

Inventory App Project

Table of Contents

Project Overview.....	2
GitHUB.....	3
S3.....	5
Lambda.....	5
DynamoDB.....	6
API Gateway.....	7

Project Overview

In this group project, you'll tie together major concepts from the course to build a cloud based inventory application. The inventory app will consist of a DynamoDB table, API Gateway and Lambda functions, similar to the ones you've already made in the course. The user will interact with the inventory app using Postman or Restfox, there is no web UI required for this project, although you can create one for bonus points. You'll use GitHub to manage the lambda function code and sample web application code as a team. Additionally, you'll use Github actions to implement some basic CI/CD automation for this project. You'll implement GitHub actions on specific events, to automatically deploy your function code to AWS Lambda, automatically deploy the sample web application code to AWS S3, and automatically check (lint) your code for syntax and formatting issues. You'll demonstrate your completed application to the instructor and walk them through each rubric item.

Once you've completed the project development, you'll need to test that the entire system works when you commit changes to the "dev" branch by completing pull requests & peer reviews, and also when you merge changes into the "main" branch by completing automated deployment. Your team's solution should be well tested and fully functional when you demonstrate it for the instructor.

GitHUB

- Select one GitHub account from your team members and create a repository for this project called: **InventoryApp**.
- Invite your team members as collaborators in this repository.
- Add your Lambda functions & website demo code into a “dev” branch for development.
 - The website code does not need to connect to the API, it can simply be a bare bones HTML “hello world” just to prove your S3 bucket is serving it correctly. See **index.html** file attached to the Dropbox for an example. Your web browser should be able to display this example code successfully.
 - Your website code should be named: **index.html**
 - The lambda functions should be fully functional, like the ones demonstrated in class, and need to be named exactly as described in the Lambda section.
- After any team member has committed production ready code to the “dev” branch, create a pull request that requires review from the other team members. Once the review has been completed, merge the “dev” changes back into the “main” branch.
 - Follow the basic peer review workflow from [here](#).
- Add an action that uses the “superlinter” functionality, *every time a pull request is opened or re-opened*.
 - Follow the basic GitHub Action steps from [here](#).
 - Follow the same steps as assignment 12.
 - Use the GitHub action code attached to the Dropbox as a reference and modify as required.
- Add an action that deploys your function code to AWS Lambda, *every time a merge is made with the “main” branch*. This action should only run when any of the 5 python function files have been committed.
 - Follow the basic GitHub Action steps from [here](#) and/or [here](#).

- Use the GitHub action code attached to the Dropbox as a reference and modify as required.
- Put each .py file in its own folder named the same as the function name, and use the name: ***lambda_function.py*** for the actual Python file.
 - i.e. lambda/get_all_inventory_items/lambda_function.py
- Make sure to create each function in AWS Lambda first, so that your GitHub action only needs to ***update*** the function code, not create the function.
- Please note: You can find the login credentials (Access key, secret access key & session token) for your AWS learner lab by clicking the "AWS Details" button at the top right of the learner lab launch page, next to the "End Lab" button. These credentials will change every time you start a new learner lab session, so you'll need to update your GitHub secrets each time. You don't need to create any policies or users for this project, just use the AWS Academy LabRole as usual.
- Add an action that deploys your demo web application code (***index.html***) to AWS S3, every time a merge is made with the “main” branch. This action should only run when ***index.html*** has been committed. Ensure you can access the website from S3.
 - Follow the basic GitHub Action steps from [here](#).
 - Use the GitHub action code attached to the Dropbox as a reference and modify as required.
 - Please note: You can find the login credentials (Access key, secret access key & session token) for your AWS learner lab by clicking the "AWS Details" button at the top right of the learner lab launch page, next to the "End Lab" button. These credentials will change every time you start a new learner lab session, so you'll need to update your GitHub secrets each time. You don't need to create any policies or users for this project, just use the AWS Academy LabRole as usual.

S3

Create an S3 bucket with a globally unique name, configured for hosting web pages.

Verify you can load web pages from your bucket using [HTTP](#). Once your S3 bucket is configured for hosting, add a CloudFront distribution and verify you can load web pages from your bucket using [HTTPS](#).

Lambda

- Lambda Function Names
 - [*get_all_inventory_items*](#)
 - [*get_inventory_item*](#)
 - [*add_inventory_item*](#)
 - [*delete_inventory_item*](#)
 - [*get_location_inventory_items*](#)

You can use the functions from the lecture slides, demonstrated in class, as a template for your inventory functions. Test your function code with the appropriate sample event data to ensure it works as expected.

DynamoDB

Create a DynamoDB table named: ***Inventory***, and configure with attributes as shown below. Design the table in NoSQL Workbench, then push your design to AWS Academy from there.

- Inventory table structure
 - Item ***id*** (PK, string, using the ULID module)
 - Item ***name*** (string)
 - Item ***description*** (string)
 - Item ***qty*** on hand (integer)
 - Item ***price*** (float)
 - Item ***location_id*** (SK, integer)
 - GSI
 - Reverse PK & SK so you can easily query what inventory is in each location.

API Gateway

Create an API on API Gateway and configure the endpoints as shown below. Test all the endpoints with Postman or Restfox to validate that the API calls the correct functions and is able to interact with the database successfully.

- API endpoints
 - *{ url }/item*
 - GET all items
 - *{ url }/item/<id>*
 - GET specific item
 - *{ url }/item*
 - POST new item
 - *{ url }/item/<id>*
 - DELETE specific item
 - *{ url }/location/<id>*
 - GET all items in specific location