Three-Tier Architecture Setup

Version History

#	Date	Version	Author	Reviewer
1	10/8/2023	1.0.0	P. Uma Poornima	

Overview

The Three-Tier Architecture POC serves as a demonstration of how to architect and build a scalable and maintainable web application. It showcases the separation of concerns by dividing the application into three distinct layers: Presentation Layer, Application Layer, and Data Storage Layer.

Three-Tier Architecture POC demonstrates the clear separation of responsibilities between layers, showcasing how user interactions flow through the frontend, backend, and database to deliver a functional and user-friendly application.

Connecting the Database to the Backend (Application Layer):

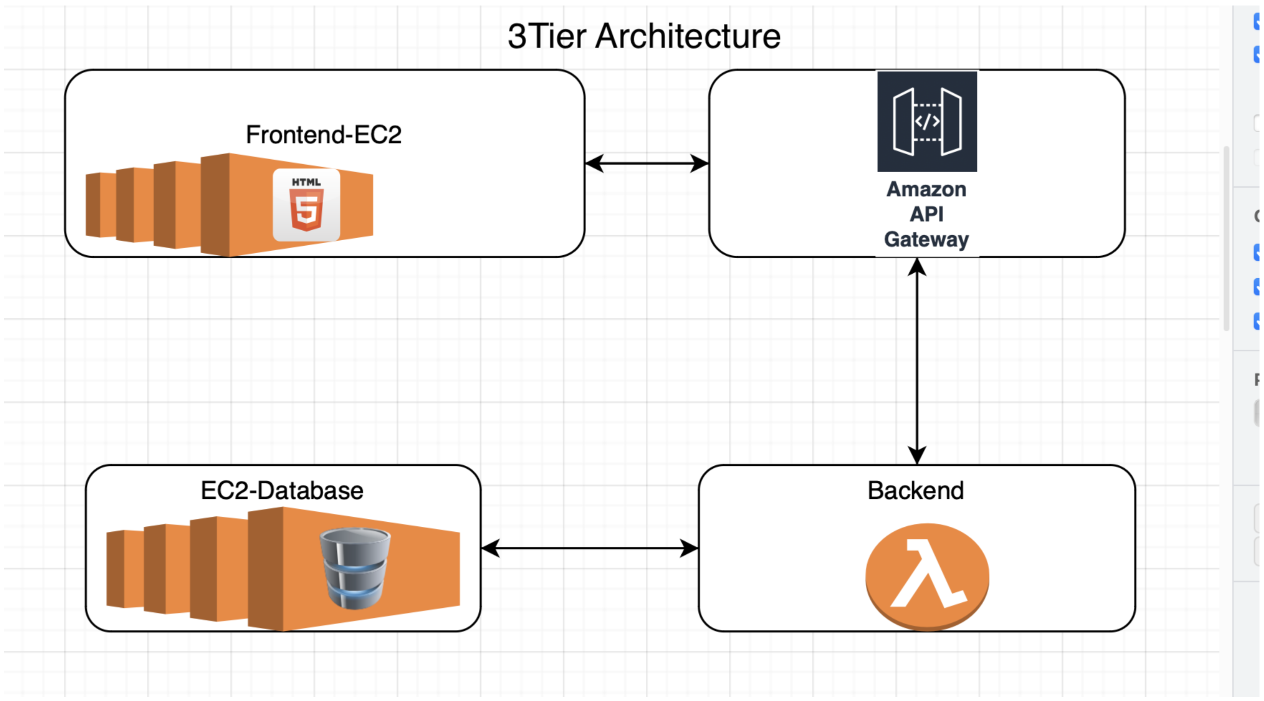
- AWS Lambda functions in the Application Layer are responsible for connecting to the MySQL database hosted on an Amazon EC2 instance.
- For retrieving employee data, a Lambda function processes an HTTP request, fetches data from the MySQL database, and sends it back to the frontend for display.
- For adding new employees, a Lambda function is triggered by the frontend, processes the input data, and inserts it into the MySQL database.

- Similar Lambda functions can be used for handling employee record deletion.
- Lambda functions use database connection libraries (e.g., MySQL libraries) to establish connections, execute queries, and handle data transactions.

Connecting Backend to Frontend (Presentation Layer):

- API Gateway is used to create APIs that expose endpoints for communication between the Presentation Layer and the Application Layer.
- API Gateway routes incoming HTTP requests from the frontend to the appropriate AWS Lambda functions in the Application Layer.
- Each action, such as retrieving, adding, or deleting employees, corresponds to a specific API endpoint.
- API Gateway also handles authentication, authorization, and request validation.
- The Presentation Layer sends HTTP requests (e.g., GET, POST) to the API Gateway endpoints to interact with the backend.

Interaction and Flow:



- User interacts with the Presentation Layer (web-based frontend).
- Presentation Layer sends an HTTP request to the appropriate API Gateway endpoint.
- API Gateway routes the request to the relevant Lambda function in the Application Layer.
- Lambda function processes the request, interacts with the MySQL database in the Data Storage Layer if needed, and generates a response.
- Response is sent back through the API Gateway to the Presentation Layer.
- Presentation Layer displays the retrieved or manipulated data to the user.

- This interaction ensures that data flows seamlessly between the frontend, backend, and database, providing a smooth user experience while maintaining data integrity and separation of concerns.

Pre-requisites

- EC2 Instance with Amazon Linux (open port no **3306**)
- Python 3.9
- Configure MYSQL Installation
- IAM Role & Policy
- Lambda Function
- API Gateway

Procede

Step 1: Set up the Database (MySQL on EC2 Instance)

- Launch one Ec2 instance with Amazon Linux and Configure the security group to allow inbound traffic on port **3306** (MySQL's default port) and HTTP Port **80**.
- ssh into instance, Install python 3.9 package in your server.
- Install mysql (for how to install mysql refer under 'Reference link').
- configure mysql installation & login to mysql "**mysql -u poornima -p**".
- create user with appropriate host and permissions for all tables and database.
- create database table and insert data into that table.
- Test remote connection for that, Install Mysql client in another instance and try to connect by using following command

  mysql -h 'Public-Ip address' -u 'username' -P 'Password' -p
- If you're able to connect with mysql and remotely create, view and modify tables and databases, you have successfully setup your mysql database.

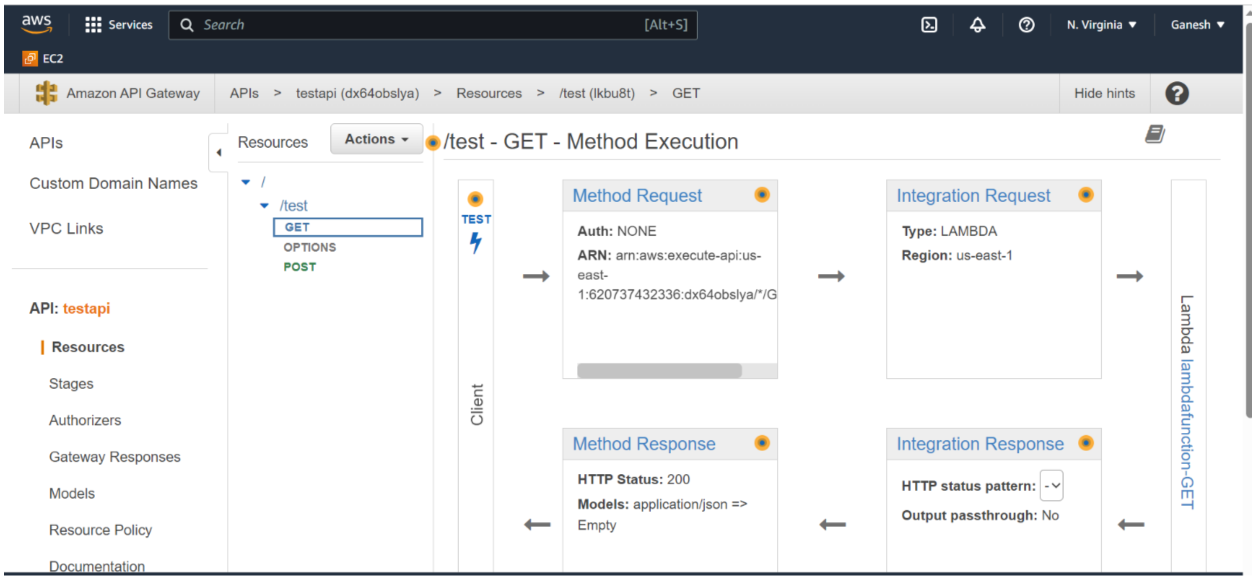
Step 2: Set up the Backend (AWS Lambda to MYSQL Database)

- Go to the Lambda dashboard in the AWS Management Console.
- Download this zip file by using below link https://3ktechnologies22-my.sharepoint.com/:u:/g/personal/vishak_a_3ktechnologies_com/EYtaoD1l4GJCv9NTxw5P7aEBORGnpTBL5tgtHDUhd8VDkw?e=jF4Oi4
- Click on "Layers" in the navigation pane and then "Create layer."
- Provide a name, description, and select the Python runtime compatible with Python 3.9.
- Upload a .zip file containing your Python libraries (ensure they are compatible with Python 3.9). For that zip file refer this link **https://www.geeksforgeeks.org/how-to-install-python-packages-for-aws-lambda-layers/**
- Create three lambda functions for GET, POST, DELETE and Choose the Python runtime as **Python3.9**. When creating or updating a Lambda function, scroll down to the **"Layers"** section.
- Click on **"Add a layer"** and select the layer you created under **Custom-Layer** section.
- Add lambda code in that respective lambda function For Fetching the data, adding and deleting the data from database, use lambda code for **GET, POST, DELETE** in python language and create trigger events for that lambda function.

- ➢ If lambda function executes successfully, Now you are successfully connected backend to your database.

Step 4: Configure the API Gateway Trigger (For connecting Front End To Backend)

- ➢ Create multiple methods like (GET, POST, DELETE) under a single resource in a single API gateway to connect to the Lambda function and attach each lambda function to its corresponding method.



/test - GET - Method Execution

Auth: NONE
ARN: arn:aws:execute-api:us-east-1:620737432336:dx64obslya/*/G

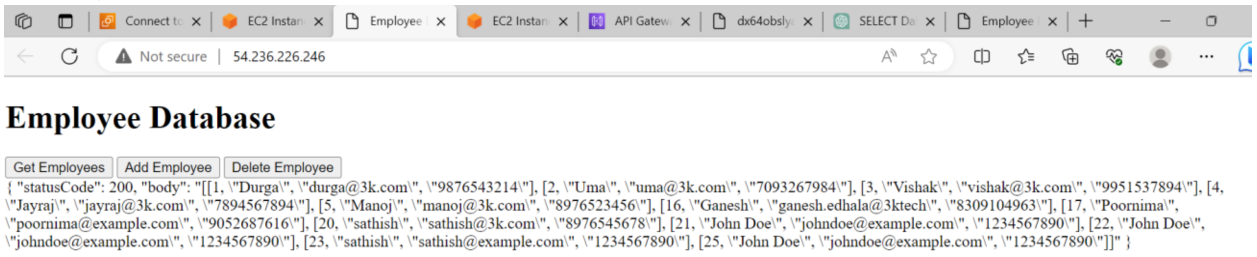
Type: LAMBDA
Region: us-east-1

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

HTTP status pattern: -
Output passthrough: No

Step 5: Set up the Frontend (HTML)

- ➢ To set up front end application, launch one ec2 instance and install Apache web server to create front end application.
- ➢ Prepare the front-end codebase, including HTML and JavaScript files. Ensure that the frontend is designed to communicate with the backend Lambda functions via **"HTTP Port: 80"** Requests.
- ➢ Create one index.html file in that write a code according to your database and save it.
- ➢ Now copy your public ip & open in your browser, now you can able to load, add and delete your database.



Employee Database

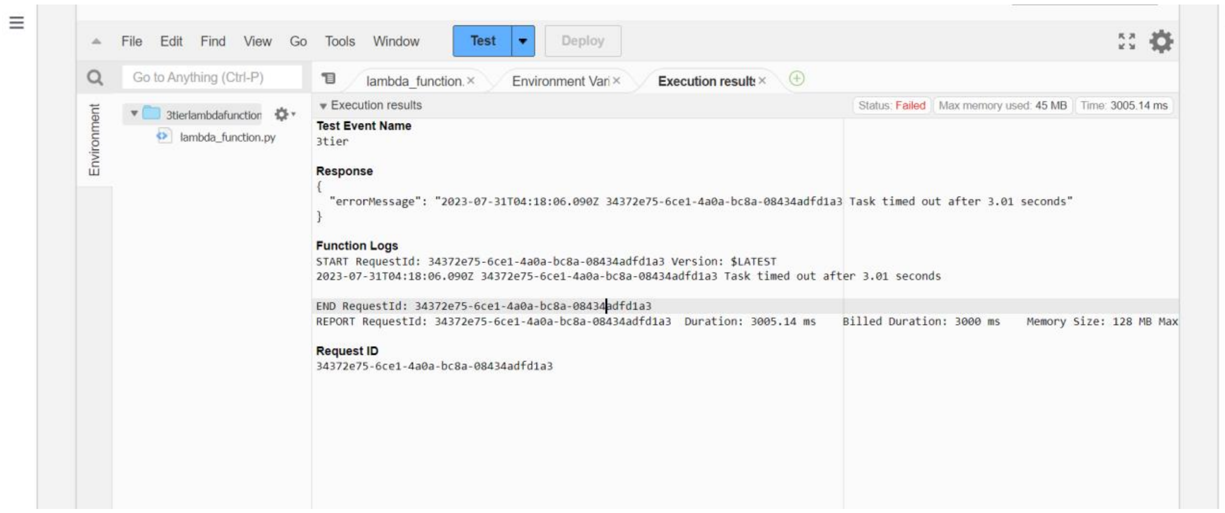
```
{ "statusCode": 200, "body": "[[1, \"Durga\", \"durga@3k.com\", \"9876543214\"], [2, \"Uma\", \"uma@3k.com\", \"7093267984\"], [3, \"Vishak\", \"vishak@3k.com\", \"9951537894\"], [4, \"Jayraj\", \"jayraj@3k.com\", \"7894567894\"], [5, \"Manoj\", \"manoj@3k.com\", \"8976523456\"], [16, \"Ganesh\", \"ganesh.edhala@3ktech\", \"8309104963\"], [17, \"Poornima\", \"poornima@example.com\", \"9052687616\"], [20, \"sathish\", \"sathish@3k.com\", \"8976545678\"], [21, \"John Doe\", \"johndoe@example.com\", \"1234567890\"], [22, \"John Doe\", \"johndoe@example.com\", \"1234567890\"], [23, \"sathish\", \"sathish@example.com\", \"1234567890\"], [25, \"John Doe\", \"johndoe@example.com\", \"1234567890\"]]" }
```

References

For any package installations, lambda function codes & any doubts and queries for this poc, please refer this below repository link
https://github.com/Umapoornima/3tier-architecture.git

Other

1.Error: Lambda functions are unable to connect to the MySQL database.

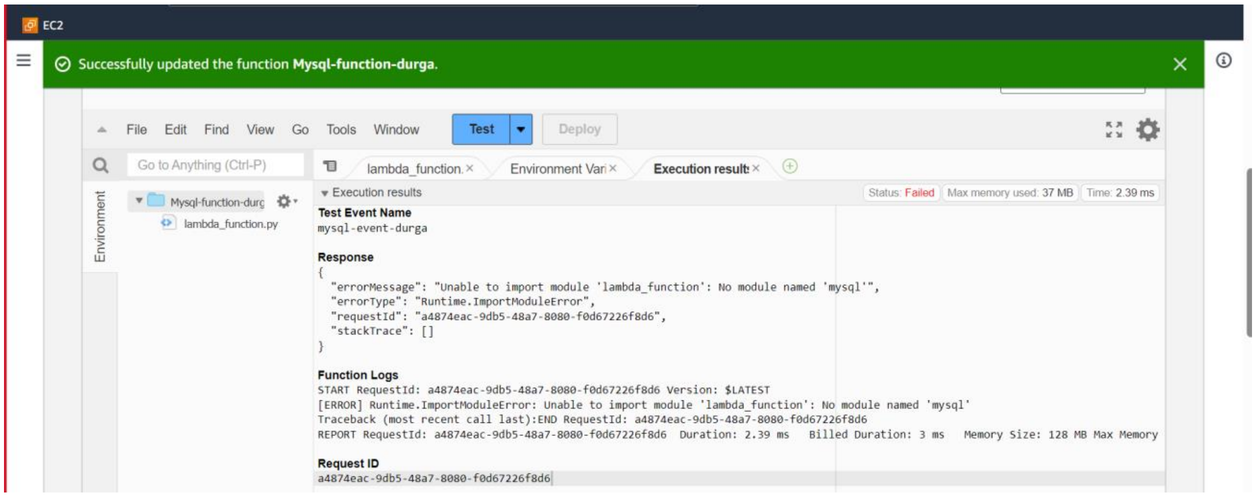


```
Execution results                         Status: Failed   Max memory used: 45 MB   Time: 3005.14 ms
Test Event Name
3tier

Response
{
  "errorMessage": "2023-07-31T04:18:06.090Z 34372e75-6ce1-4a0a-bc8a-08434adfd1a3 Task timed out after 3.01 seconds"
}

Function Logs
START RequestId: 34372e75-6ce1-4a0a-bc8a-08434adfd1a3 Version: $LATEST
2023-07-31T04:18:06.090Z 34372e75-6ce1-4a0a-bc8a-08434adfd1a3 Task timed out after 3.01 seconds

END RequestId: 34372e75-6ce1-4a0a-bc8a-08434adfd1a3
REPORT RequestId: 34372e75-6ce1-4a0a-bc8a-08434adfd1a3  Duration: 3005.14 ms    Billed Duration: 3000 ms    Memory Size: 128 MB Max

Request ID
34372e75-6ce1-4a0a-bc8a-08434adfd1a3
```

Solution:

- Verify that the database host, username, password, and database name in your Lambda function's environment variables are accurate.
- Check that the security groups and network settings of both the EC2 instance hosting the database and the Lambda functions allow incoming/outgoing traffic on the necessary ports (typically 3306 for MySQL).
- Ensure that the MySQL server is running on the EC2 instance.

2. The Error Message: "Unable to import module 'lambda_function': No module named 'mysql'" indicates that your Lambda function is trying to import the mysql module, but the module is not available in the Lambda execution environment



Successfully updated the function **Mysql-function-durga**.

```
Execution results                         Status: Failed   Max memory used: 37 MB   Time: 2.39 ms
Test Event Name
mysql-event-durga

Response
{
  "errorMessage": "Unable to import module 'lambda_function': No module named 'mysql'",
  "errorType": "Runtime.ImportModuleError",
  "requestId": "a4874eac-9db5-48a7-8080-f0d67226f8d6",
  "stackTrace": []
}

Function Logs
START RequestId: a4874eac-9db5-48a7-8080-f0d67226f8d6 Version: $LATEST
[ERROR] Runtime.ImportModuleError: Unable to import module 'lambda_function': No module named 'mysql'
Traceback (most recent call last):END RequestId: a4874eac-9db5-48a7-8080-f0d67226f8d6
REPORT RequestId: a4874eac-9db5-48a7-8080-f0d67226f8d6  Duration: 2.39 ms   Billed Duration: 3 ms   Memory Size: 128 MB Max Memory

Request ID
a4874eac-9db5-48a7-8080-f0d67226f8d6
```

Solution:

- while creating lambda function and layer choose python run time as a python 3.9.

- Use Lambda Layers: If the mysql module is not included in your deployment package, you can create a Lambda Layer that contains the mysql module and attach it to your Lambda function. This way, your function can access the required module from the layer.

3. Lambda Execution Role Permissions

Error: Lambda function encounters "Access Denied" or "Permission Denied" errors.

Solution:

- Review the IAM role attached to the Lambda function. Ensure that it has the necessary permissions to interact with other AWS services (e.g., accessing the database, logging to CloudWatch).
- Check if you have specified the correct execution role when creating the Lambda function.

4. API Gateway Configuration

Error: API Gateway is not invoking the Lambda function or returning expected responses.

Solution:

- Validate the API Gateway configuration, including resource paths, methods, and integration settings.
- Ensure that the Lambda function names and ARNs specified in the API Gateway integration are correct.
- Check the permissions of the Lambda function to ensure that API Gateway is allowed to invoke it.

5. API Gateway and Frontend Configuration

During the setup, you might encounter issues related to the interaction between the API Gateway and the frontend. Here are the challenges you faced and their solutions:

a.) CORS Enabling

Issue: Cross-Origin Resource Sharing (CORS) issues might occur when your frontend hosted on one domain attempts to interact with the API Gateway on another domain.

Solution: Enable CORS on the API Gateway to allow your frontend to make requests from different origins.

b.) Correct API Gateway Endpoint and Resource:

Issue: The front-end code might incorrectly construct the API Gateway endpoint URLs for different CRUD operations, resulting in non-existent resource extensions.

Solution: Verify that the front-end code constructs API Gateway endpoint URLs correctly and adds the relevant resource extensions for each CRUD operation.

6. Missing Authentication Token Error

Error: When making requests to the API Gateway, you might encounter a "Missing Authentication Token" error.

Solution: Ensure that along with the API URL, you include your database resource name at the end of the URL. This helps the API Gateway properly route requests and authenticate them.