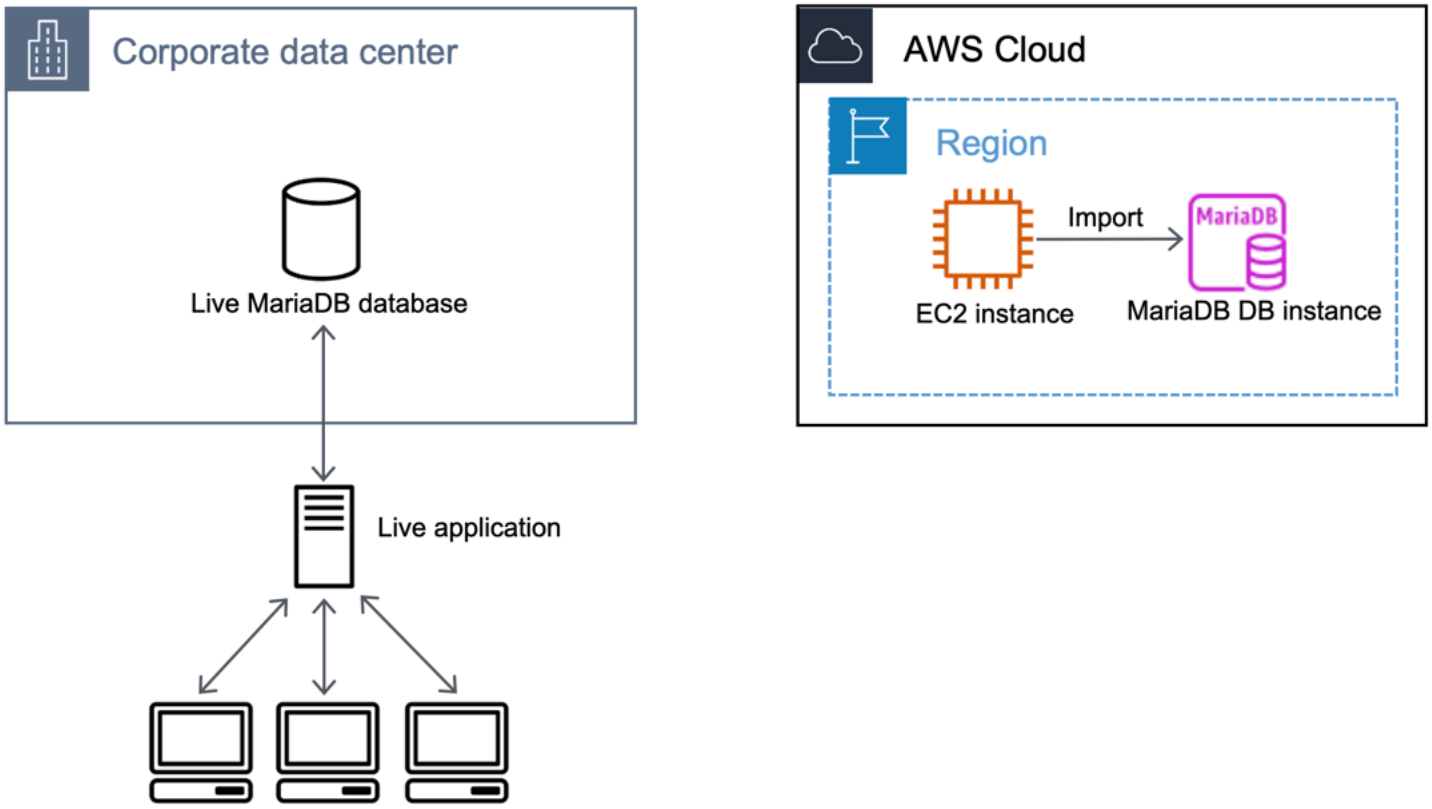
- To decompress delimited-text output, use the following command:

```
tar xzvf backup.tar.gz
```

Task 3: Create a MariaDB database and import data from your Amazon EC2 instance

By creating an RDS for MariaDB DB instance 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 MariaDB database.



To create a MariaDB 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 MariaDB database. For more information, see DB instance classes.
2. Create a DB instance in the AWS Region that contains your Amazon EC2 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, `mysampledb.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:

     ```
     MariaDB [(none)]> 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.

```
MariaDB [(none)]> create database database_name;
MariaDB [(none)]> use database_name;
```

Then create the tables.

```
MariaDB [(none)]> source table1.sql
MariaDB [(none)]> source table2.sql
etc...
```

Then import the data.

```
MariaDB [(none)]> LOAD DATA LOCAL INFILE 'table1.txt' INTO TABLE table1 FIELDS
 TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '0x0d0a';
MariaDB [(none)]> 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.

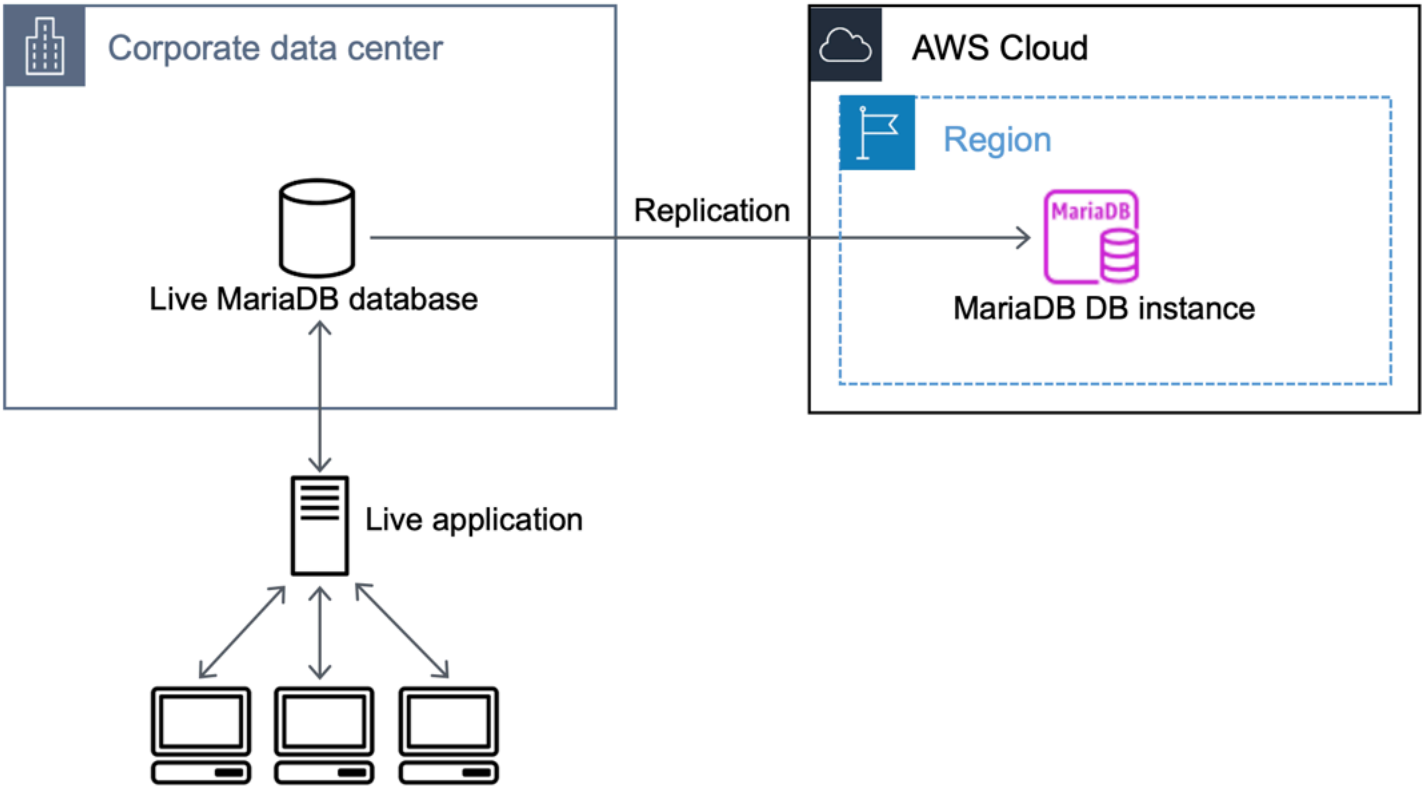
> **Note**
>
> If you used any data-formatting options with `mysqldump` or `mariadb-dump` when you initially dumped the table, make sure to use the same options with `LOAD DATA LOCAL INFILE` to ensure proper interpretation of the data file contents.

8. 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 MariaDB database. 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:

- mysql.rds_set_external_master
- 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. If you used SQL format to create your backup file and the external instance isn't MariaDB 10.0.24 or higher, look at the contents of that file by running the following command:

   ```
   cat backup.sql
   ```

The file includes a `CHANGE MASTER TO` comment that contains the master log file name and position. This comment is included in the backup file when you use the `--master-data` option with `mysqldump`. Note the values for `MASTER_LOG_FILE` and `MASTER_LOG_POS`.

```
--
-- Position to start replication or point-in-time recovery from
--

-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;
```

If you used delimited text format to create your backup file and the external instance isn't MariaDB 10.0.24 or higher, you should already have binary log coordinates from Step 1 in Task 1 when you created a backup copy of your existing database.

If the external instance is MariaDB 10.0.24 or higher, you should already have the GTID from which to start replication from Step 2 in Task 1 when you created a backup copy of your existing database.

5. Make the Amazon RDS database the replica. If the external instance isn't MariaDB 10.0.24 or higher, connect to the Amazon RDS database as the master user and identify the source database as the source replication instance by using the mysql.rds_set_external_master stored procedure.

   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 command is an example:

   ```
   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.

   If the external instance is MariaDB 10.0.24 or higher, connect to the Amazon RDS database as the master user and identify the source database as the source replication instance by using the mysql.rds_set_external_master_gtid stored procedure. Use the GTID that you determined in Step

2 in Task 1 when you created a backup copy of your existing database. The following command is an example:

```
CALL mysql.rds_set_external_master_gtid ('source_server_ip_address', 3306,
 'ReplicationUser', 'password', 'GTID', 1);
```

The *source_server_ip_address* is the IP address of source replication instance. An EC2 private DNS address isn't currently supported.

> **Note**
>
> Specify credentials other than the prompts shown here as a security best practice.

6. 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;
```

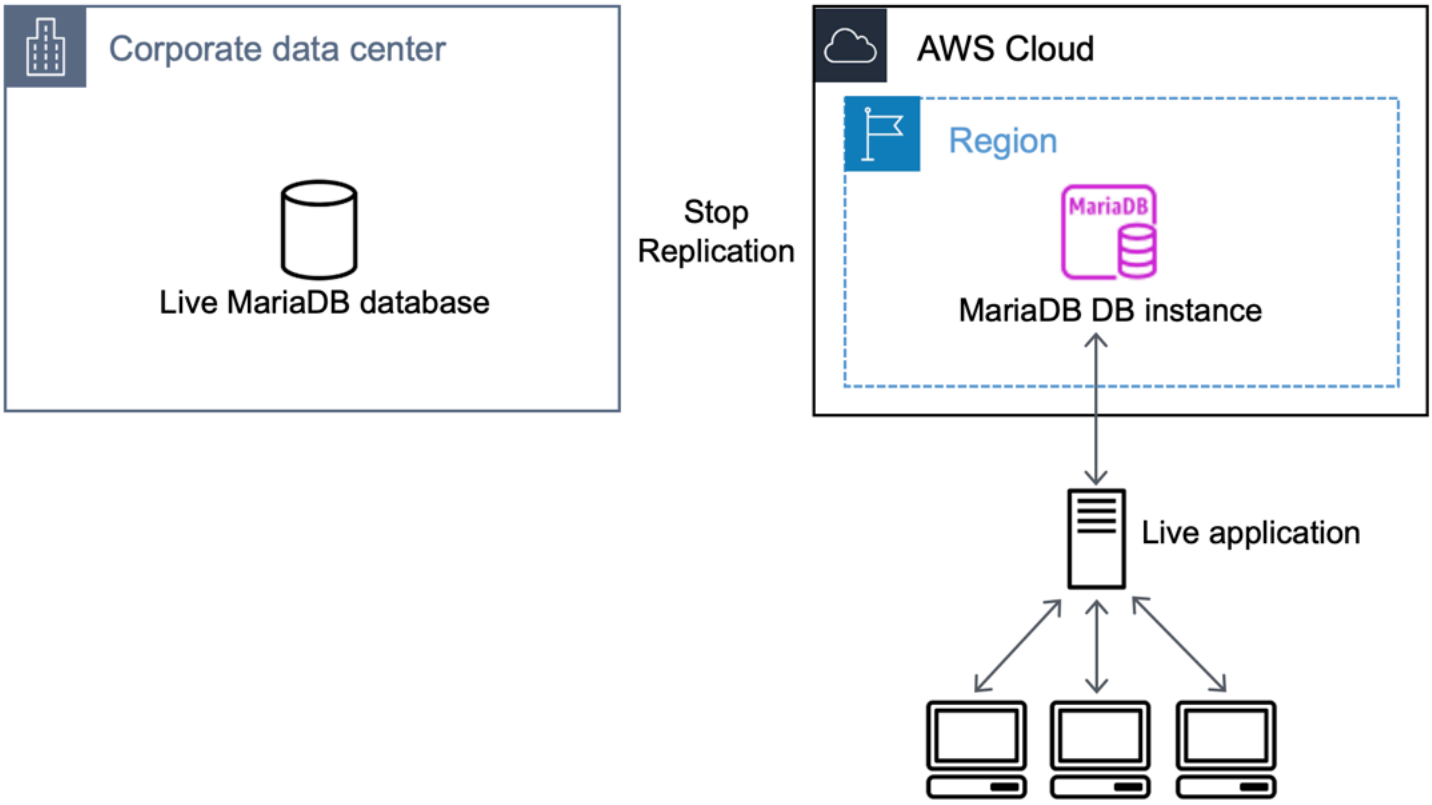
7. On the Amazon RDS database, to determine when the replica is up-to-date with the source replication instance, run the SHOW REPLICA STATUS command. The results of the `SHOW REPLICA STATUS` command include the `Seconds_Behind_Master` field. When the `Seconds_Behind_Master` field returns 0, then the replica is up-to-date with the source replication instance.

   For a MariaDB 10.5, 10.6, 10.11, 11.4, or 11.8 DB instance, use the mysql.rds_replica_status stored procedure instead of running the MySQL command.

8. 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 MariaDB 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 MariaDB 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 REPLICA STATUS command results is 0, which indicates that the replica is up-to-date with the source replication instance.

   ```
   SHOW REPLICA STATUS;
   ```

   For a MariaDB 10.5, 10.6, 10.11, 11.4, or 11.8 DB instance, use the mysql.rds_replica_status procedure instead of running the MySQL command.
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.

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 running the following command that uses the mysql.rds_reset_external_master stored procedure on your Amazon RDS database:

```
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 MariaDB DB instance

With Amazon RDS, you can migrate existing MariaDB data from any source to an RDS for MariaDB DB instance. You can transfer data from on-premises databases, other cloud providers, or existing RDS for MariaDB DB instances to your target RDS for MariaDB 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 MariaDB servers to fully managed Amazon RDS DB instances, consolidating multiple MariaDB databases into a single DB instance, or creating test environments with production data. The following sections provide step-by-step instructions for importing your MariaDB data using methods such as `mariadb-dump`, 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 MariaDB 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:DBInstanceStatus}"
```

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 MariaDB `LOAD DATA LOCAL INFILE` statement.

> **Note**
>
> You must invoke the `mariadb` 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:

```
MariaDB [(none)]> 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 INFILE in the MariaDB 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 MariaDB 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 MariaDB, see Working with MariaDB read replicas.

You can also configure replication based on binary log coordinates for a MariaDB DB instance. For MariaDB instances, you can also configure replication based on global transaction IDs (GTIDs), which provides better crash safety. For more information, see Configuring GTID-based replication with an external source instance.

The following are other replication options available with RDS for MariaDB:

- You can set up replication between an RDS for MariaDB DB instance and a MySQL or MariaDB 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.
- You can configure replication to import databases from a MySQL or MariaDB 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 MariaDB DB instance with reduced downtime and Exporting data from a MySQL DB instance by using replication.

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 MariaDB binary logging.

For information about replication compatibility between MariaDB versions, see Replication Compatibility in the MariaDB documentation.

Topics

- Working with MariaDB read replicas

- Configuring GTID-based replication with an external source instance
- Configuring binary log file position replication with an external source instance

Working with MariaDB read replicas

Following, you can find specific information about working with read replicas on Amazon RDS for MariaDB. For general information about read replicas and instructions for using them, see Working with DB instance read replicas.

- Configuring replication filters with MariaDB
- Configuring delayed replication with MariaDB
- Updating read replicas with MariaDB
- Working with Multi-AZ read replica deployments with MariaDB
- Using cascading read replicas with RDS for MariaDB
- Monitoring MariaDB read replicas
- Starting and stopping replication with MariaDB read replicas
- Troubleshooting a MariaDB read replica problem

Configuring read replicas with MariaDB

Before a MariaDB DB instance can serve as a replication source, make sure to turn on automatic backups on the source DB instance by setting 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.

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 as 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 MariaDB supports cascading read replicas. To learn how to configure cascading read replicas, see Using cascading read replicas with RDS for MariaDB.

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.

Configuring replication filters with MariaDB

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 MariaDB 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 MariaDB

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, the database that includes the specified tables must be included 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 MariaDB documentation.

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 MariaDB binary logging.

> **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 MariaDB

The following limitations apply to replication filtering for RDS for MariaDB:

- Each replication filtering parameter has a 2,000-character limit.
- Commas aren't supported in replication filters.
- The MariaDB `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 isn't supported for RDS for MariaDB version 10.2.

Replication filtering examples for RDS for MariaDB

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
- Escaping wildcard characters in names
- 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. When you set `replicate-do-db` for a read replica, only the databases specified in the parameter are replicated.

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 `orders` and `returns` 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.orders%,mydb.returns%", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-wild-do-table", "ParameterValue":
 "mydb.orders%,mydb.returns%", "ApplyMethod":"immediate"}]"
```

Example Escaping wildcard characters in names

The following example shows you how to use the escape character \ to escape a wildcard character that is part of a name.

Assume that you have several table names in database `mydb1` that start with `my_table`, and you want to include these tables in replication. The table names include an underscore, which is also a wildcard character, so the example escapes the underscore in the table names.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "[{"ParameterName": "replicate-wild-do-table", "ParameterValue": "my
\_table%", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-wild-do-table", "ParameterValue": "my
\_table%", "ApplyMethod":"immediate"}]"
```

Example Excluding databases from replication

The following example excludes the `mydb1` and `mydb2` 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":
 "mydb1,mydb2", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-ignore-db", "ParameterValue":
 "mydb1,mydb2", "ApplyMethod":"immediate"}]"
```

Example Excluding tables from replication

The following example excludes tables `table1` and `table2` in database `mydb1` from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
```

```
  --parameters "[{"ParameterName": "replicate-ignore-table", "ParameterValue":
 "mydb1.table1,mydb1.table2", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-ignore-table", "ParameterValue":
 "mydb1.table1,mydb1.table2", "ApplyMethod":"immediate"}]"
```

Example Excluding tables from replication using wildcard characters

The following example excludes tables with names that begin with `orders` and `returns` in database `mydb` 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":
 "mydb.orders%,mydb.returns%", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-wild-ignore-table", "ParameterValue":
 "mydb.orders%,mydb.returns%", "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 MariaDB client, connect to the read replica and run the `SHOW REPLICA 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.

> **Note**
>
> Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version before 10.5, then use `SHOW SLAVE STATUS`.

Configuring delayed replication with MariaDB

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.

  To stop replication, use the mysql.rds_stop_replication 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**
>
> - Delayed replication is supported for MariaDB 10.6 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.

Topics

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 MariaDB client, connect to the MariaDB 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 MariaDB 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.

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 MariaDB

Read replicas are designed to support read queries, but you might need occasional updates. For example, you might need to add an index to speed the specific types of queries accessing the replica. You can enable updates by setting the `read_only` parameter to **0** in the DB parameter group for the read replica.

Working with Multi-AZ read replica deployments with MariaDB

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 MariaDB

RDS for MariaDB supports cascading read replicas. With *cascading read replicas*, you can scale reads without adding overhead to your source RDS for MariaDB DB instance.

With cascading read replicas, your RDS for MariaDB 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 MariaDB 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 MariaDB DB instance. For example, suppose that you have an RDS for MariaDB DB instance, `mariadb-main`. You can do the following:

- Starting with `mariadb-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 `mariadb-main`. A complete series of instances from an RDS for MariaDB 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 MariaDB 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 `mariadb-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 `mariadb-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 MariaDB read replicas

For MariaDB 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.

> **Note**
>
> Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version before 10.5, then use `SHOW SLAVE STATUS`.

Common causes for replication lag for MariaDB are the following:

- A network outage.
- Writing to tables with indexes on a read replica. If the `read_only` parameter is not set to 0 on the read replica, it can break replication.
- Using a nontransactional storage engine such as MyISAM. Replication is only supported for the InnoDB storage engine on MariaDB.

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 MariaDB 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 MariaDB DB instance 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 ends 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 ended. You can create a new read replica for the source DB instance to reestablish replication.

Troubleshooting a MariaDB read replica problem

The replication technologies for MariaDB 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 MariaDB documentation, go to Replication overview.

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 MariaDB 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 MariaDB error message is returned, review the error in the MariaDB 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 that is used to specify the maximum size of DML code that can be run on the database. In some cases, the `max_allowed_packet` parameter value in the DB parameter group associated with a source DB instance is smaller than

the `max_allowed_packet` parameter value in the DB parameter group associated with the source's read replica. In these cases, the replication process can throw an error (Packet bigger than 'max_allowed_packet' bytes) and stop replication. You can fix the error by having the source 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. If you are creating indexes on a read replica, you need to have the `read_only` parameter set to **0** to create the indexes. If you are writing to tables on the read replica, it might break replication.
- 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 MariaDB.
- 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 Skipping the current replication error for RDS for MySQL. Otherwise, you can delete the read replica and 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*.

For MariaDB DB instances, in some cases read replicas can't be switched to the secondary if some binary log (binlog) events aren't flushed during the failure. 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.

Configuring GTID-based replication with an external source instance

You can set up replication based on global transaction identifiers (GTIDs) from an external MariaDB instance of version 10.0.24 or higher into an RDS for MariaDB DB instance. Follow these guidelines when you set up an external source instance and a replica on Amazon RDS:

- Monitor failover events for the RDS for MariaDB 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 binary logs (binlogs) on your source instance until you have verified that they have been applied to the replica. This maintenance ensures that you can restore your source instance in the event of a failure.
- Turn on automated backups on your MariaDB DB instance on Amazon RDS. Turning on automated backups ensures that you can restore your replica to a particular point in time if you need to resynchronize your source instance and replica. For information on backups and Point-In-Time Restore, see Backing up, restoring, and exporting data.

> **Note**
>
> The permissions required to start replication on a MariaDB DB instance are restricted and not available to your Amazon RDS master user. Because of this, you must use the Amazon RDS mysql.rds_set_external_master_gtid and mysql.rds_start_replication commands to set up replication between your live database and your RDS for MariaDB database.

To start replication between an external source instance and a MariaDB DB instance on Amazon RDS, use the following procedure.

To start replication

1. Make the source MariaDB instance read-only:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SET GLOBAL read_only = ON;
   ```

2. Get the current GTID of the external MariaDB instance. You can do this by using `mysql` or the query editor of your choice to run `SELECT @@gtid_current_pos;`.

   The GTID is formatted as `<domain-id>-<server-id>-<sequence-id>`. A typical GTID looks something like **`0-1234510749-1728`**. For more information about GTIDs and their component parts, see Global transaction ID in the MariaDB documentation.

3. Copy the database from the external MariaDB instance to the MariaDB DB instance using `mysqldump`. For very large databases, you might want to use the procedure in Importing data to an Amazon RDS for MariaDB DB instance 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. Specify a password other than the prompt shown here as a security best practice.

Use the `--host`, `--user (-u)`, `--port` and `-p` options in the `mysql` command to specify the host name, user name, port, and password to connect to your MariaDB DB instance. The host name is the DNS name from the MariaDB 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 Amazon RDS Management Console.

4. Make the source MariaDB instance writeable again.

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

5. In the Amazon RDS Management Console, add the IP address of the server that hosts the external MariaDB database to the VPC security group for the MariaDB DB instance. For more information on modifying a VPC security group, go to 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 MariaDB DB instance, so that it can communicate with your external MariaDB instance. To find the IP address of the MariaDB DB instance, use the `host` command.

   ```
   host db_instance_endpoint
   ```

   The host name is the DNS name from the MariaDB DB instance endpoint.

6. Using the client of your choice, connect to the external MariaDB instance and create a MariaDB 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 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.

7. For the external MariaDB 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';
```

8. Make the MariaDB DB instance the replica. Connect to the MariaDB DB instance as the master user and identify the external MariaDB database as the replication source instance by using the mysql.rds_set_external_master_gtid command. Use the GTID that you determined in Step 2. The following is an example.

```
CALL mysql.rds_set_external_master_gtid ('mymasterserver.mydomain.com', 3306,
 'repl_user', 'password', 'GTID', 1);
```

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

9. On the MariaDB DB instance, issue the mysql.rds_start_replication command to start replication.

```
CALL mysql.rds_start_replication;
```

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

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.

6. 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.

7. 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';
```

8. 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;
```

Options for MariaDB database engine

Following, you can find descriptions for options, or additional features, that are available for Amazon RDS instances running the MariaDB DB engine. To turn on these options, you 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 MariaDB:

Option ID	Engine versions
MARIADB_AUDIT_PLUGIN	MariaDB 10.3 and higher

MariaDB Audit Plugin support

Amazon RDS supports using the MariaDB Audit Plugin on MariaDB database instances. The MariaDB Audit Plugin records database activity such as users logging on to the database, queries run against the database, and more. 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.

> **Note**
>
> If you don't configure an option setting in the RDS console, RDS uses the default setting.

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 MariaDB database log files.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_FILE_ROTATE_SIZE	1–1000000000	1000000	The size in bytes that when reached, causes the file to rotate. For more information, see Log rotation and retention for MariaDB.
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 Log rotation and retention for MariaDB and Downloading a database log file.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_EVENTS	CONNECT, QUERY, TABLE, QUERY_DDL , QUERY_DML , QUERY_DML_NO_SELECT , QUERY_DCL	CONNECT, QUERY	The types of activity to record in the log. Installing the MariaDB Audit Plugin is itself logged. • 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. • TABLE: Log tables affected by queries when the queries are 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).
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 .

Option setting	Valid values	Default value	Description
SERVER_AUDIT_EXCL_USERS	Multiple comma-separated values	None	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. 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. **Note** CONNECT activity is always recorded for all users, even if the user is specified for this option setting.
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:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. 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.

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 **mariadb** for **Engine**, and choose **10.3** or higher 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 DB instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

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 settings for the plugin. 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.

Parameters for MariaDB

By default, a MariaDB DB instance uses a DB parameter group that is specific to a MariaDB database. This parameter group contains some but not all of the parameters contained in the Amazon RDS DB parameter groups for the MySQL database engine. It also contains a number of new, MariaDB-specific parameters. For information about working with parameter groups and setting parameters, see Parameter groups for Amazon RDS.

Viewing MariaDB parameters

RDS for MariaDB parameters are set to the default values of the storage engine that you have selected. For more information about MariaDB parameters, see the MariaDB documentation. For more information about MariaDB storage engines, see Supported storage engines for MariaDB on Amazon RDS.

You can view the parameters available for a specific RDS for MariaDB version using the RDS console or the AWS CLI. For information about viewing the parameters in a MariaDB 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 MariaDB version by running the `describe-engine-default-parameters` command. Specify one of the following values for the `--db-parameter-group-family` option:

- `mariadb11.8`
- `mariadb11.4`
- `mariadb10.11`
- `mariadb10.6`
- `mariadb10.5`
- `mariadb10.4`
- `mariadb10.3`

For example, to view the parameters for RDS for MariaDB version 10.6, run the following command.

```
aws rds describe-engine-default-parameters --db-parameter-group-family mariadb10.6
```

Your output looks similar to the following.

```
{
    "EngineDefaults": {
        "Parameters": [
            {
                "ParameterName": "alter_algorithm",
                "Description": "Specify the alter table algorithm.",
                "Source": "engine-default",
                "ApplyType": "dynamic",
                "DataType": "string",
                "AllowedValues": "DEFAULT,COPY,INPLACE,NOCOPY,INSTANT",
                "IsModifiable": true
            },
            {
                "ParameterName": "analyze_sample_percentage",
                "Description": "Percentage of rows from the table ANALYZE TABLE will
 sample to collect table statistics.",
                "Source": "engine-default",
                "ApplyType": "dynamic",
                "DataType": "float",
                "AllowedValues": "0-100",
                "IsModifiable": true
            },
            {
                "ParameterName": "aria_block_size",
                "Description": "Block size to be used for Aria index pages.",
                "Source": "engine-default",
                "ApplyType": "static",
                "DataType": "integer",
                "AllowedValues": "1024-32768",
                "IsModifiable": false
            },
            {
                "ParameterName": "aria_checkpoint_interval",
                "Description": "Interval in seconds between automatic checkpoints.",
                "Source": "engine-default",
                "ApplyType": "dynamic",
                "DataType": "integer",
                "AllowedValues": "0-4294967295",
                "IsModifiable": true
            },
        ...
```

To list only the modifiable parameters for RDS for MariaDB version 10.6, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mariadb10.6 \
   --query 'EngineDefaults.Parameters[?IsModifiable==`true`]'
```

For Windows:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mariadb10.6 ^
   --query "EngineDefaults.Parameters[?IsModifiable==`true`]"
```

MySQL parameters that aren't available

The following MySQL parameters are not available in MariaDB-specific DB parameter groups:

- bind_address
- binlog_error_action
- binlog_gtid_simple_recovery
- binlog_max_flush_queue_time
- binlog_order_commits
- binlog_row_image
- binlog_rows_query_log_events
- binlogging_impossible_mode
- block_encryption_mode
- core_file
- default_tmp_storage_engine
- div_precision_increment
- end_markers_in_json
- enforce_gtid_consistency
- eq_range_index_dive_limit
- explicit_defaults_for_timestamp
- gtid_executed

- gtid-mode
- gtid_next
- gtid_owned
- gtid_purged
- log_bin_basename
- log_bin_index
- log_bin_use_v1_row_events
- log_slow_admin_statements
- log_slow_slave_statements
- log_throttle_queries_not_using_indexes
- master-info-repository
- optimizer_trace
- optimizer_trace_features
- optimizer_trace_limit
- optimizer_trace_max_mem_size
- optimizer_trace_offset
- relay_log_info_repository
- rpl_stop_slave_timeout
- slave_parallel_workers
- slave_pending_jobs_size_max
- slave_rows_search_algorithms
- storage_engine
- table_open_cache_instances
- timed_mutexes
- transaction_allow_batching
- validate-password
- validate_password_dictionary_file
- validate_password_length
- validate_password_mixed_case_count
- validate_password_number_count

- validate_password_policy
- validate_password_special_char_count

For more information on MySQL parameters, see the MySQL documentation.

Migrating data from a MySQL DB snapshot to a MariaDB DB instance

You can migrate an RDS for MySQL DB snapshot to a new DB instance running MariaDB using the AWS Management Console, the AWS CLI, or Amazon RDS API. You must use a DB snapshot that was created from an Amazon RDS DB instance running MySQL 5.6 or 5.7. To learn how to create an RDS for MySQL DB snapshot, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Migrating the snapshot doesn't affect the original DB instance from which the snapshot was taken. You can test and validate the new DB instance before diverting traffic to it as a replacement for the original DB instance.

After you migrate from MySQL to MariaDB, the MariaDB DB instance is associated with the default DB parameter group and option group. After you restore the DB snapshot, you can associate a custom DB parameter group with the new DB instance. However, a MariaDB parameter group has a different set of configurable system variables. For information about the differences between MySQL and MariaDB system variables, see System Variable Differences between MariaDB and MySQL. To learn about DB parameter groups, see Parameter groups for Amazon RDS. To learn about option groups, see Working with option groups.

Performing the migration

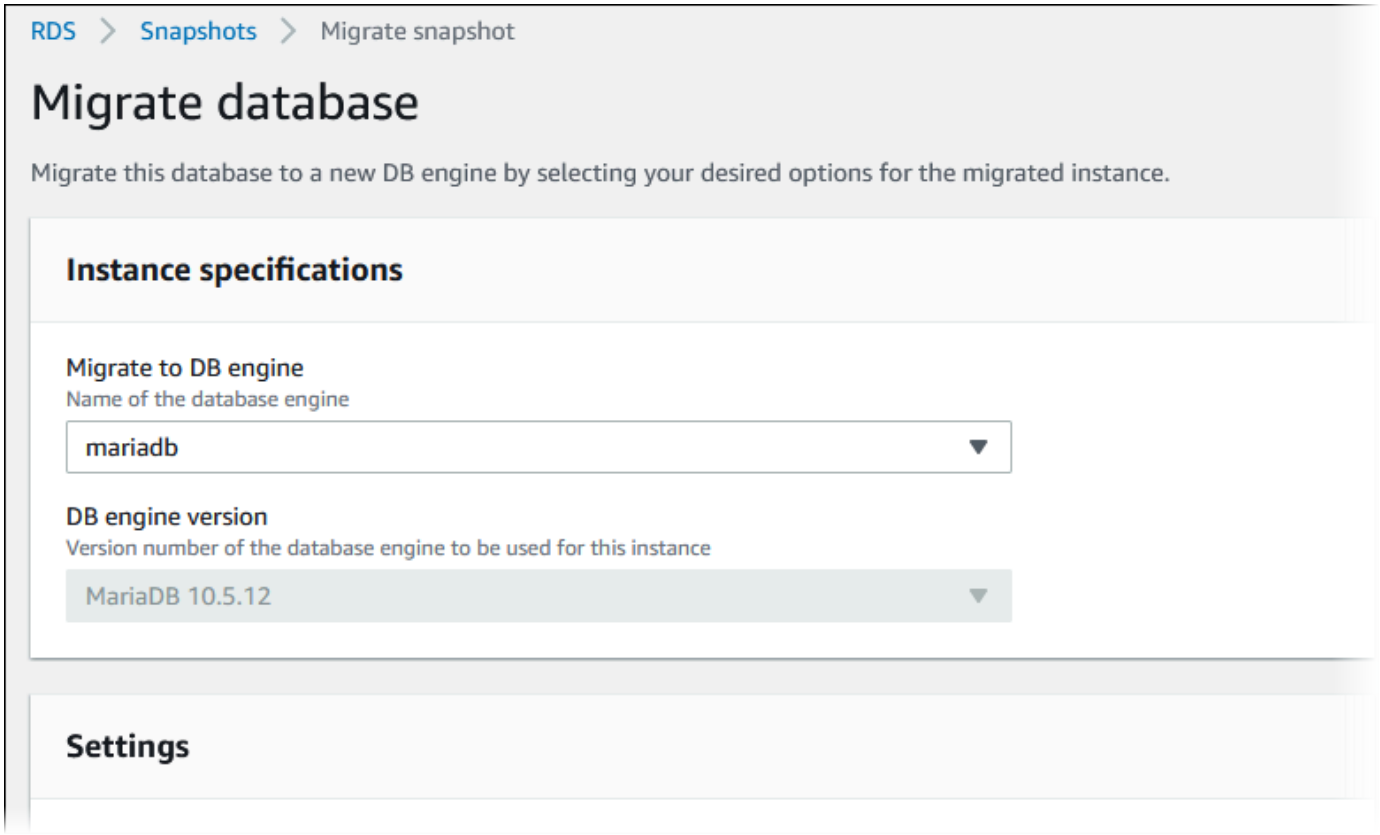
You can migrate an RDS for MySQL DB snapshot to a new MariaDB DB instance using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To migrate a MySQL DB snapshot to a MariaDB 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 navigation pane, choose **Snapshots**, and then select the MySQL DB snapshot you want to migrate.
3. For **Actions**, choose **Migrate snapshot**. The **Migrate database** page appears.
4. For **Migrate to DB Engine**, choose **mariadb**.

   Amazon RDS selects the **DB engine version** automatically. You can't change the DB engine version.



5. For the remaining sections, specify your DB instance settings. For information about each setting, see Settings for DB instances.
6. Choose **Migrate**.

AWS CLI

To migrate data from a MySQL DB snapshot to a MariaDB DB instance, run the AWS CLI `restore-db-instance-from-db-snapshot` command with the following options:

- --db-instance-identifier – Name of the DB instance to create from the DB snapshot.
- --db-snapshot-identifier – The identifier for the DB snapshot to restore from.
- --engine – The database engine to use for the new instance.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier newmariadbinstance \
```

```
    --db-snapshot-identifier mysqlsnapshot \
    --engine mariadb
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier newmariadbinstance ^
    --db-snapshot-identifier mysqlsnapshot ^
    --engine mariadb
```

API

To migrate data from a MySQL DB snapshot to a MariaDB DB instance, call the Amazon RDS API operation `RestoreDBInstanceFromDBSnapshot`.

Incompatibilities between MariaDB and MySQL

Incompatibilities between MySQL and MariaDB include the following:

- You can't migrate a DB snapshot created with MySQL 8.0 to MariaDB.
- If the source MySQL database uses a SHA256 password hash, make sure to reset user passwords that are SHA256 hashed before you connect to the MariaDB database. The following code shows how to reset a password that is SHA256 hashed.

  ```
  SET old_passwords = 0;
  UPDATE mysql.user SET plugin = 'mysql_native_password',
  Password = PASSWORD('new_password')
  WHERE (User, Host) = ('master_user_name', %);
  FLUSH PRIVILEGES;
  ```

- If your RDS master user account uses the SHA-256 password hash, make sure to reset the password using the AWS Management Console, the `modify-db-instance` AWS CLI command, or the ModifyDBInstance RDS API operation. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.
- MariaDB doesn't support the Memcached plugin. However, the data used by the Memcached plugin is stored as InnoDB tables. After you migrate a MySQL DB snapshot, you can access the data used by the Memcached plugin using SQL. For more information about the innodb_memcache database, see InnoDB memcached Plugin Internals.

MariaDB on Amazon RDS SQL reference

Following, you can find descriptions of system stored procedures that are available for Amazon RDS instances running the MariaDB DB engine.

You can use the system stored procedures that are available for MySQL DB instances and MariaDB DB instances. These stored procedures are documented at RDS for MySQL stored procedure reference. MariaDB DB instances support all of the stored procedures, except for `mysql.rds_start_replication_until` and `mysql.rds_start_replication_until_gtid`.

Additionally, the following system stored procedures are supported only for Amazon RDS DB instances running MariaDB:

- mysql.rds_replica_status
- mysql.rds_set_external_master_gtid
- mysql.rds_kill_query_id
- mysql.rds_execute_operation

mysql.rds_replica_status

Shows the replication status of a MariaDB read replica.

Call this procedure on the read replica to show status information on essential parameters of the replica threads.

Syntax

```
CALL mysql.rds_replica_status;
```

Usage notes

This procedure is only supported for MariaDB DB instances running MariaDB version 10.5 and higher.

This procedure is the equivalent of the `SHOW REPLICA STATUS` command. This command isn't supported for MariaDB version 10.5 and higher DB instances.

In prior versions of MariaDB, the equivalent `SHOW SLAVE STATUS` command required the `REPLICATION SLAVE` privilege. In MariaDB version 10.5 and higher, it requires the `REPLICATION REPLICA ADMIN` privilege. To protect the RDS management of MariaDB 10.5 and higher DB instances, this new privilege isn't granted to the RDS master user.

Examples

The following example shows the status of a MariaDB read replica:

```
call mysql.rds_replica_status;
```

The response is similar to the following:

```
*************************** 1. row ***************************
               Replica_IO_State: Waiting for master to send event
                    Source_Host: XX.XX.XX.XXX
                    Source_User: rdsrepladmin
                    Source_Port: 3306
                  Connect_Retry: 60
                Source_Log_File: mysql-bin-changelog.003988
            Read_Source_Log_Pos: 405
                 Relay_Log_File: relaylog.011024
                  Relay_Log_Pos: 657
          Relay_Source_Log_File: mysql-bin-changelog.003988
             Replica_IO_Running: Yes
            Replica_SQL_Running: Yes
                Replicate_Do_DB:
            Replicate_Ignore_DB:
             Replicate_Do_Table:
         Replicate_Ignore_Table:
mysql.rds_sysinfo,mysql.rds_history,mysql.rds_replication_status
        Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
                     Last_Errno: 0
                     Last_Error:
                   Skip_Counter: 0
            Exec_Source_Log_Pos: 405
                Relay_Log_Space: 1016
                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: 0
  Source_SSL_Verify_Server_Cert: No
                  Last_IO_Errno: 0
                  Last_IO_Error:
                 Last_SQL_Errno: 0
                 Last_SQL_Error:
    Replicate_Ignore_Server_Ids:
               Source_Server_Id: 807509301
                 Source_SSL_Crl:
             Source_SSL_Crlpath:
                     Using_Gtid: Slave_Pos
                   Gtid_IO_Pos: 0-807509301-3980
        Replicate_Do_Domain_Ids:
    Replicate_Ignore_Domain_Ids:
                  Parallel_Mode: optimistic
                      SQL_Delay: 0
            SQL_Remaining_Delay: NULL
      Replica_SQL_Running_State: Reading event from the relay log
             Replica_DDL_Groups: 15
Replica_Non_Transactional_Groups: 0
    Replica_Transactional_Groups: 3658
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.000 sec)
```

mysql.rds_set_external_master_gtid

Configures GTID-based replication from a MariaDB instance running external to Amazon RDS to a MariaDB DB instance. This stored procedure is supported only where the external MariaDB instance is version 10.0.24 or higher. When setting up replication where one or both instances do not support MariaDB global transaction identifiers (GTIDs), use mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower).

Using GTIDs for replication provides crash-safety features not offered by binary log replication, so we recommend it in cases where the replicating instances support it.

Syntax

```
CALL mysql.rds_set_external_master_gtid(
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , gtid
  , ssl_encryption
);
```

Parameters

host_name

String. The host name or IP address of the MariaDB instance running external to Amazon RDS that will become the source instance.

host_port

Integer. The port used by the MariaDB instance running external to Amazon RDS to be configured as the source 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

String. The ID of a user with `REPLICATION SLAVE` permissions in the MariaDB DB instance to be configured as the read replica.

replication_user_password

String. The password of the user ID specified in `replication_user_name`.

gtid

String. The global transaction ID on the source instance that replication should start from.

You can use `@@gtid_current_pos` to get the current GTID if the source instance has been locked while you are configuring replication, so the binary log doesn't change between the points when you get the GTID and when replication starts.

Otherwise, if you are using `mysqldump` version 10.0.13 or greater to populate the replica instance prior to starting replication, you can get the GTID position in the output by using the `--master-data` or `--dump-slave` options. If you are not using `mysqldump` version

10.0.13 or greater, you can run the `SHOW MASTER STATUS` or use those same `mysqldump` options to get the binary log file name and position, then convert them to a GTID by running `BINLOG_GTID_POS` on the external MariaDB instance:

```
SELECT BINLOG_GTID_POS('<binary log file name>', <binary log file position>);
```

For more information about the MariaDB implementation of GTIDs, go to Global transaction ID in the MariaDB documentation.

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 `mysql.rds_set_external_master_gtid` procedure must be run by the master user. It must be run on the MariaDB DB instance that you are configuring as the replica of a MariaDB instance running external to Amazon RDS. Before running `mysql.rds_set_external_master_gtid`, you must have configured the instance of MariaDB running external to Amazon RDS as a source instance. For more information, see Importing data into an Amazon RDS for MariaDB DB instance.

> **Warning**
>
> Do not use `mysql.rds_set_external_master_gtid` to manage replication between two Amazon RDS DB instances. Use it only when replicating with a MariaDB instance running external to 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_gtid` to configure an Amazon RDS DB instance as a read replica, you can call mysql.rds_start_replication on the 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_gtid` 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 MariaDB DB instance, the following example configures it as the replica of an instance of MariaDB running external to Amazon RDS.

```
call mysql.rds_set_external_master_gtid
 ('Sourcedb.some.com',3306,'ReplicationUser','SomePassW0rd','0-123-456',0);
```

mysql.rds_kill_query_id

Ends a query running against the MariaDB server in order to terminate long-running or problematic queries. You can identify the query ID and effectively stop a specific query to address performance issues and maintain optimal database operation.

Syntax

```
CALL mysql.rds_kill_query_id(queryID);
```

Parameters

queryID

Integer. The identity of the query to be ended.

Usage notes

To stop a query running against the MariaDB server, use the `mysql.rds_kill_query_id` procedure and pass in the ID of that query. To obtain the query ID, query the MariaDB Information schema PROCESSLIST table, as shown following:

```
SELECT USER, HOST, COMMAND, TIME, STATE, INFO, QUERY_ID FROM
                INFORMATION_SCHEMA.PROCESSLIST WHERE USER = '<user name>';
```

The connection to the MariaDB server is retained.

Examples

The following example ends a query with a query ID of 230040:

```
call mysql.rds_kill_query_id(230040);
```

mysql.rds_execute_operation

Executes InnoDB operations to manage buffer pool states and temporary tablespace. This procedure can be used to dynamically control InnoDB operations such as dumping and loading buffer pool states or truncating temporary tablespace.

Syntax

```
CALL mysql.rds_execute_operation(operation);
```

Parameters

operation

String. The InnoDB operations to execute. Valid values are:

- *innodb_buffer_pool_dump_now* - Operation that dumps the current state of the buffer pool.
- *innodb_buffer_pool_load_now* - Operation that loads the saved buffer pool state.
- *innodb_buffer_pool_load_abort* - Operation that aborts a buffer pool load operation.
- *innodb_truncate_temporary_tablespace_now* - Operation that truncates the temporary tablespace.

Usage notes

This procedure is only supported for MariaDB DB instances running MariaDB version 11.8 and higher.

During execution, binary logging is temporarily disabled to prevent replication of these administrative commands.

The procedure maintains an audit trail by logging all operations in the `mysql.rds_history` table.

Examples

The following example demonstrates temporary tablespace shrinking using `mysql.rds_execute_operation`:

To check current temporary tablespace size, run the following query:

```
SELECT FILE_SIZE FROM information_schema.innodb_sys_tablespaces WHERE name LIKE
 'innodb_temporary';
+------------+
| FILE_SIZE  |
+------------+
| 6723469312 |  -- 6.3 GB
+------------+
```

When you drop temporary tables, it doesn't reduce storage usage in the global tablespace. To reduce the size of the global tablespace, run the `mysql.rds_execute_operation` command to shrink the temporary tablespace.

```
CALL mysql.rds_execute_operation('innodb_truncate_temporary_tablespace_now');
Query OK, 2 rows affected (0.004 sec)
```

After you run the procedure, verify that the space was reclaimed.

```
SELECT FILE_SIZE FROM information_schema.innodb_sys_tablespaces WHERE name LIKE
 'innodb_temporary';
+-----------+
| FILE_SIZE |
+-----------+
|  12582912 |  -- 12 MB
+-----------+
```

> **Note**
>
> The shrink operation might take time, depending on the temporary tablespace size and current workload.

Important

The temporary tablespace shrinks only when all temporary tables that contributed to its size are no longer in use. We recommend that you run this procedure when there are no active temporary tablespaces on the instance.

Local time zone for MariaDB DB instances

By default, the time zone for a MariaDB 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 MariaDB, it ships with the latest time zone data at the time of the release. When you use the latest RDS for MariaDB 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 RDS for MariaDB

The following items are known issues and limitations when using RDS for MariaDB.

> **Note**
>
> This list is not exhaustive.

Topics

MariaDB file size limits in Amazon RDS

For MariaDB DB instances, the maximum size of a table is 16 TB when using InnoDB file-per-table tablespaces. This limit also constrains the system tablespace to a maximum size of 16 TB. InnoDB file-per-table tablespaces (with tables each in their own tablespace) are set by default for MariaDB DB instances. This limit isn't related to the maximum storage limit for MariaDB DB instances. For more information about the storage limit, see Amazon RDS DB instance storage.

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.

To determine the size of all InnoDB tablespaces

- Use the following SQL command to determine if any of your tables are too large and are candidates for partitioning.

> **Note**
>
> For MariaDB 10.6 and higher, this query also returns the size of the InnoDB system tablespace.
> For MariaDB versions earlier than 10.6, you can't determine the size of the InnoDB system tablespace by querying the system tables. We recommend that you upgrade to a later version.

```
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 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. You can use this command to move a table from the global tablespace to its own tablespace. Or you can move a table from its own tablespace to the global tablespace. Following is an example.

```
ALTER TABLE table_name ENGINE=InnoDB, ALGORITHM=COPY;
```

InnoDB reserved word

`InnoDB` is a reserved word for RDS for MariaDB. You can't use this name for a MariaDB database.

Custom ports

Amazon RDS blocks connections to custom port 33060 for the MariaDB engine. Choose a different port for your MariaDB engine.

Performance Insights

InnoDB counters are not visible in Performance Insights for RDS for MariaDB version 10.11 because the MariaDB community no longer supports them.