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

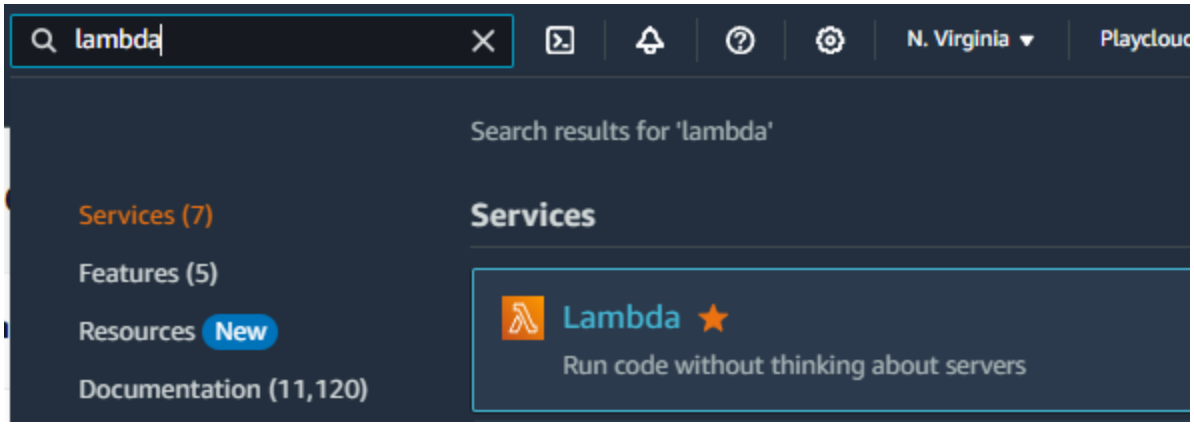
In this lab, you will:

- 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

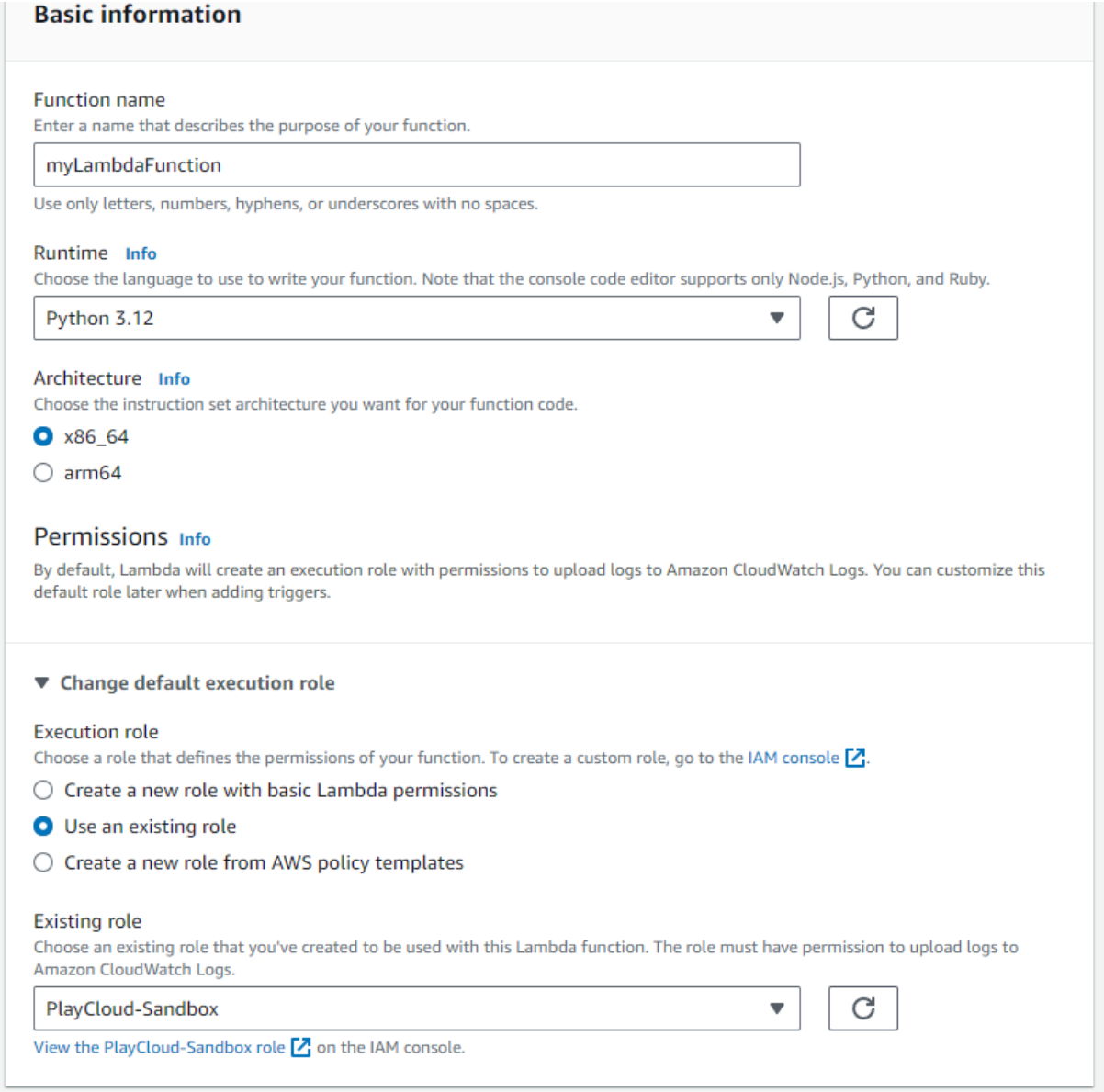
Create a Simple Lambda Function

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-Sanbox



- 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.*

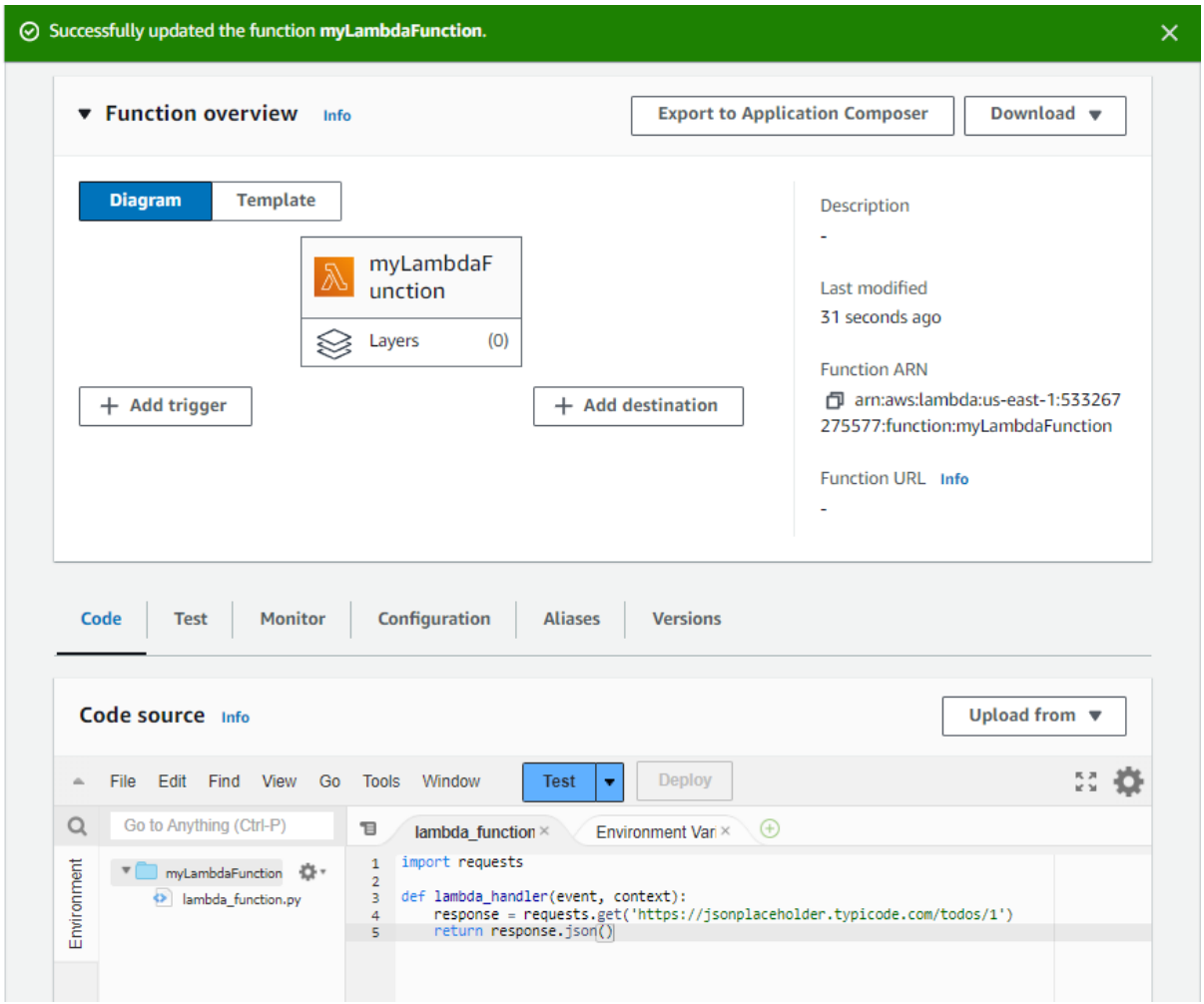
```
import requests


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.

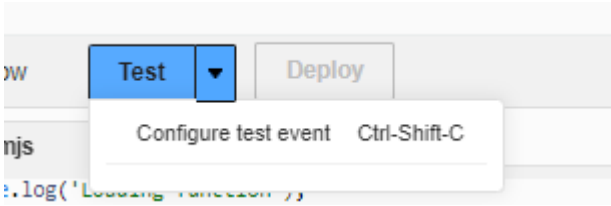
---

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



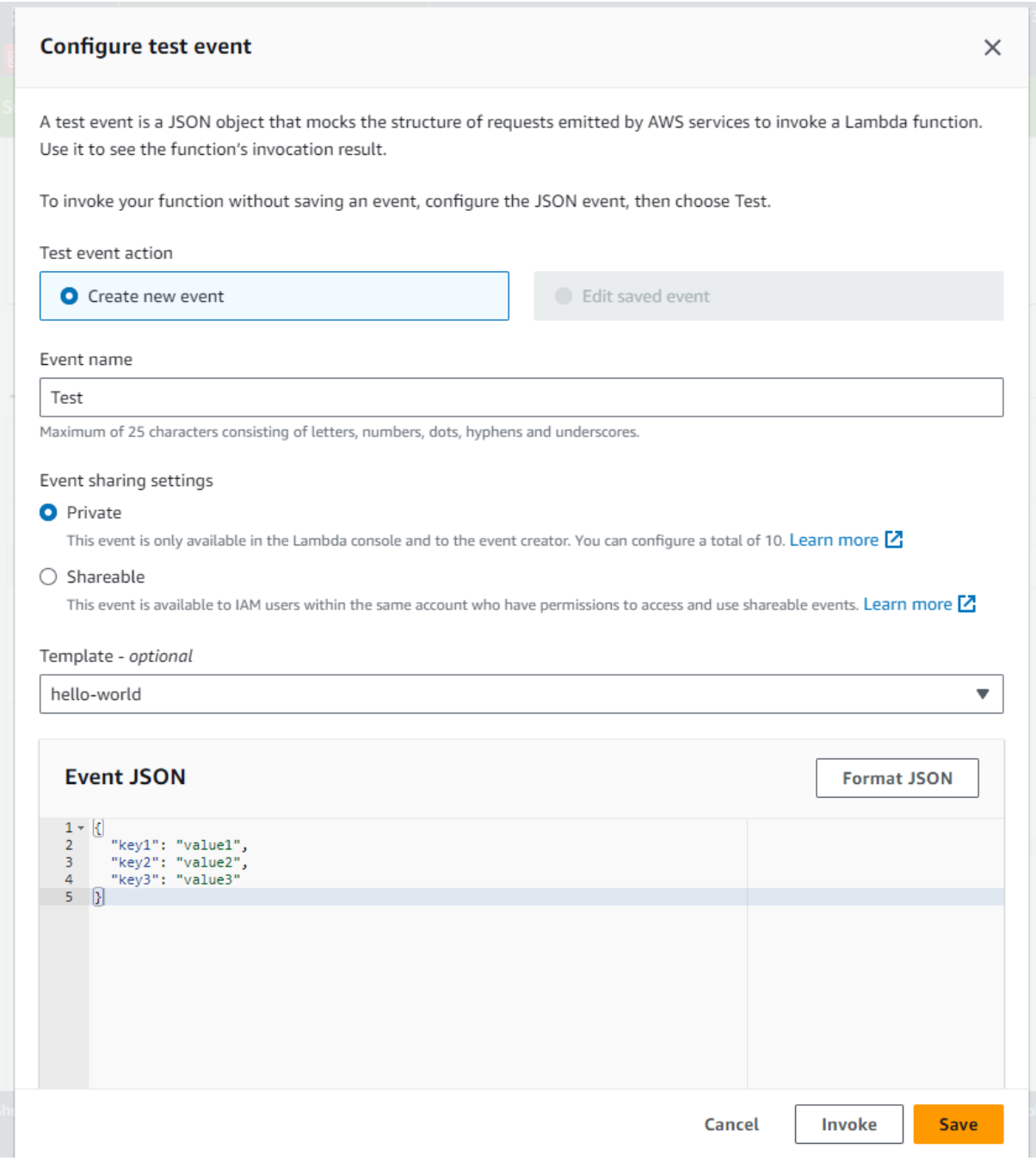
Testing the Lambda Function Without the Required Library

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:

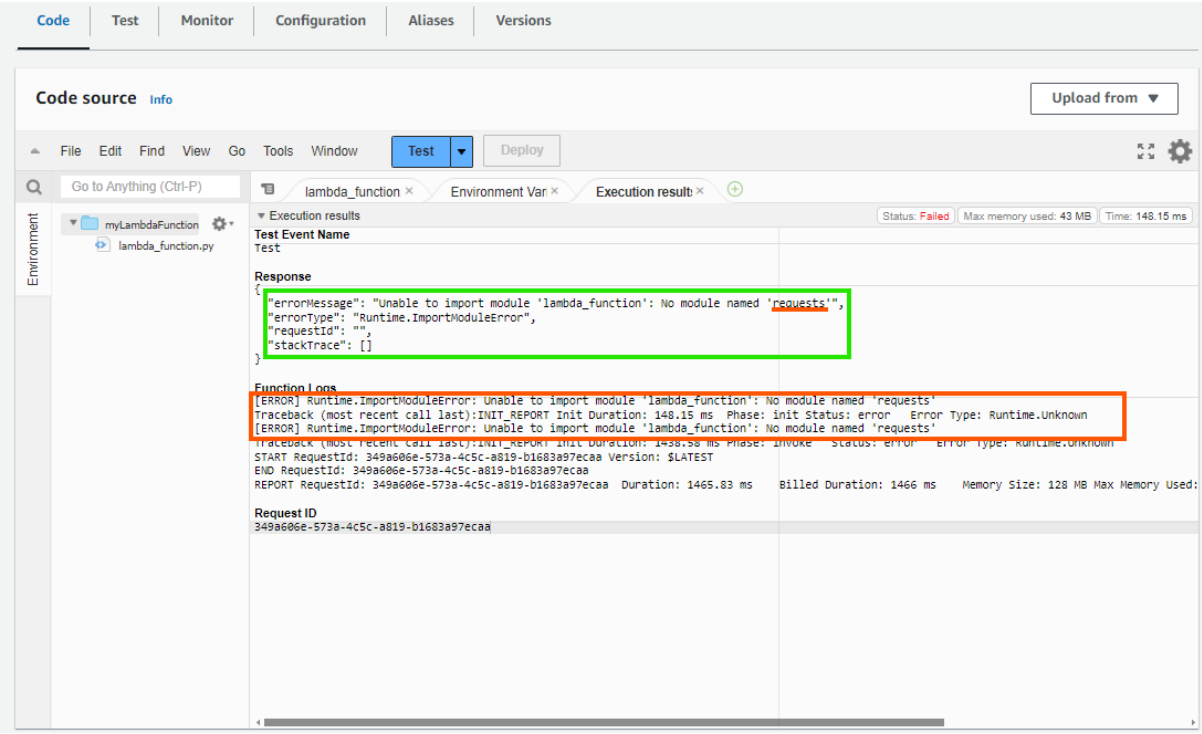
- Event name: Test
- Template- optional: hello-world
    - Leave the rest as default



- 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/

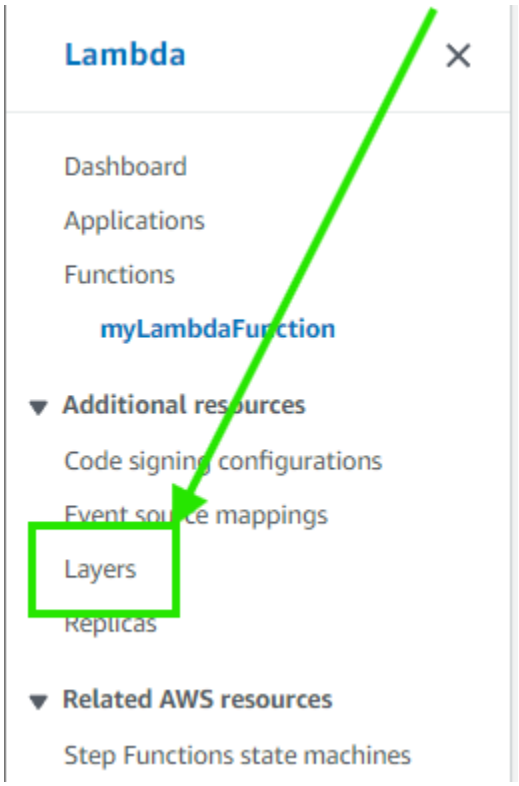
- **Zip the contents of the** python **directory:**
    - **On Unix-like systems:**
        - Navigate inside the python directory and run:

zip -r my_lambda_layer.zip .

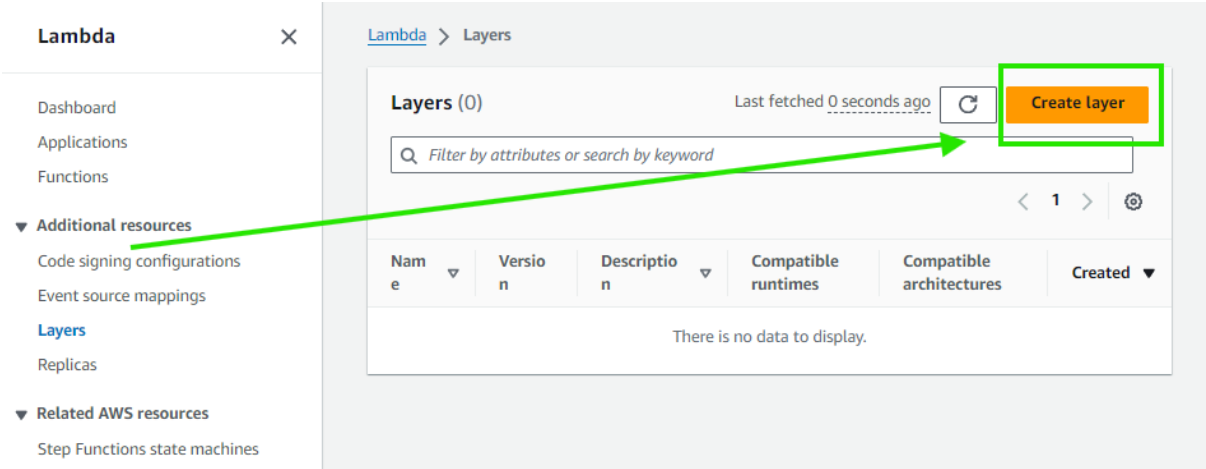
-
    - **On Windows:**
        - Compress the python folder to a zip file named my_lambda_layer.zip.
        - Ensure that the structure is maintained correctly.

2. **Create a Layer:**

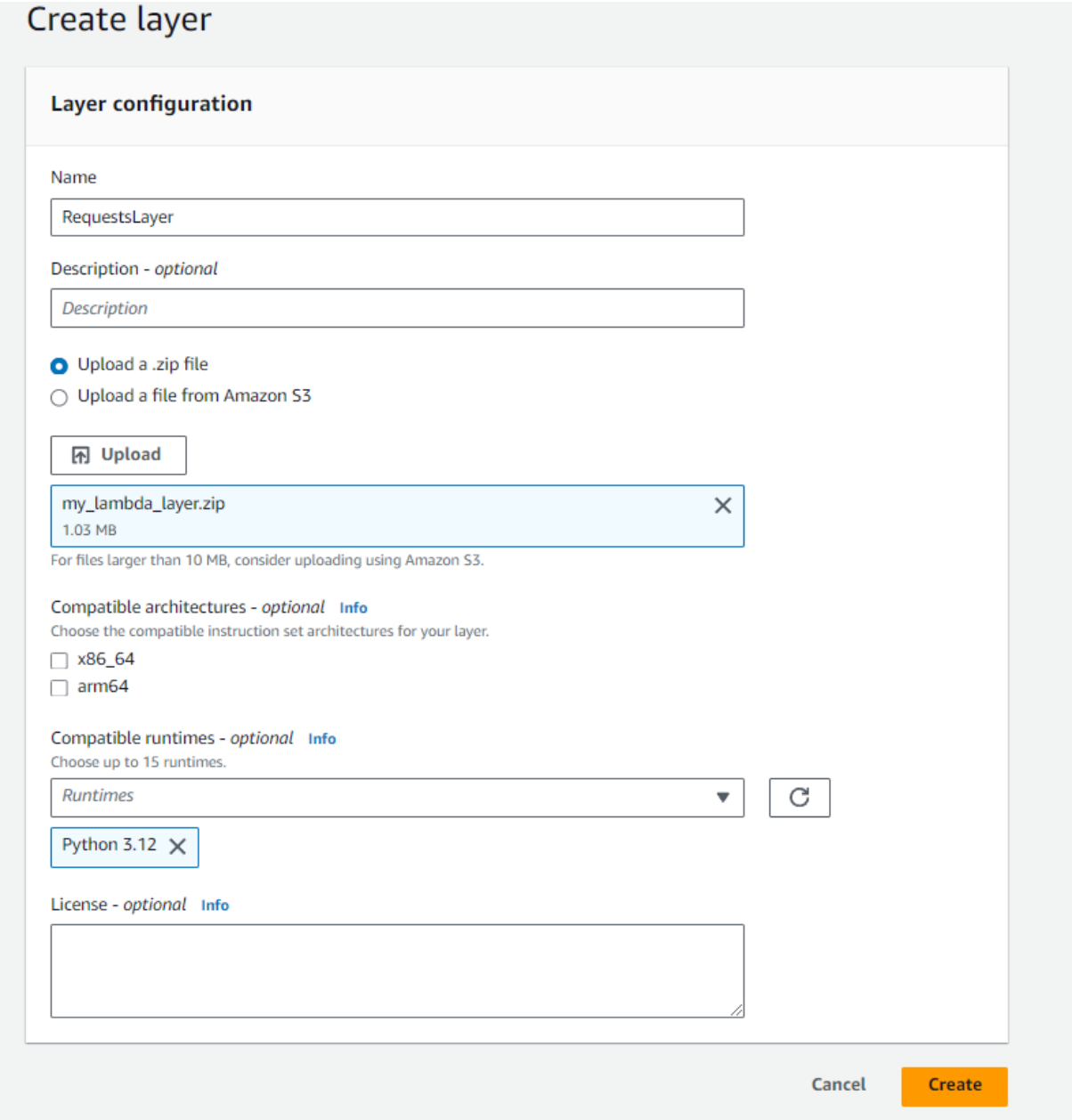
- Go back to the AWS Lambda console.
- In the left pane, select **Layers**.



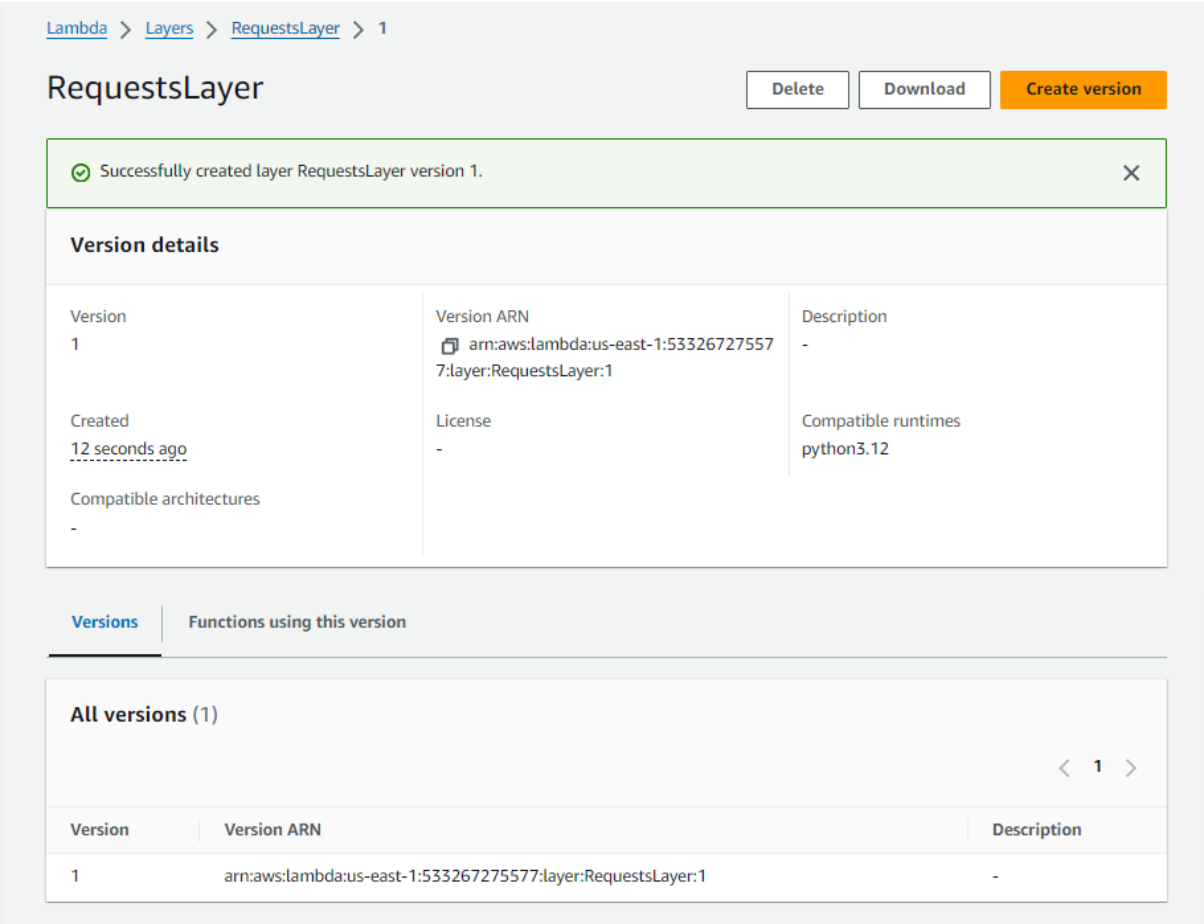
- Choose **Create layer**.



- 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.

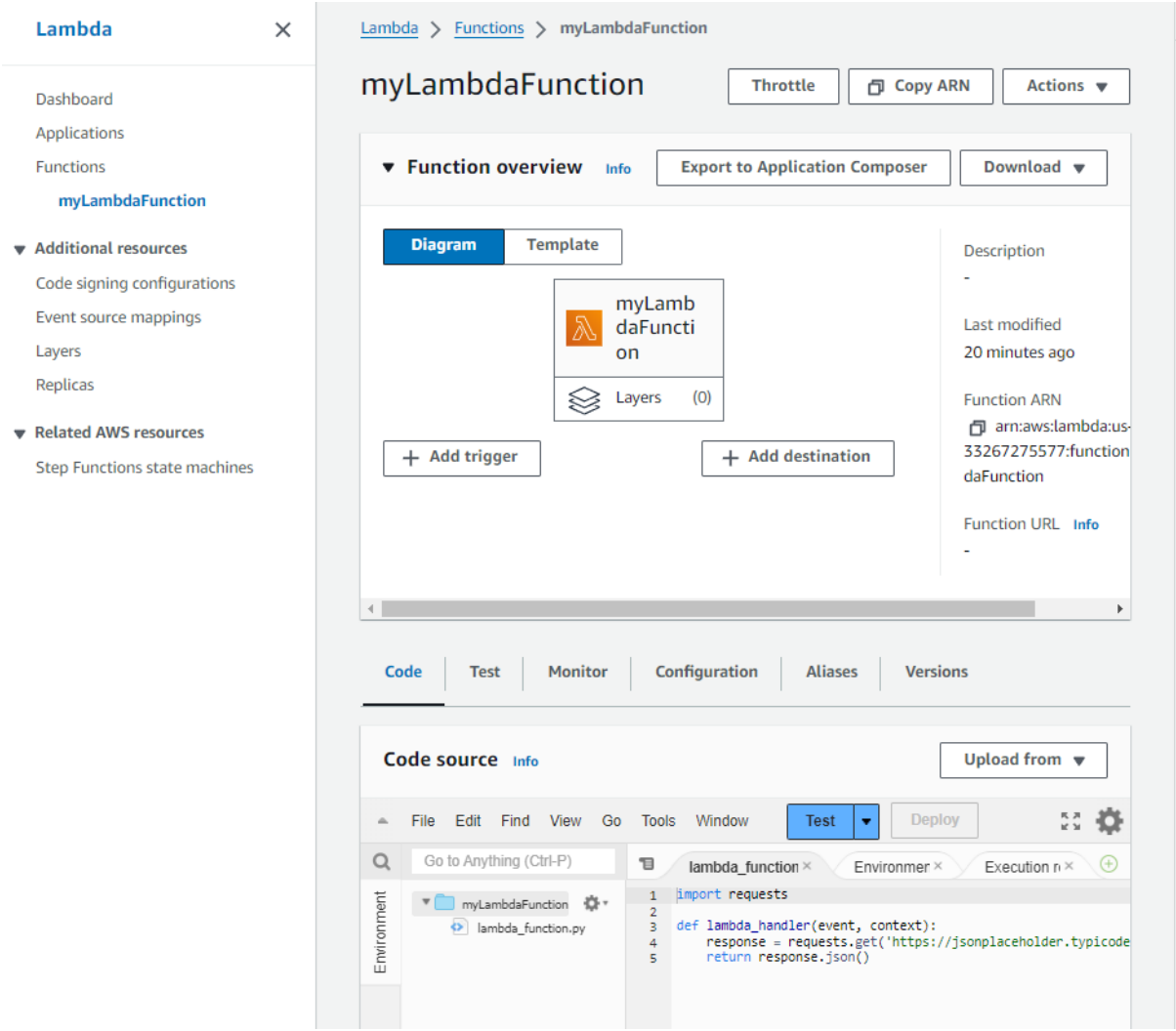


- Click **Create**.

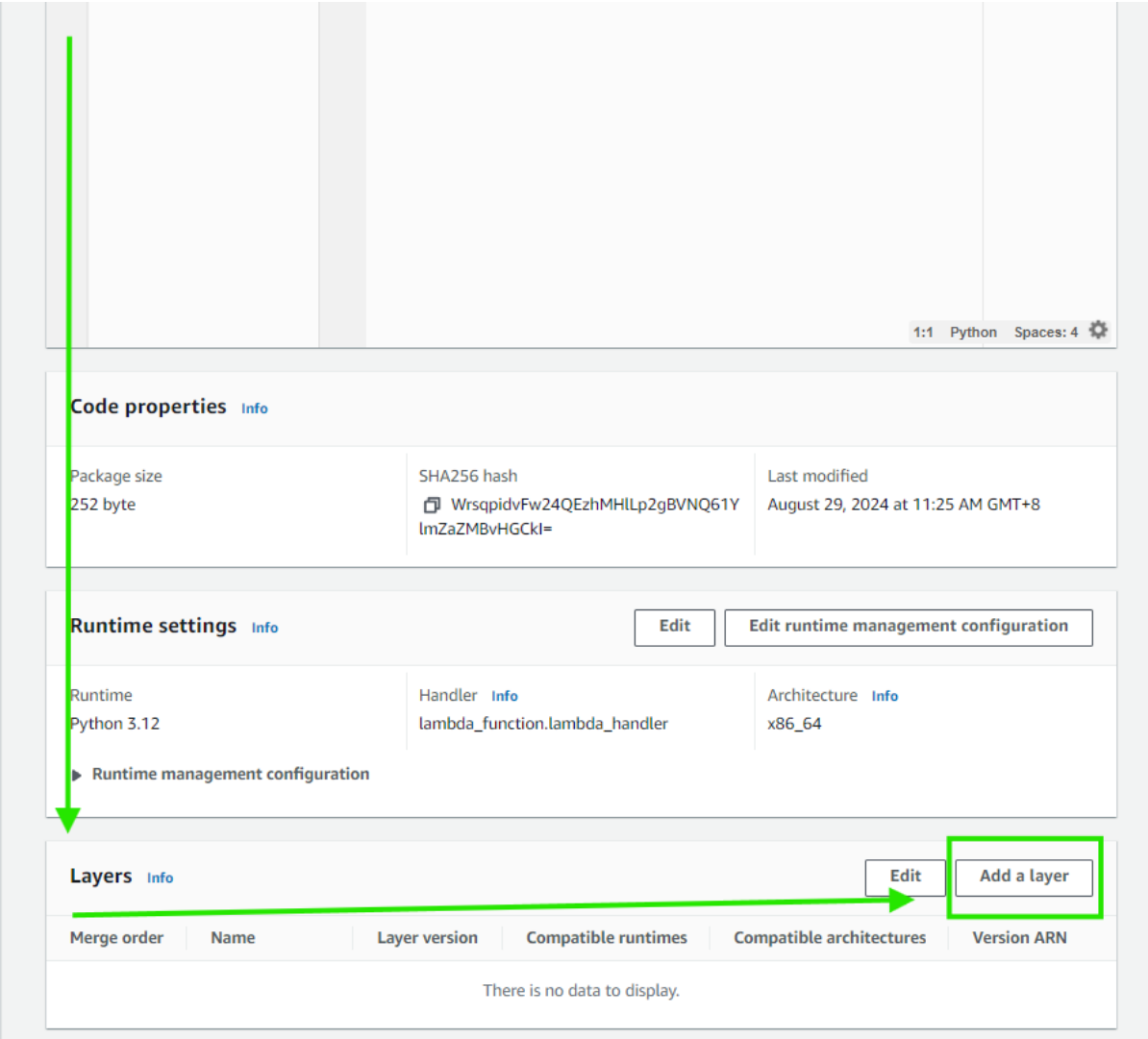


3. **Add the Layer to Your Lambda Function:**

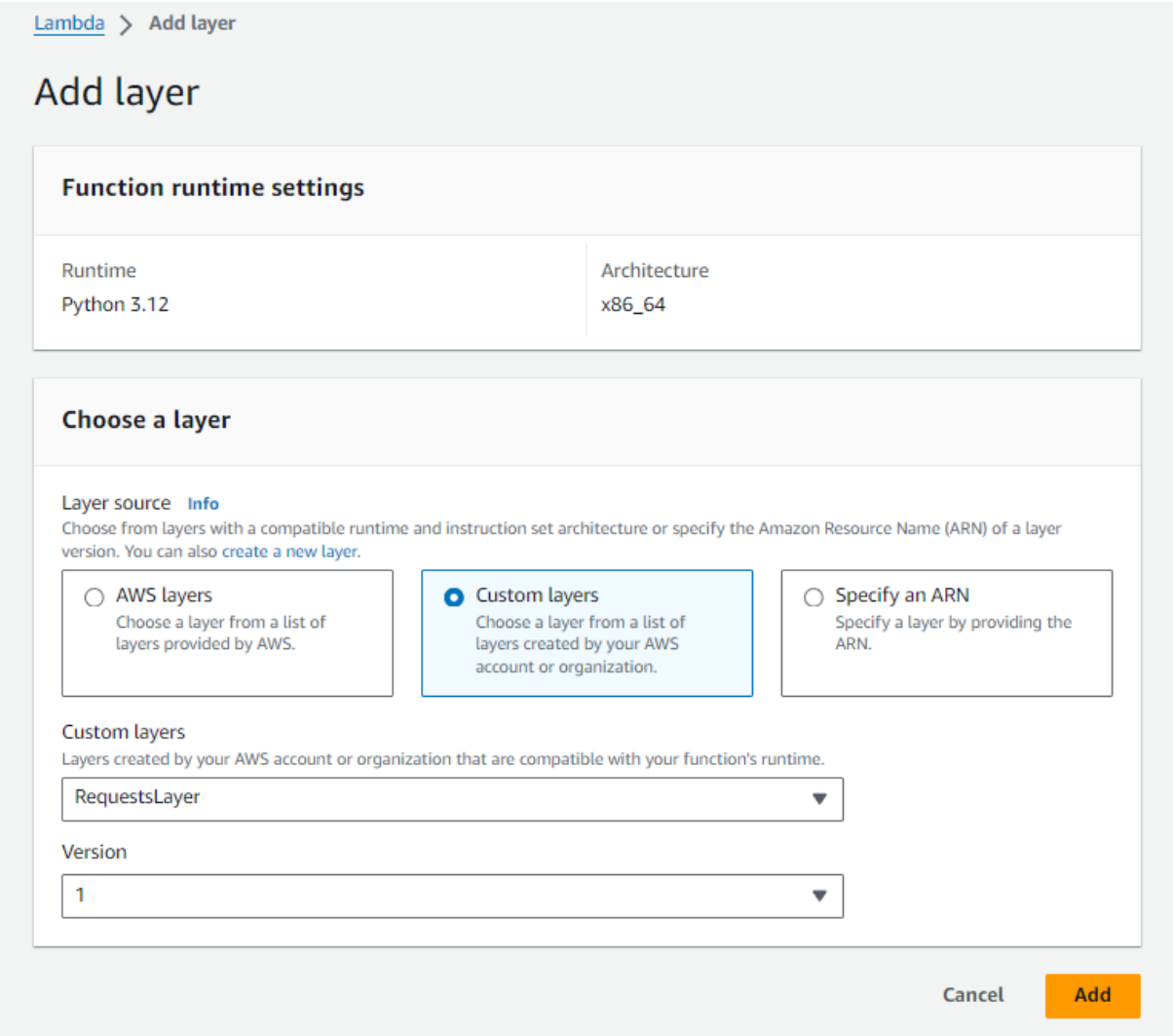
- Go to your myLambdaFunction function.



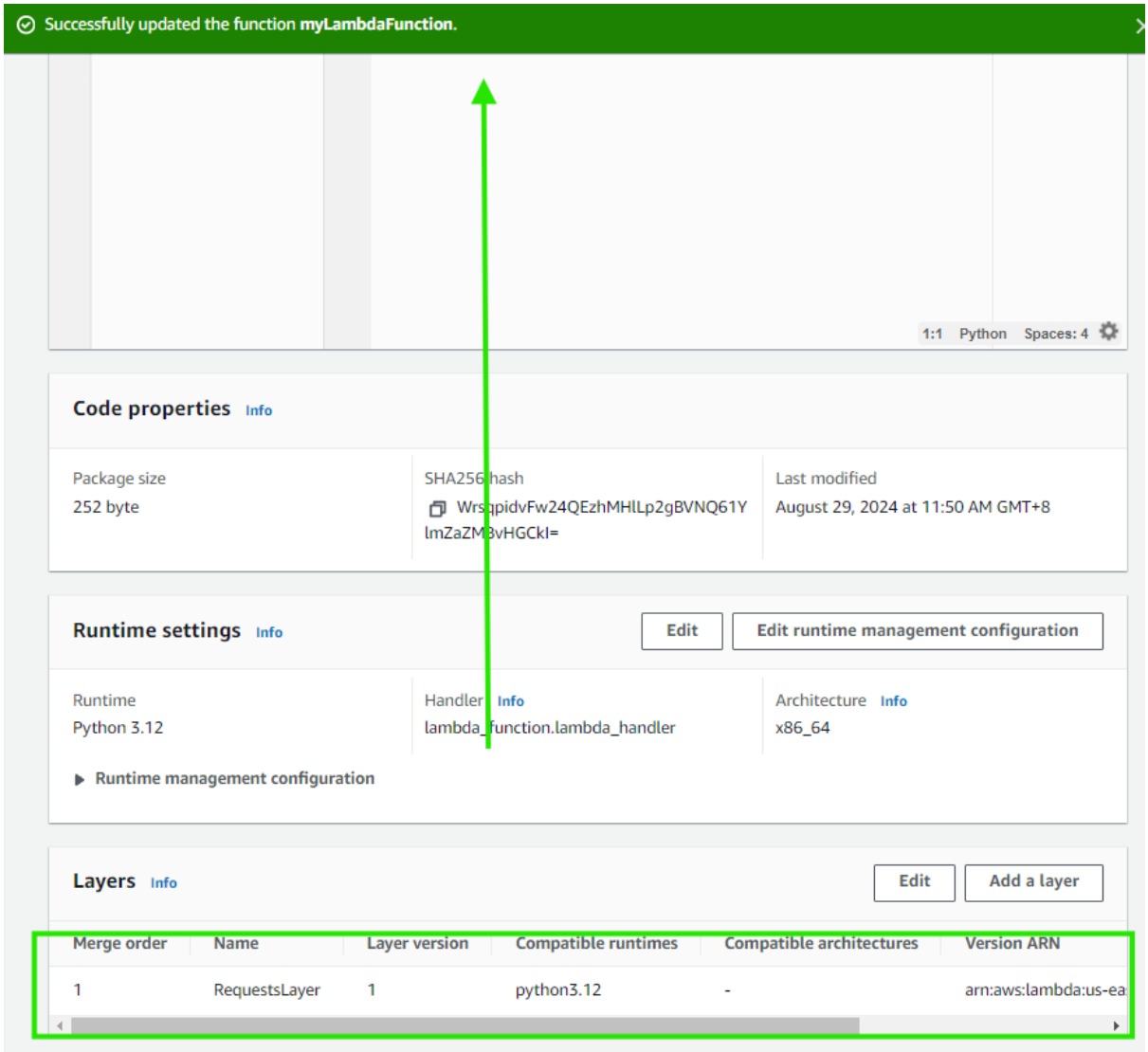
- Scroll down from the Code tab of the console. In the **Layers** section, click **Add a layer**.



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

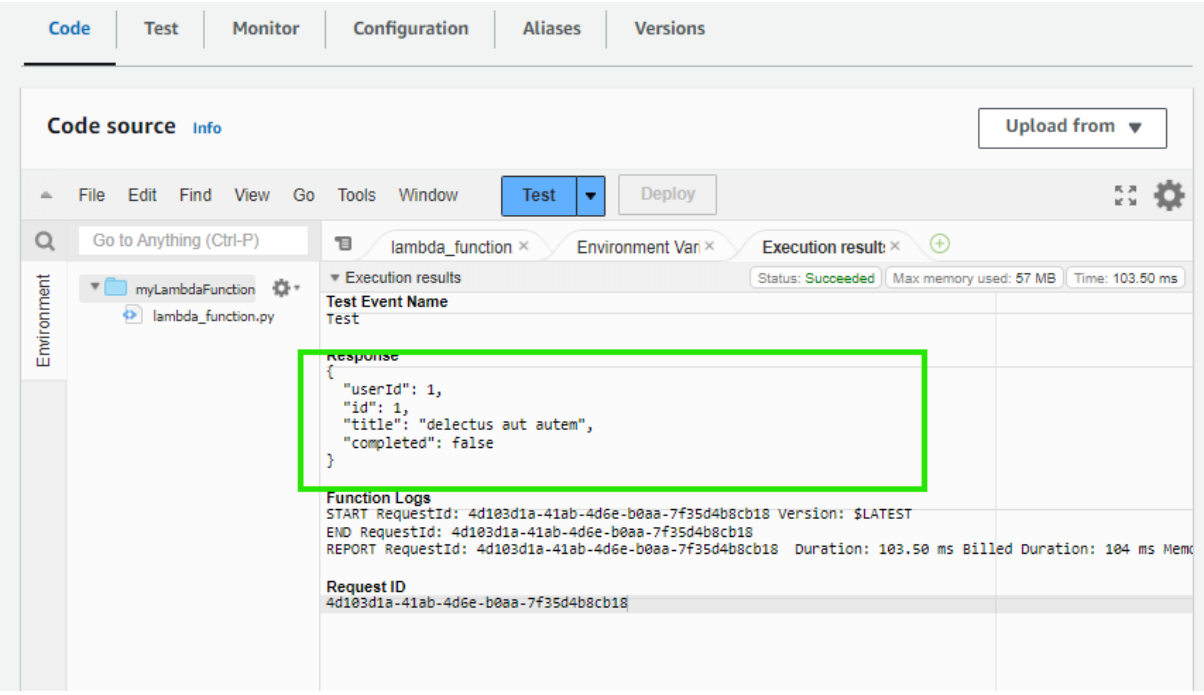


- Click **Add**. You will notice the Lambda Function will update, and you should also see the RequestLayer added to the Layer section. Just wait for it to finish.



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!