Project in AWS
Practice Lab

Using Secrets Manager to Authenticate with an RDS Database Using Lambda

Andra-Diana Popescu

2025

ABOUT THIS LAB

AWS Secrets Manager helps you protect secrets needed to access your applications, services, and IT resources. The service enables you to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle.

In this lab, we connect to a MySQL RDS database from an AWS Lambda function using a username and password, and then we hand over credential management to the AWS Secrets Manager service. We then use the Secrets Manager API to connect to the database instead of hard-coding credentials in our Lambda function. By the end of this lab, you will understand how to store a secret in AWS Secrets Manager and access it from a Lambda function.

LEARNING OBJECTIVES

- Create a Lambda Function
- Create the SQS Trigger
- Create a Secret in Secrets Manager
- Test Connectivity from Lambda to RDS Using Credentials from AWS Secrets Manager
  Create Table in the RDS Database Using Lambda to Check Connectivity
- Modify the Lambda IAM Role

AWS Documentation about Secrets Manager, RDS and Lambda:

https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html

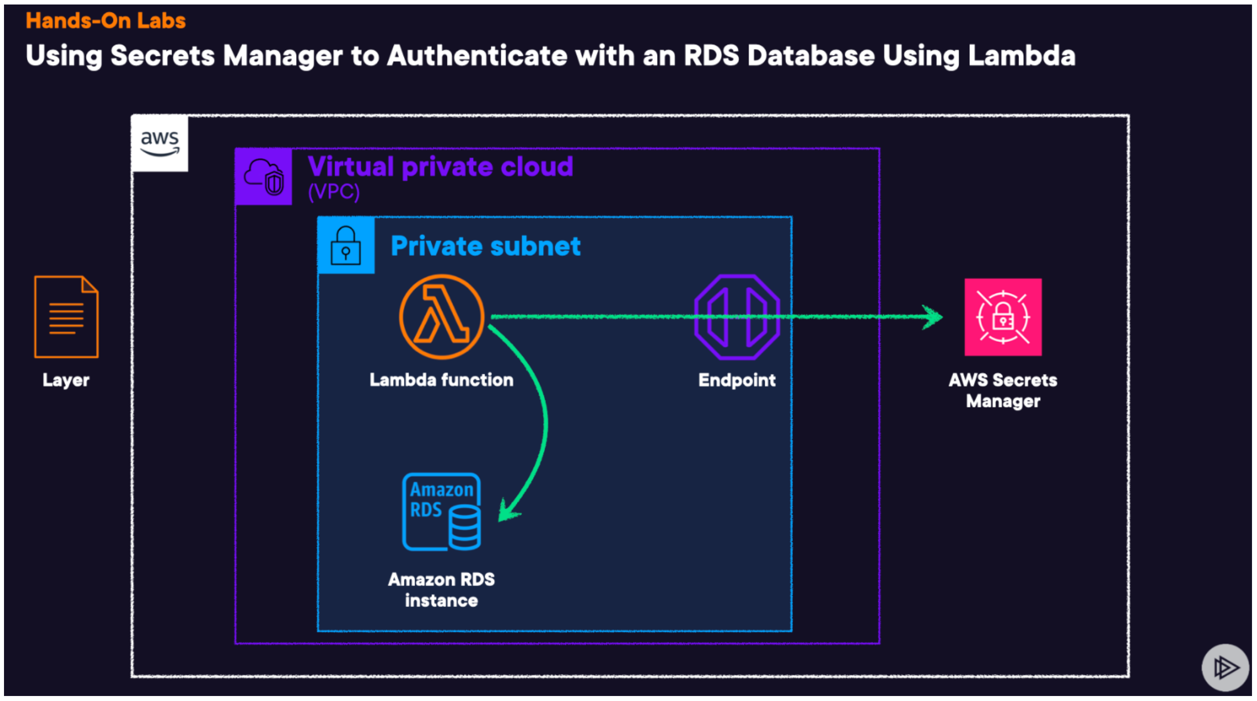
https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

Source: https://learn.acloud.guru/course/certified-solutions-architect-associate/

Table of Contents

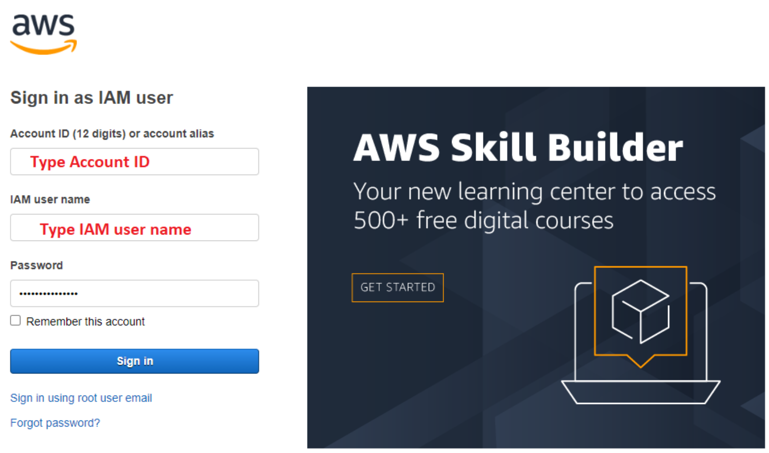
Lab Diagrams



We have the AWS account in **us-east-1** Region. In this lab, you are working for a well-known bank who is looking to transform some of their core processing to Lambda. You've been handed the project as the lead engineer and told this will only be permitted if you can demonstrate how you can effectively configure Lambda to authenticate without the need to hard code credentials. Your VP of security is also mandated that due to the sensitivity of the information held in this application, you'll need to find a way to rotate passwords on a regular basis.

You'll start this lab having been provided with a VPC with 2 private subnets. In these subnets, we have provided you with an RDS database running MySQL. You will first create your Lambda function using the code provided, selecting the deployment method as enabling VPC. Once created, you will create and attach a MySQL layer to the function and test connectivity with the RDS endpoint. Once this has been successfully tested, you'll create an AWS Secrets Manager Secret and enable the password rotation. Finally, once the password's been rotated, you will deploy the code provided to have the Lambda function retrieve the secret from Secrets Manager and authenticate the RDS database.

Log in to your AWS account


Sign in as IAM user

Account ID (12 digits) or account alias

Type Account ID

IAM user name

Type IAM user name

Password

Remember this account

Sign in

Sign in using root user email

Forgot password?

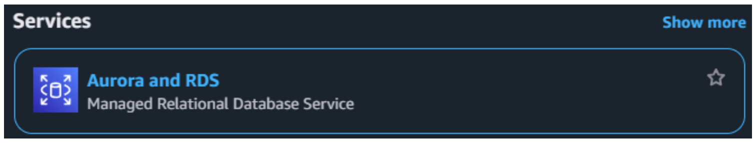
AWS Skill Builder

Your new learning center to access 500+ free digital courses

GET STARTED

1. Create the Lambda Function

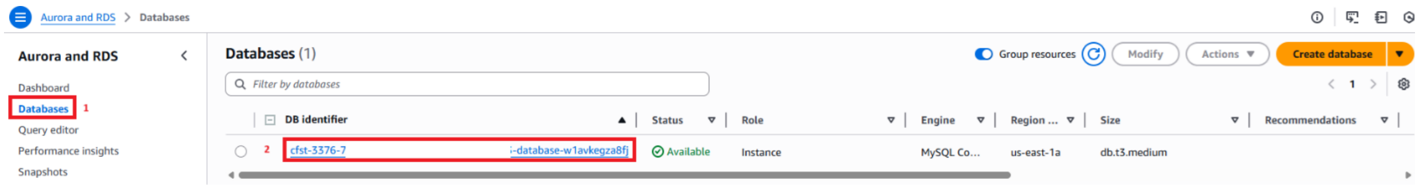
1. Once you are logged into the AWS Management Console, navigate to **RDS** in a new tab.


Services — Show more

Aurora and RDS

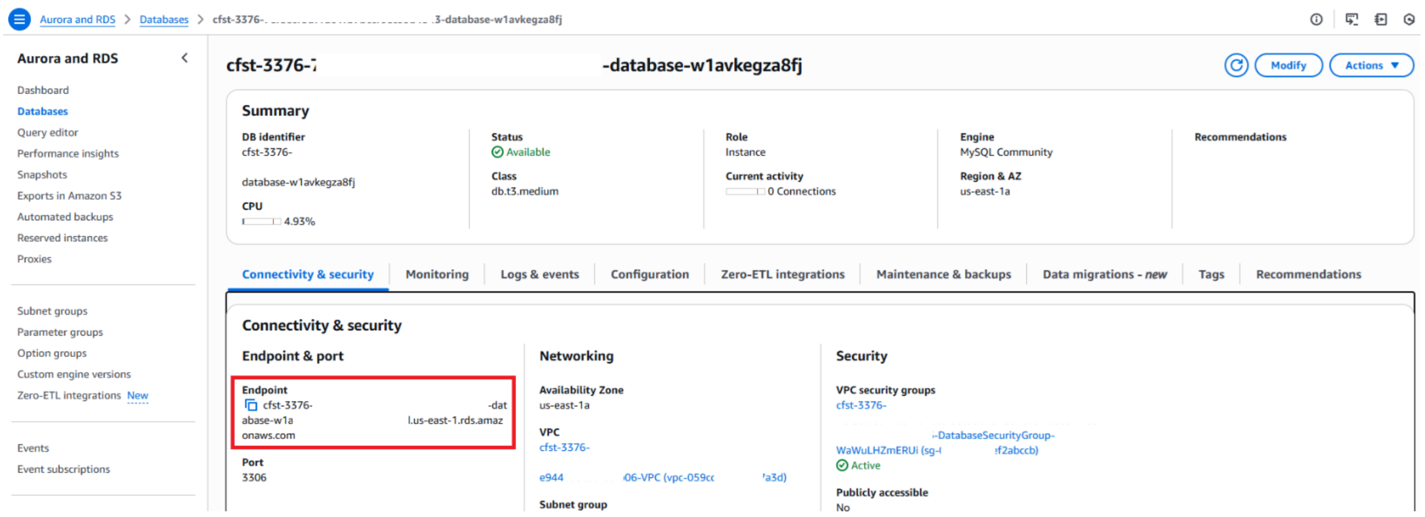
Managed Relational Database Service

2. From the left navigation menu, click **Databases**.
3. Click the link for the displayed RDS database.



	DB identifier	Status	Role	Engine	Region ...	Size	Recommendations
○	cfst-3376-7 -database-w1avkegza8fj	Available	Instance	MySQL Co...	us-east-1a	db.t3.medium	

4. Under the **Connectivity & security** section, copy the value for **Endpoint**. Save the value to a text file to use for later.



Aurora and RDS > Databases > cfst-3376-... -3-database-w1avkegza8fj

Aurora and RDS

- Dashboard
- Databases
- Query editor
- Performance insights
- Snapshots
- Exports in Amazon S3
- Automated backups
- Reserved instances
- Proxies
- Subnet groups
- Parameter groups
- Option groups
- Custom engine versions
- Zero-ETL integrations New
- Events
- Event subscriptions

cfst-3376- -database-w1avkegza8fj

Modify | Actions

Summary

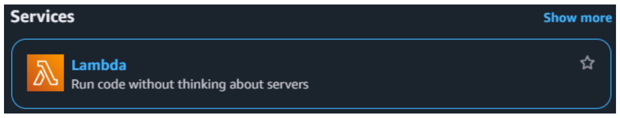
DB identifier	Status	Role	Engine	Recommendations
cfst-3376- database-w1avkegza8fj	Available	Instance	MySQL Community	
CPU 4.93%	Class db.t3.medium	Current activity 0 Connections	Region & AZ us-east-1a	

Connectivity & security | Monitoring | Logs & events | Configuration | Zero-ETL integrations | Maintenance & backups | Data migrations - new | Tags | Recommendations

Connectivity & security

Endpoint & port	Networking	Security
Endpoint: cfst-3376- -dat abase-w1a l.us-east-1.rds.amaz onaws.com	Availability Zone: us-east-1a	VPC security groups: cfst-3376- -DatabaseSecurityGroup-WaWuLHZmERUi (sg- f2abccb) Active
Port: 3306	VPC: cfst-3376- e944 06-VPC (vpc-059c a3d)	Publicly accessible: No
	Subnet group	

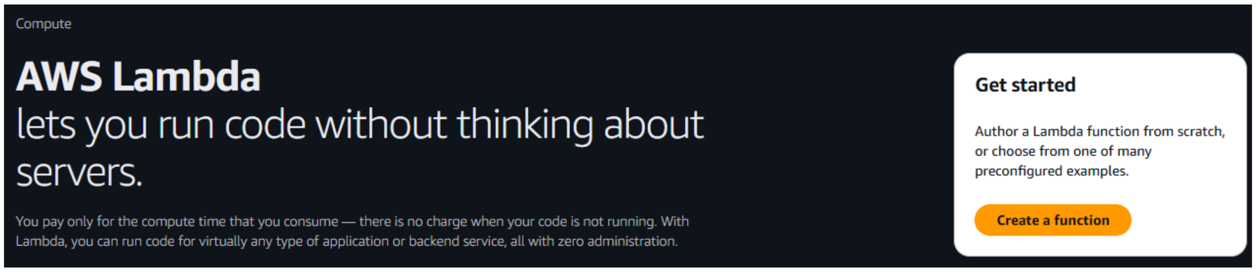
5. In a new tab, navigate to **Lambda**.



Services — Show more

Lambda — Run code without thinking about servers

6. Click **Create a function**.



Compute

AWS Lambda lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

Get started

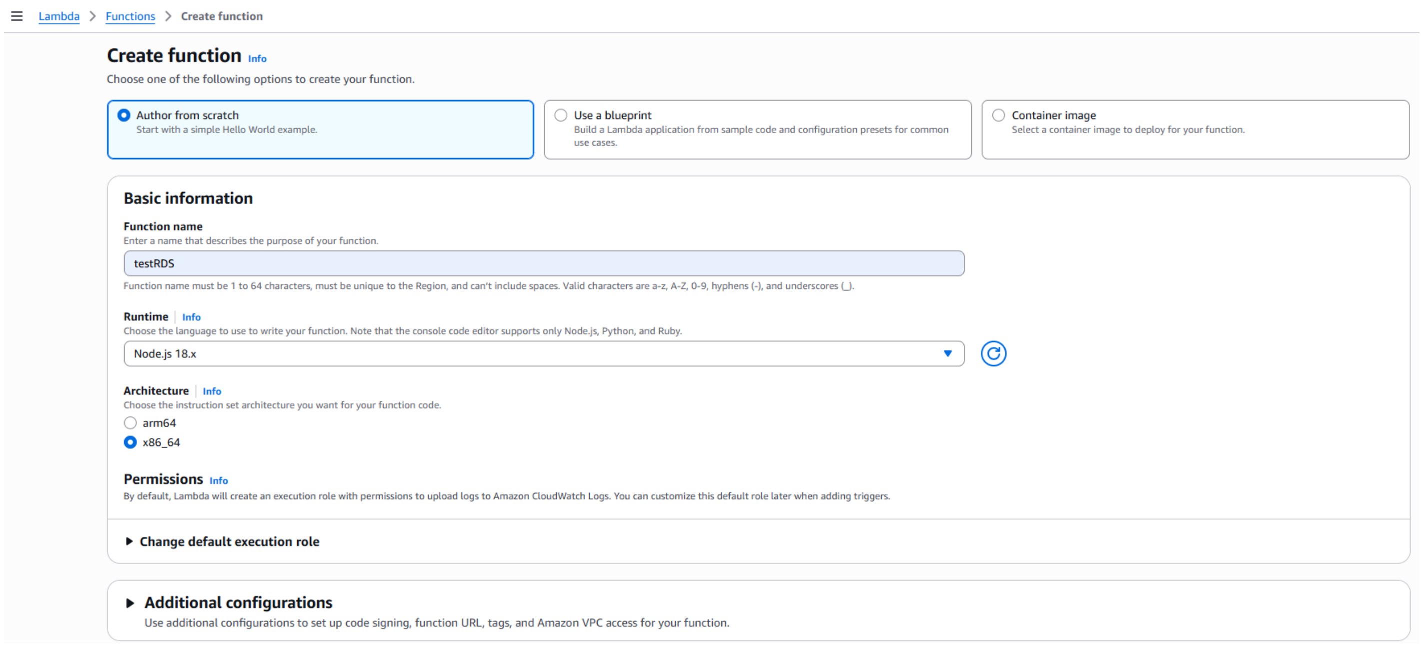
Author a Lambda function from scratch, or choose from one of many preconfigured examples.

Create a function

7. Make sure the **Author from scratch** option at the top is selected, and then use the following settings:

   a. ***Function name***: Enter **testRDS**.

   b. ***Runtime***: Select **Node.js 18.x**.


Lambda > Functions > Create function

Create function Info

Choose one of the following options to create your function.

- [✓ Author from scratch] Start with a simple Hello World example.
- Use a blueprint — Build a Lambda application from sample code and configuration presets for common use cases.
- Container image — Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.
testRDS
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime | Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 18.x

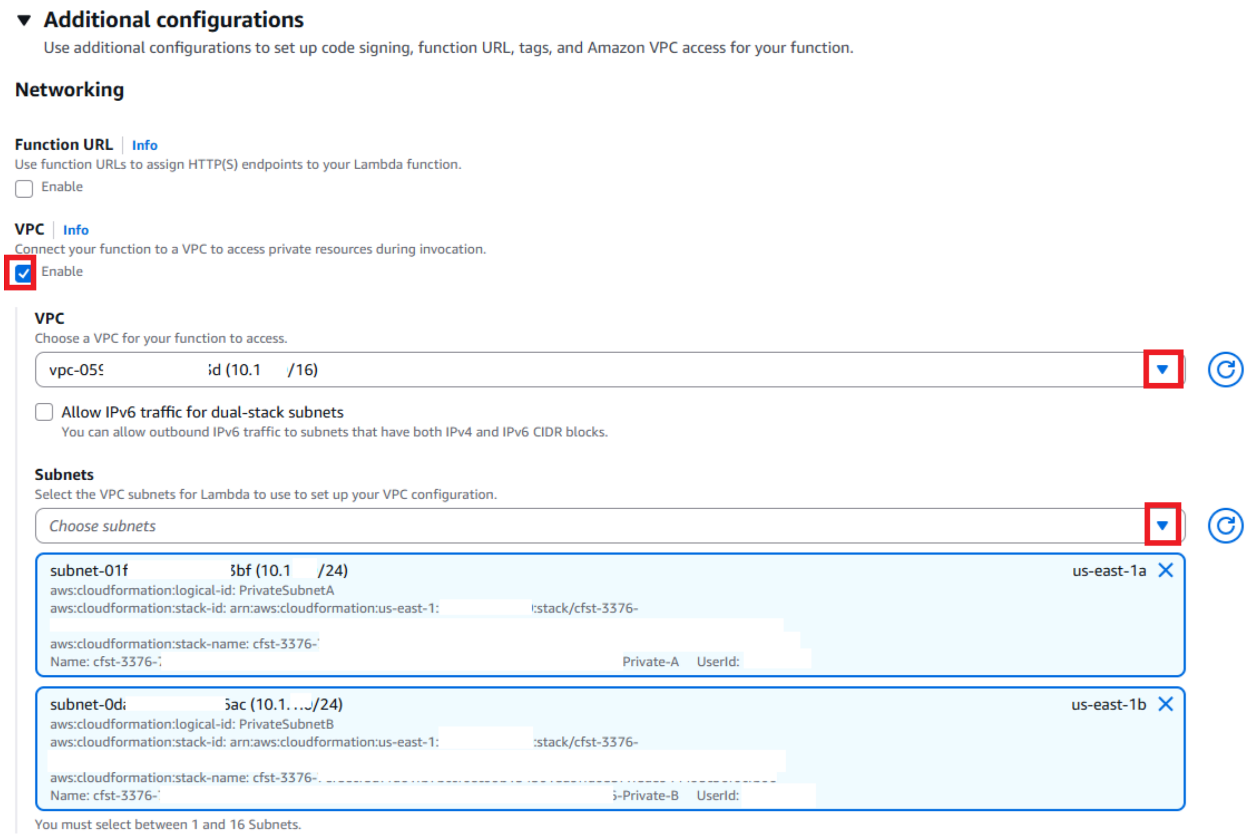
Architecture | Info
Choose the instruction set architecture you want for your function code.
arm64 [✓ x86_64]

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▶ Change default execution role

▶ Additional configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

8. Expand **Additional configurations**, click **Enable VPC**, and set the following values:
   a. ***VPC***: Select the lab-provided VPC.
   b. ***Subnets***: Select the two subnets.


▼ Additional configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Networking

Function URL | Info
Use function URLs to assign HTTP(S) endpoints to your Lambda function.
☐ Enable

VPC | Info
Connect your function to a VPC to access private resources during invocation.
☑ Enable

VPC
Choose a VPC for your function to access.
vpc-05ɩ ɩd (10.1 /16)

☐ Allow IPv6 traffic for dual-stack subnets
You can allow outbound IPv6 traffic to subnets that have both IPv4 and IPv6 CIDR blocks.

Subnets
Select the VPC subnets for Lambda to use to set up your VPC configuration.
Choose subnets

subnet-01f 3bf (10.1 /24) — us-east-1a
aws:cloudformation:logical-id: PrivateSubnetA
aws:cloudformation:stack-id: arn:aws:cloudformation:us-east-1: :stack/cfst-3376-
aws:cloudformation:stack-name: cfst-3376-
Name: cfst-3376- Private-A UserId:

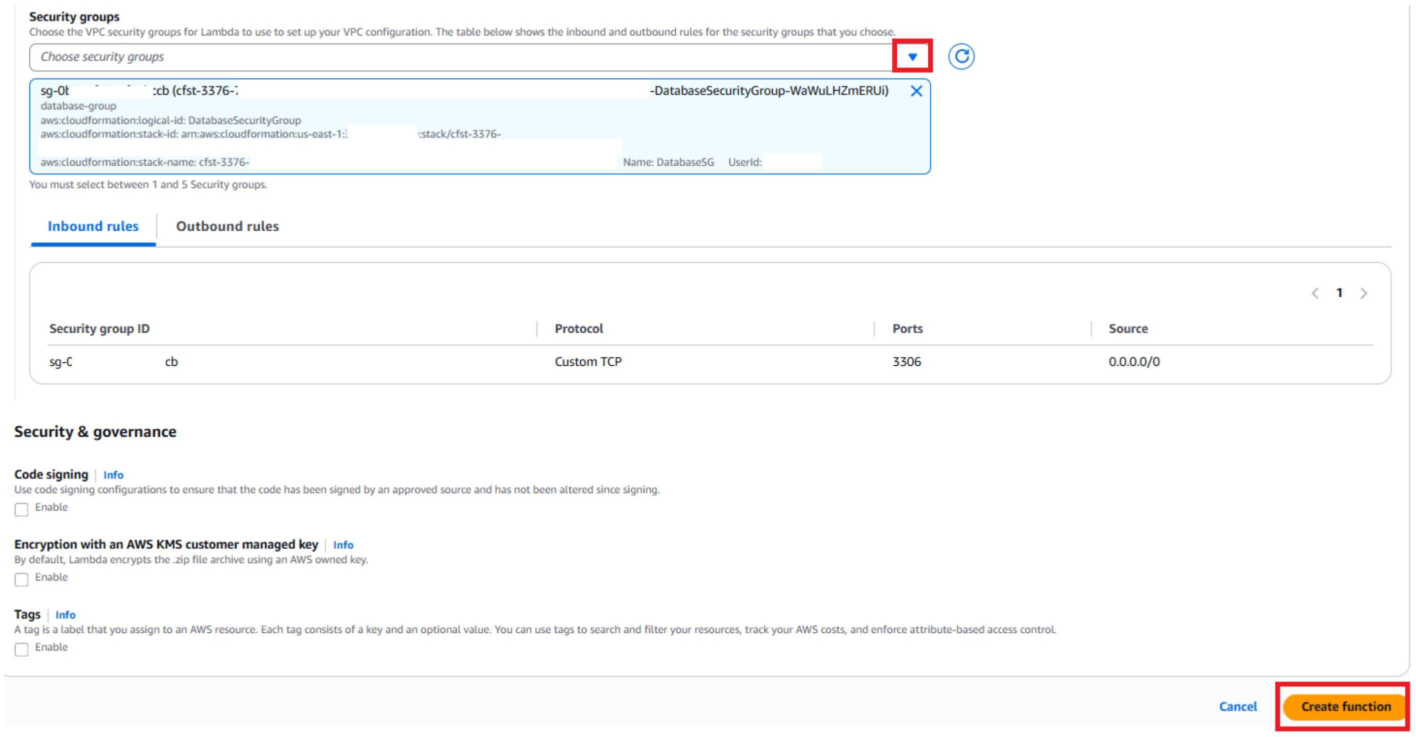
subnet-0d 5ac (10.1. /24) — us-east-1b
aws:cloudformation:logical-id: PrivateSubnetB
aws:cloudformation:stack-id: arn:aws:cloudformation:us-east-1: :stack/cfst-3376-
aws:cloudformation:stack-name: cfst-3376-
Name: cfst-3376- -Private-B UserId:

You must select between 1 and 16 Subnets.

c. ***Security groups***: Select the lab-provided **DatabaseSecurityGroup** security group (**NOT** the default security group).

9. Click **Create function**.

   Note: It may take up to 5 to 10 minutes to finish creating. The blue bar at the top of the page will indicate the function is being created.



Security groups
Choose the VPC security groups for Lambda to use to set up your VPC configuration. The table below shows the inbound and outbound rules for the security groups that you choose.
Choose security groups

sg-0b [redacted] cb (cfst-3376- [redacted] -DatabaseSecurityGroup-WaWuLHZmERUi)
database-group
aws:cloudformation:logical-id: DatabaseSecurityGroup
aws:cloudformation:stack-id: arn:aws:cloudformation:us-east-1: [redacted] :stack/cfst-3376- [redacted]
aws:cloudformation:stack-name: cfst-3376- [redacted] Name: DatabaseSG UserId: [redacted]

You must select between 1 and 5 Security groups.

Inbound rules | Outbound rules

Security group ID	Protocol	Ports	Source
sg-C [redacted] cb	Custom TCP	3306	0.0.0.0/0

Security & governance

Code signing Info
Use code signing configurations to ensure that the code has been signed by an approved source and has not been altered since signing.
☐ Enable

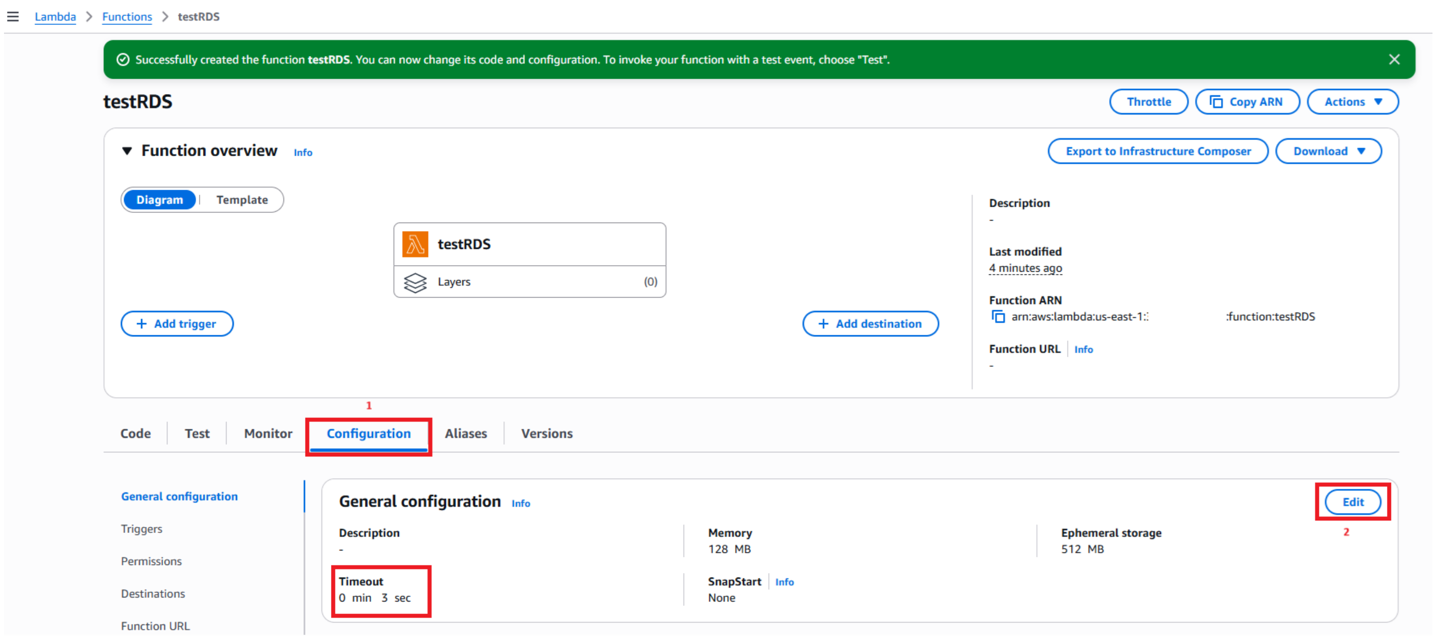
Encryption with an AWS KMS customer managed key Info
By default, Lambda encrypts the .zip file archive using an AWS owned key.
☐ Enable

Tags Info
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources, track your AWS costs, and enforce attribute-based access control.
☐ Enable

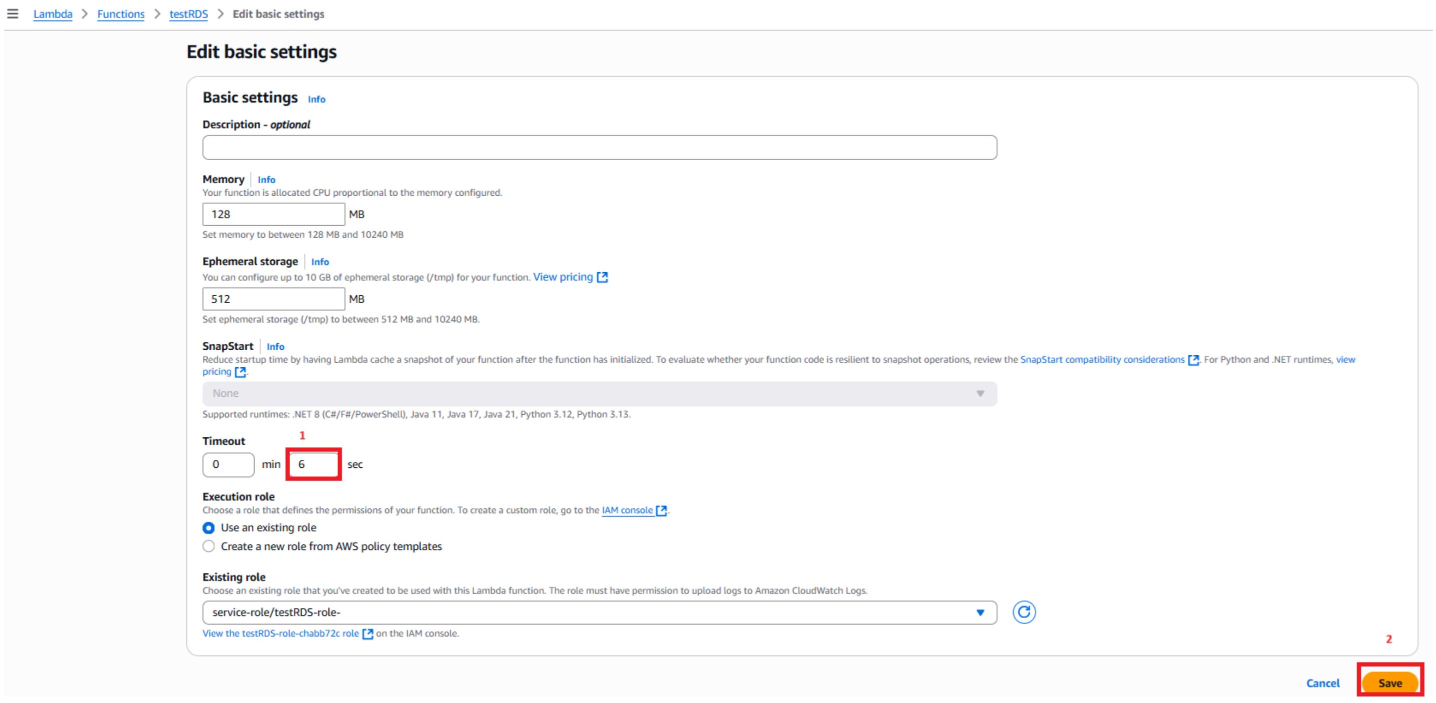
Cancel | Create function

2. Create the MySQL Layer, and Copy Your Code to the Lambda Function

1. Once the function has been created, select the **Configuration** tab.
2. Next to **General configuration**, click **Edit**.

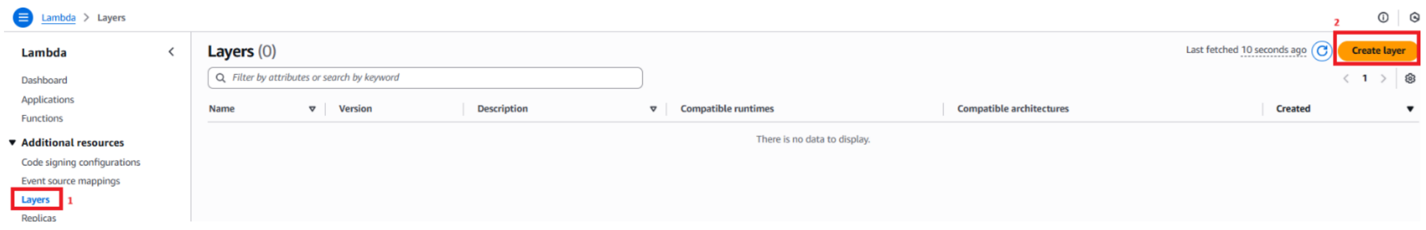


3. Change the **Timeout** setting from 3 seconds to **6** seconds, and click **Save**. This will allow the Lambda function a slightly time to execute for the code that we're providing it.


Lambda > Functions > testRDS > Edit basic settings
Edit basic settings
Basic settings Info
Description - optional
Memory Info
Your function is allocated CPU proportional to the memory configured.
128 MB
Set memory to between 128 MB and 10240 MB
Ephemeral storage Info
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. View pricing
512 MB
Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.
SnapStart Info
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the SnapStart compatibility considerations. For Python and .NET runtimes, view pricing.
None
Supported runtimes: .NET 8 (C#/F#/PowerShell), Java 11, Java 17, Java 21, Python 3.12, Python 3.13.
Timeout
1
0 min 6 sec
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Use an existing role
Create a new role from AWS policy templates
Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
service-role/testRDS-role-
View the testRDS-role-chabb72c role on the IAM console.
2
Cancel
Save

4. Select the hamburger menu on the left-hand side, then click **Layers**.

5. Click **Create layer**.



6. Set the following values:

   a. ***Name***: Enter **mysql**.

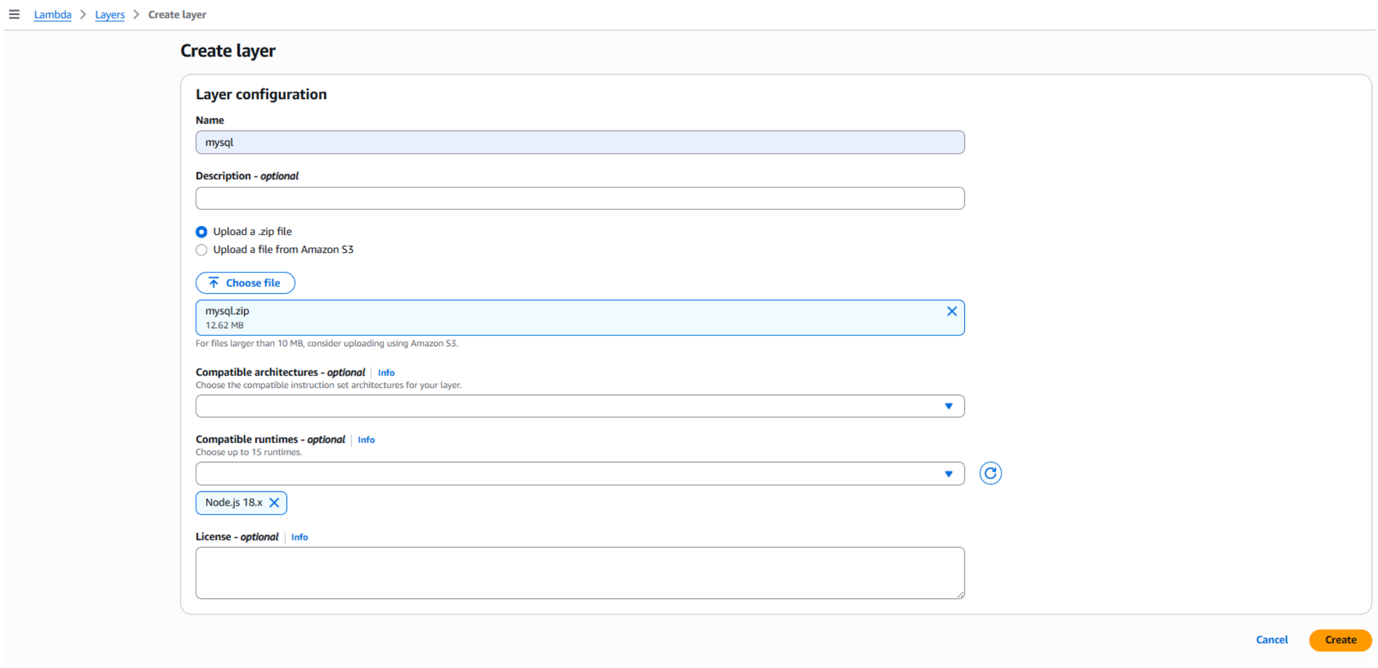
   b. ***Upload a .zip file***: Select this option, and upload the file:

      - Click **Upload**.

      - Upload the **MySQL Library ZIP file** you downloaded earlier.

   c. ***Compatible runtimes***: Select **Node.js 18.x**.

7. Click **Create**.


Lambda > Layers > Create layer
Create layer
Layer configuration
Choose file
Compatible architectures - optional
Info
Choose the compatible instruction set architectures for your layer.
Compatible runtimes - optional
Info
Choose up to 15 runtimes.
License - optional
Info
Cancel
Create

Name: mysql

Description - optional:

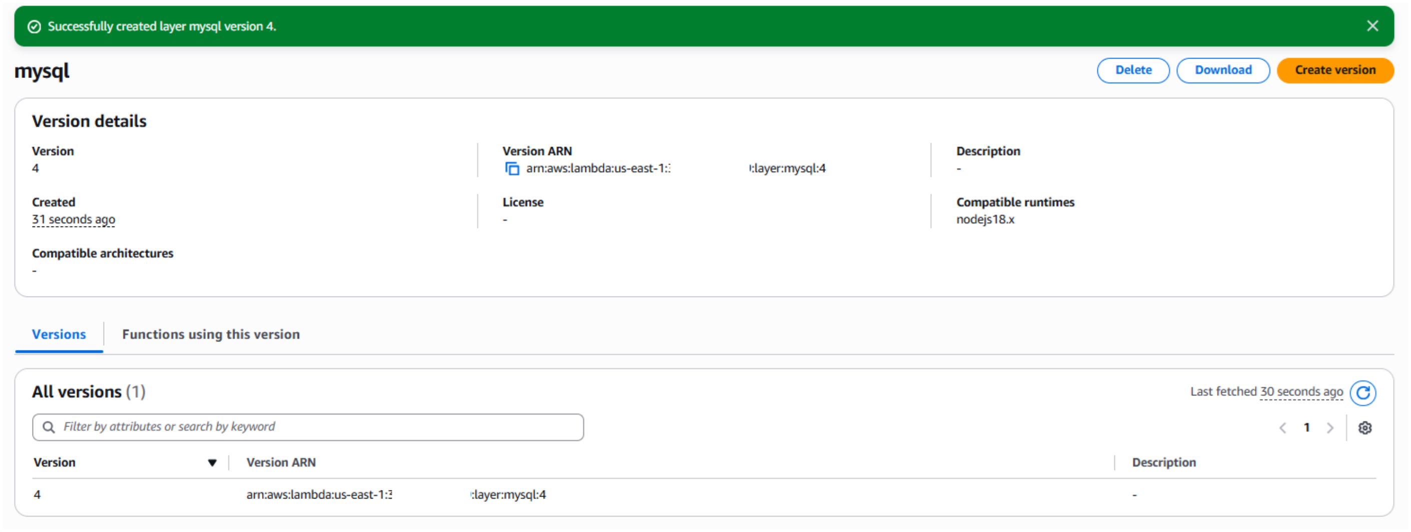
Upload a .zip file [✓]

Upload a file from Amazon S3

mysql.zip
12.62 MB

For files larger than 10 MB, consider uploading using Amazon S3.

Compatible runtimes: Node.js 18.x


Successfully created layer mysql version 4.

mysql

Delete
Download
Create version

Version details

Version	Version ARN	Description
4	arn:aws:lambda:us-east-1:[redacted]:layer:mysql:4	-

Created	License	Compatible runtimes
31 seconds ago	-	nodejs18.x

Compatible architectures
-

Versions
Functions using this version

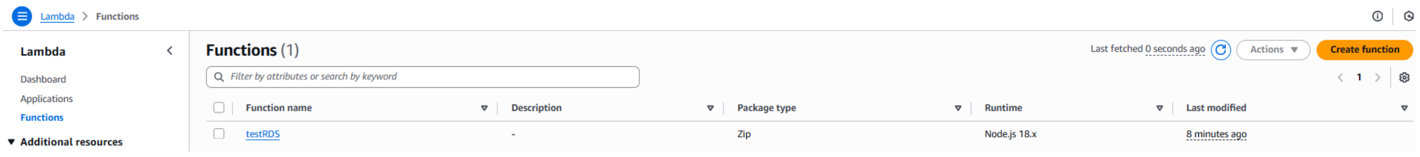
All versions (1)

Last fetched 30 seconds ago

Filter by attributes or search by keyword

Version	Version ARN	Description
4	arn:aws:lambda:us-east-1:[redacted]:layer:mysql:4	-

8. On the left, click the hamburger menu, and select **Functions**.
9. Select the **testRDS** function.

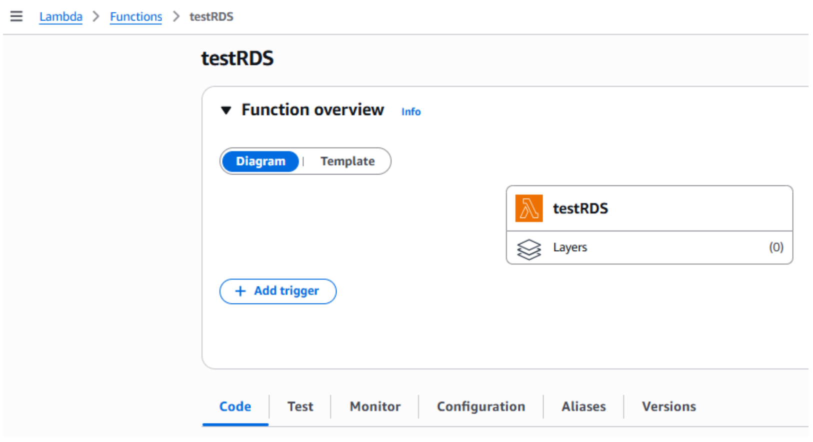


Functions (1)

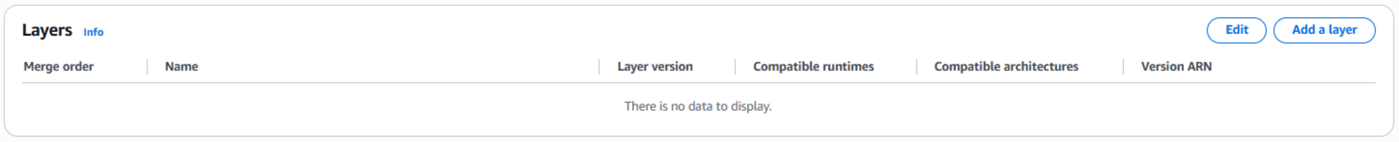
Last fetched 0 seconds ago

Function name	Description	Package type	Runtime	Last modified
testRDS	-	Zip	Node.js 18.x	8 minutes ago

10. Once on the **testRDS** page, ensure the **Code** tab is selected, and scroll down to **Layers**.



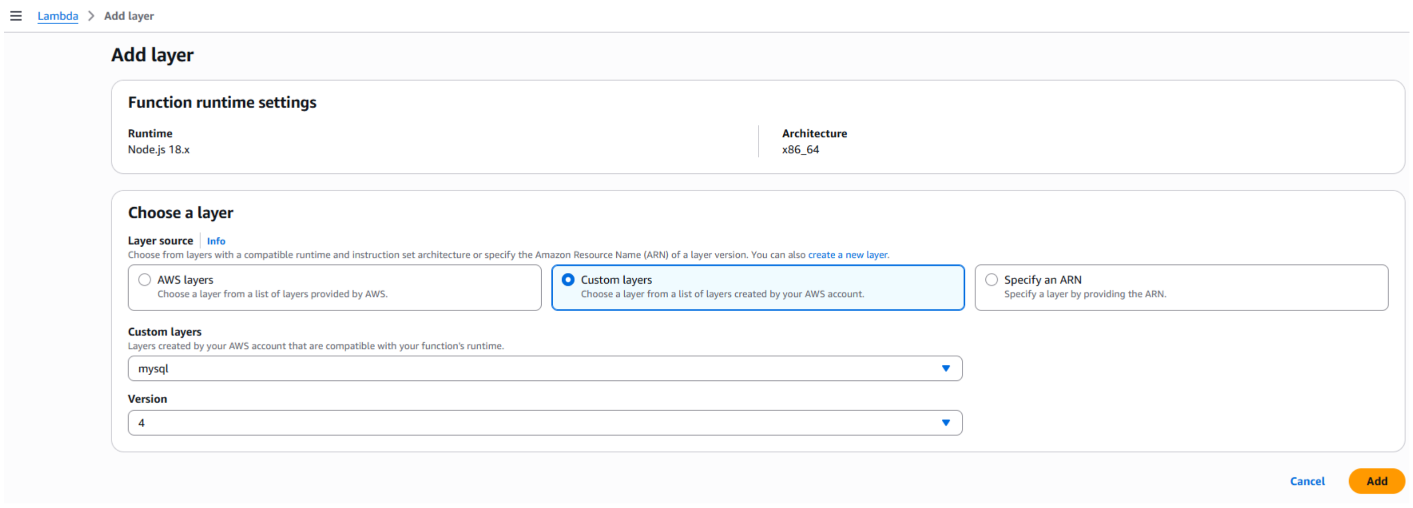
11. In the **Layers** section, click **Add a layer**.



Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN

There is no data to display.

12. Select **Custom layers**, and select the following values:
    a. ***Custom layers***: Select **mysql**.
    b. ***Version***: Select the displayed version.
13. Click **Add**. Wait a minute for the function to update.


Lambda > Add layer

Add layer

Function runtime settings

Runtime: Node.js 18.x

Architecture: x86_64

Choose a layer

Layer source | Info

Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also create a new layer.

- ○ AWS layers — Choose a layer from a list of layers provided by AWS.
- ● Custom layers — Choose a layer from a list of layers created by your AWS account.
- ○ Specify an ARN — Specify a layer by providing the ARN.

Custom layers

Layers created by your AWS account that are compatible with your function's runtime.

mysql

Version

4

Now our layer's been added to our Lambda function. The next thing we're going to do is modify the code in the ***index.mjs*** file.

3. Create Table in the RDS Database Using Lambda to Check Connectivity

1. In the **Code source** section, replace the existing code in the **index.mjs** file with the following code:

```
import mysql from 'mysql2/promise';

export const handler = async (event, context, callback) => {
  try {
    const connection = await mysql.createConnection({
      host: "<RDS Endpoint>",
      user: "username",
      password: "password",
      database: "example",
    });

    // Create 'pets' table
    await connection.execute(`
      CREATE TABLE IF NOT EXISTS pets (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        age INT NOT NULL
      )
    `);

    console.log('Table created: pets');

    // List all tables
    const [rows] = await connection.execute('SHOW TABLES');
    console.log('Tables:');
    rows.forEach((row) => {
      console.log(row[`Tables_in_example`]);
    });

    connection.end();

    callback(null, {
      statusCode: 200,
      body: 'Tables listed successfully',
    });
  } catch (err) {
    console.error(err);
    callback(err, {
      statusCode: 500,
      body: 'Error listing tables',
    });
  }
};
```

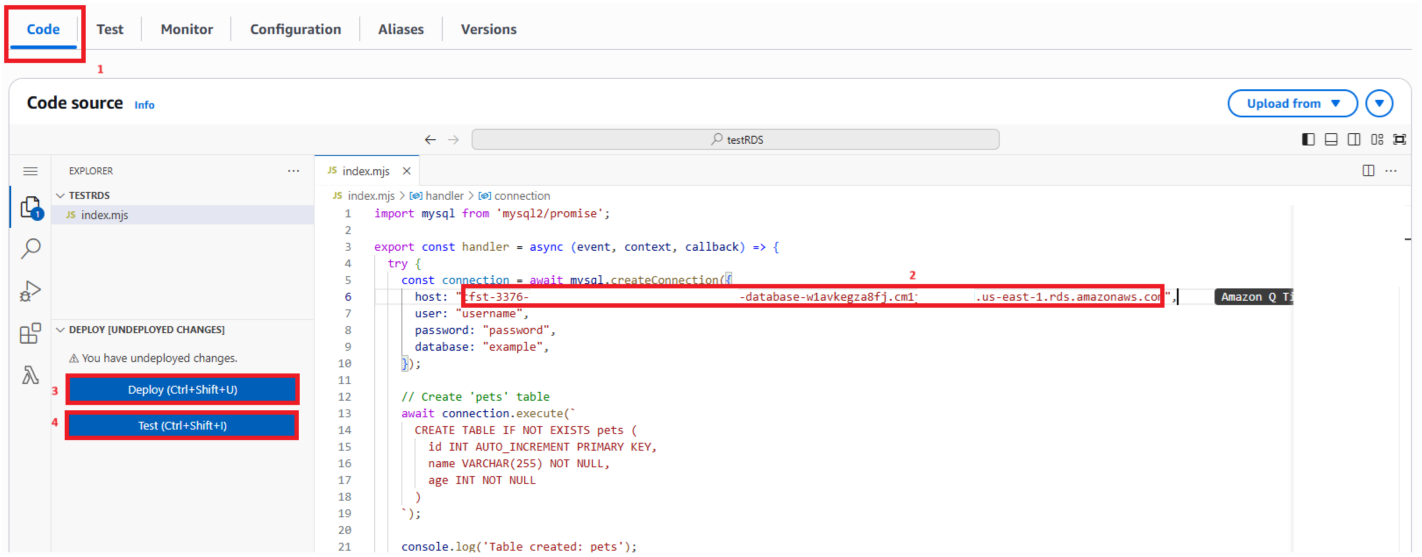
2. Replace the **<RDS Endpoint>** placeholder with the endpoint value you previously copied from RDS. Ensure it remains wrapped in quotes.

   The code will create a new MySQL connection to our database endpoint, using the username as ***username*** and password as ***password***, to gain access. It will be targeting the ***example*** database and it's going to create a table called ***Pets***. Once this is done, a further command's going to be issued, which is a SHOW TABLES command.

   If everything has run successfully, we should get a **200-status code** back and the ***Pets*** table name is listed successfully.

3. To the left of the file, click **Deploy**.

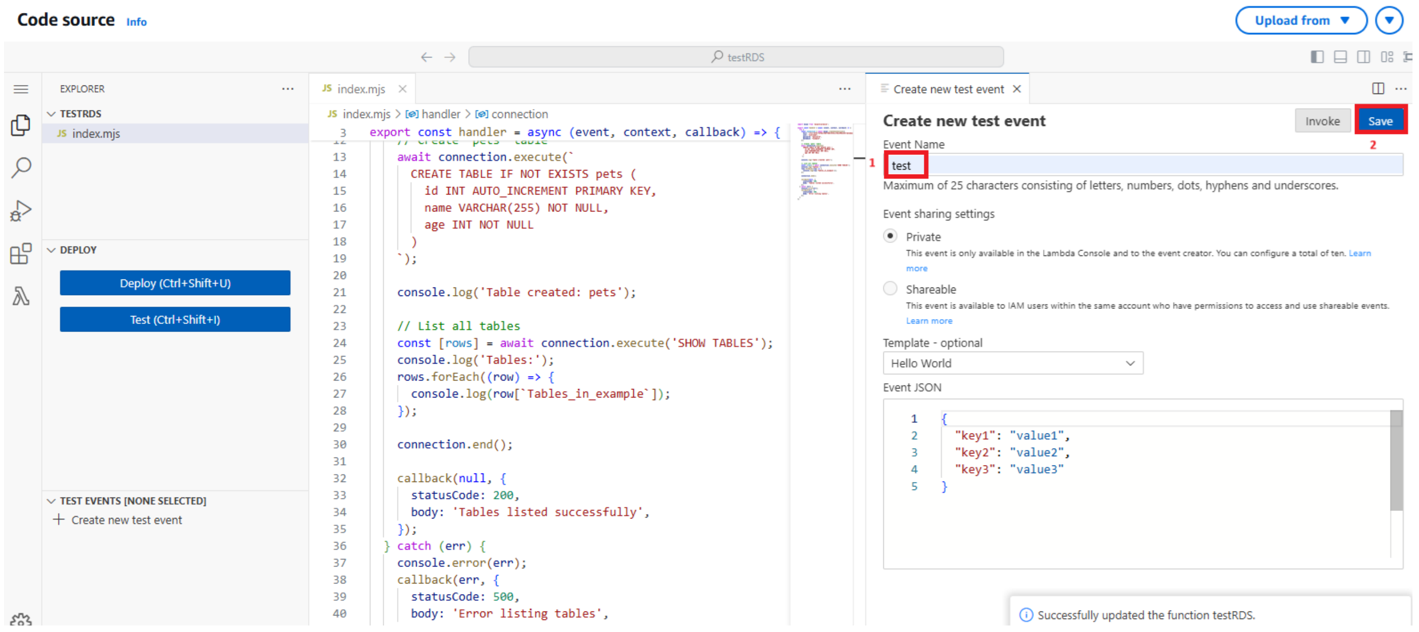
4. Once the function is updated, click the **Test** tab.



```
import mysql from 'mysql2/promise';

export const handler = async (event, context, callback) => {
  try {
    const connection = await mysql.createConnection({
      host: "fst-3376-[redacted]-database-w1avkegza8fj.cm1[redacted].us-east-1.rds.amazonaws.com",
      user: "username",
      password: "password",
      database: "example",
    });

    // Create 'pets' table
    await connection.execute(`
      CREATE TABLE IF NOT EXISTS pets (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        age INT NOT NULL
      )
    `);

    console.log('Table created: pets');
```

5. For **Event name**, enter **test**.

6. Click **Save**.



```
export const handler = async (event, context, callback) => {
    // Create 'pets' table
    await connection.execute(`
      CREATE TABLE IF NOT EXISTS pets (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        age INT NOT NULL
      )
    `);

    console.log('Table created: pets');

    // List all tables
    const [rows] = await connection.execute('SHOW TABLES');
    console.log('Tables:');
    rows.forEach((row) => {
      console.log(row[`Tables_in_example`]);
    });

    connection.end();

    callback(null, {
      statusCode: 200,
      body: 'Tables listed successfully',
    });
  } catch (err) {
    console.error(err);
    callback(err, {
      statusCode: 500,
      body: 'Error listing tables',
```

Create new test event

Event Name: test

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

- [✓ Private] This event is only available in the Lambda Console and to the event creator. You can configure a total of ten. Learn more
- Shareable This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - optional: Hello World

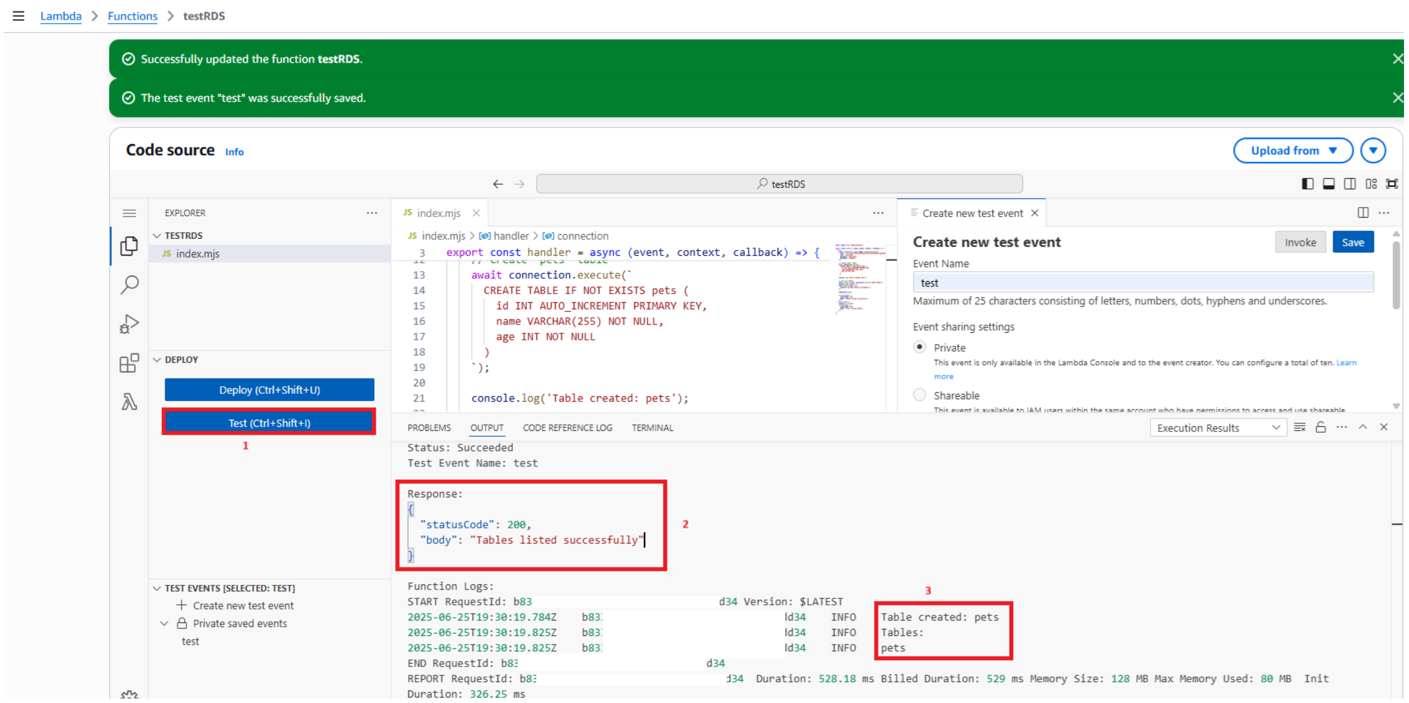
Event JSON

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

Successfully updated the function testRDS.

7. Click **Test**.

8. Expand **Details**, and note the response includes a **statusCode** of **200**.


Lambda > Functions > testRDS
Successfully updated the function testRDS.
The test event "test" was successfully saved.
Code source Info
Upload from
testRDS
EXPLORER
TESTRDS
index.mjs
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
1
TEST EVENTS [SELECTED: TEST]
Create new test event
Private saved events
test
index.mjs > handler > connection
Create new test event
Invoke
Save
Event Name
test
Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.
Event sharing settings
Private
This event is only available in the Lambda Console and to the event creator. You can configure a total of ten. Learn more
Shareable
PROBLEMS
OUTPUT
CODE REFERENCE LOG
TERMINAL
Execution Results
2
3

```
export const handler = async (event, context, callback) => {
    await connection.execute(`
      CREATE TABLE IF NOT EXISTS pets (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        age INT NOT NULL
      )
    `);

    console.log('Table created: pets');
```

```
Status: Succeeded
Test Event Name: test

Response:
{
  "statusCode": 200,
  "body": "Tables listed successfully"
}

Function Logs:
START RequestId: b83[redacted]d34 Version: $LATEST
2025-06-25T19:30:19.784Z    b83[redacted]d34    INFO    Table created: pets
2025-06-25T19:30:19.825Z    b83[redacted]d34    INFO    Tables:
2025-06-25T19:30:19.825Z    b83[redacted]d34    INFO    pets
END RequestId: b83[redacted]d34
REPORT RequestId: b83[redacted]d34  Duration: 528.18 ms Billed Duration: 529 ms Memory Size: 128 MB Max Memory Used: 80 MB  Init Duration: 326.25 ms
```

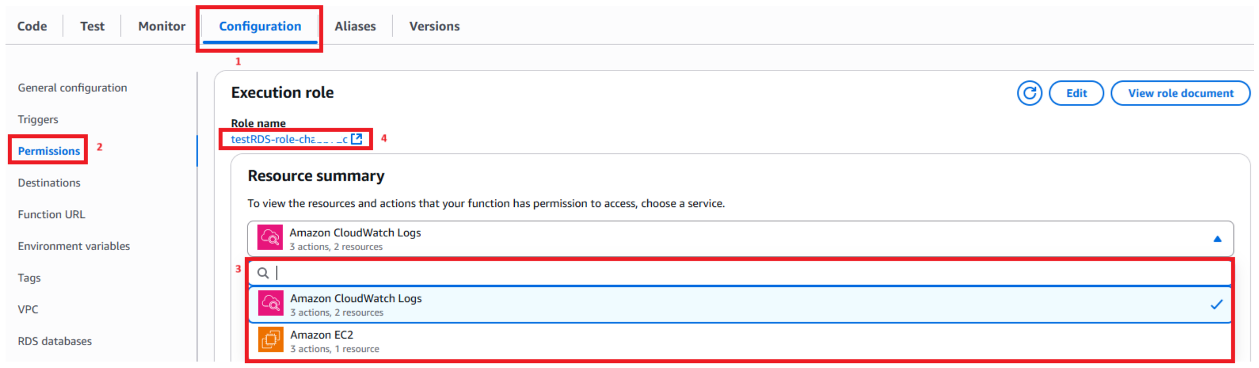
Next, we’ll configure the Lambda execution role to have access to Secrets Manager, and we’ll create our Secrets Manager secret.

4. Modify the Lambda IAM Role

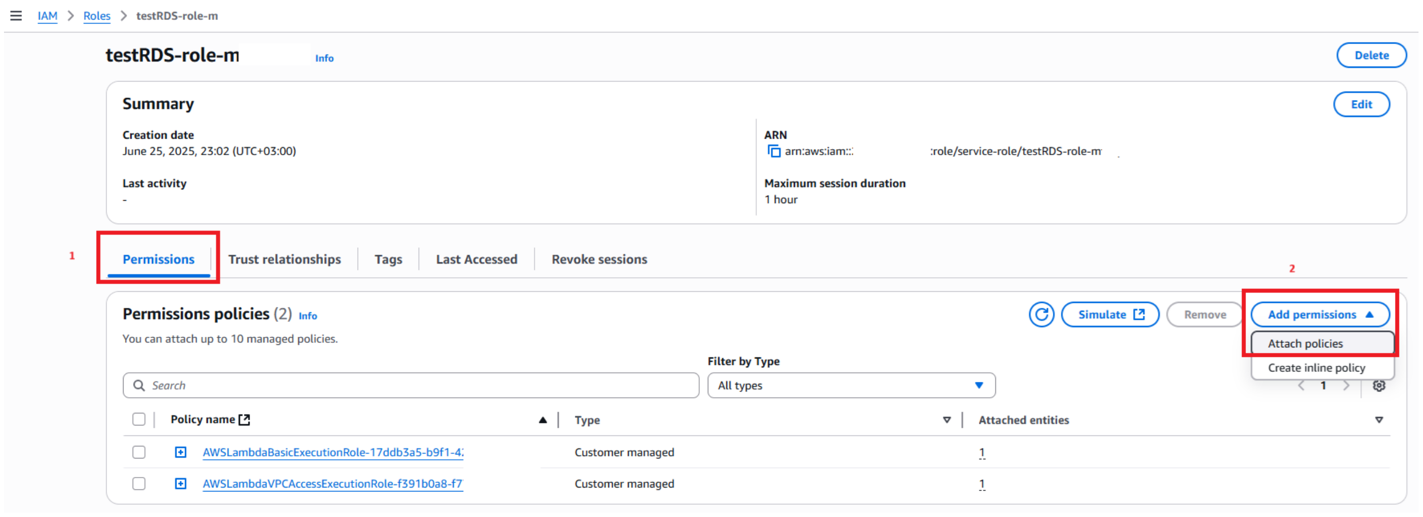
1. Click the **Configuration** tab.
2. From the left menu, select **Permissions**. Review the permitted actions in the **Resource summary** section.

   As we can see, it provides a very nice resource summary, which allows us to see the permissions the Lambda execution role has. At the top, our execution role is called ***testRDS-role*** and a string of characters.

   Under resource summary, we can see the Lambda execution role currently has 3 actions permitted to do against 2 resources in CloudWatch Logs and 3 actions permitted to do against 1 resource on Amazon EC2. Let’s make the changes!
3. Click the **testRDS-role-** link above **Resource summary** to open IAM.



4. As we can see, under permissions, we don’t have permission at the moment allowing access to Secrets Manager. On the right side of the **Permission policies** box, click **Add permissions → Attach policies**.



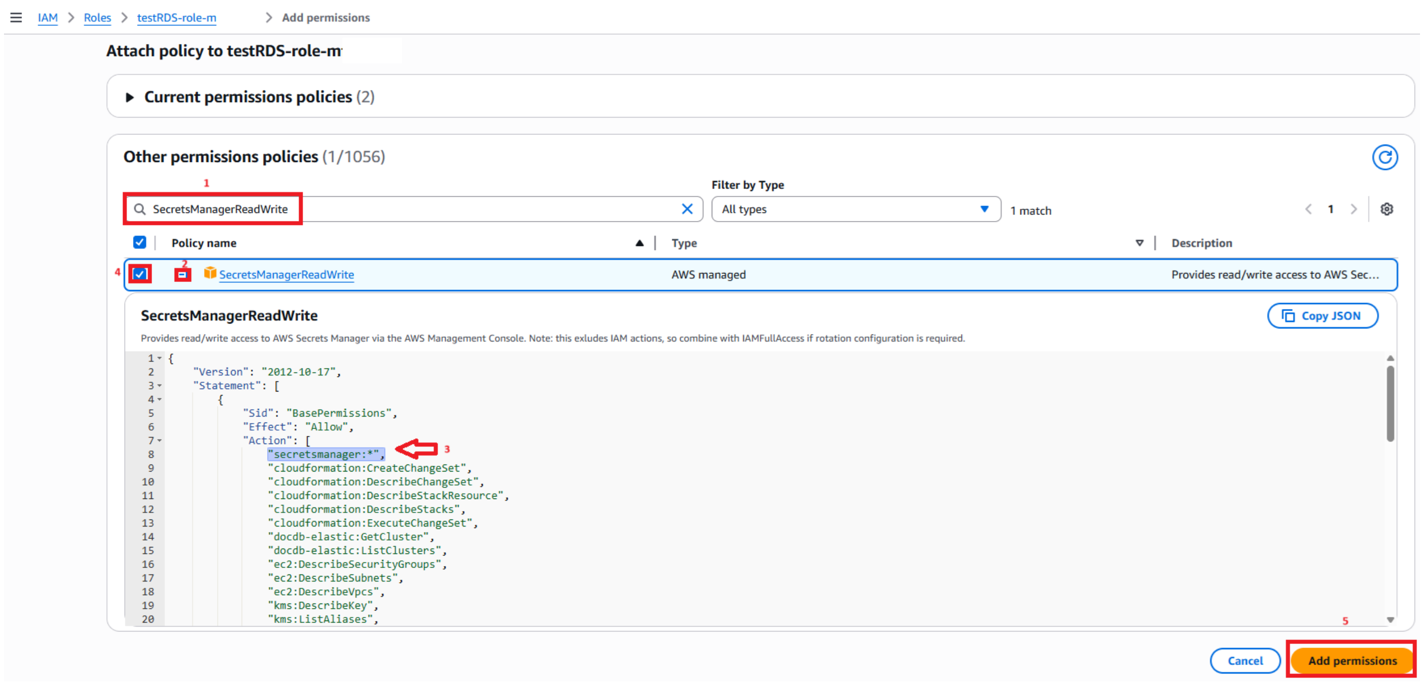
Policy name	Type	Attached entities
AWSLambdaBasicExecutionRole-17ddb3a5-b9f1-4	Customer managed	1
AWSLambdaVPCAccessExecutionRole-f391b0a8-f7	Customer managed	1

5. Search for and check the box next to the **SecretsManagerReadWrite** policy name. This is an AWS-managed policy. Click on the “+” to see what is allowed through the policy.

6. This policy has a few actions to be allowed. This is ok to do in a learning environment, but if this is your production environment, please make sure you’re following the principle of least privilege and instead, configure a custom policy allowing only the permissions you need for the Lambda execution role, which in this case, we can remove the actions to “**secretsmanager:***”.

7. For the lab we are ok with the policy as it is, so click the checkbox of the policy.

8. Click **Add permissions**.


IAM > Roles > testRDS-role-m > Add permissions
Attach policy to testRDS-role-m
Current permissions policies (2)
Other permissions policies (1/1056)
1
SecretsManagerReadWrite
Filter by Type
All types
1 match
Policy name
Type
Description
4
2
Copy JSON
3
5
Cancel
Add permissions

Policy name	Type	Description
SecretsManagerReadWrite	AWS managed	Provides read/write access to AWS Sec...

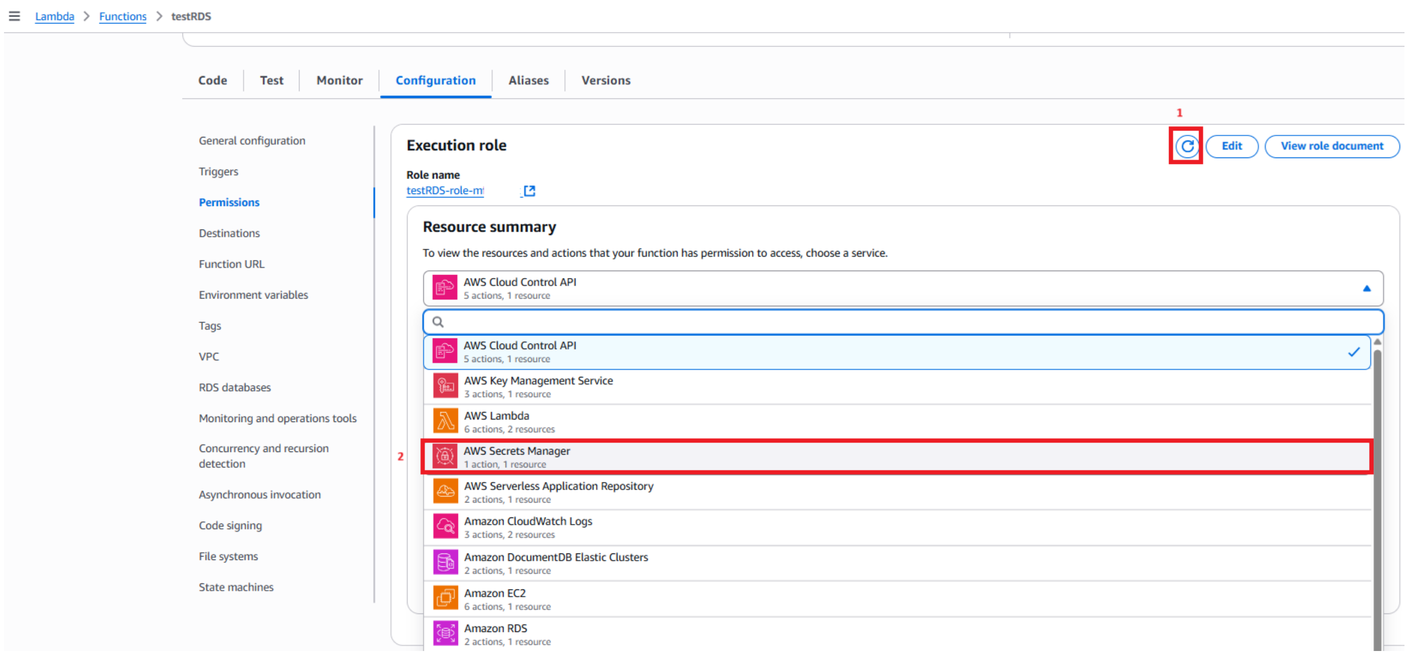
SecretsManagerReadWrite

Provides read/write access to AWS Secrets Manager via the AWS Management Console. Note: this exludes IAM actions, so combine with IAMFullAccess if rotation configuration is required.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BasePermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:*",
                "cloudformation:CreateChangeSet",
                "cloudformation:DescribeChangeSet",
                "cloudformation:DescribeStackResource",
                "cloudformation:DescribeStacks",
                "cloudformation:ExecuteChangeSet",
                "docdb-elastic:GetCluster",
                "docdb-elastic:ListClusters",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "kms:DescribeKey",
                "kms:ListAliases",
```

9. Wait a few moments for your configurations to take effect.

10. Go back to Lambda, and click the **Refresh** icon at the top of the page. Observe all the additional permissions the role has access to. We are interested in AWS Secrets Manager.



Execution role

Role name
testRDS-role-m

Resource summary

To view the resources and actions that your function has permission to access, choose a service.

AWS Cloud Control API
5 actions, 1 resource

AWS Cloud Control API
5 actions, 1 resource

AWS Key Management Service
3 actions, 1 resource

AWS Lambda
6 actions, 2 resources

AWS Secrets Manager
1 action, 1 resource

AWS Serverless Application Repository
2 actions, 1 resource

Amazon CloudWatch Logs
3 actions, 2 resources

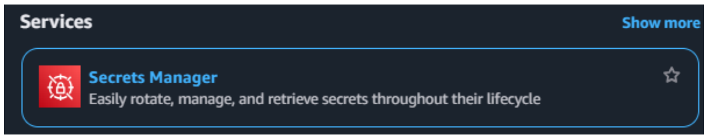
Amazon DocumentDB Elastic Clusters
2 actions, 1 resource

Amazon EC2
6 actions, 1 resource

Amazon RDS
2 actions, 1 resource

5. Create a Secret in Secrets Manager

1. In a new browser tab, navigate to **Secrets Manager**.



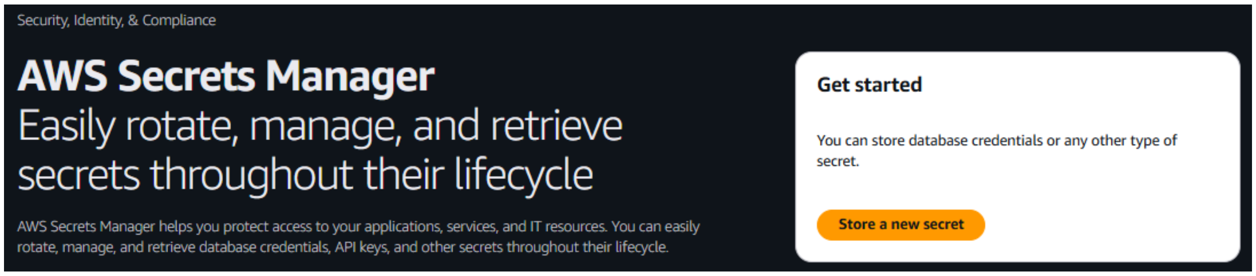
Services

Show more

Secrets Manager

Easily rotate, manage, and retrieve secrets throughout their lifecycle

2. Click **Store a new secret**.



Security, Identity, & Compliance

AWS Secrets Manager

Easily rotate, manage, and retrieve secrets throughout their lifecycle

AWS Secrets Manager helps you protect access to your applications, services, and IT resources. You can easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle.

Get started

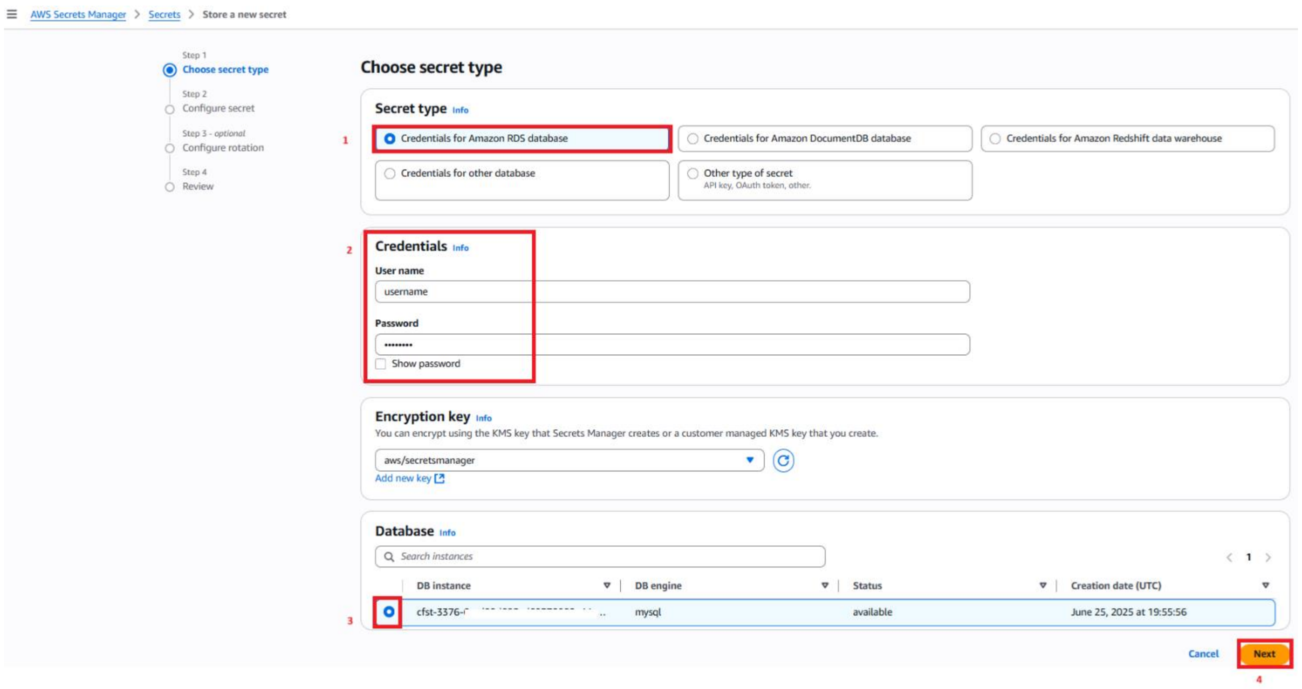
You can store database credentials or any other type of secret.

Store a new secret

3. With **Credentials for Amazon RDS database** selected, set the following values:

   a. ***Username***: Enter username
   b. ***Password***: Enter password
   c. ***Encryption key***: Leave this as the default option.
   d. ***Database***: Select the listed DB instance.

4. Click **Next**.



AWS Secrets Manager > Secrets > Store a new secret

Step 1 Choose secret type
Step 2 Configure secret
Step 3 - optional Configure rotation
Step 4 Review

Choose secret type

Secret type Info

1 [✓ Credentials for Amazon RDS database] Credentials for Amazon DocumentDB database Credentials for Amazon Redshift data warehouse Credentials for other database Other type of secret (API key, OAuth token, other.)

2 Credentials Info

User name: username

Password: ••••••••

Show password

Encryption key Info

You can encrypt using the KMS key that Secrets Manager creates or a customer managed KMS key that you create.

aws/secretsmanager

Add new key

Database Info

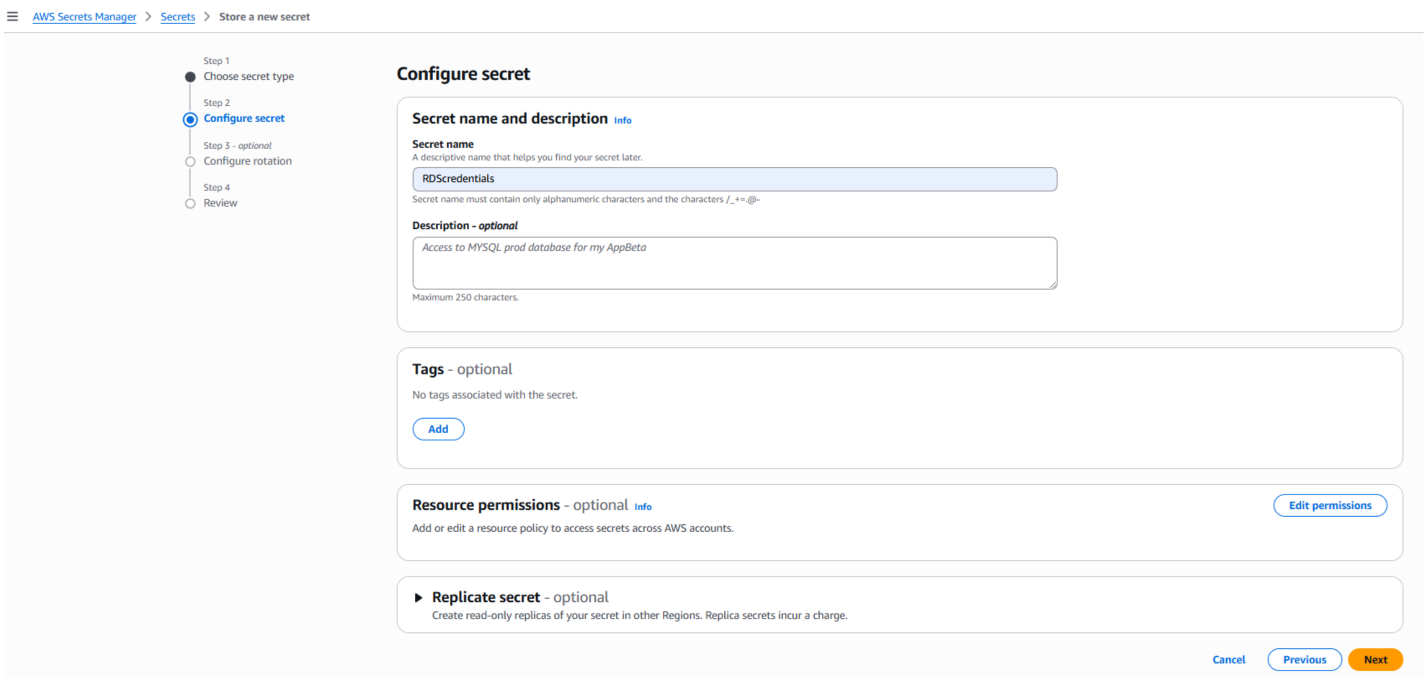
Search instances

	DB instance	DB engine	Status	Creation date (UTC)
3 ●	cfst-3376-[redacted]	mysql	available	June 25, 2025 at 19:55:56

Cancel Next 4

5. On the next page, for **Secret name**, enter **RDScredentials**.

6. Leave the rest of the defaults as they are, and click **Next**.


AWS Secrets Manager > Secrets > Store a new secret

Step 1 Choose secret type
Step 2 Configure secret
Step 3 - *optional* Configure rotation
Step 4 Review

Configure secret

Secret name and description Info

Secret name
A descriptive name that helps you find your secret later.
RDScredentials
Secret name must contain only alphanumeric characters and the characters /_+=.@-

Description - *optional*
Access to MYSQL prod database for my AppBeta
Maximum 250 characters.

Tags - optional
No tags associated with the secret.
Add

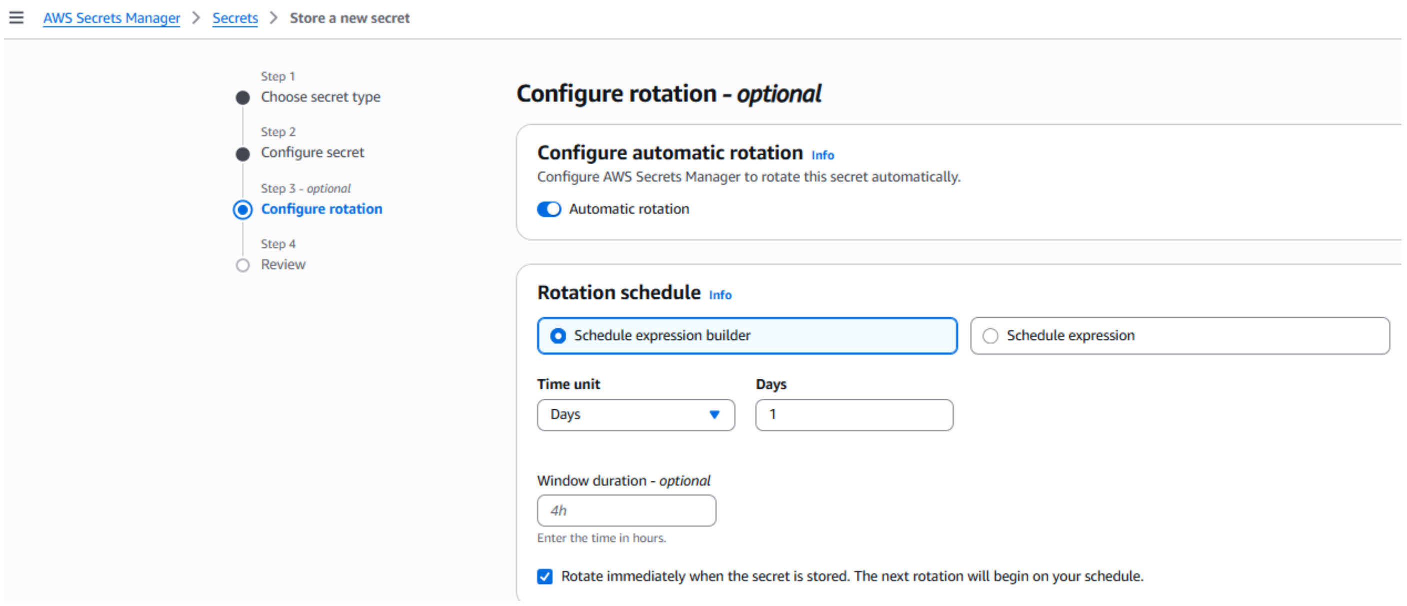
Resource permissions - optional Info
Add or edit a resource policy to access secrets across AWS accounts.
Edit permissions

Replicate secret - optional
Create read-only replicas of your secret in other Regions. Replica secrets incur a charge.

Cancel Previous Next

7. On the next page, set the following values:

   a. Toggle the ***Automatic rotation*** option to enable it.
   b. Leave ***Schedule expression builder*** selected.
   c. ***Time unit***: Select **Days**, and enter **1**. So, for every 1 day, the password is going to be rotated.


AWS Secrets Manager > Secrets > Store a new secret

Step 1 Choose secret type
Step 2 Configure secret
Step 3 - *optional* Configure rotation
Step 4 Review

Configure rotation - *optional*

Configure automatic rotation Info
Configure AWS Secrets Manager to rotate this secret automatically.
Automatic rotation

Rotation schedule Info
[✓ Schedule expression builder] Schedule expression

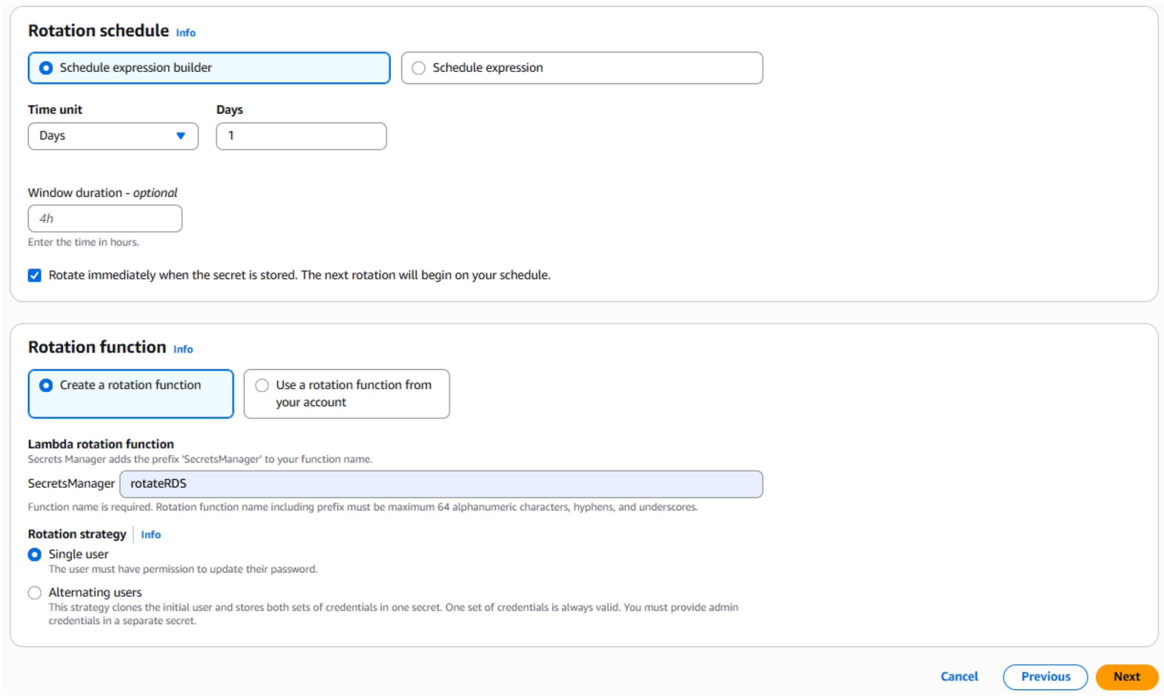
Time unit: Days
Days: 1

Window duration - *optional*
4h
Enter the time in hours.

☑ Rotate immediately when the secret is stored. The next rotation will begin on your schedule.

d. Leave ***Create a rotation*** function selected. So, in order for the password to be rotated, a new Lambda function (an application) is going to be created in the background, which is going to do the pass of the rotation for us.
e. ***SecretsManager***: Enter **rotateRDS**.
f. Under ***Rotation strategy***, leave **Single User** selected.

8. Click **Next**.



Rotation schedule Info

(●) Schedule expression builder ( ) Schedule expression

Time unit: Days
Days: 1

Window duration - *optional*: 4h
Enter the time in hours.

[✓] Rotate immediately when the secret is stored. The next rotation will begin on your schedule.

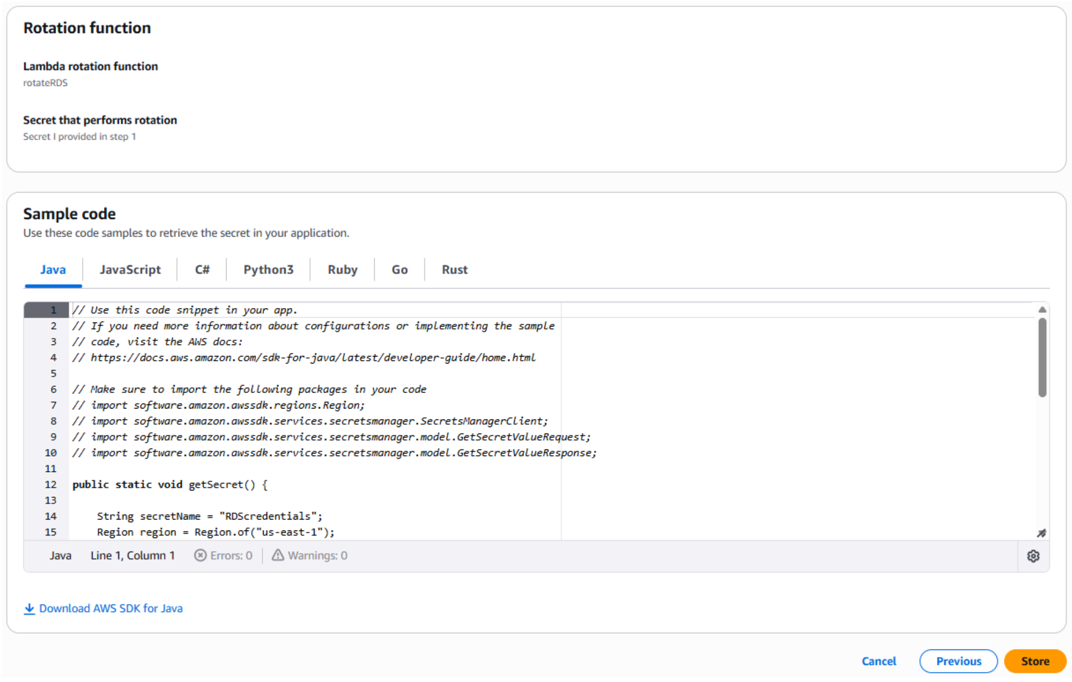
Rotation function Info

(●) Create a rotation function ( ) Use a rotation function from your account

Lambda rotation function
Secrets Manager adds the prefix 'SecretsManager' to your function name.
SecretsManager: rotateRDS
Function name is required. Rotation function name including prefix must be maximum 64 alphanumeric characters, hyphens, and underscores.

Rotation strategy | Info
(●) Single user
The user must have permission to update their password.
( ) Alternating users
This strategy clones the initial user and stores both sets of credentials in one secret. One set of credentials is always valid. You must provide admin credentials in a separate secret.

9. Scroll down and click **Store**.



Rotation function

Lambda rotation function
rotateRDS

Secret that performs rotation
Secret I provided in step 1

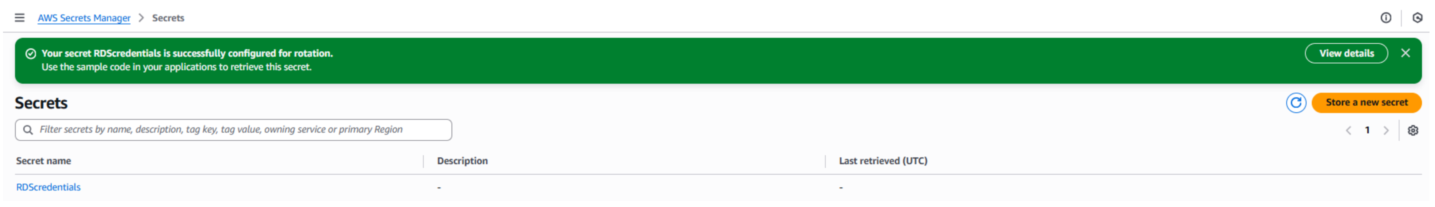
Sample code
Use these code samples to retrieve the secret in your application.

```
// Use this code snippet in your app.
// If you need more information about configurations or implementing the sample
// code, visit the AWS docs:
// https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/home.html

// Make sure to import the following packages in your code
// import software.amazon.awssdk.regions.Region;
// import software.amazon.awssdk.services.secretsmanager.SecretsManagerClient;
// import software.amazon.awssdk.services.secretsmanager.model.GetSecretValueRequest;
// import software.amazon.awssdk.services.secretsmanager.model.GetSecretValueResponse;

public static void getSecret() {

    String secretName = "RDScredentials";
    Region region = Region.of("us-east-1");
```

Java Line 1, Column 1 Errors: 0 Warnings: 0

Note: If it doesn't show, click on the Refresh button.

10. Once it's done, refresh your page and click **RDScredentials**.



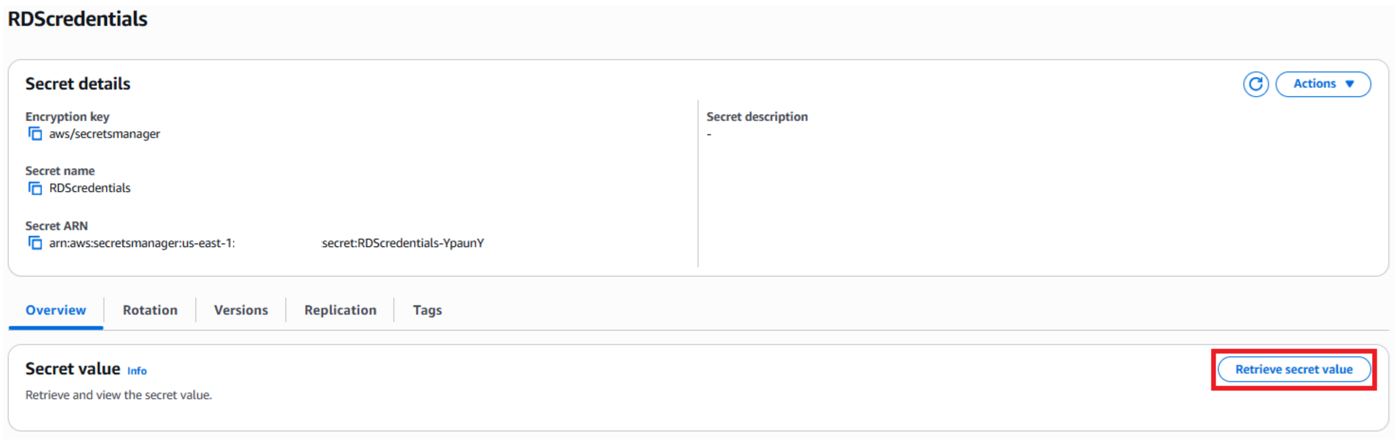
AWS Secrets Manager > Secrets

Your secret RDScredentials is successfully configured for rotation.
Use the sample code in your applications to retrieve this secret.

View details
Secrets
Store a new secret
Filter secrets by name, description, tag key, tag value, owning service or primary Region

Secret name	Description	Last retrieved (UTC)
RDScredentials	-	-

11. In the **Secret value** section, click **Retrieve secret value**. You should see the password listed as ***password***.



RDScredentials

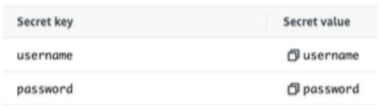
Secret details

Encryption key
aws/secretsmanager

Secret name
RDScredentials

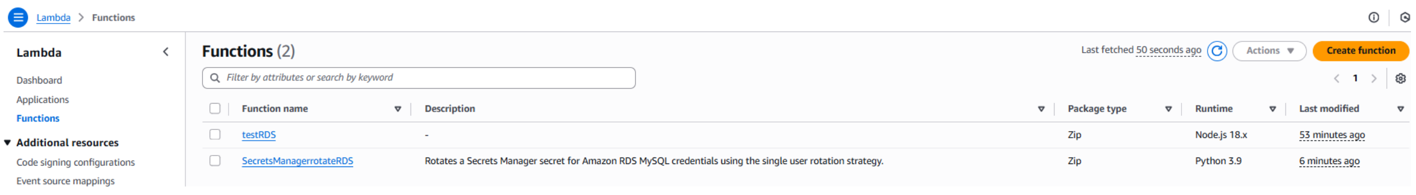
Secret ARN
arn:aws:secretsmanager:us-east-1: secret:RDScredentials-YpaunY

Secret description
-



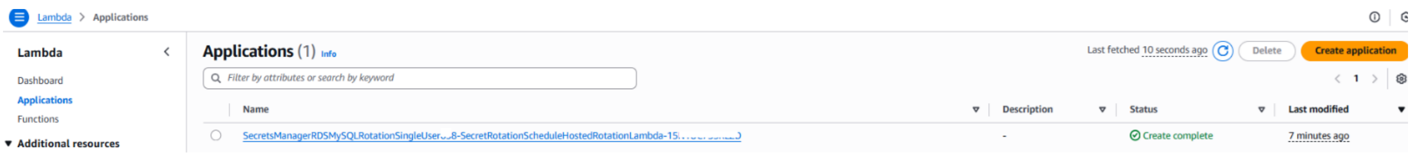
Secret key	Secret value
username	username
password	password

12. Go back to the **Lambda** browser tab, and click the hamburger menu in the upper-left. Then, select **Functions**. You should also see the **SecretsManagerrotateRDS** function.



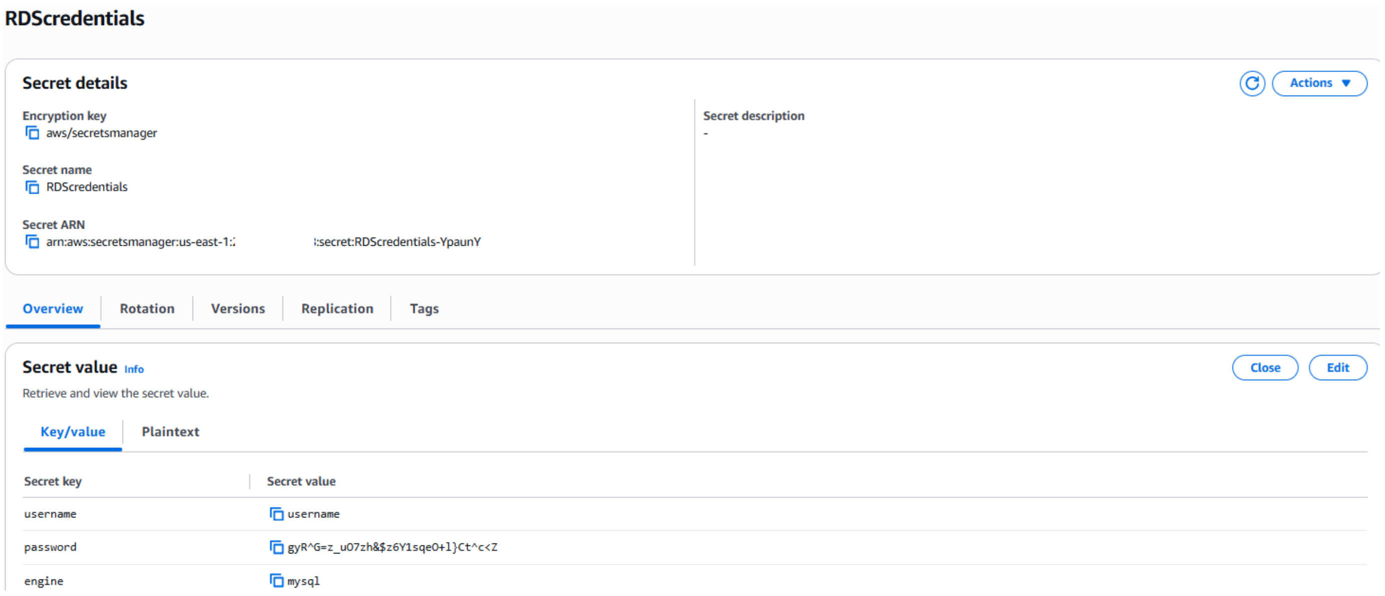
Function name	Description	Package type	Runtime	Last modified
testRDS	-	Zip	Node.js 18.x	53 minutes ago
SecretsManagerrotateRDS	Rotates a Secrets Manager secret for Amazon RDS MySQL credentials using the single user rotation strategy.	Zip	Python 3.9	6 minutes ago

13. To check if the function is running, from the left menu, click **Applications**. The function should show as **Create complete**.



Name	Description	Status	Last modified
SecretsManagerRDSMySQLRotationSingleUser[illegible]8-SecretRotationScheduleHostedRotationLambda-15[illegible]	-	Create complete	7 minutes ago

14. Back in **Secrets Manager** browser tab, click the **Refresh** button, and click **Retrieve secret value** to see the secret again. You will see the ***password*** is now a series of random characters.


RDScredentials

Secret details

Encryption key	aws/secretsmanager
Secret name	RDScredentials
Secret ARN	arn:aws:secretsmanager:us-east-1: :secret:RDScredentials-YpaunY
Secret description	-

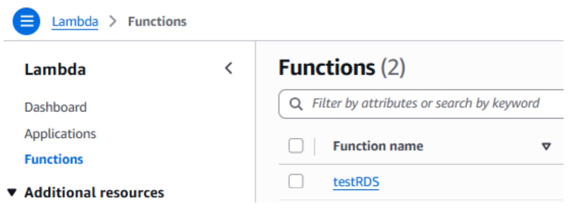
Secret value Info

Retrieve and view the secret value.

Secret key	Secret value
username	username
password	gyR^G=z_uO7zh&$z6Y1sqeO+l}Ct^c<Z
engine	mysql

6. Test Connectivity from Lambda to RDS Using Credentials from AWS Secrets Manager

1. Go back to **Lambda** (so it's able to pull the secret from Secrets Manager and log in to MySQL RDS instance).
2. From the left menu, select **Functions**.
3. Select the **testRDS** function.



4. Click the **Code** tab.
5. Replace the code in ***index.mjs*** with the following:

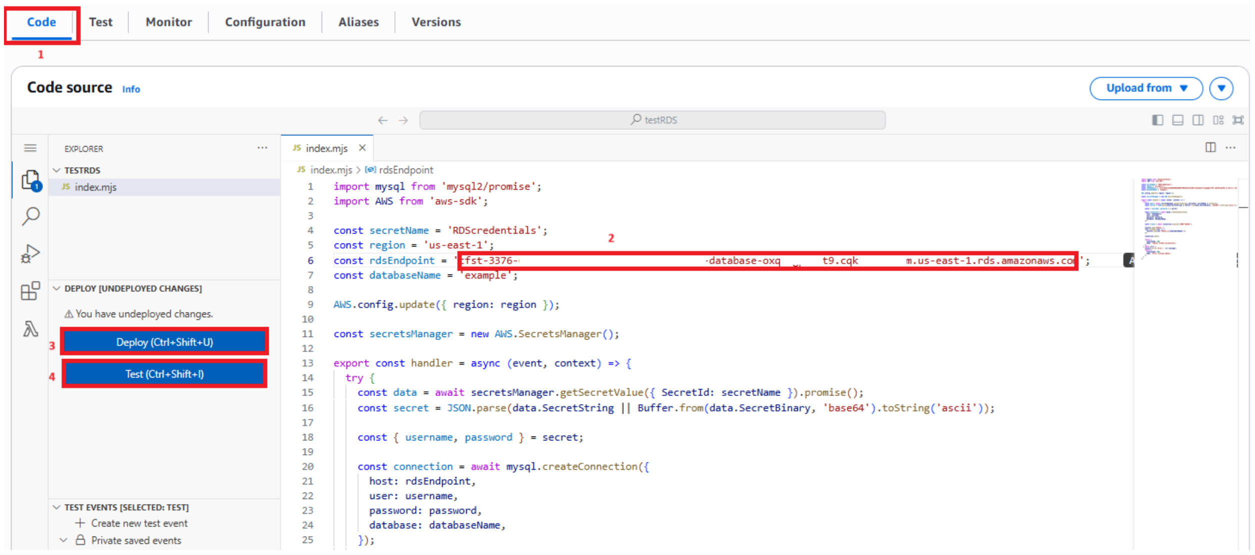
import mysql from 'mysql2/promise';

import AWS from 'aws-sdk';

```
const secretName = 'RDScredentials';
const region = 'us-east-1';
const rdsEndpoint = '<RDS Endpoint>';
const databaseName = 'example';

AWS.config.update({ region: region });

const secretsManager = new AWS.SecretsManager();

export const handler = async (event, context) => {
  try {
    const data = await secretsManager.getSecretValue({ SecretId: secretName }).promise();
    const secret = JSON.parse(data.SecretString || Buffer.from(data.SecretBinary,
'base64').toString('ascii'));

    const { username, password } = secret;

    const connection = await mysql.createConnection({
      host: rdsEndpoint,
      user: username,
      password: password,
      database: databaseName,
    });

    const [rows] = await connection.execute('SHOW TABLES');

    console.log('Tables:');
    rows.forEach((row) => {
      console.log(row[`Tables_in_${databaseName}`]);
    });
```

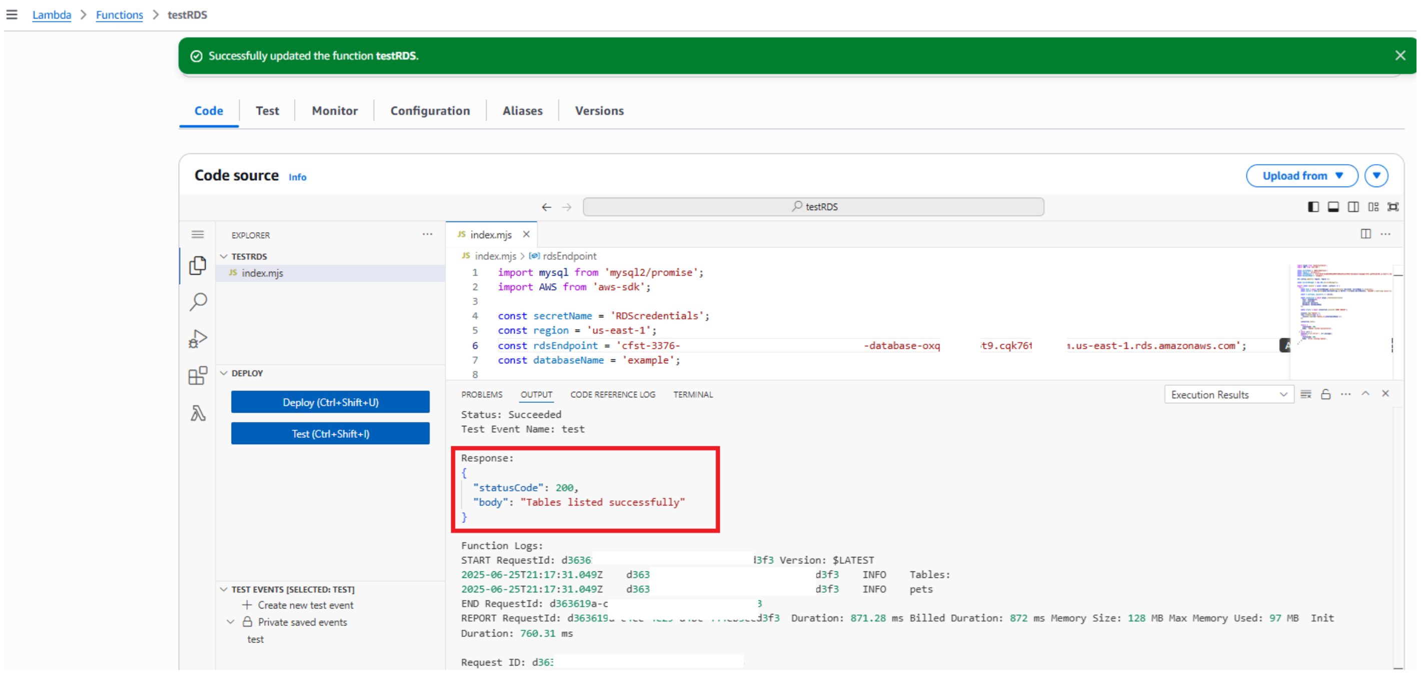
```
      connection.end();

      return {
        statusCode: 200,
        body: 'Tables listed successfully',
      };
    } catch (err) {
      console.error('Error:', err.message);
      return {
        statusCode: 500,
        body: 'Error listing tables',
      };
    }
  };
```

6. Replace the **<RDS Endpoint>** placeholder with the endpoint you copied earlier in the lab. This is going to retrieve RDS credential secret name from Secrets Manager. It will open a connection to the MySQL RDS database using the host as the RDS endpoint and the user and password taken from the information of the RDS secret.
7. Click **Deploy**.
8. Once the function is updated, click the **Test** tab.



```
import mysql from 'mysql2/promise';
import AWS from 'aws-sdk';

const secretName = 'RDScredentials';
const region = 'us-east-1';
const rdsEndpoint = 'cfst-3376-            -database-oxq      t9.cqk       m.us-east-1.rds.amazonaws.com';
const databaseName = 'example';

AWS.config.update({ region: region });

const secretsManager = new AWS.SecretsManager();

export const handler = async (event, context) => {
  try {
    const data = await secretsManager.getSecretValue({ SecretId: secretName }).promise();
    const secret = JSON.parse(data.SecretString || Buffer.from(data.SecretBinary, 'base64').toString('ascii'));

    const { username, password } = secret;

    const connection = await mysql.createConnection({
      host: rdsEndpoint,
      user: username,
      password: password,
      database: databaseName,
    });
```

9. Once you have the green banner at the top, click **Test**. Expand Details, and note the response includes a **statusCode** of **200** and the tables are listed successfully.



```
import mysql from 'mysql2/promise';
import AWS from 'aws-sdk';

const secretName = 'RDScredentials';
const region = 'us-east-1';
const rdsEndpoint = 'cfst-3376-            -database-oxq      t9.cqk761      .us-east-1.rds.amazonaws.com';
const databaseName = 'example';
```

```
Status: Succeeded
Test Event Name: test

Response:
{
  "statusCode": 200,
  "body": "Tables listed successfully"
}

Function Logs:
START RequestId: d3636                    3f3 Version: $LATEST
2025-06-25T21:17:31.049Z    d363                    d3f3    INFO    Tables:
2025-06-25T21:17:31.049Z    d363                    d3f3    INFO    pets
END RequestId: d363619a-c                    3
REPORT RequestId: d363619          d3f3  Duration: 871.28 ms Billed Duration: 872 ms Memory Size: 128 MB Max Memory Used: 97 MB  Init Duration: 760.31 ms

Request ID: d363
```