

Adv DevOps Exp-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.

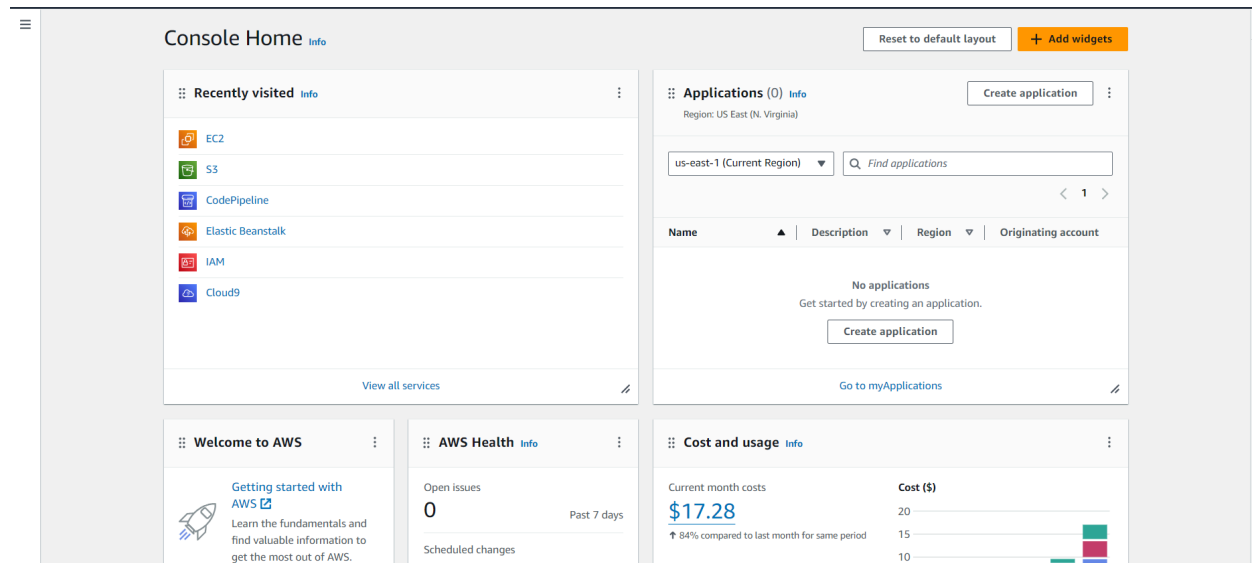
- **Java:** Typically used for more complex, compute-intensive tasks. While it's robust, cold start times can be higher.
- **Node.js:** Excellent for I/O-bound tasks like handling APIs or streaming data, with fast startup times and efficient memory usage.

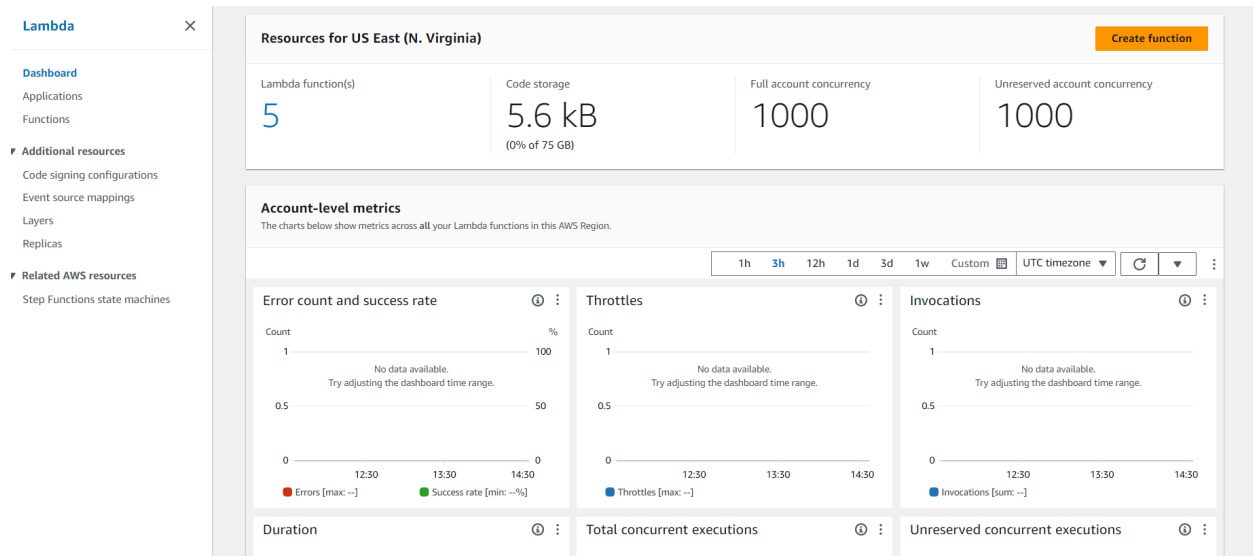
Prerequisites: AWS Personal/Academy Account

Prerequisites: AWS Personal/Academy Account


Steps To create the lambda function:

Step 1: Login to your AWS Personal/Academy Account. 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.

 Services [Alt+S]

Lambda > Functions > Create function

Create function [Info](#)

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.

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.



Architecture [Info](#)
Choose the instruction set architecture you want for your function code.


☒ x86_64
☐ arm64

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.

▼ **Change default execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☒ Create a new role with basic Lambda permissions
☐ Use an existing role
☐ Create a new role from AWS policy templates

 Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named `KCS_Lambda-role-kssqesm9`, with permission to upload logs to Amazon CloudWatch Logs.

► **Advanced settings**

Cancel

Create function

✓ Successfully created the function **KCS_Lambda**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > KCS_Lambda

KCS_Lambda

Throttle Copy ARN Actions

▼ Function overview Info

Export to Application Composer Download

Diagram Template

KCS_Lambda

Layers (0)

+ Add trigger + Add destination

Description
-

Last modified
3 seconds ago

Function ARN
arn:aws:lambda:us-east-1:235494807211:fu
nction:KCS_Lambda

Function URL Info
-

✓ Successfully created the function **KCS_Lambda**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Code Test Monitor Configuration Aliases Versions

Code source Info

Upload from

File Edit Find View Go Tools Window Test Deploy Changes not deployed

Go to Anything (Ctrl-P)

Environment

KCS_Lambda /

lambda_function.py

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello KCS from Lambda!')
8     }
9
```

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

Code Test Monitor Configuration Aliases Versions

General configuration Info

Edit

Description -	Memory 128 MB	Ephemeral storage 512 MB
Timeout 0 min 3 sec	SnapStart None	

Here, you can enter a description and change Memory and Timeout. I've changed the Timeout period to 2 sec since that is sufficient for now.

Basic settings [Info](#)

Description - optional

Memory [Info](#)
Your function is allocated CPU proportional to the memory configured.
 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](#)
 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
 min sec

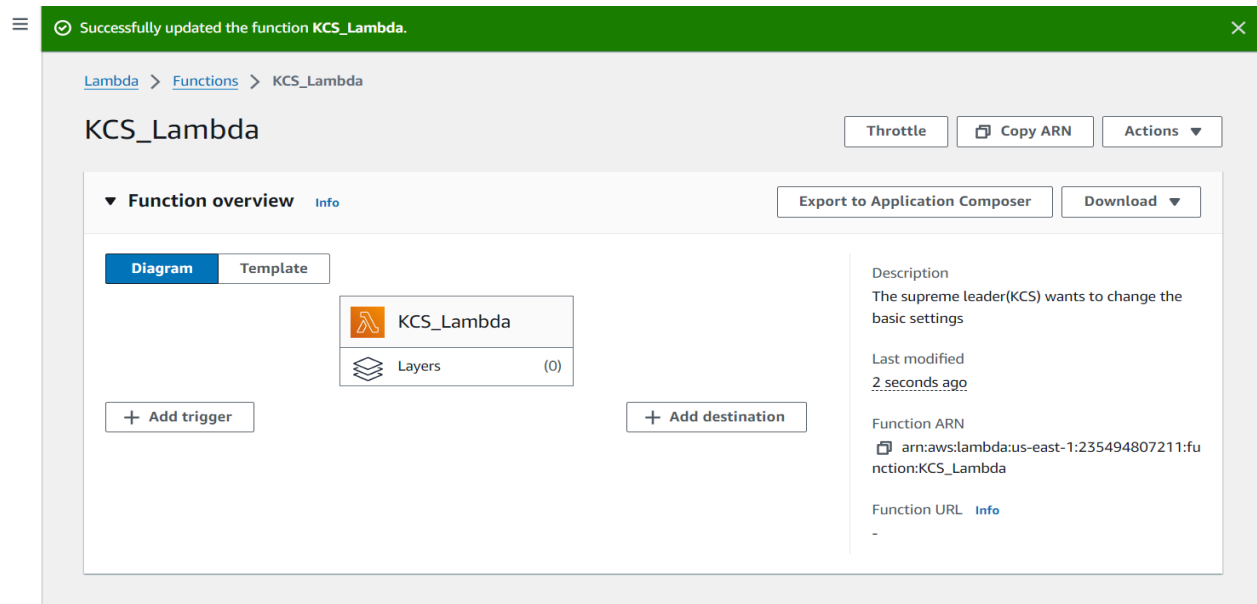
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).
☒ Use an existing role
☐ Create a new role from AWS policy templates

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.

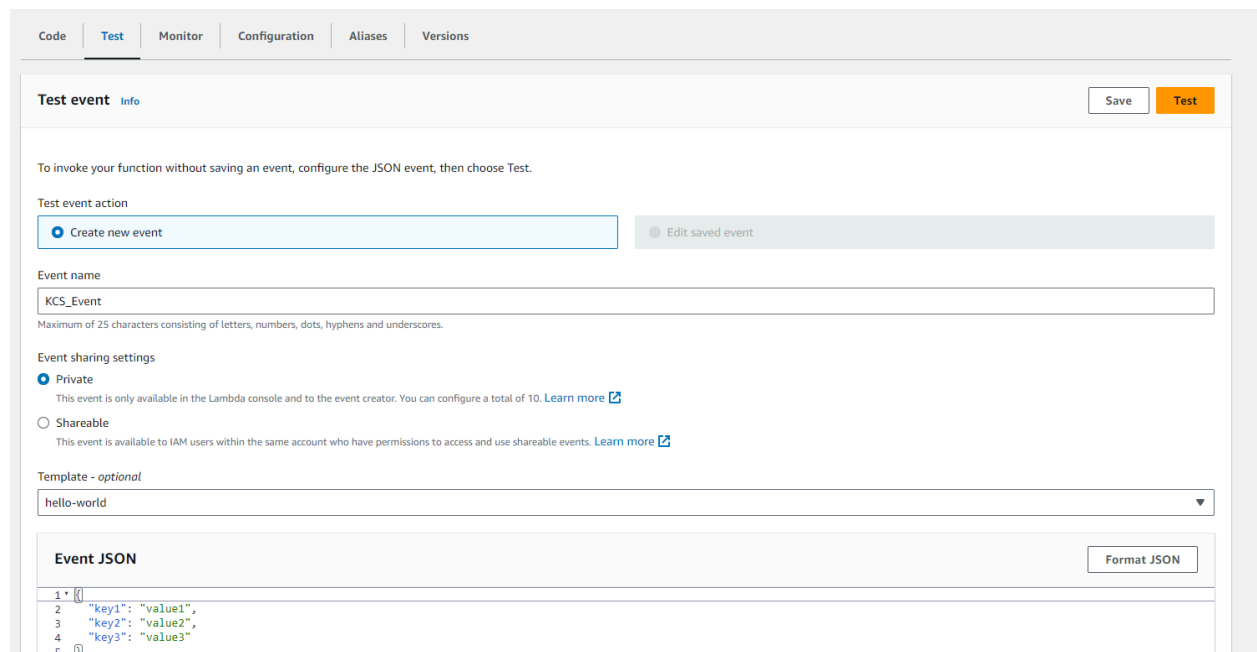
service-role/KCS_Lambda-role-9nzyyxbk ▼



↻

[View the KCS_Lambda-role-9nzyyxbk 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.



 The test event **KCS_Event** was successfully saved. 

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

KCS_Event

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

☒ Private

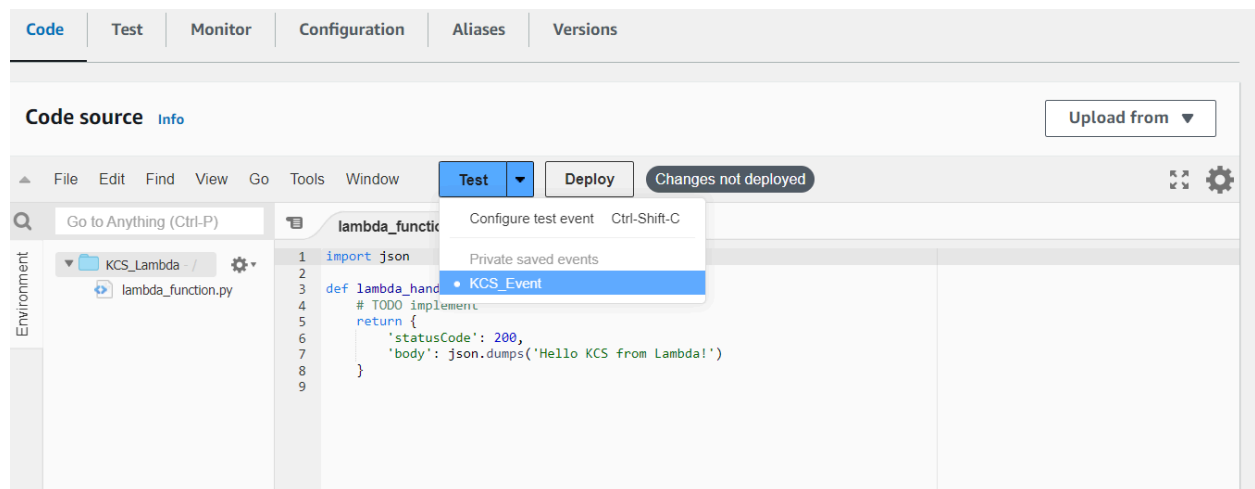
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

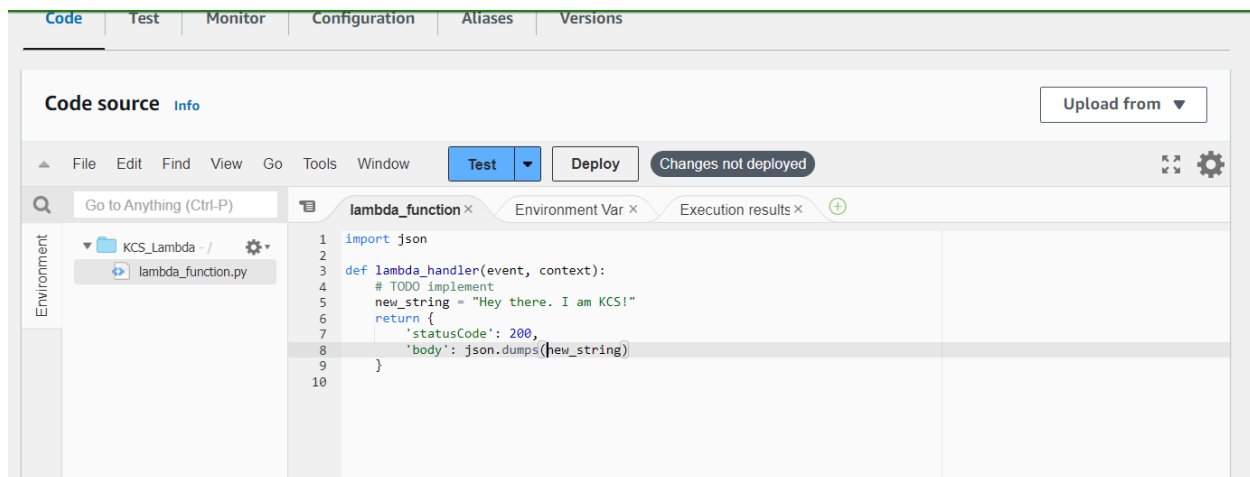
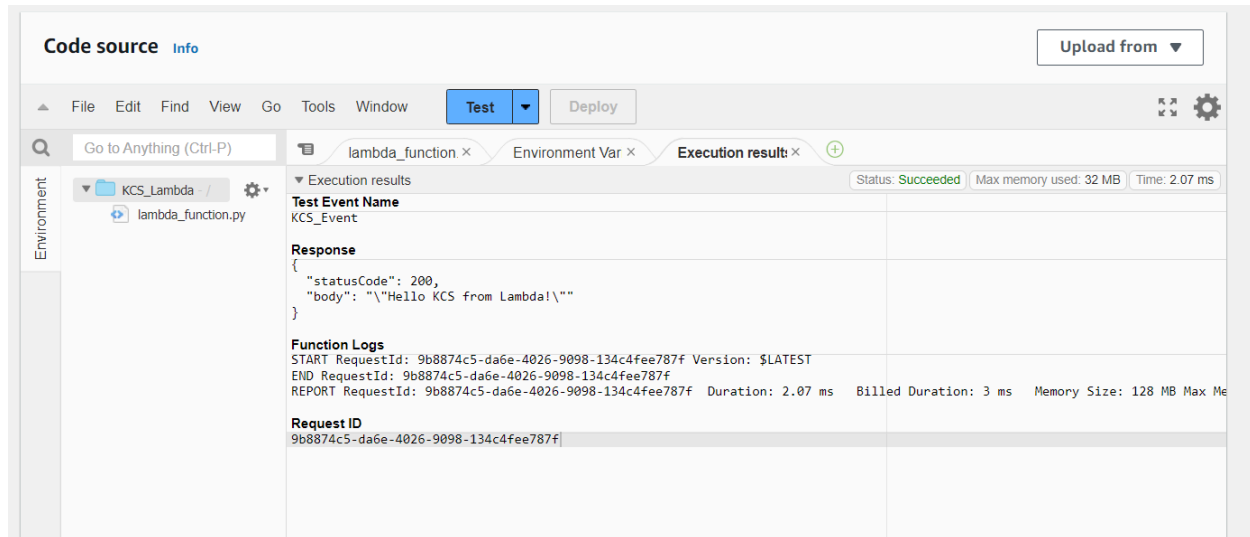
☐ Shareable

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

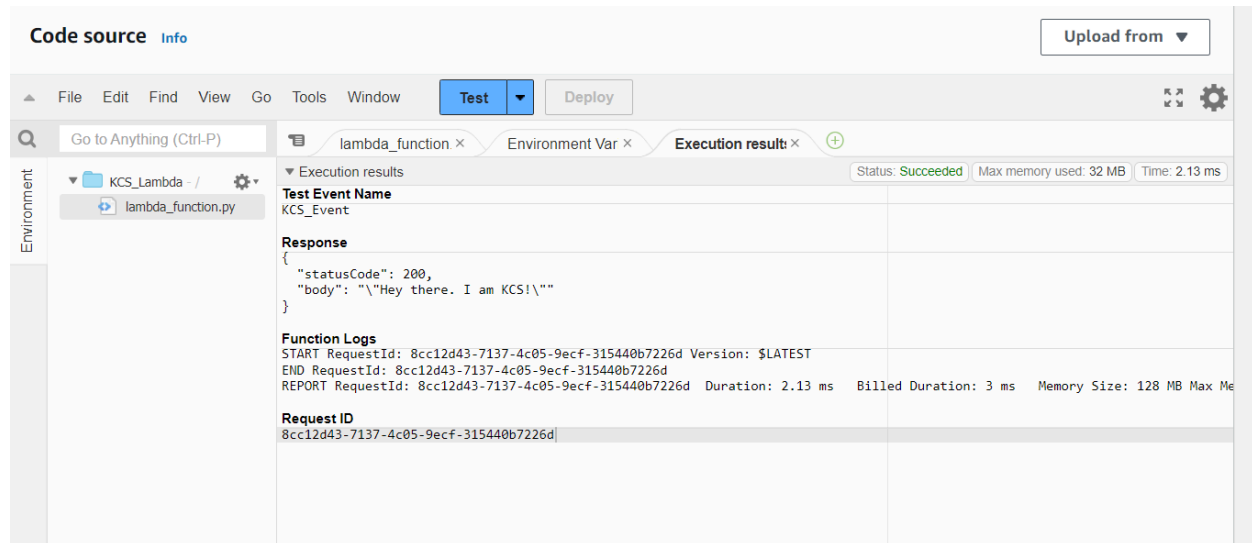
Template - optional

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.





Now ctrl+s to save and click on deploy to deploy the changes



You can see the desired output.

Conclusion: In this experiment, we successfully developed an AWS Lambda function, covering the key steps involved. Starting with the Python-based setup, we configured the function's fundamental settings, including setting the timeout to 1 second. We proceeded to create a test event, deployed the function, and verified its output. Additionally, we made updates to the Lambda function's code and redeployed it, observing the real-time changes. This hands-on experience highlighted AWS Lambda's efficiency and adaptability, enabling rapid serverless application development while AWS handles infrastructure and scaling effortlessly.