MySQL on Amazon RDS

**Contents**

[Scope 2](#_Toc115190395)

[Problem Statement 2](#_Toc115190396)

[Introduction 2](#_Toc115190397)

[Encryption of data at rest 2](#_Toc115190396)

[Encryption of data in transit 5](#_Toc115190396)

[Steps to encrypt MySQL on Amazon RDS 7](#_Toc115190396)

[Encrypt Amazon RDS Snapshots using a KMS key 10](#_Toc115190396)

MySQL security on Amazon RDS [11](#_Toc115190398)

[Using the Password Validation Plugin for RDS for MySQL 12](#_Toc115190399)

[Encrypting client connections to MySQL DB instances with SSL/TLS 13](#_Toc115190398)

Using SSL/TLS Certificates for MySQL DB instances. [15](#_Toc115190398)

[Using Kerberos Authentication for MySQL 27](#_Toc115190398)

[Kerberos Authentication Limitations 31](#_Toc115190400)

[Automation of back ups 36](#_Toc115190413)

[Work Flow - RDS 42](#_Toc115190413)

[Limitations](#_Toc115190413) 42

**MySQL on Amazon RDS**

Document Name: [MySQL on Amazon RDS](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_MySQL.html)

Owner: Hema Paul

**Scope**

Amazon RDS for MySQL instances are pre-configured with parameters and settings tuned for the instance type. As you have a massive amount of control over these parameters with easy to manage database parameter groups that provide granular control and tuning options for your database instances.

**Problem Statement**

As a managed service, Amazon RDS provides a high level of security for your MySQL databases. These include network isolation using Amazon VPC (virtual private cloud), encryption at rest using keys that you create and control through the AWS Key Management Service (KMS). Data can also be encrypted through the wire in transit using SSL.

**Introduction**

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

**Encryption of data at rest**

Many AWS customers using RDS MySQL-related database engines rely on [encrypting RDS resources](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.html). With RDS-encrypted resources, data is encrypted at rest, including the underlying storage for a database (DB) instance, its automated backups, read replicas, and snapshots. This capability uses the open standard AES-256 encryption algorithm to encrypt your data, which is transparent to your database engine.

For RDS MySQL and MariaDB, [Amazon EBS](https://aws.amazon.com/ebs/) provides the underlying storage and snapshot capability. Aurora uses a purpose-built, distributed, and log-structured storage service. Encrypted Aurora DB clusters enable you to encrypt data persistently stored by the storage service, along with associated backups stored in Amazon S3.

This encryption option protects against physical exfiltration or access of your data bypassing the DB instances. It is therefore critical to complement encrypted resources with an effective encryption key management and database credential management practice to mitigate any unauthorized access. Otherwise, compromised credentials or insufficiently protected keys might allow unauthorized users to access the plaintext data directly through the database engine.

Encryption key management is provided using the [AWS KMS](https://docs.aws.amazon.com/kms/latest/developerguide/), which allows you to create encryption keys and define the policies that control the use of these keys. RDS MySQL-related engines can use two types of keys:

* The default key
* AWS KMS customer master keys (CMKs)

A default key is available with each AWS account and enables a one-press solution for encrypting your RDS resources. It is easy to use but does not allow you to take any advanced credential management actions, such as rotation, revocation, or deletion. The default key is ideal for customers that do not want to take on the burden of key management, preferring to delegate that responsibility to AWS.

[CMKs](https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html#master_keys), in contrast, provide you with full control of the lifecycle of the encryption keys, access controls, and audit-ability, but require additional management overhead on your part.

Encrypted resources provide an additional layer of security to your backups and snapshots. However, you may need to [take extra steps](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.html) when you share snapshots, restore DB instances and clusters from encrypted snapshots, or copy snapshots to another Region. For example, snapshots using the default key cannot be shared directly with other AWS accounts. First, you copy the snapshot and then change the key to a CMK.

Sharing a CMK-encrypted snapshot also requires you to grant the target AWS account access to the CMK used by the snapshot. During copy operations, the data remains encrypted as RDS employs [envelope encryption](https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html#enveloping), where individual data keys that physically encrypt the data are themselves encrypted using the indicated KMS key.

Encrypted resources leverage the hardware acceleration provided by modern CPUs (AES-NI or newer instruction set versions). The storage level implementation ensures that the performance overhead of enabling encryption is negligible.

**Which encryption techniques have been used in RDS to store data?**

With RDS-encrypted resources, data is encrypted at rest, including the underlying storage for a database (DB) instance, its automated backups, read replicas, and snapshots. This capability uses the **open standard AES-256 encryption algorithm** to encrypt your data, which is transparent to your database engine.

The AES Encryption algorithm (also known as the Rijndael algorithm) is a symmetric block cipher algorithm with a block/chunk size of 128 bits. It converts these individual blocks using keys of 128, 192, and 256 bits. Once it encrypts these blocks, it joins them together to form the ciphertext.

**Types of Encryption**

Although they are popular, RDS-encrypted resources are not the only way to implement data encryption at rest. Some use cases require the implementation of encryption at the logical layer—for example, to limit data access between different users. Many customers implement both, as they can serve different purposes. There are two ways to do so:

* Encryption of data in fields using [*encryption functions*](https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html#function_aes-encrypt)
* Encryption of data *client-side* before transmission to the database server

RDS MySQL-related database engines **do not provide any built-in encryption** **key management** capabilities. Managing the encryption function keys, therefore, becomes an implementation concern. The risk of exposure increases as the keys pass to the database engine—in SQL statements, logs, the engine’s process list, or other monitoring capabilities. The risk remains even if the client connections to the database are encrypted. Consequently, this approach might not meet all the data protection requirements applicable to you.

On the other hand, by performing encryption on the client side, you avoid the risk of inadvertently exposing keys in the database engine logging and monitoring features. Client-side encryption also removes from the database engine any burden of performing encryption/decryption operations.

Both approaches to logically encrypting the data introduce limitations to your data’s indexability and searchability. The database engine doesn’t store the plaintext values in the encrypted fields. The construction of indexes using the ciphertext values affects the ordering of your indexes and cardinality. Cyphertext indexing might impact execution plan decisions and thus performance. Similarly, database engines can at most evaluate encrypted fields in

terms of “equal to” or “not equal to” predicates. Any other evaluations must be performed after data decryption.

As such, logical data encryption has generally been implemented only in narrow use cases, such as encrypting a person’s social security number (SSN) in a user data table. In that case, logical encryption limits access to the SSN to users with the required key. But search queries in the database are practically also limited to identifying users with a particular SSN only.

## **Encryption of data in transit**

RDS MySQL-related database engines allow you to establish [Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security) (TLS) [encrypted connections](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html) to the database engine. These connections are frequently also called Secure Sockets Layer (SSL) encryption, although that term specifically refers to the now-deprecated predecessor cryptographic protocol of TLS, which is not supported by RDS.

RDS creates an SSL certificate and installs the certificate on the DB instance when RDS provisions the instance. A certificate authority signs these certificates. The SSL certificate includes the DB instance endpoint as the common name (CN) for the SSL certificate to guard against spoofing attacks.

Aurora MySQL DB clusters also include the cluster endpoints as subject alternative names (SAN) in the certificate. Your database driver or client must support SAN to use cluster database endpoints for encrypted connections with Aurora and to be able to verify the server certificate or identity.

From a server-side perspective, MySQL-based engines enforce the requirement to use SSL connections at the database-user level. Ensure that any user accounts employing remote connections require the use of SSL. The specific command varies based on the version of the engine and is available in the respective engine’s documentation.

Clients can also require the use of SSL connections. AWS recommends verifying the server identity using the [*ssl\_mode*](https://dev.mysql.com/doc/refman/5.7/en/connection-options.html#option_general_ssl-mode) = VERIFY\_IDENTITY option (‘ssl-verify-server-cert’ in older versions), to mitigate the risk of connecting to unintended server endpoints.

The version of TLS cryptographic protocol also varies with the database engine version, with RDS MySQL 5.6, older versions of MariaDB 10.0 and 10.1, and Aurora MySQL 5.6 only supporting TLS 1.0. Furthermore, the cryptographic library used by RDS changed, transitioning with newer versions of the database engines to OpenSSL from yaSSL.

For RDS MySQL-related engine versions that support multiple versions of TLS (such as MySQL 8.0), you can use the [*tls\_version*](https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html#sysvar_tls_version) parameter in the DB instance [parameter group](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html) to indicate the permitted protocol versions. Similar client parameters exist for most client tools or database drivers. By default, the database engine attempts to use the highest TLS protocol version allowed by both the server and client configuration.

Establishing encrypted database connections involves overhead, both in terms of compute resources and response latency to first query. However, the newer engine versions, using the OpenSSL cryptographic library, offer lower overhead.

# Encrypt an unencrypted Amazon RDS DB instance for MySQL

Amazon RDS has the following [limitations for encrypted DB instances](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.html#Overview.Encryption.Limitations):

* You can't modify an existing unencrypted Amazon RDS DB instance to encrypt the instance.
* You can't create an encrypted read replica from an unencrypted instance.

**Steps to Encrypt an Unencrypted Snapshot**

1.    Encrypt an unencrypted snapshot that you take from an unencrypted read replica of the DB instance.

2.    Restore a new DB instance from the encrypted snapshot to deploy a new encrypted DB instance.

3.    Use MySQL replication to synchronize changes from the source to the new encrypted DB instance.

4.    Verify that the new, encrypted DB instance is in sync with the source DB instance.

5.    Switch your connections and redirect your traffic to the new DB instance.

### **Setting up replication with minimal downtime**

1.    [Create a temporary read replica](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReadRepl.html#USER_ReadRepl.Create) for the source unencrypted Amazon RDS DB instance. In this example, the source unencrypted DB instance is called **SOURCE-EU** and the temporary read replica is called **TEMP-RR**.

2.    Connect to **TEMP-RR**, and monitor the replica lag until **Seconds\_Behind\_Master** is stable at value **0**. This indicates that **TEMP-RR** is in sync with **SOURCE-EU**:

mysql> SHOW SLAVE STATUS \G

Seconds\_Behind\_Master: 0

3.    [Stop the replication](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/mysql_rds_stop_replication.html) process on **TEMP-RR**.

MySQL > call mysql.rds\_stop\_replication;

+---------------------------+

| Message |

+---------------------------+

| Slave is down or disabled |

+---------------------------+

4.    Note the values for **Relay\_Master\_Log\_File** and **Exec\_Master\_Log\_Pos** from **TEMP-RR**:

mysql> SHOW SLAVE STATUS \G

Relay\_Master\_Log\_File: mysql-bin-changelog.000012

Exec\_Master\_Log\_Pos: 123

5.    In **SOURCE-EU**, set the **binlog retention hours** parameter to preserve binary logs for the time that is required to complete the operation. In the following example, **binlog retention hours** is set to 24 hours:

mysql> call mysql.rds\_set\_configuration('binlog retention hours', 24);

6.    Take a snapshot of **TEMP-RR**. Optionally, delete **TEMP-RR** after taking the snapshot.

7.    [Copy the snapshot](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CopySnapshot.html) of **TEMP-RR**, and set **Enable Encryption** to **Yes**.

8.    Restore a new DB instance from the copied snapshot that has encryption enabled. In this example, the new encrypted DB instance is called **NEW-RR-EN**.

9.    Modify the inbound rules in **SOURCE-EU** security group to allow traffic from **NEW-RR-EN**. If you use the same [security group](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html#VPCSecurityGroups) on both DB instances, then you can use the same security group ID reference as **SOURCE-EU**.

**Note:** Be sure to allow outbound traffic to **SOURCE-EU** from **NEW-RR-EN**.

10.    Log in to **SOURCE-EU**, set up a replication user, and then grant the necessary permissions to the user:

mysql> create user 'repl\_user'@'%' identified by 'password123';

mysql> grant replication slave, replication client on \*.\* to 'repl\_user'@'%';

mysql> show grants for 'repl\_user'@'%';

**Note:** Replace **repl\_user** with your own replication user name and **password123** with your own password.

11.    Connect to **NEW-RR-EN**, and establish a replication connection to **SOURCE-EU**:

mysql> CALL mysql.rds\_set\_external\_master ( 'rds-endpoint' , 3306 , 'repl\_user' , 'password123' , 'mysql-bin.000012' , 123 , 0 );

The **rds-endpoint** is the [DB instance endpoint](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html#USER_ConnectToInstance.EndpointAndPort) for **SOURCE-EU**.

The user name (**repl\_user**) and password (**password123**) are the user name and password that you created in Step 10. Use the captured values of **Relay\_Master\_Log\_File** and **Exec\_Master\_Log\_Pos** from Step 4 to set up replication with **mysql.rds\_set\_external\_master** procedure.

**Note**: If **SOURCE-EU** is [publicly accessible](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_VPC.WorkingWithRDSInstanceinaVPC.html#USER_VPC.Hiding) and **NEW-RR-EN** (new encrypted DB instance) is set to "private", use the private IP address (of **SOURCE-EU**) instead of **rds-endpoint**.

12.    From **NEW-RR-EN**, start replication:

mysql > CALL mysql.rds\_start\_replication;

13.    From **NEW-RR-EN**, confirm that the replication was successful and in sync between **SOURCE-EU** and **NEW-RR-EN**.

mysql> SHOW SLAVE STATUS \G

If your connection between the source DB instance and read replica are successful, your output looks like this:

Slave\_IO\_State: Waiting for master to send event

Seconds Behind master: 0

14.    After **Seconds\_Behind\_Master** is stable at value **0**, stop the traffic and close the connections on **SOURCE-EU**. Downtime then begins.

**Note:** Stop all application servers and clients that connect to **SOURCE-EU** to make sure that no new changes are made to **SOURCE-EU**. Optionally, you can temporarily lock the security group that **SOURCE-EU** uses. This prevents inbound traffic from any application or client, except from **NEW-RR-EN** and the host that the user is performing these actions from.

15.    As the database master user, connect to **NEW-RR-EN**, and then stop replication:

MySQL > call mysql.rds\_stop\_replication;

**Important:** After you run this command, **NEW-RR-EN** no longer replicates data from **SOURCE-EU**.

16.    Promote **NEW-RR-EN** to a standalone server by stopping the replication relationship between **SOURCE-EU** and **NEW-RR-EN**:

MySQL > call mysql.rds\_reset\_external\_master;

17.    Point all applications, clients, and database connections to **NEW-RR-EN** by specifying the **NEW-RR-EN** DNS endpoint in all connection strings. Or, rename **SOURCE-EU**, and then modify **NEW-RR-EN** to use the same name that **SOURCE-EU** uses.

18.    Confirm that the security group rules on **NEW-RR-EN** allow inbound traffic from the appropriate applications and clients.

19.    After you point the applications and clients to **NEW-RR-EN** and test the environment, delete **SOURCE-EU**.

**Tip:** It's a best practice to test this operation on a test instance before applying this operation in a production environment.

**Encrypt Amazon RDS snapshots using a KMS key**

You can't take an encrypted snapshot of an unencrypted DB instance. The following steps are applicable to Amazon RDS for MySQL, Oracle, SQL Server, PostgreSQL, or MariaDB.

1.    Open the [Amazon RDS console](https://console.aws.amazon.com/rds), and then choose **Snapshots** from the navigation pane.

2.    Select the snapshot that you want to encrypt.

3.    Under **Snapshot Actions**, choose **Copy Snapshot**.

4.    Choose your **Destination Region**, and then enter your **New DB Snapshot Identifier**.

5.    Change **Enable Encryption** to **Yes**.

6.    Select your **AWS KMS Key** from the list.

7.    Choose **Copy Snapshot**.

After the snapshot status is *available*, the **Encrypted** field is set to "True" to indicate that the snapshot is encrypted. You can now use this encrypted DB snapshot to [restore the DB instance from the DB snapshot](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_RestoreFromSnapshot.html#USER_RestoreFromSnapshot.CON).

**To restore a DB instance from 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 DB snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore snapshot** page, for **DB instance identifier**, enter the name for your restored DB instance.
6. Specify other settings.
7. Choose **Restore DB instance**.

**MySQL security on Amazon RDS**

Security for MySQL DB instances is managed at three levels:

* AWS Identity and Access Management controls who can perform Amazon RDS management actions on DB instances. When you connect to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS management operations.
* When you create a DB instance, you use a VPC security group to control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance. These connections can be made using Secure Sockets Layer (SSL) and Transport Layer Security (TLS). In addition, firewall rules at your company can control whether devices running at your company can open connections to the DB instance.
* To authenticate login and permissions for a MySQL DB instance, you can take either of the following approaches, or a combination of them.

You can take the same approach as with a stand-alone instance of MySQL. Commands such as **CREATE USER, RENAME USER, GRANT, REVOKE, and SET PASSWORD** work just as they do in on-premises databases, as does directly modifying database schema tables.

You can also use IAM database authentication. With IAM database authentication, you authenticate to your DB instance by using an IAM user or IAM role and an authentication token. An *authentication token* is a unique value that is generated using the Signature Version 4 signing process. By using IAM database authentication, you can use the same credentials to control access to your AWS resources and your databases.

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

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
* replication client
* replication slave
* select
* show databases
* show view
* trigger
* update

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

To allow management of the DB instance, the standard kill and *kill\_query* commands have been restricted. The Amazon RDS commands *rds\_kill* and *rds\_kill\_query* are provided to allow you to end user sessions or queries on DB instances.

# Using the Password Validation Plugin for RDS for MySQL

MySQL provides the *validate\_password* plugin for improved security. The plugin enforces password policies using parameters in the DB parameter group for your MySQL DB instance.

**To enable the validate\_password plugin for a MySQL DB instance**

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

*INSTALL PLUGIN validate\_password SONAME 'validate\_password.so';*

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

**Reference*:*** *https://dev.mysql.com/doc/refman/8.0/en/validate-password-options-variables.html*

After installing and enabling the *password\_validate* plugin, reset existing passwords to comply with your new validation policies.

Amazon RDS doesn't validate passwords. The MySQL DB instance performs password validation. If you set a user password with the AWS Management Console, the *modify-db-instance* AWS CLI command, or the *ModifyDBInstance* RDS API operation, the change can succeed even if the new password doesn't satisfy your password policies. However, a new password is set in the MySQL DB instance only if it satisfies the password policies. In this case, Amazon RDS records the following event.

"RDS-EVENT-0067" - An attempt to reset the master password for the DB instance has failed.

**Encrypting client connections to MySQL DB instances with SSL/TLS**

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

## **Using SSL/TLS with a MySQL DB instance**

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.

MySQL uses OpenSSL for secure connections. Amazon RDS for MySQL supports Transport Layer Security (TLS) versions 1.0, 1.1, and 1.2

You can require SSL/TLS connections for specific users accounts. For example, you can 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;

## **Requiring SSL/TLS for all connections to a MySQL DB instance**

You can require that all user connections to your MySQL DB instance use SSL/TLS by using the *require\_secure\_transport* parameter. By default, the *require\_secure\_transport* parameter is set to OFF. You can set the *require\_secure\_transport* parameter to ON to require SSL/TLS for connections to your DB instance.

You can set the *require\_secure\_transport* parameter value by updating the DB parameter group for your DB instance. You don't need to reboot your DB instance for the change to take effect.

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

*MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --require\_secure\_transport=ON.*

**For setting parameters of a DB instance , modify the parameter group.**

You can modify parameter values in a customer-created DB parameter group; you can't change the parameter values in a default DB parameter group. Changes to parameters in a customer-created DB parameter group are applied to all DB instances that are associated with the DB parameter group.

Changes to some parameters are applied to the DB instance immediately without a reboot. Changes to other parameters are applied only after the DB instance is rebooted. The RDS console shows the status of the DB parameter group associated with a DB instance on the **Configuration** tab. For example, suppose that the DB instance isn't using the latest changes to its associated DB parameter group. If so, the RDS console shows the DB parameter group with a status of **pending-reboot**. To apply the latest parameter changes to that DB instance, manually reboot the DB instance.

**To modify a DB parameter group using 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 **Parameter groups**.
3. In the list, choose the parameter group that you want to modify.
4. For **Parameter group actions**, choose **Edit**.
5. Change the values of the parameters that you want to modify. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

You **can't** change values in a default parameter group.

1. Choose **Save changes**.

**Using AWS CLI**

To modify a DB parameter group, use the AWS CLI [modify-db-parameter-group](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-parameter-group.html) command with the following required options:

* --db-parameter-group-name
* --parameters

The following example modifies the *max\_connections* and *max\_allowed\_packet* values in the DB parameter group named mydbparametergroup.

**Example**

For **Linux, macOS, or Unix**:

aws rds modify-db-parameter-group \

--db-parameter-group-name *mydbparametergroup* \

--parameters "ParameterName=*max\_connections*,ParameterValue=*250*,ApplyMethod=*immediate*" \

"ParameterName=*max\_allowed\_packet*,ParameterValue=*1024*,ApplyMethod=*immediate*"

For **Windows**:

aws rds modify-db-parameter-group ^

--db-parameter-group-name *mydbparametergroup* ^

--parameters "ParameterName=*max\_connections*,ParameterValue=*250*,ApplyMethod=*immediate*" ^

"ParameterName=*max\_allowed\_packet*,ParameterValue=*1024*,ApplyMethod=*immediate*"

The command produces output like the following:

DBPARAMETERGROUP *mydbparametergroup*

## **Connecting from the MySQL command-line client with SSL/TLS (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*

**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.
2. Use a MySQL command-line client to connect to a DB instance with SSL/TLS encryption. For the -h parameter, substitute the DNS name (endpoint) for your DB instance. For the --ssl-ca parameter, substitute the SSL/TLS certificate file name. For the -P parameter, substitute the port for your DB instance. For the *-u* parameter, substitute the user name of a valid database user, such as the master user. Enter the master user password when prompted.

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

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

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

mysql -h *mysql–instance1.123456789012.us-east-1.rds.amazonaws.com* --ssl-ca=*global-bundle.pem* --ssl-mode=VERIFY\_IDENTITY -P 3306 -u *myadmin* -p

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

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

1. Enter the master user password when prompted

You will see output similar to the following.

*Welcome to the MySQL monitor. Commands end with ; or \g.*

*Your MySQL connection id is 9738*

*Server version: 8.0.23 Source distribution*

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

*mysql>*

# Using new SSL/TLS certificates for MySQL DB instances

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

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

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB instances.

Before you update your DB instances to use the new CA certificate, make sure that you update your clients or applications connecting to your RDS databases.

Amazon RDS provides new CA certificates as an AWS security best practice.

Reference: [*https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL-certificate-rotation.html*](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL-certificate-rotation.html)

## Sample script for importing certificates into your trust store

**Linux :**

mydir=tmp/certs

if [ ! -e "${mydir}" ]

then

mkdir -p "${mydir}"

fi

truststore=${mydir}/rds-truststore.jks

storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" > ${mydir}/global-bundle.pem

awk 'split\_after == 1 {n++;split\_after=0} /-----END CERTIFICATE-----/ {split\_after=1}{print > "rds-ca-" n ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-\*; do

alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/; s/.\*(CN=|CN = )//; print')

echo "Importing $alias"

keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore ${truststore} -noprompt

rm $CERT

done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut -d " " -f3- | while read alias

do

expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.\*?)\n/) { print "$1\n"; }'`

echo " Certificate ${alias} expires in '$expiry'"

done

**MacOS**

mydir=tmp/certs

if [ ! -e "${mydir}" ]

then

mkdir -p "${mydir}"

fi

truststore=${mydir}/rds-truststore.jks

storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" > ${mydir}/global-bundle.pem

split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-\*; do

alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/; s/.\*(CN=|CN = )//; print')

echo "Importing $alias"

keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore ${truststore} -noprompt

rm $CERT

done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut -d " " -f3- | while read alias

do

expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.\*?)\n/) { print "$1\n"; }'`

echo " Certificate ${alias} expires in '$expiry'"

done

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

If you are using Amazon RDS for MySQL version 5.7 or 8.0 and the Performance Schema is enabled, run the following query to check if connections are using SSL/TLS

mysql> SELECT id, user, host, connection\_type

FROM performance\_schema.threads pst

INNER JOIN information\_schema.processlist isp

ON pst.processlist\_id = isp.id;

**Output:**

+----+-----------------+------------------+-----------------+

| id | user | host | connection\_type |

+----+-----------------+------------------+-----------------+

| 8 | admin | 10.0.4.249:42590 | SSL/TLS |

| 4 | event\_scheduler | localhost | NULL |

| 10 | webapp1 | 159.28.1.1:42189 | SSL/TLS |

+----+-----------------+------------------+-----------------+

3 rows in set (0.00 sec)

## **Determining whether a client requires certificate verification to connect**

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

### **JDBC**

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

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);*

### **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.

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**

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

*System.setProperty("javax.net.ssl.trustStore", certs);*

*System.setProperty("javax.net.ssl.trustStorePassword", "password");*

When you start the application, set the following properties.

*java -Djavax.net.ssl.trustStore=/path\_to\_truststore/MyTruststore.jks -Djavax.net.ssl.trustStorePassword=my\_truststore\_password com.companyName.MyApplication*

## **Example Java code for establishing SSL connections**

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

public class MySQLSSLTest {

private static final String DB\_USER = "username";

private static final String DB\_PASSWORD = "password";

// This 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 test(String[] args) throws Exception {

Class.forName("com.mysql.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.setProperty("sslMode", "VERIFY\_IDENTITY");

properties.put("user", DB\_USER);

properties.put("password", DB\_PASSWORD);

Connection connection = null;

Statement stmt = null;

ResultSet rs = null;

try {

connection = DriverManager.getConnection("jdbc:mysql://mydatabase.123456789012.us-east-1.rds.amazonaws.com:3306",properties);

stmt = connection.createStatement();

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

} finally {

if (rs != null) {

try {

rs.close();

} catch (SQLException e) {

}

}

if (stmt != null) {

try {

stmt.close();

} catch (SQLException e) {

}

}

if (connection != null) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

return;

}

}

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-2019 certificates.

**Using Kerberos authentication for MySQL**

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

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

## **Region and version availability**

Feature availability and support varies across specific versions of each database engine, and across AWS Regions

## **Overview of Setting up Kerberos authentication for MySQL DB instances**

1. Use AWS Managed Microsoft AD to create an AWS Managed Microsoft AD directory. You can use the AWS Management Console, the AWS CLI, or the AWS Directory Service to create the directory. For details about doing so, see [Create your AWS Managed Microsoft AD directory](https://docs.aws.amazon.com/directoryservice/latest/admin-guide/ms_ad_getting_started_create_directory.html) in the AWS Directory Service Administration Guide.
2. Create an AWS Identity and Access Management (IAM) role that uses the managed IAM policy *AmazonRDSDirectoryServiceAccess.* The role allows Amazon RDS to make calls to your directory.

For the role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see [Activating and deactivating AWS STS in an AWS Region](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp_enable-regions.html#sts-regions-activate-deactivate) in the IAM User Guide.

1. Create and configure users in the AWS Managed Microsoft AD directory using the Microsoft Active Directory tools. For more information about creating users in your Active Directory, see [Manage users and groups in AWS managed Microsoft AD](https://docs.aws.amazon.com/directoryservice/latest/admin-guide/ms_ad_manage_users_groups.html) in the AWS Directory Service Administration Guide.
2. Create or modify a MySQL DB instance. If you use either the CLI or RDS API in the create request, specify a domain identifier with the Domain parameter. Use the d-\* identifier that was generated when you created your directory and the name of the role that you created.

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

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

## **Setting up Kerberos authentication for MySQL DB instances**

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

### **Step 1: Create a directory using AWS Managed Microsoft AD**

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

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

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

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

* Creates a security group for the directory controllers.

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

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

* Create, update, or delete users
* Add resources to your domain such as file or print servers, and then assign permissions for those resources to users in your OU
* Create additional OUs and containers
* Delegate authority
* Restore deleted objects from the Active Directory Recycle Bin
* Run AD and DNS Windows PowerShell modules on the Active Directory Web Service.

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

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

**To create a directory with AWS Managed Microsoft AD**

* 1. Sign in to the AWS Management Console and open the AWS Directory Service console at <https://console.aws.amazon.com/directoryservicev2/>
  2. In the navigation pane, choose **Directories** and choose **Set up Directory**.
  3. Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option that you can currently use with Amazon RDS.
  4. Enter the following information:

**Directory DNS name**

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

**Directory NetBIOS name**

The short name for the directory, such as CORP.

**Directory description**

(Optional) A description for the directory.

**Admin password**

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

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

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

**Confirm password**

The administrator password retyped.

* 1. Choose **Next**.
  2. Enter the following information in the **Networking** section and then choose **Next**:

**VPC**

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

**Subnets**

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

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


              Directory details page during creation
            

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

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


          Directory details page
        

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

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

When a DB instance is created using the AWS Management Console and the console user has the *iam:CreateRole* permission, the console creates this role automatically. In this case, the role name is *rds-directoryservice-kerberos-access-role.* Otherwise, you must create the IAM role manually. When you create this IAM role, choose *Directory Service*, and attach the AWS managed policy *AmazonRDSDirectoryServiceAccess* to it.

**Note**

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

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

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "",

"Effect": "Allow",

"Principal": {

"Service": [

"directoryservice.rds.amazonaws.com",

"rds.amazonaws.com"

]

},

"Action": "sts:AssumeRole"

}

]

}

The role must also have the following IAM role policy.

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"ds:DescribeDirectories",

"ds:AuthorizeApplication",

"ds:UnauthorizeApplication",

"ds:GetAuthorizedApplicationDetails"

],

"Effect": "Allow",

"Resource": "\*"

}

]

}

### **Step 3: Create and configure users**

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

To create users in an AWS Directory Service directory, you must be connected to an Amazon EC2 instance based on Microsoft Windows. This instance must be a member of the AWS Directory Service directory and be logged in as a user that has privileges to create users.

### **Step 4: Create or modify a MySQL DB instance**

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

* Create a new MySQL DB instance using the console, the [create-db-instance](https://docs.aws.amazon.com/cli/latest/reference/rds/create-db-instance.html) CLI command, or the [*CreateDBInstance*](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_CreateDBInstance.html) RDS API operation.
* Modify an existing MySQL DB instance using the console, the [*modify-db-instance*](https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-instance.html) CLI command, or the[*ModifyDBInstance*](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_ModifyDBInstance.html) RDS API operation.
* Restore a MySQL DB instance from a DB snapshot using the console, the [restore-*db-instance-from-db-snapshot*](https://docs.aws.amazon.com/cli/latest/reference/rds/restore-db-instance-from-db-snapshot.html) CLI command, or the [*RestoreDBInstanceFromDBSnapsho*t](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_RestoreDBInstanceFromDBSnapshot.html) RDS API operation.
* Restore a MySQL DB instance to a point-in-time using the console, the [*restore-db-instance-to-point-in-time*](https://docs.aws.amazon.com/cli/latest/reference/rds/restore-db-instance-to-point-in-time.html) CLI command, or the [*RestoreDBInstanceToPointInTime*](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_RestoreDBInstanceToPointInTime.html) RDS API operation.
* Kerberos authentication is only supported for MySQL DB instances in a VPC. The DB instance can be in the same VPC as the directory, or in a different VPC. The DB instance must use a security group that allows egress within the directory's VPC so the DB instance can communicate with the directory.
* When you use the console to create a DB instance, choose **Password and Kerberos authentication** in the **Database authentication** section. Choose **Browse Directory** and then select the directory, or choose **Create a new directory**.


          Kerberos authentication setting when creating a DB instance
        

When you use the console to modify or restore a DB instance, choose the directory in the **Kerberos authentication** section, or choose **Create a new directory**.


          Kerberos authentication setting when modifying or restoring a DB instance
        

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

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

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

For **Linux, macOS, or Unix:**

aws rds modify-db-instance \

--db-instance-identifier *mydbinstance* \

--domain d-*ID* \

--domain-iam-role-name *role-name*

For **Windows:**

aws rds modify-db-instance ^

--db-instance-identifier *mydbinstance* ^

--domain d-*ID* ^

--domain-iam-role-name *role-name*

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

### **Step 5: Create Kerberos authentication MySQL logins**

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

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

*CREATE USER 'testuser'@'%' IDENTIFIED WITH 'auth\_pam';*

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

To require an SSL/TLS encrypted connection for your AD user, run the following command:

UPDATE mysql.user SET ssl\_type = 'any' WHERE ssl\_type = '' AND PLUGIN = 'auth\_pam' and USER = 'testuser';

FLUSH PRIVILEGES;

## **Managing a DB instance in a domain**

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

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

* To reattempt enabling Kerberos authentication for a failed membership, use the *ModifyDBInstance* API operation and specify the current membership's directory ID.
* To update the IAM role name for membership, use the *ModifyDBInstance* API operation and specify the current membership's directory ID and the new IAM role.
* To disable Kerberos authentication on a DB instance, use the *ModifyDBInstance* API operation and specify none as the domain parameter.
* To move a DB instance from one domain to another, use the *ModifyDBInstance* API operation and specify the domain identifier of the new domain as the domain parameter.
* To list membership for each DB instance, use the *DescribeDBInstances* API operation.

### **Understanding domain membership**

After you create or modify your DB instance, it becomes a member of the domain. You can view the status of the domain membership for the DB instance by running the [describe-db-instances](https://docs.aws.amazon.com/cli/latest/reference/rds/describe-db-instances.html) CLI command. The status of the DB instance can be one of the following:

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

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

## **Connecting to MySQL with Kerberos authentication**

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

To create a database user that you can connect to using Kerberos authentication, use an IDENTIFIED WITH clause on the CREATE USER statement. For instructions, see [Step 5: Create Kerberos authentication MySQL logins](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/mysql-kerberos.html#mysql-kerberos-setting-up.create-logins).

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

.At a command prompt, connect to one of the endpoints associated with your MySQL DB instance. Follow the general procedures in [Connecting to a DB instance running the MySQL database engine](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html). When you're prompted for the password, enter the Kerberos password associated with that user name.

## **Restoring a MySQL DB instance and adding it to a domain**

You can restore a DB snapshot or complete a point-in-time restore for a MySQL DB instance and then add it to a domain. After the DB instance is restored, modify the DB instance using the process explained in [Step 4: Create or modify a MySQL DB instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/mysql-kerberos.html#mysql-kerberos-setting-up.create-modify) to add the DB instance to a domain.

## **Kerberos authentication for MySQL limitations**

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

**Automation of Back Ups**

Amazon RDS automatically creates a [backup](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithAutomatedBackups.html) of the underlying storage volumes associated with your Amazon RDS DB instance. All of your data is backed up into Amazon Simple Storage Service (Amazon S3), based on a retention policy that runs for up to 35 days. These backups allow you to perform a [point-in-time recovery (PITR)](https://docs.aws.amazon.com/aws-backup/latest/devguide/point-in-time-recovery.html) at any time within your retention period.

By default, this automation occurs only once each day during a [30 minute backup window](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithAutomatedBackups.html#USER_WorkingWithAutomatedBackups.BackupWindow). If you need the automated backup to run more frequently, you can use the following methods:

* [Create an automated backup job](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html) by AWS Lambda or a cron job on Amazon Elastic Compute Cloud (Amazon EC2). The automated backup job must call the [CreateDBSnapshot](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_CreateDBSnapshot.html) action at a required interval and a subsequent [DeleteDBSnapshot](https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_DeleteDBSnapshot.html) action to automate the [deletion of these snapshots](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_DeleteSnapshot.html).  
  --Or--
* Use [AWS Backup](https://aws.amazon.com/getting-started/hands-on/amazon-rds-backup-restore-using-aws-backup/), automating the scheduled backup.

To create an automated backup job in Amazon RDS, which captures daily snapshots at a specific interval, perform the following steps:

1.    Open the [AWS Backup console](https://console.aws.amazon.com/backup/).

2.    Choose **Build a new plan** to create a new backup plan.

3.    Enter the **Backup plan name** and any relevant tag information.

4.    Under **Backup configuration**, update the following:

Backup plan name  
Backup vault  
Backup frequency  
Backup window

For example, if you want to schedule your automated backup to run every six hours, and to complete within three hours, update the following:  
For Backup frequency, select the **Custom cron expression** and type "cron(0 0/6 ? \* \* \*)" for your cron expression. For Backup window, select "Start within 1 hour" and "Complete within 3 hours".

5.    Choose **Copy automatically to a target region**.

**Note:** Not all AWS Regions are supported for automatic cross-Region copies. For more information, see [Replicating automated backups to another AWS region.](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReplicateBackups.html)

6.    Specify the desired retention period for your automated backup. Amazon RDS Backup will retain the backup for the specified duration until the backup is automatically deleted.

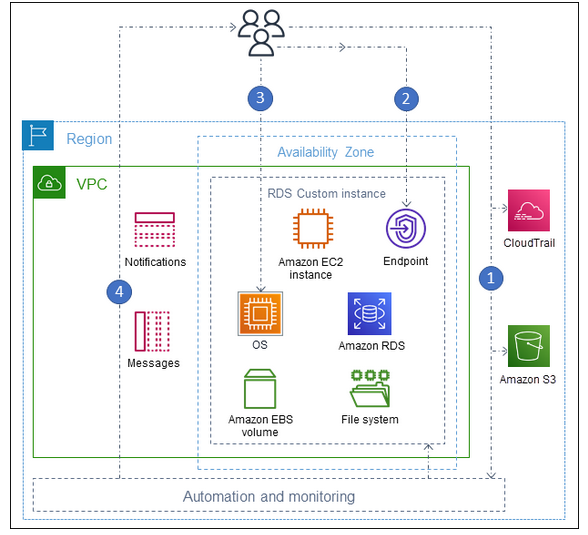
7.    Choose **Create plan**.

8.    Under the **Newly created backup plan** tab, choose **Assign resources**.

9.    Assign your resources by updating the following information:

* Resource assignment name.
* AWS Identity Access Management (IAM) role (to create and manage DB instance recovery points).
* Amazon RDS resource (by tag or resource ID)

**Work Flow**



**Limitations**

1. **InnoDB** is a reserved word for RDS for MySQL. You can't use this name for a MySQL database.
2. When storage becomes full for a MySQL DB instance, there can be metadata inconsistencies, dictionary mismatches, and orphan tables.
3. Inconsistent InnoDB buffer pool size.
4. Index merge optimization returns incorrect results.
5. Some MySQL parameters require special considerations when used with an Amazon RDS DB instance.
6. For MySQL DB instances, the maximum provisioned storage limit constrains the size of a table to a maximum size of 16 TB when using InnoDB file-per-table tablespaces.
7. Currently, Amazon RDS for MySQL does not support the MySQL *keyring\_aws* Amazon Web Services Keyring Plugin.