Naveed Ahmad

Sep 3, 2021

·

6 min read

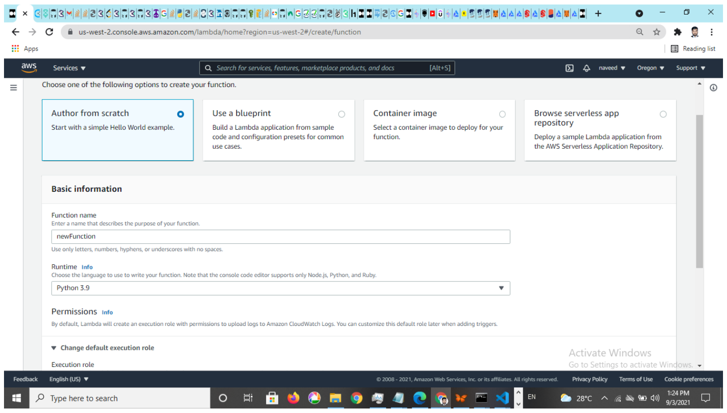
·

Listen

Writing a REST API on AWS using Lambda Functions and RDS (MySQL)

Creating a new Lambda Function on AWS

Login to aws.amazon.com and from search select Lambda Functions. Click on the button Create Function. Select the option to write a function from Scratch. Write newFunction as the name of the function and choose python 3.9 as the runtime. For the execution role, select “Create a new role with basic Lambda permissions”. Click on the “Create Function” button to create a new function.



Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

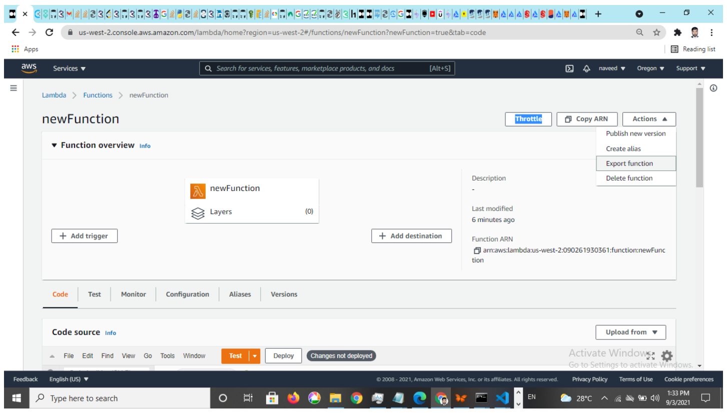
Basic information

Function name
Enter a name that describes the purpose of your function.
newFunction
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

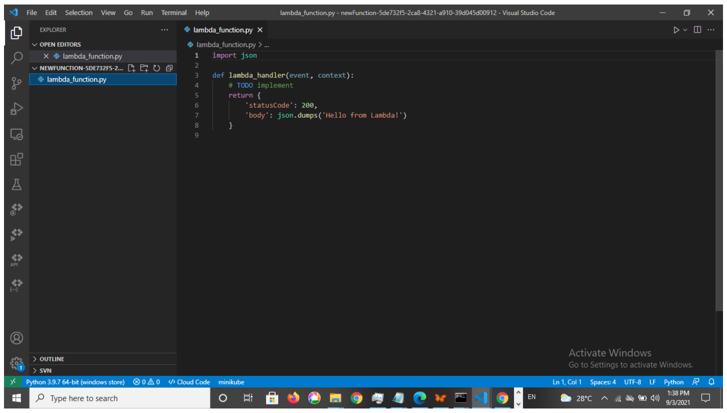
Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

To access the MySQL from the AWS RDS, PyMSQL and Cryptography libraries need to be installed. To add these libraries to your lambda functions, you have to first export the lambda function to your local machine. From the Actions, dropdown, select the export function option to download the Lambda Function.



From the Export your Function

dialog, click the button “Download Deployment Package”. This would download the zip file containing your Lambda Function to your local drive. Unzip the folder and open it in VSCode. Click on the lamda_function.py file, as we have to add some code in it.


lambda_function.py - newFunction-5de732f5-2ca8-4321-a910-39d045d00912 - Visual Studio Code
EXPLORER
OPEN EDITORS
lambda_function.py
NEWFUNCTION-5DE732F5-2...
lambda_function.py

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

Add the following code above the function lambda_handler.

```
import json
 #!/usr/bin/python
 import sys
 import logging
 import rds_config
 import pymysql


port = 3306
```

```
logger = logging.getLogger()
 logger.setLevel(logging.INFO)

try:

conn =
pymysql.connect(host='database-
3.ctjq5ulwjvv8.us-west-
2.rds.amazonaws.com',user='root
',password =
"rootuser",db='tree_database')

except:

logger.error("ERROR: Unexpected
error: Could not connect to
MySql instance.")

sys.exit()
 logger.info("SUCCESS:
Connection to RDS mysql
instance succeeded")
```

The above code would make a pymysql connection available to your code.

You can see that above code uses a database name and a user name and password to access this database. You would first have to create this database, using the AWS RDS service.

Go to the AWS console, and open RDS service. Select Databases from the left menu and click, Create Database. Choose i. Standard Create ii. MySQL and iii. Free Tier from templates. Choose a user name and password for the database. Click Create Database. Your database would show up in the list of databases, after some time. Once the database status becomes available, you have to copy the database end-point from the Connectivity and Security tab, and use that for host value in your pymysql.connect function. Also replace the database name, user name and password with the database name, user name and password, you have selected for your database.

Now add the following code to your lambda_handler function.

```
sql_query = “select * from
customers”
 lock_table_sql = “LOCK TABLES
```

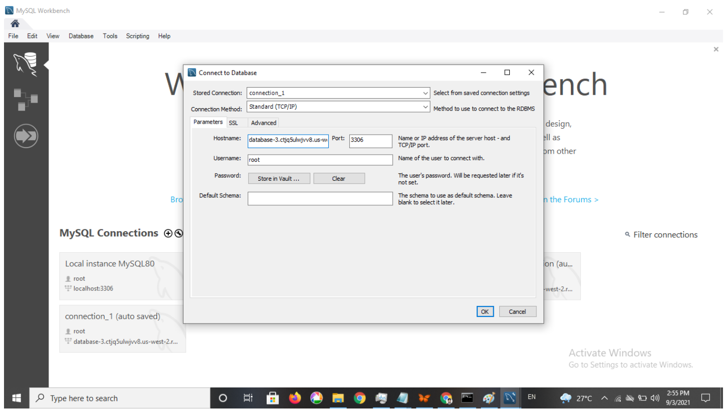
```
customers READ;"
 unlock_table_sql = "UNLOCK
TABLES;"

cur = conn.cursor()

cur.execute(lock_table_sql)
 cur.execute(sql_query)
 output = cur.fetchall()
 cur.execute(unlock_table_sql)
 print(output)

return {
 'result':json.dumps(output),
 'statusCode': 200,
 'body': json.dumps('Hello from
Lambda!')
 }
```

The above code uses the customers table. You have to create the table named customers, which is being accessed in the code. For table creation a SQL command needs to be run. And for that you would have to install MySQL WorkBench client on your machine.



Connect to Database

Stored Connection: connection_1 — Select from saved connection settings

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Parameters | SSL | Advanced

Hostname: database-3.ctjq5ulwjvv8.us-w Port: 3306 — Name or IP address of the server host - and TCP/IP port.

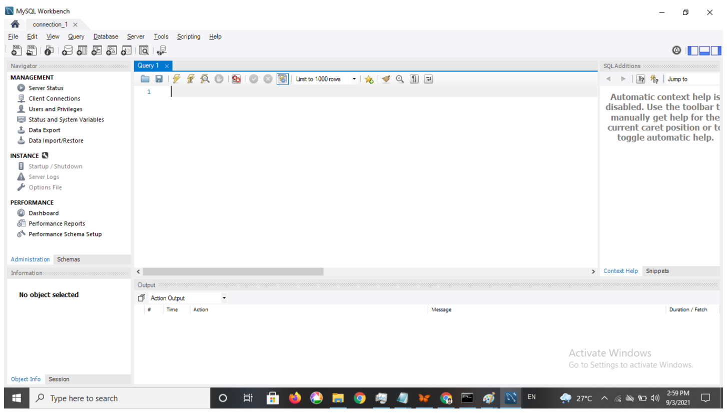
Username: root — Name of the user to connect with.

Password: Store in Vault ... Clear — The user's password. Will be requested later if it's not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

OK Cancel

After installing MySQL workbench, from the database drop-down select the option to Connect to Database. Enter the hostname of the database that you created and user name and password in the dialog box. And click the ok button to connect to the MySQL RDS instance. Once you are connected, you would see the following screen.



Now you enter the SQL command to

Get started Sign In

Search

Naveed Ahmad

2 Followers

create the customers table.

```
use tree_database;
 create table
customers_1(customer_id INT,
customer_name varchar(25));
```

You also have to provide your Lambda Function execution role access rights to call the RDS service to be able to execute the database queries. To do this, click on the Lambda Function and open permissions from the Configuration tab. In the permission, click Attach Policies and select Amazon RDS Full access check box and click Attach Policy button.

Now lets move to completing the Lambda function code in the file you downloaded.

You still have to install the PyMySQL and cryptography libraries for your Lambda function to be able to run. Open the root directory of your

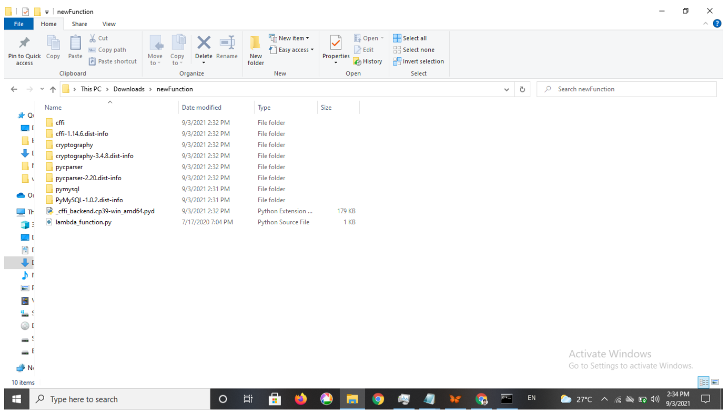
Lambda function and enter the following two commands.

```
pip install — target
C:\Users\LENOVO\Downloads\newFu
nction PyMySQL — no—user
```

Above command would install pymysql in the same directory which contains your lambda_function.py file. Then run the following command

```
pip install — target
C:\Users\LENOVO\Downloads\newFu
nction cryptography — no—user
```

Your directory would now look something like this



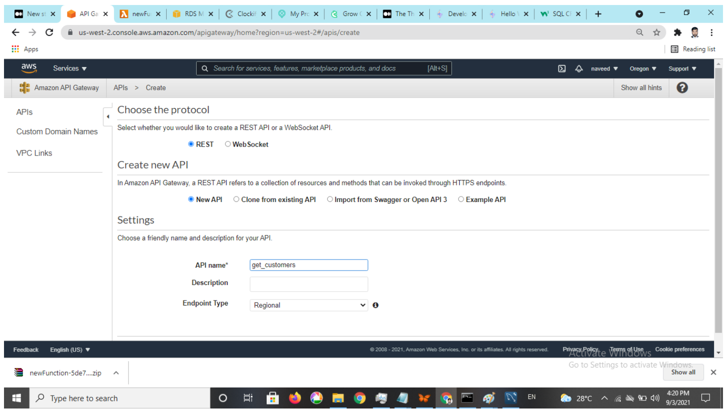
Name	Date modified	Type	Size
cffi	9/3/2021 2:32 PM	File folder	
cffi-1.14.6.dist-info	9/3/2021 2:32 PM	File folder	
cryptography	9/3/2021 2:32 PM	File folder	
cryptography-3.4.8.dist-info	9/3/2021 2:32 PM	File folder	
pycparser	9/3/2021 2:32 PM	File folder	
pycparser-2.20.dist-info	9/3/2021 2:32 PM	File folder	
pymysql	9/3/2021 2:31 PM	File folder	
PyMySQL-1.0.2.dist-info	9/3/2021 2:31 PM	File folder	
_cffi_backend.cp39-win_amd64.pyd	9/3/2021 2:32 PM	Python Extension ...	179 KB
lambda_function.py	7/17/2020 7:04 PM	Python Source File	1 KB

Now you can upload the Lamdba function to AWS. First, zip the folder contain ting the Lambda Function. Then, enter the following command

```
aws lambda update-function-code
— function-name newFunction —
zip-file
fileb://newFunction.zip
```

This would upload your code to the AWS Lambda Service. You can open the Lambda function code to verify that the code was updated. Click on the Test function to verify that the function is working as expected.

But before you could use this function as an API, you have to deploy this function on the API Gateway. From the AWS services, choose API Gateway. Click on the button Create API on the API Gateway dashboard. From the REST API card select Build. Enter the API name get_customers, and click the Create API button.



us-west-2.console.aws.amazon.com/apigateway/home?region=us-west-2#/apis/create

Amazon API Gateway APIs > Create

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

[✓ REST] WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

[✓ New API] Clone from existing API Import from Swagger or Open API 3 Example API

Settings

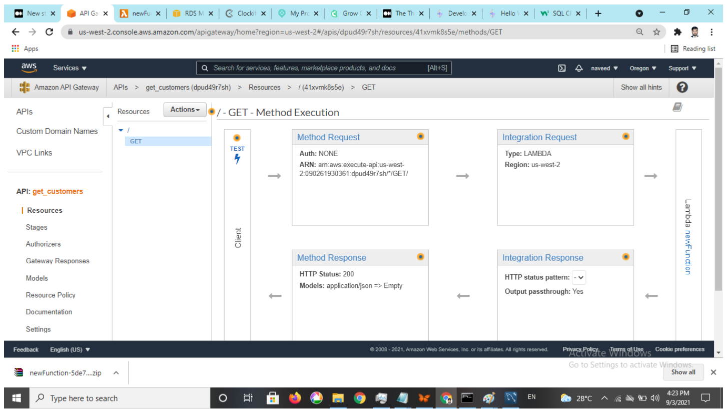
Choose a friendly name and description for your API.

API name* get_customers

Description

Endpoint Type Regional

From the actions drop-down menu, select create method and choose Get and click the tick mark. This would open up the Get setup method. From the Lambda Function drop down select the name of the Lambda function, which would be newFunction and click the Save button. This would open the following screen.


us-west-2.console.aws.amazon.com/apigateway/home?region=us-west-2#/apis/dpud49r7sh/resources/41xvmk8s5e/methods/GET
Amazon API Gateway > APIs > get_customers (dpud49r7sh) > Resources > / (41xvmk8s5e) > GET
APIs
Custom Domain Names
VPC Links
API: get_customers
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Settings
Actions
TEST
Client
Lambda newFunction

/ - GET - Method Execution

Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-west-2:090261930361:dpud49r7sh/*/GET/

Integration Request
Type: LAMBDA
Region: us-west-2

Method Response
HTTP Status: 200
Models: application/json => Empty

Integration Response
HTTP status pattern: -
Output passthrough: Yes

If you are passing parameters through the Get and Post http request methods, you will have to define mapping of these parameters. From the above page, click on the method request card and click on URL query string parameters. Enter page_number in the name input field and click the right arrow. Now move to the previous page and click on the Integration Request card. Open the mapping request section, click on add mapping template. In the content-type write application/json and click the right arrow button. In the editor below, enter the following code and click the save button.

```
{
   “page_number”:
  “$input.params(‘page_number’)”
   }
```

From the Actions drop-down select Enable CORS, and click on the button Deploy API. From the deployment stage, select [New Stage]. Enter Dev, as the stage name and Click the Deploy Button.

Whoa!

You have deployed your REST API. You would find an invoke URL at the top the page (stage editor). Use Advanced REST client (ARC) or any other REST client to verify if the URL is working or not.

In the next few episodes we would explore how to use different databases (Aurora, Dynamo and others), different languages (Go, Ruby, .Net, Java and Node) for writing Lambda Functions. We would also explore HTTP APIs on AWS. So stay tuned!

1