**Using Secrets Manager to Authenticate with an RDS Database Using Lambda:**

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.

**RDS:** Copy the endpoint. The db was already created.

* RDS – Databases – select the only db there – Connectivity & Security –
* Enpoint, copy it and keep it in a notepad.

**Lambda**: Create the lambda function.

* Lambda – Create function – Author from scratch – Function name: testRDS –
* Runtime: Node.js 18.x – Advanced settings – tick Enable VPC –
* Select provided VPC and both subnets –
* Security groups: select the db one – Create function – will take 5 mins or so.
* Configuration tab – Edit – Timeout: 6 sec – Save.
* AWS Lambda – Additional resources – Layers – Create layer –
* Name: mysql (make sure is lowercase) –
* download the mysql.zip file from the exercise and upload it –
* Compatible runtimes - Node.js 18.x - Create
* AWS Lambda – Functions – testRDS – scroll down to Layers –
* Add a layer – Custom layers – Custom layers: mysql –
* Version: select the one you have there – Add
* **Add the code**: AWS Lambda – Functions – testRDS – Code source –
* Index.mjs – remove the default code there and paste the one below –
* Host: “<RDS endpint>” 🡪 Update the endpoint copied in the first step, you have it in the notepad
* This code will create a new mysql connection to our database endpoint, will create then a table.
* Deploy – wait till successfully updates the function.

* **Check the code:** Hit the Test button – Create new event – Event name: test – Save
* Once the test event is created hit again Test. Output code:

Test Event Name

test

Response

{

"statusCode": 200,

"body": "Tables listed successfully"

}

Function Logs

START RequestId: b94e43ea-1e14-41bc-996f-2a433adbbce3 Version: $LATEST

2024-03-10T13:04:59.802Z b94e43ea-1e14-41bc-996f-2a433adbbce3 INFO Table created: pets

2024-03-10T13:04:59.835Z b94e43ea-1e14-41bc-996f-2a433adbbce3 INFO Tables:

2024-03-10T13:04:59.835Z b94e43ea-1e14-41bc-996f-2a433adbbce3 INFO pets

END RequestId: b94e43ea-1e14-41bc-996f-2a433adbbce3

REPORT RequestId: b94e43ea-1e14-41bc-996f-2a433adbbce3 Duration: 366.69 ms Billed Duration: 367 ms Memory Size: 128 MB Max Memory Used: 80 MB Init Duration: 231.72 ms

Request ID

b94e43ea-1e14-41bc-996f-2a433adbbce3

* It successfully worked.

**Lambda:** Modify Lambda execution role to include Secrets Manager IAM policy. Once configured we will create the Secrets manager secret so we can call the lambda code.

* AWS Lambda – Functions – testRDS – Configuration – Permissions –
* Role name: testRDS-role-vksdfs3, click on it – you will be redirected to:
* IAM - Roles - testRDS-role- vksdfs3 – Add permissions – Attach policies –
* Search for: SecretsManagerReadWrite – tick and Add permissions (careful if you do this in prod as you are opening too much things, so modify as necessary to make it safer)
* Go back to Lambda window and hit refresh to see the new policy added –
* You will see there are a lot more permissions.

**Secrets Manager:**

* AWS Secrets Manager – Secrets – Store a new secret – Secret type – Credentials for Amazon RDS database –
* Credentials – User name: username – Password: password – leave encryption key
* Database – select the one provided – Next – Secret name: RDScredentials – Next
* Configure automatic rotation – Automatic rotation – change rotation schedule to days and put 1 day, so the pass is gonna be rotated once a day.

**CODE TO PASTE:** Create Table in the RDS Database Using Lambda to Check Connectivity

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',

});

}

};