# **AWS Lambda Function Creation & Deployment in Github Copilot**

## **Introduction**

## In this exercise, you'll learn how to create and deploy an AWS Lambda function using GitHub Copilot. This hands-on guide covers setting up a project structure, coding your Lambda function, writing tests, creating deployment templates, and integrating a CI/CD pipeline with GitHub Actions.

## **Objectives**

## **Setup a Project:** Initialize a new GitHub repository and structure your project for AWS Lambda development.

## **Develop Lambda Function:** Create a Python-based Lambda function to process images from S3, extract metadata, and store it in DynamoDB.

## **Testing & CI/CD:** Implement tests using Pytest with mocked AWS services and automate deployments with GitHub Actions.

## **Documentation & Best Practices:** Generate comprehensive documentation and follow best practices for error handling, logging, and deployment.

## **Prerequisites**

* GitHub account with Copilot subscription
* AWS account with appropriate permissions
* AWS CLI installed and configured
* Pythoninstalled (for this example)
* Visual Studio Code with GitHub Copilot extension installed

|  |
| --- |
| **Note**: Below prompts are provided solely as examples of effective prompt construction. As participants, we must engage in an iterative process to refine these examples and develop the optimal prompt for the task at hand. |

## **Step 1: Project Initialization**

Begin by instructing Copilot Chat to create your project scaffold. Instead of manually creating files, you will send a prompt that creates a shell script to set up your repository structure.

Start with this prompt:

|  |
| --- |
| Create a file named project\_setup.sh that initializes an AWS Lambda project. The script should: - Create directories: src/lambda, tests, deployment. - Follow Python best practices. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

Since this basic command might yield a minimal scaffold, refine it by sending:

|  |
| --- |
| Update project\_setup.sh to also create a virtual environment folder (venv) and a docs folder for documentation. |

Then further refine:

|  |
| --- |
| Update project\_setup.sh so that it creates directories: src/lambda, tests, deployment, docs. Also, in the project root, generate empty files named README.md and requirements.txt. Ensure all directory names are in lowercase and follow Python best practices. |

All file and folder creation is done through Copilot Chat, so no manual file creation is required.

## **Step 2: Create the Lambda Handler**

Next, instruct Copilot Chat to generate the Lambda handler code automatically. Ask:

|  |
| --- |
| Create a file named handler.py in the src/lambda directory. Write a Python AWS Lambda function that: 1. Processes images uploaded to S3. 2. Extracts metadata from the image. 3. Stores the metadata in a DynamoDB table. 4. Returns a success or error response. Include basic error handling and logging. |

To improve the quality, refine the prompt:

|  |
| --- |
| Update handler.py: Enhance the function to use Python's logging module for detailed logs and include try-except blocks for S3 and DynamoDB exceptions. |

Then refine further:

|  |
| --- |
| Further update handler.py by adding inline comments for each step and modularizing error handling into helper functions. |

Finally, send:

|  |
| --- |
| Write a complete Python Lambda function in handler.py that processes S3 image uploads by extracting metadata and storing it in DynamoDB. Ensure detailed logging (using Python's logging module), modular error handling via helper functions, inline comments for clarity, and robust exception handling for AWS service errors. |

All the code in handler.py is generated solely by Copilot Chat.

## **Step 3: Define the AWS SAM Template**

Instruct Copilot Chat to create the AWS SAM template file. Use this prompt:

|  |
| --- |
| Create a file named template.yaml in the project root. Generate an AWS SAM YAML template that: - Defines a Lambda function using Python 3.11 triggered by an S3 bucket upload. - Includes a DynamoDB table for metadata storage. - Specifies IAM roles and permissions for the Lambda to access S3 and DynamoDB. |

Then refine:

|  |
| --- |
| Update template.yaml to include descriptions for each resource, proper indentation, and resource properties for timeouts, memory size, and environment variables. |

Finally, send:

|  |
| --- |
| Create a comprehensive AWS SAM YAML template in template.yaml that defines a Python 3.11 Lambda function triggered by an S3 bucket upload, a DynamoDB table for metadata storage, detailed resource properties (timeouts, memory size), and IAM roles with specific permissions and descriptions. |

Copilot Chat generates the complete template.yaml file automatically.

## **Step 4: Implement Testing with Pytest and Moto**

Now instruct Copilot Chat to generate tests. Use this prompt:

|  |
| --- |
| Create a file named test\_handler.py in the tests directory. Write a Pytest suite for the AWS Lambda image processing function. Use Moto to mock S3 and DynamoDB, set up fixtures for AWS credentials, and test both successful processing and error scenarios. |

Then refine the prompt:

|  |
| --- |
| Update test\_handler.py: Enhance the tests by adding cases for edge scenarios like invalid image formats and missing metadata. |

Further refine:

|  |
| --- |
| Further update test\_handler.py to refine fixtures for proper setup and teardown of mocked AWS resources and add assertions to verify logging outputs. |

Final prompt:

|  |
| --- |
| Write a comprehensive Pytest file in test\_handler.py using Moto to mock S3 and DynamoDB. Include detailed fixtures for AWS credentials and resource management, and test both successful image processing and various error scenarios with clear assertions. |

Copilot Chat will generate all testing code in test\_handler.py without manual intervention.

## **Step 5: Set Up GitHub Actions for CI/CD**

Next, instruct Copilot Chat to create the CI/CD workflow. Use this prompt:

|  |
| --- |
| Create a file named deploy.yml in the .github/workflows directory. Write a GitHub Actions workflow that: - Triggers on push to the main branch. - Installs Python dependencies. - Runs Pytest and generates a coverage report. - Builds and deploys the Lambda using AWS SAM if tests pass. - Uses GitHub secrets for AWS credentials. |

Refine by adding:

|  |
| --- |
| Update deploy.yml: Add steps to cache Python dependencies and set up the AWS CLI. |

Then refine further:

|  |
| --- |
| Further update deploy.yml to include conditional steps that prevent deployment if tests fail and add notifications on failure. |

Final prompt:

|  |
| --- |
| Create a robust GitHub Actions workflow in deploy.yml that triggers on push to the main branch. The workflow should cache Python dependencies, set up the AWS CLI using GitHub secrets, run Pytest with coverage reporting, and conditionally build and deploy the Lambda using AWS SAM only if tests pass. Include notification steps for any failures. |

Copilot Chat handles all file generation for the CI/CD workflow.

## **Step 6: Define Python Dependencies**

Instruct Copilot Chat to generate your dependencies file. Use this prompt:

|  |
| --- |
| Create a file named requirements.txt in the project root. List all required libraries for the AWS Lambda project, including boto3, Pillow, loguru, pytest, and moto. |

Then refine:

|  |
| --- |
| Update requirements.txt to specify version numbers for each library and include any additional dependencies needed for AWS SAM deployment. |

Final prompt:

|  |
| --- |
| Create a requirements.txt file that includes boto3, Pillow, loguru, pytest, moto, and any additional libraries required for AWS SAM deployments, with specific version numbers for reproducibility. |

Copilot Chat will output the complete requirements.txt file.

## **Step 7: Generate README Documentation**

Now, instruct Copilot Chat to generate the project documentation. Use this prompt:

|  |
| --- |
| Create a file named README.md in the project root. Generate a README for the AWS Lambda image processing project that includes: - Project Overview - An architecture diagram explaining the interactions between S3, Lambda, and DynamoDB - Project structure explanation - Local development setup instructions - Testing guidelines - Deployment process using AWS SAM and GitHub Actions - A description of the AWS resources and their roles |

Then refine:

|  |
| --- |
| Update README.md to expand the architecture section with a detailed diagram and add troubleshooting tips and best practices. |

Final prompt:

|  |
| --- |
| Generate a detailed README.md that includes an overview, a clear architecture diagram with explanations, comprehensive instructions for project structure, local setup, testing, and deployment using AWS SAM and GitHub Actions, as well as a thorough description of the AWS resources, troubleshooting tips, and best practices. |

Copilot Chat will generate the final README.md file.

## **Step 8: Local Testing Setup**

Instruct Copilot Chat to create a local testing script. Use this prompt:

|  |
| --- |
| Create a file named local\_test.py in the project root. Write a Python script that: - Sets up local environment variables. - Simulates a sample S3 event. - Invokes the Lambda handler. - Prints the function response. |

Then refine:

|  |
| --- |
| Update local\_test.py to add robust error handling that catches exceptions and logs them for debugging. |

Next, refine:

|  |
| --- |
| Further update local\_test.py to include command-line argument parsing so that different test events can be triggered easily. |

Final prompt:

|  |
| --- |
| Write a Python script in local\_test.py that sets local environment variables, simulates an S3 event, invokes the Lambda handler, and prints the response. The script should include robust error handling and allow triggering different test events via command-line arguments. |

Copilot Chat automatically generates the complete local\_test.py script.

## **Step 9: Execute and Deploy the Complete Project Using Copilot Only**

Finally, instruct Copilot Chat to integrate all files and deploy the project entirely via CI/CD. First, create an integration script:

|  |
| --- |
| Create a file named integrate\_and\_test.sh in the project root. Write a script that: - Lists the project directory structure to verify that all files (project\_setup.sh, handler.py, template.yaml, test\_handler.py, deploy.yml, requirements.txt, README.md, and local\_test.py) are in place. - Runs local\_test.py with various simulated S3 events. - Logs all results and reports any errors clearly. |

Then refine:

|  |
| --- |
| Update integrate\_and\_test.sh to include commands that display the directory structure and capture outputs from local\_test.py for review. |

Final prompt:

|  |
| --- |
| Write a comprehensive integration script in integrate\_and\_test.sh that verifies the directory structure, runs local\_test.py with multiple simulated S3 events, and logs all outputs with clear error reporting. |

After integration is verified locally, send a final deployment prompt:

|  |
| --- |
| Commit all changes and push to the main branch so that the GitHub Actions workflow is triggered. Confirm that the CI/CD pipeline runs the tests, builds the project using AWS SAM, and deploys the Lambda function to AWS. Provide a summary of the deployment status once completed. |

This final instruction directs Copilot Chat to simulate the commit/push process. The pre-configured GitHub Actions workflow will automatically execute, running tests, building the project, and deploying the Lambda function to AWS—all fully automated through Copilot Chat.

# **Conclusion**

By following this exercise, you've set up an end-to-end project that covers AWS Lambda development, testing, deployment, and CI/CD integration using GitHub Copilot. This exercise not only demonstrates the power of automation tools like Copilot but also instills best practices for cloud-native development. As a next step, consider extending the functionality of your Lambda, improving security measures, and exploring additional AWS services to enhance your application's robustness.