Homework Assignment #4

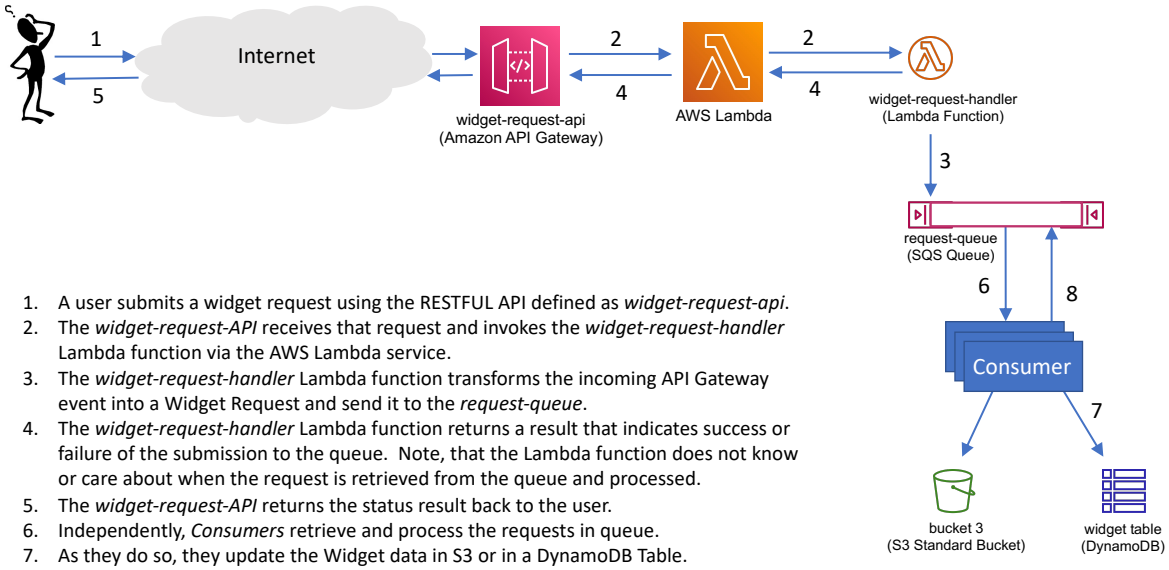
Estimated time: 240 – 360 minutes

Objectives

- Gain some experience with Lambda functions
- Gain some experience with API Gateways

Overview

In this assignment, you will replace the Producer program in the Widget application with a Lambda function that handles RESTful requests that come through a Gateway API, transforms them into Widget Requests, and places them into the request queue used by your Consumer.



1. A user submits a widget request using the RESTFUL API defined as *widget-request-api*.
2. The *widget-request-API* receives that request and invokes the *widget-request-handler* Lambda function via the AWS Lambda service.
3. The *widget-request-handler* Lambda function transforms the incoming API Gateway event into a Widget Request and send it to the *request-queue*.
4. The *widget-request-handler* Lambda function returns a result that indicates success or failure of the submission to the queue. Note, that the Lambda function does not know or care about when the request is retrieved from the queue and processed.
5. The *widget-request-API* returns the status result back to the user.
6. Independently, *Consumers* retrieve and process the requests in queue.
7. As they do so, they update the Widget data in S3 or in a DynamoDB Table.
8. After successfully processing a request, the Consumer deletes the request from the queue.

Figure 1 – Submitting Widget Requests via an API Gateway and Lambda function.

See Figure #1.

You will build the *widget-request-handler* lambda function and set up the *widget-request*-api. You do not need to change your *request-queue* or *Consumer* from HW3. However, if feel a need to improve your Consumer program or change the structure of the incoming requests, you may do so.

Instructions

Step 1 – Design your API and Widget Request Handler

In this step, you will design the details of our RESTful API and your Lambda function. Here are some things to consider:

- Keep your design simple, easy to test, maintainable, and scalable.
- The only purpose of this API is to allow client (like a mobile app) to submit a Widget Request. So, you only needed one type of HTTP Method, namely a POST.
- Consider passing the JSON serialization of a Widget Request, i.e., a JSON string, as the body of the POST method request.
- The Lambda function must accept an incoming Widget Request event (from the API Gateway), to validate the request, and to place it in the queue.
- For your Lambda function, keep the actual event handler function a simple as possible. Put most of your logic in other helper functions or classes and methods. This will enable you to test most of your logic independent of calling the event handler.
- Follow the principles for good software design, e.g., good abstraction, modularity, and encapsulation. These three general principles encompass many of others that you may have be familiar with, such as low coupling, high cohesion, and the SOLID principles.

Document your design as submit it with your other deliverables for this assignment.

Step 2 – Implement and Test Your Widget Request Handler

In this step, your goal is to implement your *widget-request-handler* logic and test it as thoroughly as possible using executable unit test cases. You may do this using any programming language and runtime supported by AWS Lambda. You may use an IDE on your own workstation, or you may use Cloud9.

You must manage your work artifacts with Git and do frequent commits with meaning messages. Your Git history will be examined during the review and must show meaningful workflow. A single commit at the end is not acceptable.

Also, you will need to take a snapshot of your Git repository's commit log or print it to a PDF and submit that artifact with your assignment.

Here are some suggestions.

- Keep it simple – don't implement extraneous features.
- Make sure your executable test cases are as thorough as possible.
- Output log messages that will help you debug your code.

Step 3 – Deploy your Lambda Function

Using any mechanism that you desire, including the AWS Lambda Console, deploy your *widget-request-handler* function.

Then, test it in the AWS Lambda Console with test API Gateway Events that simulate create, update, and delete Widget Requests coming in from an API. The bodies of these test events need to be consistent with your design. Save these test events with the Lambda function in the AWS Lambda Console. Also, copy-n-paste them into files in your project, so they get included in your Git Repository.

Take three or more screen snapshots that illustrate your Lambda function running successfully using these test events.

Step 4 – Create and Test a RESTful API

Using any mechanism of your choice, create a RESTful API that will be the front-end for your widget-request-handler Lambda function.

Then, test the API in the AWS API Gateway Console using the appropriate HTTP methods and even bodies. Your testing of the API should try method requests that submit create, update, and delete requests.

Take three or more screen snapshots that illustrate your API processing create, update, and delete requests.

Step 5 – Deploy your API and Additional Testing

In this step, deploy your API to a production, labeled "prod" stage and test using a generic HTTP client, like Postman.

Take three or more screen snapshots that illustrate your API processing create, update, and delete requests.

Submission

Your submission must include

- All the screens mentioned in the above steps
- Your revised design document
- An archive file of your entire project, expect build-time or runtime-artifacts

Grading Criteria

Criteria	Points
A revised design document that satisfies the new requirements and is consistent with the implementation	15
An implementation that satisfies the requirements and design, and that includes meaningful executable unit test cases. Also, snapshot of your Git repository's history that illustrate regular progress and commits.	25
Snapshots that illustrate successful deployment and testing of the Lambda function.	30
Snapshots that illustrate successful creation and testing of a RESTful API.	20
Snapshots that illustrate successful deploy and additional testing of a RESTful API.	10
Deductions (applied on top of scores given for the above): • Late (-20 points per day up to -50 points) • Failure to complete a design and code review (-50) • Sloppy presentation of the comparison and questions/answers (up to -25) • Cheating, e.g., copying someone else's work. This will be an automatic -200 (i.e., a negative twice the maximum points).	-200 to 0
Max Points	**100**