

Guided Lab: Using Layers in AWS Lambda Functions

Description

What is a Layer in AWS Lambda?

A Layer in AWS Lambda is a way to manage code, libraries, and dependencies separately from the main Lambda function code. Layers allow you to package libraries and other dependencies that your Lambda function requires, making it easier to manage and share code across multiple Lambda functions. By using Layers, you can avoid including large libraries directly in your function code, which helps optimize function deployment and manage code more efficiently.

Uses of Layers:

- Share common code across multiple Lambda functions.
- Reduce the deployment package size.
- Simplify dependency management.
- Enable version control of dependencies.

Prerequisites

This lab assumes you have a basic understanding of AWS Lambda and Python programming.

If you find any gaps in your knowledge, consider taking the following lab:

- Creating an AWS Lambda function

Objectives

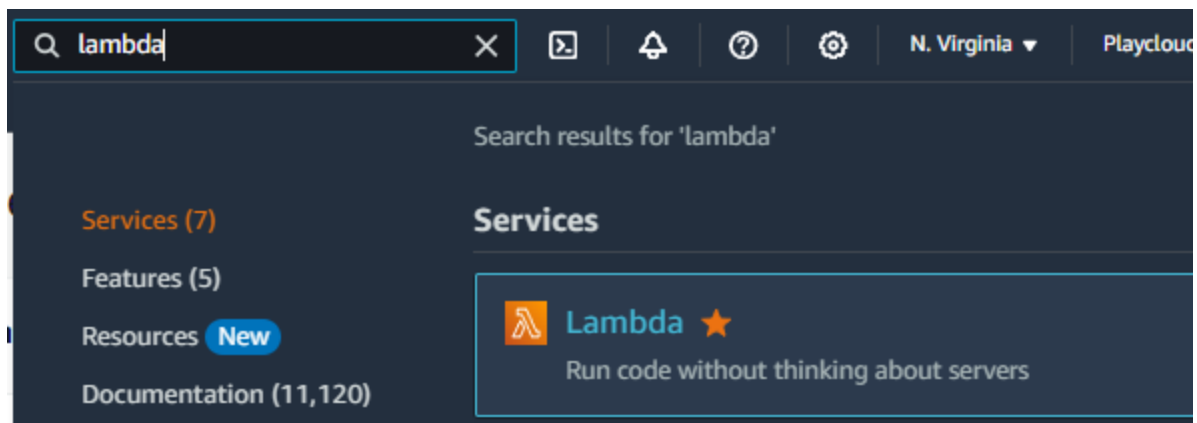
In this lab, you will:

- Understand the concept of AWS Lambda Layers.
- Create a Lambda function that uses an external library (requests).
- Test the function with and without the necessary library.
- Learn how to create and deploy a Layer to include the required library.

Lab Steps

Create a Simple Lambda Function

1. Navigate to the AWS Lambda Console



2. Create a new Lambda function using the following configurations:

- Choose **Author from scratch**.
- Function name: myLambdaFunction
- Select Python 3.12 as the runtime.
- **Execution role:**
 - Select Use an Existing Role: PlayCloud-Sandbox

Basic information

Function name

Enter a name that describes the purpose of your function.

myLambdaFunction

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 3.12

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

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.

PlayCloud-Sandbox

[View the PlayCloud-Sandbox role](#) on the IAM console.

- Click **Create function**

3. Replace the default code with the following Python code:

Note: We are using the **old console editor** for this lab. You're welcome to switch between the old and new editors if you prefer; the steps remain the same, though the interface may have a slightly different appearance in the new editor.

```
import requests
```

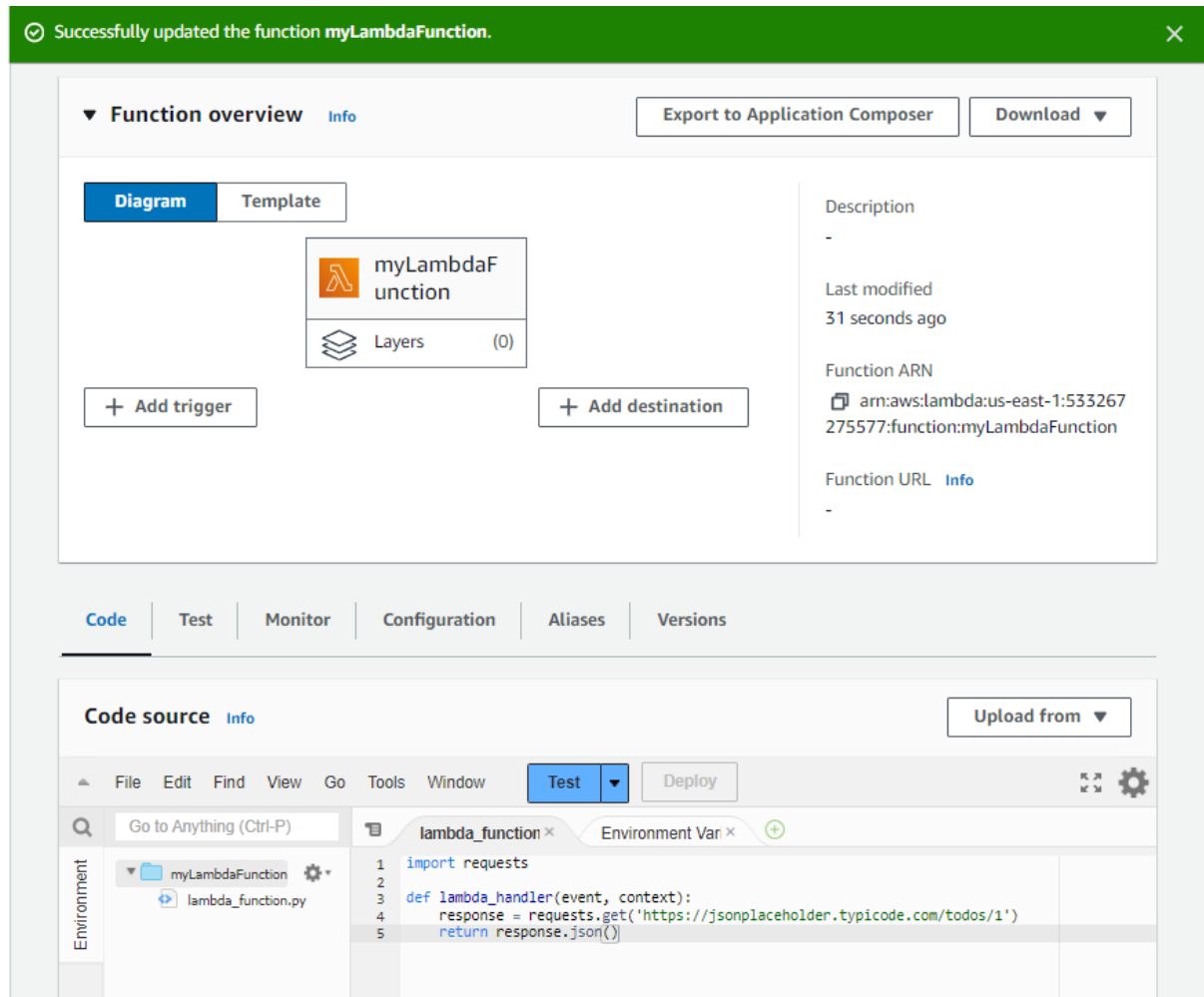
```
def lambda_handler(event, context):
```

```
    response = requests.get('https://jsonplaceholder.typicode.com/todos/1')
```

```
    return response.json()
```

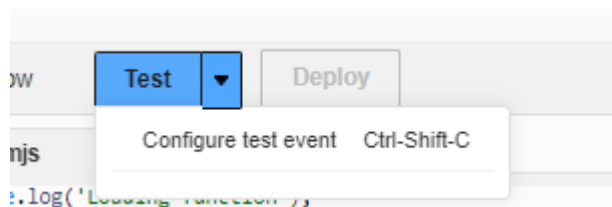
This Lambda function uses the requests library to make an HTTP GET request to a public API (jsonplaceholder.typicode.com) to fetch a sample TODO item. The requests library simplifies the process of sending HTTP requests in Python.

4. Click on **Deploy** to save changes.



Testing the Lambda Function Without the Required Library

1. Once your function is deployed. Click the arrow dropdown of the **Test** button



2. Click on **Configure test event**, and follow the configuration below:

- Event name: Test
- Template- optional: hello-world
 - Leave the rest as default

Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

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

Test

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

```
1 {  
2   "key1": "value1",  
3   "key2": "value2",  
4   "key3": "value3"  
5 }
```

Cancel

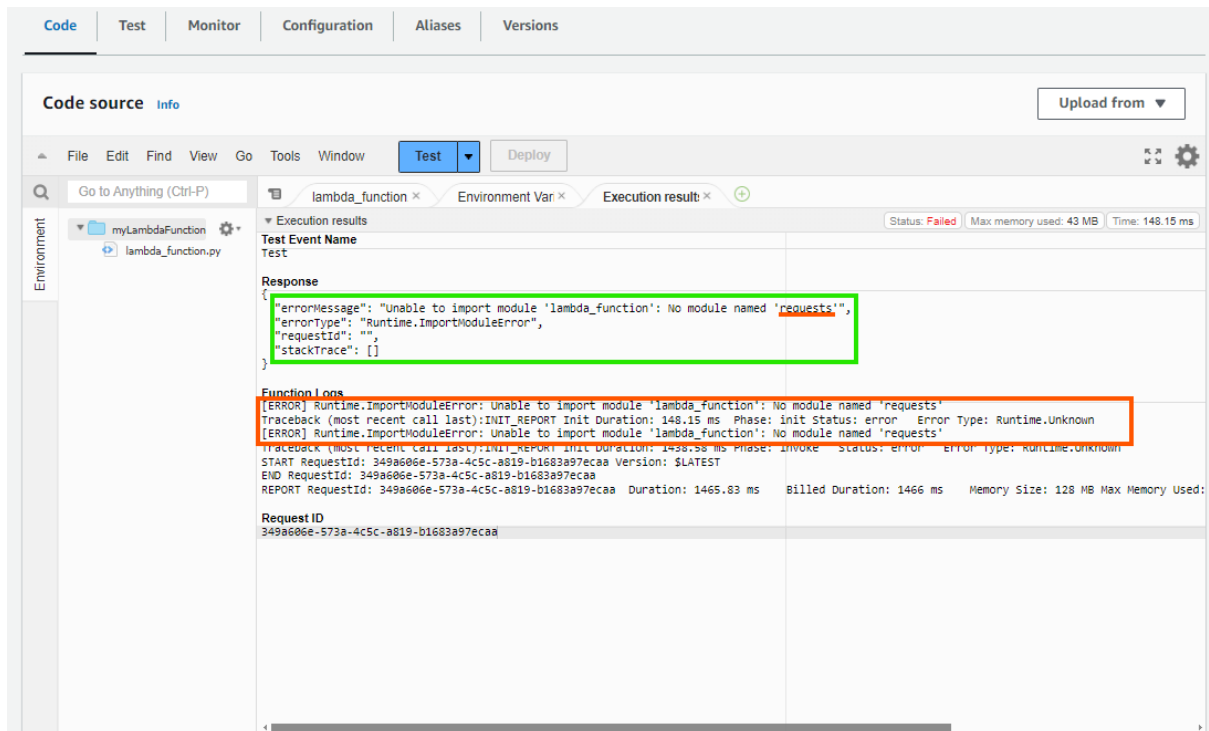
Invoke

Save

- Click on Save

3. Now, click on **Test** to execute the function.

The function should fail with an error stating that the requests module is not found. This happens because the requests library is not in the AWS Lambda environment by default.



This error occurs because the requests library is an external dependency that is not natively available in the Lambda execution environment. To resolve this, we need to include the requests library in our function, where Lambda Layers come into play.

Creating and Adding a Layer with the Requests Library

1. Download the Requests Library: (For simplicity, you can use the .zip file below)

https://media.tutorialsdojo.com/public/my_lambda_layer.zip

- Or you can also download your own from your local PC:
 - Open your terminal (GitBash, Putty) and create a directory for your layer:

```
mkdir python
```

```
cd python
```

This will create a python directory that you can navigate.

- - Download the Requests Library:
 - Run the following command to download the requests library and its dependencies into the python directory:

```
pip install requests -t .
```

- - - **Alternate Command:** Depending on the Python version, you might need to use:

```
py -m pip install requests -t .
```

These commands download the requests library and its dependencies into your current directory. The -t . option specifies the target directory for the installation.

- - **Your directory structure should look like this:**

```
python/
```

```
└─ requests/
```

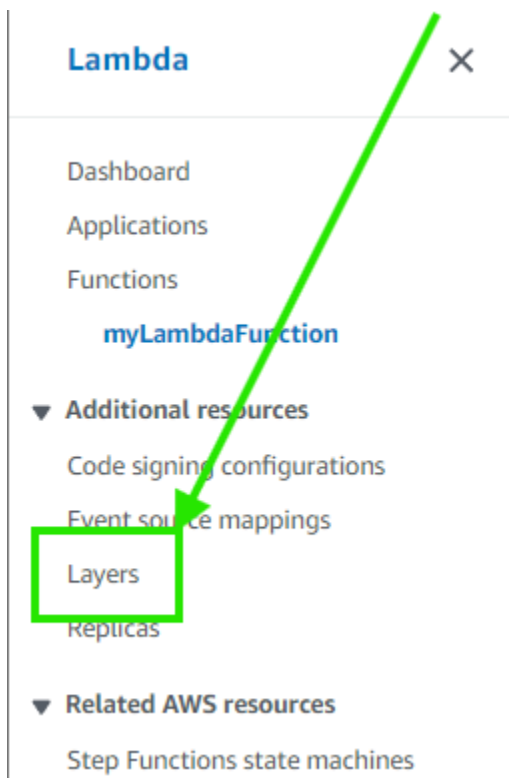
- **Zip the contents of the python directory:**
 - **On Unix-like systems:**
 - Navigate inside the python directory and run:

```
zip -r my_lambda_layer.zip .
```

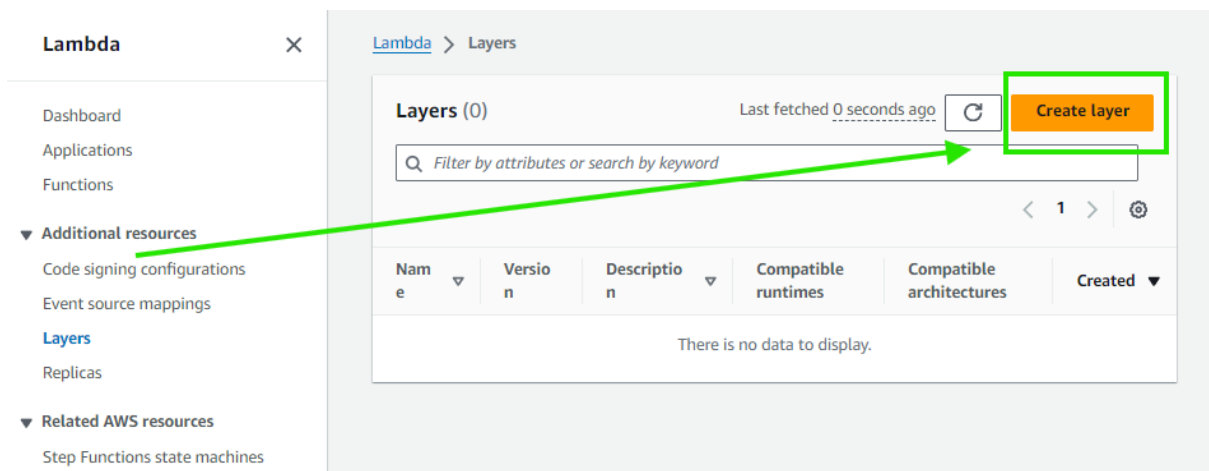
- - **On Windows:**
 - Compress the python folder to a zip file named my_lambda_layer.zip.
 - Ensure that the structure is maintained correctly.

2. Create a Layer:

- Go back to the AWS Lambda console.
- In the left pane, select **Layers**.



- Choose **Create layer**.



- Name your layer RequestsLayer. You can add a Description if desired.
- Upload the my_lambda_layer.zip file containing the requests library.
- Choose **Python 3.12** as the runtime.

Create layer

Layer configuration

Name


RequestsLayer

Description - *optional*

Description

☒ Upload a .zip file

☐ Upload a file from Amazon S3

 Upload

my_lambda_layer.zip

1.03 MB



For files larger than 10 MB, consider uploading using Amazon S3.

Compatible architectures - *optional* [Info](#)

Choose the compatible instruction set architectures for your layer.

☐ x86_64

☐ arm64

Compatible runtimes - *optional* [Info](#)

Choose up to 15 runtimes.

Runtimes



Python 3.12



License - *optional* [Info](#)

Cancel

Create

- Click **Create**.


[Lambda](#) > [Layers](#) > [RequestsLayer](#) > 1

RequestsLayer

DeleteDownloadCreate version

✔ Successfully created layer RequestsLayer version 1.✕

Version details

Version 1	Version ARN  arn:aws:lambda:us-east-1:533267275577:layer:RequestsLayer:1	Description -
Created 12 seconds ago	License -	Compatible runtimes python3.12
Compatible architectures -		

VersionsFunctions using this version

All versions (1)

< 1 >

Version	Version ARN	Description
1	arn:aws:lambda:us-east-1:533267275577:layer:RequestsLayer:1	-

3. Add the Layer to Your Lambda Function:

- Go to your myLambdaFunction function.

Lambda X

Dashboard
Applications
Functions
myLambdaFunction

▼ **Additional resources**
Code signing configurations
Event source mappings
Layers
Replicas

▼ **Related AWS resources**
Step Functions state machines

Lambda > Functions > myLambdaFunction

myLambdaFunction

Throttle Copy ARN Actions ▼

▼ **Function overview** Info Export to Application Composer Download ▼

Diagram Template

myLambdaFunction
Layers (0)

+ Add trigger + Add destination

Description
-
Last modified
20 minutes ago
Function ARN
arn:aws:lambda:us-33267275577:function:myLambdaFunction
Function URL Info
-

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)

Environment

- myLambdaFunction
 - lambda_function.py

```
1 import requests
2
3 def lambda_handler(event, context):
4     response = requests.get('https://jsonplaceholder.typicode.com/todos/1')
5     return response.json()
```

- Scroll down from the Code tab of the console. In the **Layers** section, click **Add a layer**.

1:1 Python Spaces: 4

Code properties [Info](#)

Package size 252 byte	SHA256 hash WrsqpidvFw24QEzhMHllp2gBVNQ61Y lmZaZMBvHGCKl=	Last modified August 29, 2024 at 11:25 AM GMT+8
--------------------------	---	--

Runtime settings [Info](#)

Edit Edit runtime management configuration

Runtime Python 3.12	Handler Info lambda_function.lambda_handler	Architecture Info x86_64
------------------------	--	---

▶ Runtime management configuration

Layers [Info](#)

Edit Add a layer

Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN
There is no data to display.					

- Choose **Custom layers** and select the RequestsLayer you just created and its current version.

Add layer

Function runtime settings

Runtime
Python 3.12

Architecture
x86_64

Choose a layer

Layer source [Info](#)

Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also [create a new layer](#).

☐ AWS layers

Choose a layer from a list of layers provided by AWS.

☒ Custom layers

Choose a layer from a list of layers created by your AWS account or organization.

☐ Specify an ARN

Specify a layer by providing the ARN.

Custom layers

Layers created by your AWS account or organization that are compatible with your function's runtime.

RequestsLayer

Version

1

Cancel

Add

- Click **Add**. You will notice the Lambda Function will update, and you should also see the RequestLayer added to the Layer section. Just wait for it to finish.

✔ Successfully updated the function myLambdaFunction.

1:1 Python Spaces: 4 ⚙

Code properties [Info](#)

Package size
252 byte

SHA256 hash
WrsppidvFw24QEzhMHLp2gBVNQ61Y
lmZaZM3vHGCKl=

Last modified
August 29, 2024 at 11:50 AM GMT+8

Runtime settings [Info](#)

Edit Edit runtime management configuration

Runtime
Python 3.12

Handler [Info](#)
lambda_function.lambda_handler

Architecture [Info](#)
x86_64

▶ Runtime management configuration

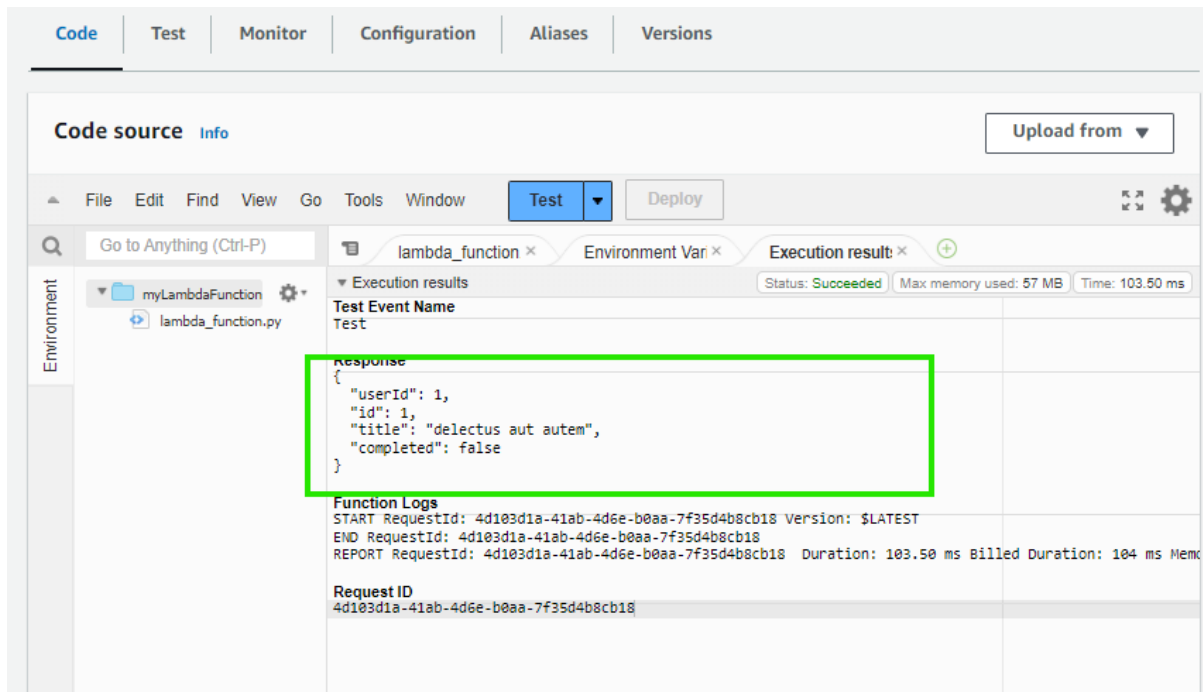
Layers [Info](#)

Edit Add a layer

Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN
1	RequestsLayer	1	python3.12	-	arn:aws:lambda:us-ea

4. Test the Lambda Function Again:

- Test the function using the same test event.
- This time, the function should execute successfully and return the expected JSON response.



That's it! Congratulations! In this lab, you learned how to use AWS Lambda Layers to manage external dependencies, specifically adding the requests library to a Lambda function. Initially, the function failed without the library, highlighting the importance of handling dependencies properly. By creating and deploying a Layer, you successfully enabled the function to execute as intended, demonstrating the flexibility and efficiency of Layers in serverless environments.

AWS Lambda Layers are a powerful tool for sharing code across multiple functions, reducing deployment package sizes, and maintaining a clean, organized codebase. This lab provided practical experience in using Layers, a crucial skill for developing scalable and maintainable serverless applications in AWS. Happy learning!