### **What is an AWS Lambda Layer?**

AWS Lambda Layers allow you to manage and share dependencies across multiple Lambda functions. Instead of packaging external libraries with every function, you can create a layer and attach it to multiple functions, reducing deployment size and complexity.

### **Why Use an AWS Lambda Layer?**

* Reusability: Share common libraries across multiple Lambda functions.
* Reduces deployment package size.
* Simplifies function updates by updating the layer instead of modifying each function.

### **Steps to Set Up an AWS Lambda Layer**

#### **1. Prepare Dependencies for the Layer**

If you need to include external libraries (e.g., psycopg2, requests), install them in a specific directory structure.

| mkdir python pip install psycopg2-binary requests -t python/ |
| --- |

This installs the required Python packages inside a python/ directory, which AWS Lambda recognizes as a layer structure.

#### **2. Package the Layer**

Create a .zip archive of the python/ directory:

| zip -r layer.zip python/ |
| --- |

#### **3. Upload the Layer to AWS Lambda**

1. Go to the **AWS Lambda Console**.
2. Navigate to **Layers** on the left panel.
3. Click **Create layer**.
4. Provide a name (e.g., MyPythonLayer).
5. Upload the layer.zip file.
6. Select the appropriate **runtime** (e.g., Python 3.x).
7. Click **Create**.

#### **4. Attach the Layer to Your Lambda Function**

1. Open your **AWS Lambda function**.
2. Scroll down to the **Layers** section.
3. Click **Add a layer**.
4. Select **Custom layers** and choose the layer you created.
5. Click **Add**.

#### **5. Verify the Layer in Your Lambda Code**

In your Lambda function, import the installed dependencies:

| import psycopg2 import requests  def lambda\_handler(event, context):  print("AWS Lambda Layer is working!") |
| --- |

#### **6. Test the Lambda Function**

* Deploy the function.
* Invoke a test event to confirm that the dependencies load correctly.