Using Secrets Manager to Authenticate with an RDS Database Using Lambda

Using Secrets Manager to Authenticate with an RDS Database Using Lambda

Introduction

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.

Solution

Log in to the live *AWS environment* using the credentials provided. Use an *incognito or private* browser window to ensure you're using the lab account rather than your own.

Make sure you're in the N. Virginia (`us-east-1`) region throughout the lab.

Download the MySQL Library zip file you'll need for the first lab objective.

Create Lambda Function

1. Navigate to **Lambda**.
2. Click **Create function**.
3. Make sure the **Author from scratch** option at the top is selected, and then use the following settings:
   - *Function name:* **testRDS**
   - *Runtime:* **Node.js 14.x**
4. Expand **Advanced settings**, click **Enable Network**, and set the following values:
   - *VPC:* Select the lab-provided VPC
   - *Subnets:* Select the 2 subnets that have `Public` in their name/ID
   - *Security groups:* Select the lab-provided **Database-Security-Group** security group (not the default security group)
5. Click **Create function**.
   - It may take up to 5 to 10 minutes to finish creating. The blue bar at the top of the page will indicate that the function is being created.
6. Once the function has been created, navigate to the Configuration tab and edit the General configuration
7. Change the timeout setting to 6 seconds and click **Save**
8. In the left-hand menu, click **Layers**.
9. Click **Create layer**.
10. Set the following values:
    - *Name:* **mysql**
    - *Description:* **mysql library**
    - *Upload a .zip file:* Select
      1. Click **Upload**.
      2. Upload the MySQL Library zip file you downloaded earlier.
    - *Compatible runtimes:* **Node.js 14.x**
11. Click **Create.**
12. Navigate back to your **testRDS** function.
13. Once on the testRDS page, scroll down to **Layers.**
14. Click **Add a layer** in the Layers section.
15. Select **Custom layers**, and set the following values:
    - *Custom layers:* **mysql**
    - *Version:* **1**
16. Click **Add**. Wait a minute for the function to update.

Copy Code into Lambda Function

1. In the *Code* source section, expand **testRDS > index.js**.
2. Select the existing code in the *index.js* tab and replace it with the following:

```
var mysql = require('mysql');

exports.handler = (event, context, callback) => {

    var connection = mysql.createConnection({
      host: '<RDS Endpoint>',
      user: 'username',
      password: 'password',
      database: 'example',
    });

    connection.query('show tables', function (error, results, fields) {
      if (error) {
          connection.destroy();
          throw error;
      } else {
          // connected!
          console.log('Query result:');
          console.log(results);
          callback(error, results);
          connection.end(function (err) { callback(err, results);});
      }
    });
};
```

3. In a new browser tab, navigate to **RDS > DB Instances.**
4. Click the listed database.
5. Copy the endpoint (in the *Connectivity & security* section).
6. Back in the Lambda function code, replace `<RDS Endpoint>` on line 4 with the endpoint you just copied.
7. Click **Deploy.**
8. Click **Test**.
9. In the *Configure test event* dialog, enter an *Event name* of "test".
10. Click **Create**.
11. Click **Test** again.
    - The *Response* should be 2 square brackets, which is correct since we don't have any tables defined in this database.
12. Click the *index.js* tab.
13. Replace line 11 with the following:

```
connection.query('CREATE TABLE pet (name VARCHAR(20), species VARCHAR(20))',function (error, results, fields) {
```

14. Click **Deploy.**
15. Click **Test**.
    - This time, the *Response* should have information within the curly brackets.
16. Click the *index.js* tab.
17. Undo the code change (**Ctrl+Z** or **Cmd+Z**) to get it back to the original code we pasted in.
18. Click **Deploy**.
19. Click **Test**.
    - This time, we should see the `pet` table listed in the *Response*.

Create a Secret in Secrets Manager

1. In a new browser tab, navigate to **Secrets Manager**.
2. Click **Store a new secret**.
3. With **Credentials for RDS database** selected, set the following values:
   - *User name:* **username**
   - *Password:* **password**
   - *Select the encryption key:* **DefaultEncryptionKey**
   - *Select which RDS database this secret will access:* Select the listed DB instance
4. Click **Next.**
5. On the next page, give it a *Secret name* of "RDScredentials".
6. Leave the rest of the defaults, and click **Next**.
7. On the next page, set the following values:
   - *Enable automatic rotation:* Check
   - *Select rotation interval:* **Custom**, **1**
   - *Create a new Lambda function to perform rotation:* Select
   - *Secrets Manager rotation:* **rotateRDS**
   - *Select which secret will be used to perform the rotation:* **Use this secret**
8. Click **Next**.
9. In the *Sample code* section, click **JavaScript**.
10. Click **Store**.
    - It may take up to 5 to 10 minutes to finish the configuration.
11. Once it's done, click **RDScredentials**.
12. In the *Secret value* section, click **Retrieve secret value**. You should see the password is now a long string rather than "password."

    Note: *If yours still says "password," give it a few minutes and refresh the page again before proceeding to avoid errors ahead. Your Lambda function may still be in the process of getting set up.*
13. Back in the Lambda function, click **Test**.
    - You will see errors saying access denied because the password has changed.
14. Click the *index.js* tab.
15. Remove all of the code except the `host` line that has your RDS endpoint.
16. Beneath that line, enter the following:

```
var mysql = require('mysql');

var AWS = require('aws-sdk'),
    region = "us-east-1",
    secretName = "RDScredentials",
    secret,
    decodedBinarySecret;

var client = new AWS.SecretsManager({
    region: "us-east-1"
});

exports.handler = (event, context, callback) => {

    client.getSecretValue({SecretId: secretName}, function(err, data) {
        if (err) {
            console.log(err);
        }
        else {
            // Decrypts secret using the associated KMS CMK
            // Depending on whether the secret is a string or binary, one of these fields will be populated.
            if ('SecretString' in data) {
                secret = data.SecretString;
            } else {
                let buff = new Buffer(data.SecretBinary, 'base64');
                decodedBinarySecret = buff.toString('ascii');
            }
        }

        var parse = JSON.parse(secret);
        var password = parse.password;

        var connection = mysql.createConnection({
          host: '<RDS Endpoint>',
          user: 'username',
          password: password,
          database: 'example',
        });

        connection.query('show tables', function (error, results, fields) {
          if (error) {
              connection.destroy();
              throw error;
          } else {
              // connected!
              console.log('Query result:');
              console.log(results);
              callback(error, results);
              connection.end(function (err) { callback(err, results);});
          }
        });

    });
```

17. Copy your RDS endpoint value in the line you left at the top, and paste it to replace the `<RDS Endpoint>` value in the new code.
18. Delete the `host` line at the top that you left from the old code, and make sure the new code begins on line 1.
19. Click **Deploy**.

Modify Permissions

1. In a new browser tab, navigate to **VPC > Endpoints**.
2. Click **Create Endpoint**.
3. In the search filter box, enter "secret".
4. Select the only result.
5. In the *VPC* dropdown, select the listed VPC.
6. For *Subnets*, ensure the 2 public subnets are selected.
7. For *Security group*, un-select the default and select the **database-group** security group.
8. Leave the other defaults, and click **Create endpoint**.
9. Navigate to **EC2 > Security Groups**.
10. Select the checkbox next to **DatabaseSG**.
11. Click the **Inbound rules** tab.
12. Click **Edit inbound rules**.
13. Click **Add rule**, and set the following values:
    - *Type:* **HTTPS**
    - *Source:* Scroll down in the dropdown and select **DatabaseSG**
14. Click **Save rules.**
15. Navigate to **IAM > Roles**.
16. In the search filter box, enter "test".
17. Click the only result.
18. Click **Attach policies**.
19. In the search filter box, enter "secret".
20. Select **SecretsManagerReadWrite**.
21. Click **Attach policy**.

Test New Code

1. Back on the Lambda function page, click **Test**.
   - This time, we should see our table name again, which means we were successful.

Conclusion

Congratulations on successfully completing this hands-on lab!

Tools

Lab Diagram Instant Terminal

Credentials

How do I connect?

Additional Resources

Log in to the live *AWS environment* using the credentials provided. Use an *incognito or private* browser window to ensure you're using the lab account rather than your own.

Make sure you're in the N. Virginia (`us-east-1`) region throughout the lab.

MySQL Library Zip file

Learning Objectives

0 of 5 completed

Optional: Run progress checks to confirm you've completed the objectives

- Create a lambda function
- Create the mysql layer, and copy your code to the lambda function
- Create a secret in secrets manager
- Modify permissions
- Test new code