# Create the AWS Lambda Function

This Lambda function will be the core of our image processing pipeline. It will:

- Trigger on new image uploads: Specifically, when an image is placed in our input S3 bucket.
- Resize/compress: It will contain the logic to transform the image (in our case, resize it).
- Save the processed image: The output will be stored in our designated output S3 bucket.

Let's build it step-by-step:

**Create the Lambda Function Skeleton**

1. Go to the Lambda Console: Navigate to the AWS Lambda service in your console.
2. Initiate Function Creation: Click the "Create function" button.
3. Configure Basic Settings:
    - "Author from scratch": Select this option.
    - "Function name": Enter ImageProcessorFunction
    - "Runtime": Choose Python 3.12
    - "Permissions": This is where we link our IAM role!
        - Expand the "Change default execution role" section.
        - Choose "Use an existing role".
        - From the dropdown, select your previously created role: lambda-image-processing-role.
4. Create the Function: Click "Create function".

**Add S3 Trigger to the Lambda Function**

Now that the function exists, we need to tell it when to run. We'll configure an S3 trigger so it's invoked every time a new image lands in our input bucket.

1. Access Your Lambda Function: If you're not already there, go back to the Lambda Console and open your newly created function: ImageProcessorFunction.
2. Add a New Trigger:
    - In the "Function overview" section, click on "Add trigger".
3. Configure the S3 Trigger:
    - "Select a trigger": Choose "S3".
    - "Bucket": Select your input bucket from the dropdown: image-processing-input. (Make sure you pick the correct one!)
    - "Event type": Choose "All objects create events" (or specifically "Put" for new uploads).
    - "Prefix / Suffix": These are optional but highly recommended for efficiency.
        - For "Suffix": Enter .jpg (or .png, .jpeg, etc., separated by | if you want multiple). This ensures your Lambda only triggers for actual image files, saving unnecessary invocations.
    - "Enable trigger": Make sure this is checked.
4. Finalize Trigger Addition: Click "Add".

**Upload Image Processing Code (with Pillow) via ZIP**

Lambda environments are minimal by default. For image manipulation, we need libraries like Pillow, which aren't built-in. This means we'll create a "deployment package" – a ZIP file containing our Lambda code and all its dependencies (Pillow in this case).

What our Lambda will do:

- Receive a trigger event from the S3 input bucket.
- Download the newly uploaded image.
- Resize it to 128x128 pixels.

- Upload the resized image to the S3 output bucket.

Required Files:

1. lambda_function.py: This is your core Lambda handler script.

```
from PIL import Image
import boto3
import os
import json

s3 = boto3.client('s3')

def lambda_handler(event, context):
    print("Lambda triggered with event:")
    print(json.dumps(event))

    try:
        # Extract bucket and key from the S3 event
        input_bucket = event['Records'][0]['s3']['bucket']['name']
        input_key = event['Records'][0]['s3']['object']['key']

        # Get the output bucket name from environment variables
        # This is important for flexibility and security
        output_bucket = os.environ.get('OUTPUT_BUCKET')
        if not output_bucket:
            raise ValueError("OUTPUT_BUCKET environment variable is not set.")

        # Define the key for the processed image in the output bucket
        output_key = f"resized-{input_key}"

        print(f"Downloading from bucket: {input_bucket}, key: {input_key}")
        # Download the image to the Lambda's temporary storage
        s3.download_file(input_bucket, input_key, '/tmp/input.jpg')

        print("Opening and resizing image...")
```

```
        # Open, resize, and save the image
        img = Image.open('/tmp/input.jpg')
        img = img.resize((128, 128)) # Resizing to 128x128
        img.save('/tmp/output.jpg')

        print(f"Uploading to bucket: {output_bucket}, key: {output_key}")
        # Upload the processed image to the output bucket
        s3.upload_file('/tmp/output.jpg', output_bucket, output_key)

        print("Process completed successfully.")
        return {"status": "Image resized and uploaded."}

    except Exception as e:
        print(f"Error occurred: {str(e)}")
        # Re-raise the exception for CloudWatch logging
        raise e
```

Important Note: The code uses os.environ['OUTPUT_BUCKET']. This means we'll need to set an environment variable in your Lambda function configuration named OUTPUT_BUCKET with the value image-processing-output. We'll do this after uploading the code.

**Step-by-Step to Create the Deployment Package using Docker:**

Using Docker is the most reliable way to create a Lambda deployment package with specific dependencies like Pillow, ensuring compatibility with the Lambda execution environment.

1. Create Project Directory: Open your terminal or command prompt and run:

   ```
   mkdir lambda-pillow
   cd lambda-pillow
   ```
2. Create Your lambda_function.py: Inside the lambda-pillow directory, create a file named lambda_function.py and paste the Python code provided above into it.
3. Create Dockerfile: In the same lambda-pillow directory, create a file named Dockerfile (no extension) and paste the following content:

```
FROM python:3.12-slim
RUN apt-get update && apt-get install -y zip
WORKDIR /package
# Install Pillow here (NOT in /app)
RUN pip install Pillow -t .
# Now copy your lambda handler
COPY lambda_function.py .
CMD ["bash"]
```

4. Build the Package in Docker: Make sure you have Docker Desktop running. In your terminal, from within the lambda-pillow directory, run:

   ```
   docker build -t lambda-packager .
   ```

This command builds a Docker image that contains Python 3.12, Pillow, and your Lambda code.

5. Zip Your Package: Now, we'll use the Docker image to create the ZIP file.
   - For Windows (using Command Prompt or PowerShell):
     ```
     docker run --rm -v "%cd%:/output" lambda-packager bash -c "cd /package && zip -r9 /output/lambda-pillow.zip ."
     ```
   - For macOS/Linux (using Bash):
     ```
     docker run --rm -v "$(pwd):/output" lambda-packager bash -c "cd /package && zip -r9 /output/lambda-pillow.zip ."
     ```
6. This command runs the Docker container, zips up the Pillow library and your lambda_function.py into lambda-pillow.zip, and places it in your current lambda-pillow directory.

7. Check Contents (Optional but Recommended): You can inspect the contents of the ZIP file to ensure everything is there:

   - For macOS/Linux:

     ```
     tar -tvf lambda-pillow.zip
     ```

   - For Windows (you might need a tool like 7-Zip or just open the .zip file directly): You can simply navigate to your lambda-pillow directory in File Explorer and open lambda-pillow.zip to confirm lambda_function.py and the PIL directory (from Pillow) are present.