Name: Shejal Tiwari Div/Roll No: D15C/57

Experiment No: 11

Aim: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs.

Theory:

AWS Lambda

A fully managed, serverless computing service where you run code without provisioning or managing servers. Lambda automatically scales your application based on the number of incoming requests or events, ensuring efficient resource utilization. You are only charged for the time your code is running, with no upfront cost, making it cost-effective for on-demand workloads.

Lambda Workflow

- **Create a Function**: Write the function code and define its handler (entry point). You can use the AWS Console, CLI, or upload a deployment package.
- **Set Event Sources**: Define how the function is triggered (e.g., when an object is uploaded to S3 or a DynamoDB table is updated).
- **Execution**: When triggered, Lambda runs your function, executes the logic, and automatically scales to handle the incoming event volume.
- **Scaling and Concurrency**: Lambda scales automatically by launching more instances of the function to handle simultaneous invocations. There are also options for configuring **reserved concurrency** to manage traffic.
- **Monitoring and Logging**: Lambda integrates with Amazon CloudWatch for logging and monitoring. Logs for each invocation are sent to CloudWatch, allowing you to track performance and troubleshoot errors.

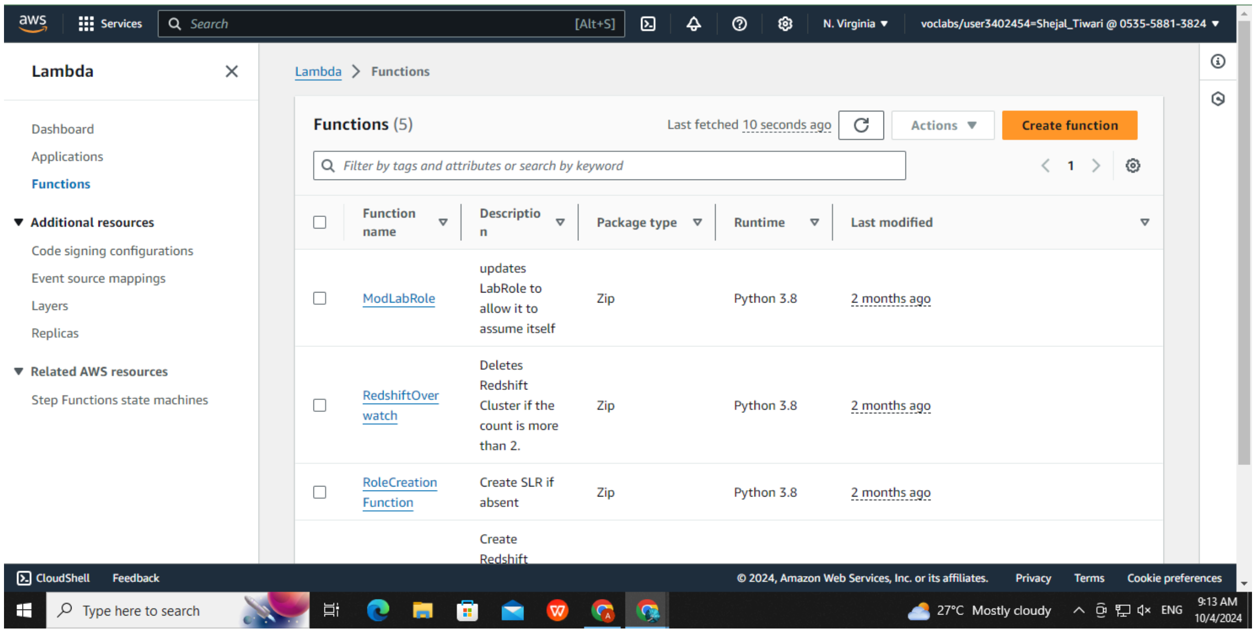
AWS Lambda Functions

- **Python**: Great for quick development with its rich standard library and support for lightweight tasks.
- **Java**: Typically used for more complex, compute-intensive tasks. While it's robust, cold start times can be higher.
- **Node.js**: Excellent for I/O-bound tasks like handling APIs or streaming data, with fast startup times and efficient memory usage.

Name: Shejal Tiwari Div/Roll No: D15C/57

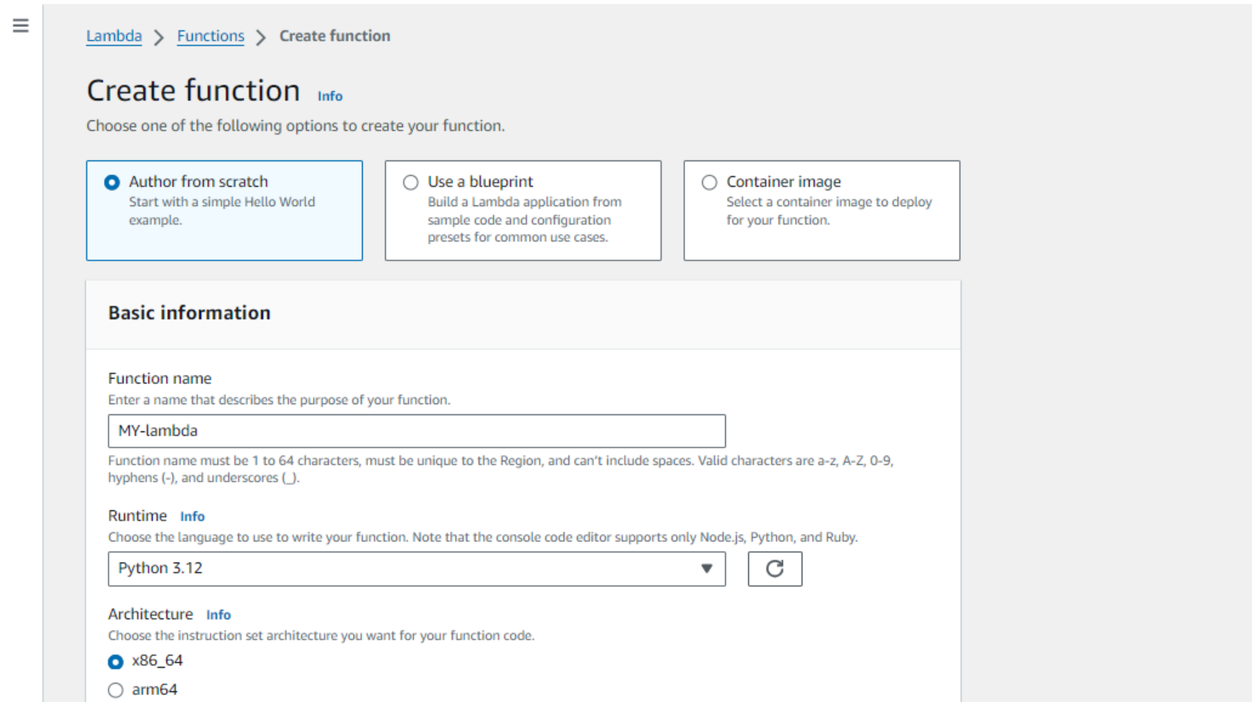
Steps To create the lambda function:

Step 1: Login to your AWS Personal/Academy Accout.Open lambda and click on create function button.
Step 2: Now Give a name to your Lambda function, Select the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby. So will select Python 3.12 , Architecture as x86, and Execution role to Create a new role with basic Lambda permissions.



Functions (5)

Function name	Description	Package type	Runtime	Last modified
ModLabRole	updates LabRole to allow it to assume itself	Zip	Python 3.8	2 months ago
RedshiftOverwatch	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	2 months ago
RoleCreationFunction	Create SLR if absent	Zip	Python 3.8	2 months ago
	Create Redshift			



Create function

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

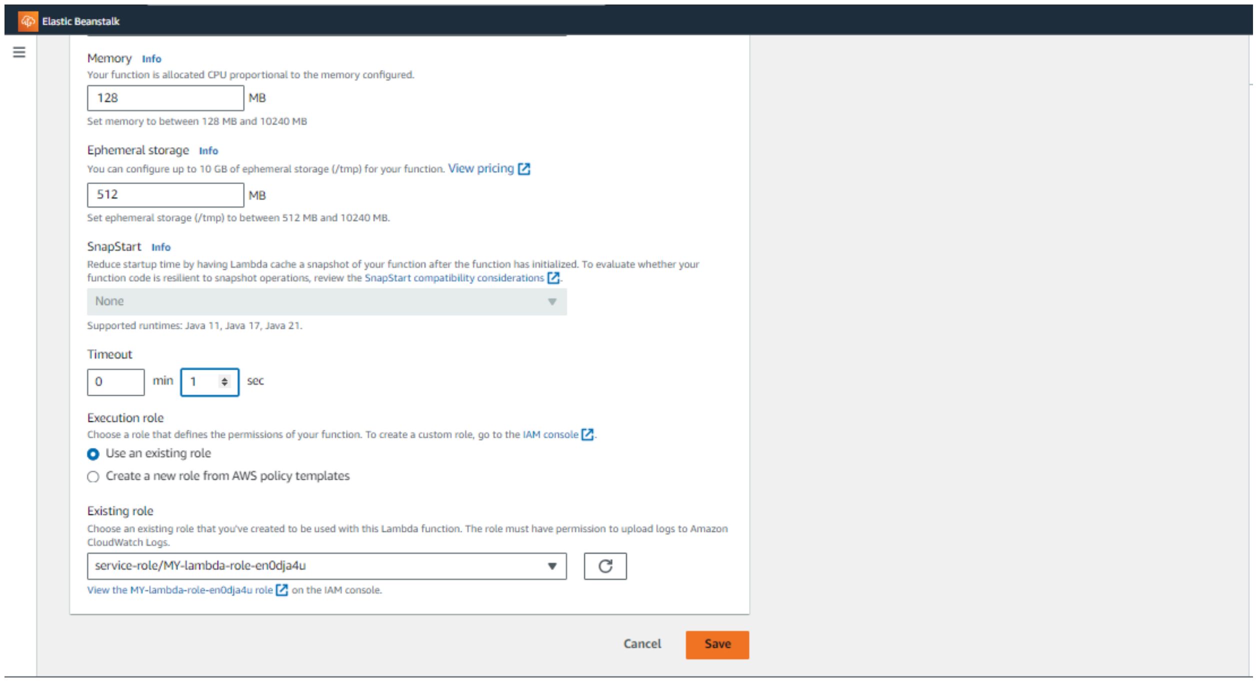
Function name
Enter a name that describes the purpose of your function.
MY-lambda
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.12

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

So See or Edit the basic settings go to configuration then click on edit general setting.

Here, you can enter a description and change Memory and Timeout. I've changed the Timeout period to 1 sec since that is sufficient for now.



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.
None
Supported runtimes: Java 11, Java 17, Java 21.

Timeout
0 min 1 sec

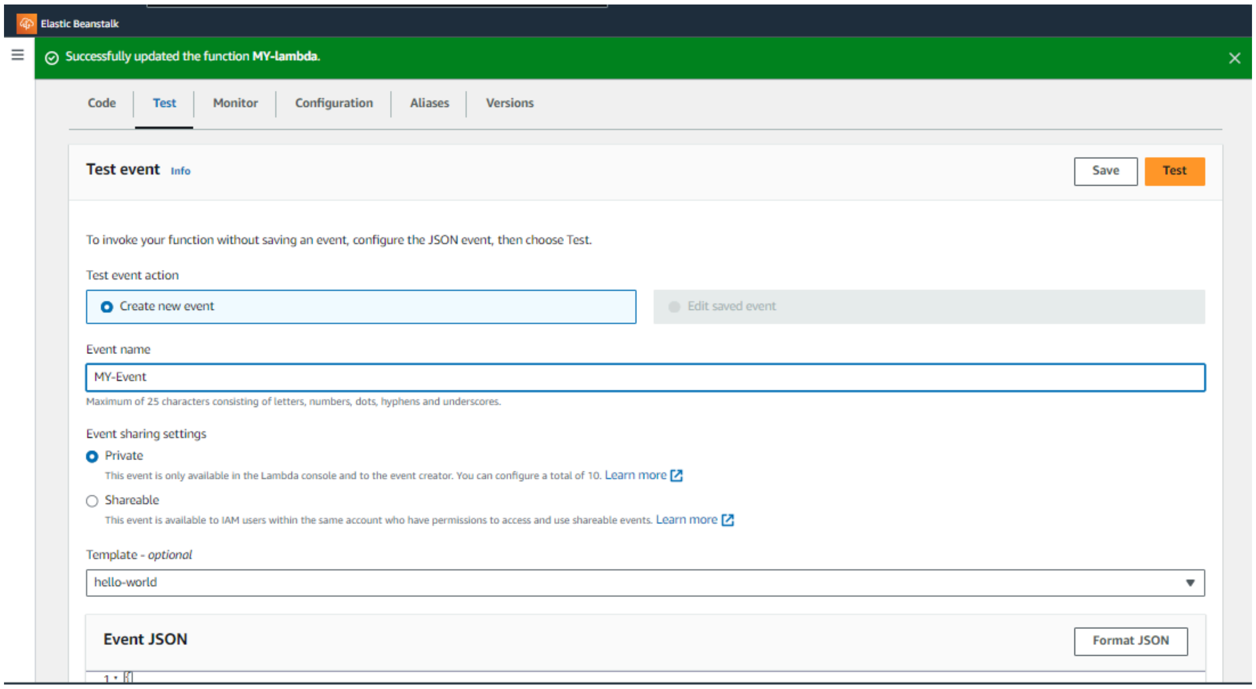
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
[✓ Use an existing role]
Create a new role from AWS policy templates

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/MY-lambda-role-en0dja4u
View the MY-lambda-role-en0dja4u role on the IAM console.

Step 3: Now Click on the Test tab then select Create a new event, give a name to the event and select Event Sharing to private, and select hello-world template.

Step 4: Now In Code section select the created event from the dropdown of test then click on test . You will see the below output.

Name: Shejal Tiwari Div/Roll No: D15C/57



Elastic Beanstalk

Successfully updated the function **MY-lambda**.

Code | Test | Monitor | Configuration | Aliases | Versions

Test event Info

Save | Test

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

● Create new event ○ Edit saved event

Event name

MY-Event

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 10. Learn more

○ Shareable

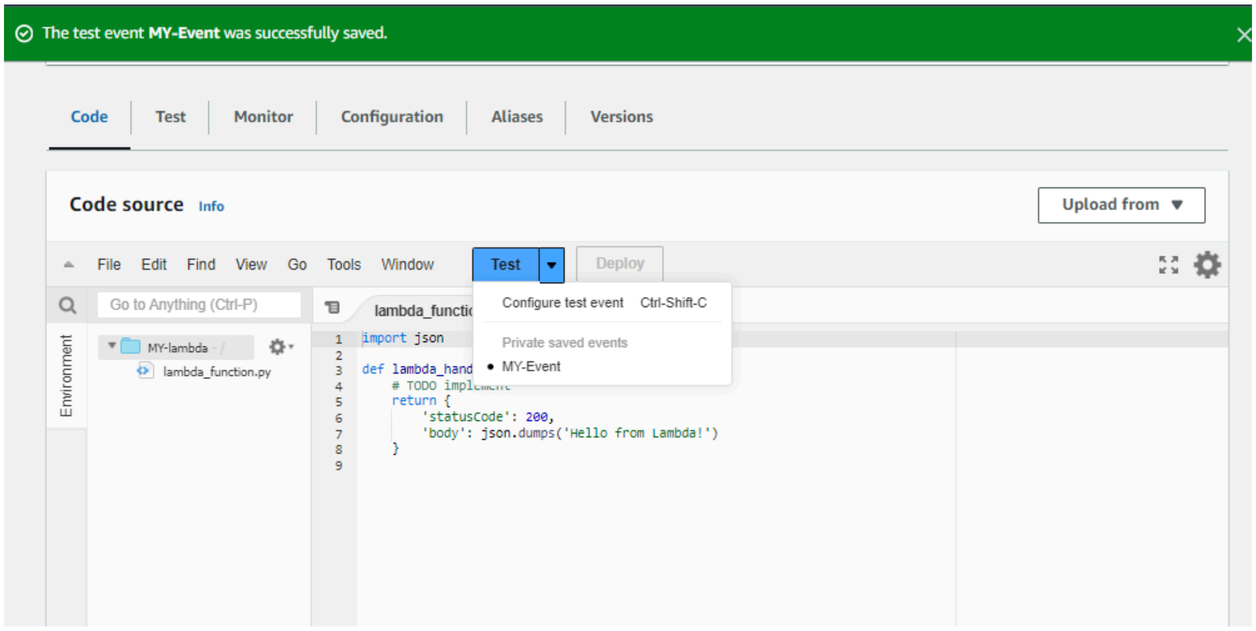
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

hello-world

Event JSON

Format JSON



The test event **MY-Event** was successfully saved.

Code | Test | Monitor | Configuration | Aliases | Versions

Code source Info

Upload from

File Edit Find View Go Tools Window | Test | Deploy

Go to Anything (Ctrl-P)

Configure test event Ctrl-Shift-C

Private saved events

- MY-Event

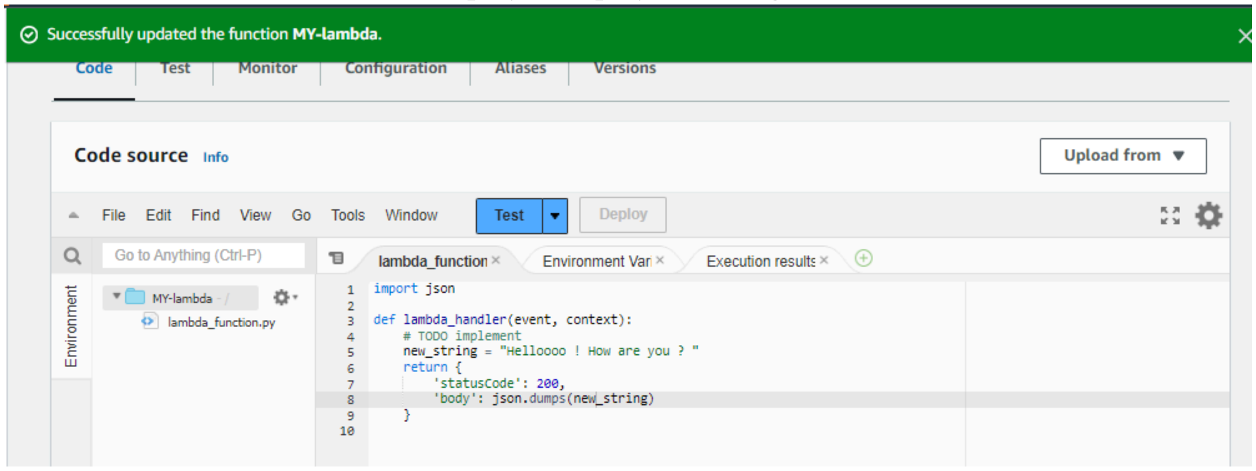
MY-lambda

lambda_function.py

```
import json

def lambda_hand
    # TODO impl
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

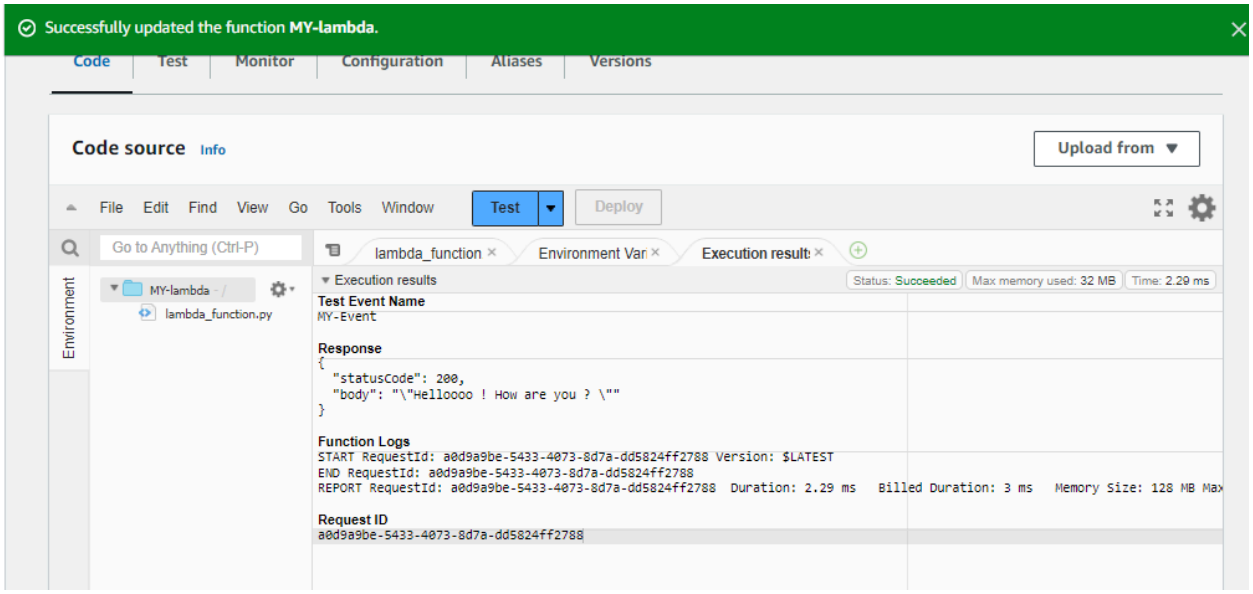
Step 5: You can edit your lambda function code. I have changed the code to display the new String. Now ctrl+s to save and click on deploy to deploy the changes.



```
import json

def lambda_handler(event, context):
    # TODO implement
    new_string = "Helloooo ! How are you ? "
    return {
        'statusCode': 200,
        'body': json.dumps(new_string)
    }
```

Step 6: Now click on the test and observe the output. We can see the status code 200 and your string output and function logs. On successful deployment.



```
Test Event Name
MY-Event

Response
{
  "statusCode": 200,
  "body": "\"Helloooo ! How are you ? \""
}

Function Logs
START RequestId: a0d9a9be-5433-4073-8d7a-dd5824ff2788 Version: $LATEST
END RequestId: a0d9a9be-5433-4073-8d7a-dd5824ff2788
REPORT RequestId: a0d9a9be-5433-4073-8d7a-dd5824ff2788  Duration: 2.29 ms   Billed Duration: 3 ms   Memory Size: 128 MB Max

Request ID
a0d9a9be-5433-4073-8d7a-dd5824ff2788
```

Conclusion:

In this experiment, we successfully created and deployed our first AWS Lambda function using Python, gaining an understanding of its workflow and capabilities. The function was executed and tested, allowing us to observe the output and logs. While the overall process was smooth, there were several challenges that we encountered:

- **Role and Permissions Configuration:** Setting up the correct execution role with the necessary permissions was critical, as misconfigurations could prevent the function from running or accessing other AWS services. Debugging permission issues was time-consuming, especially when using new roles.

- **Timeout Issues:** Initially, the default timeout setting was insufficient for some operations. Adjusting the timeout to a value that suits the function's workload was necessary, especially for more complex operations beyond basic tasks.

- **Event Testing:** Configuring and testing events within Lambda required careful attention. Choosing the wrong template or incorrect event parameters often led to failures during the test phase, requiring adjustments to the event settings.