Event-Driven Architecture Assignment

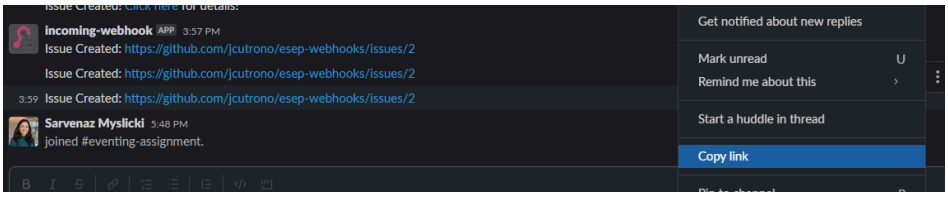
Table of Contents

Overview

- Event-driven architectures embody asynchronous communication between applications. Unlike applications that are self-contained and write directly to a datastore, eventing can introduce additional complexity.
- Experiencing these issues will help develop best practices and defensive programming skills.
- For submission: You will submit the link to your **PUBLIC** GitHub repo. We will use this for automation to submit an issue to each repo.

Grading (6 points total)

- **Full points** will be awarded for successfully getting the two integrations tied together and posting via the webhook integration.
    - **Submit a link to your GitHub repo in Canvas**
- **Half points** awarded for being able to trigger the GitHub webhook when an issue is submitted manually.
    - *This is only if you can't get the full project working!* **Submit a link to the slack message in Canvas**

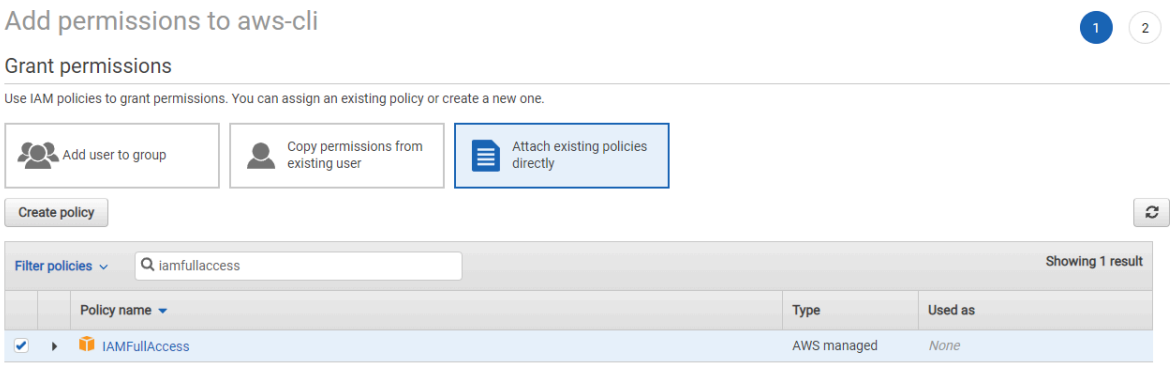


Instructions

- Complete steps in [Prerequisites](#)
- Create a public repository for your project **https://github.com/{username}/esep-webhooks**
- Build a Lambda function that will receive a [Webhook](#) from GitHub
    - The function will execute when it receives a notification from GitHub for a specific event. We will set this up later in the instructions
    - Use the Command Line Interface (CLI) for your programming language to generate a generic lambda function. Example if using .NET:
        - *dotnet new lambda.EmptyFunction --name EsepWebhook*

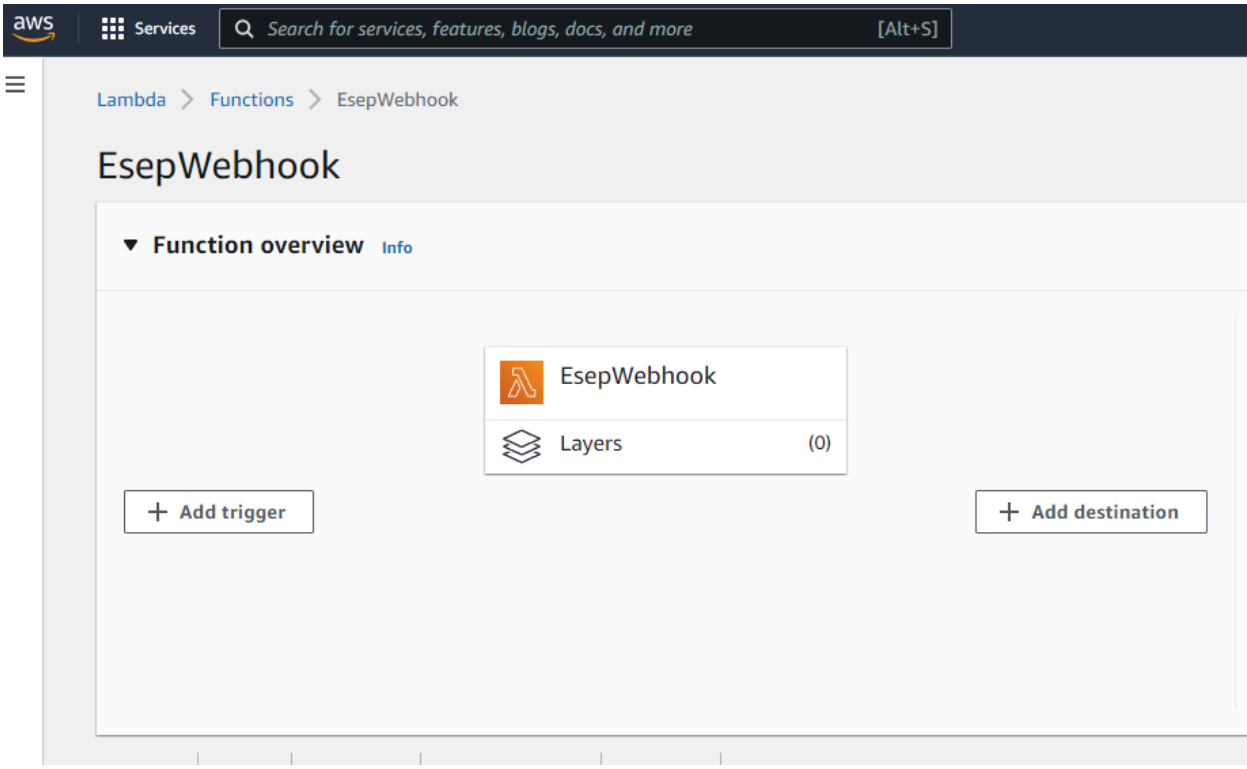
- With only the base template, let's get that deployed, tested, and verified using the CLI for your programming language to deploy your function:
    - Before doing this, you must create the IAM Role First
        - In AWS, go to the IAM dashboard by searching for IAM
        - Name: Esep-Webhook
        - In the left, click users -> add users -> add name -> attach policies and add "IAMFullAccess" and "AWSLambda_FullAccess"
        - Finish and click on the user to go into the security credentials tab and create an access key and click CLI, save these keys somewhere as you will need them
        - In the CLI, make sure you are in the /src/esepwebhook directory
        - In the CLI run aws configure and configure using the keys, region (us-east-2) and output (JSON)

    - In the CLI run the command "dotnet tool install -g Amazon.Lambda.Tools"
    - Also run "dotnet new -i Amazon.Lambda.Templates"
    - Now run *dotnet lambda deploy-function EsepWebhook*

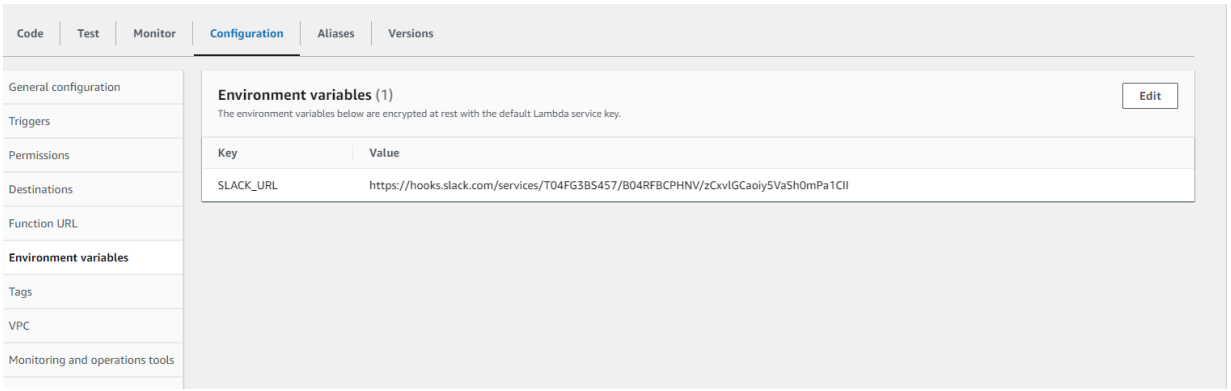


-

- You should now be able to see your function deployed in the AWS UI

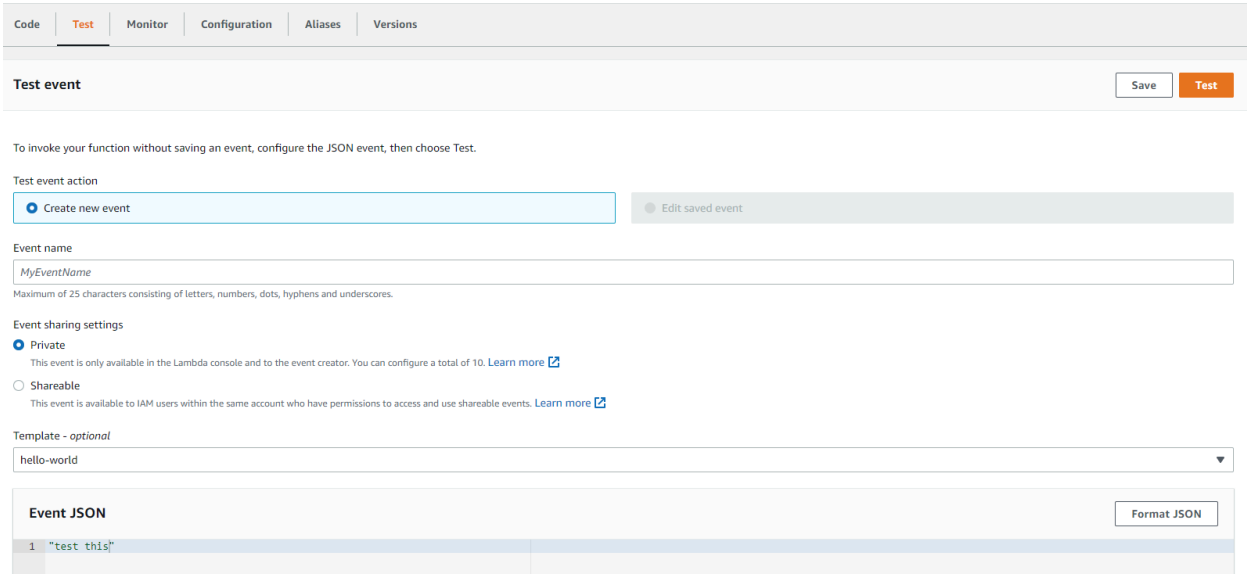


- Next you will need to add a Configuration -> Environment Variable with the URL for the webhook

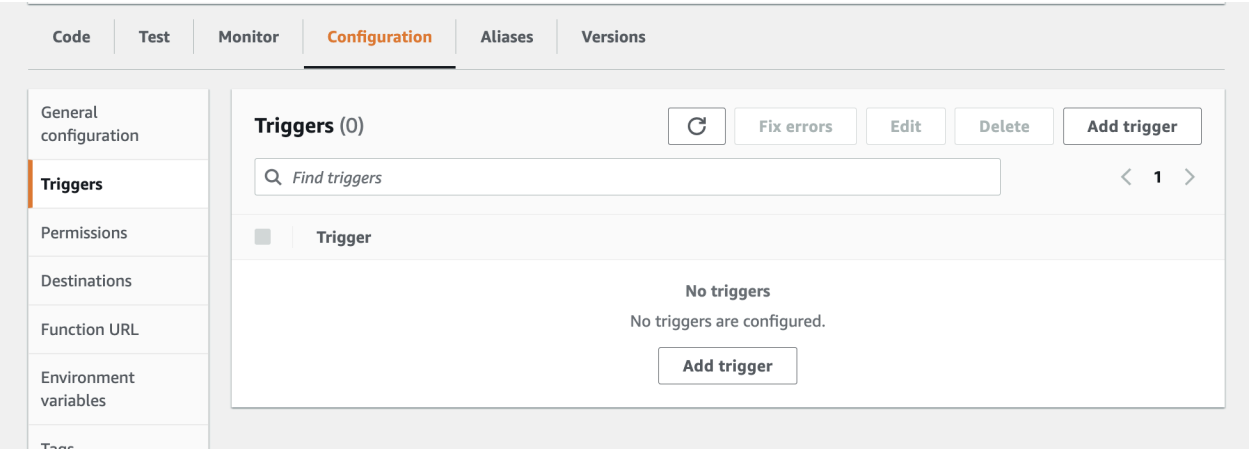


- **Important!! If you push the slack URL/link below to Github it will invalidate the link for the entire class and YOU WILL LOSE 2 POINTS! Be sure to use the environment variable to avoid this.**
- Create an Environment Variable with Key: "`SLACK_URL`" and Value: `https://hooks.slack.com/services/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`

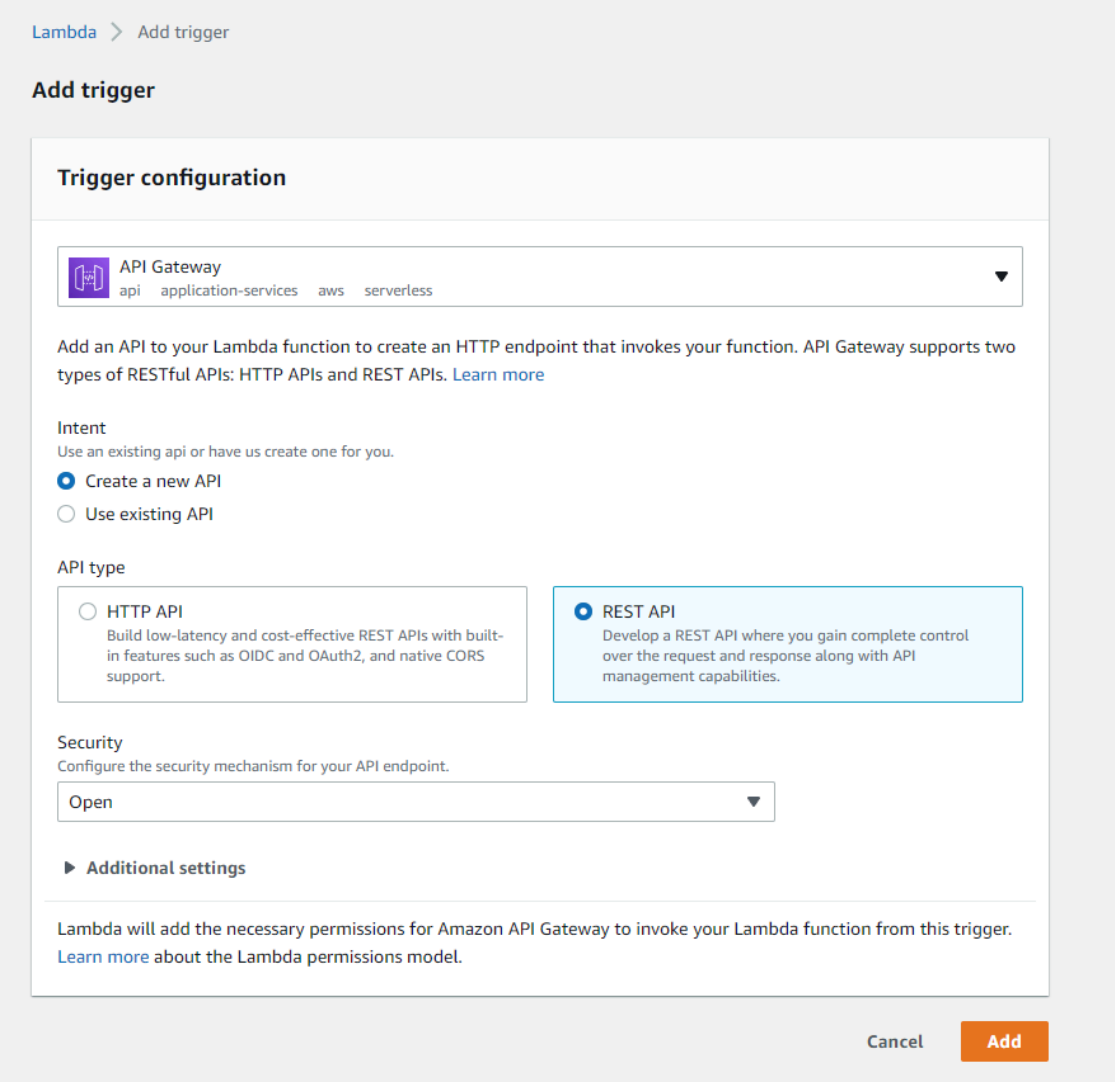
- You should now be able to test your function and ensure it executes. Change the event json to a string as shown in the picture below.



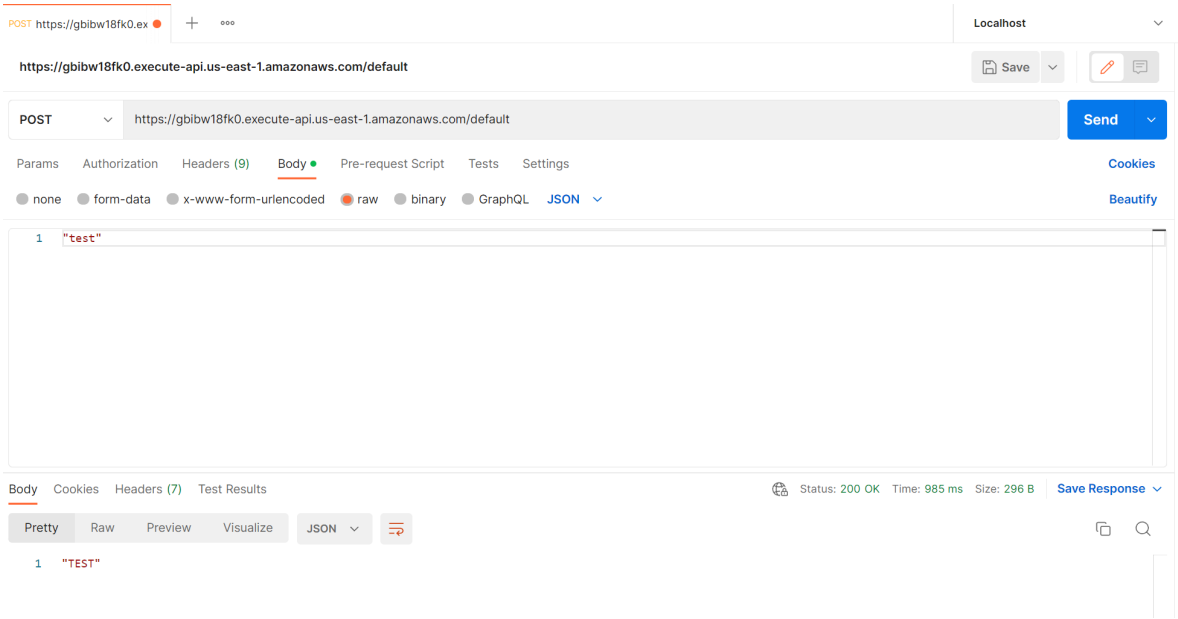
- Add a trigger to the function through the UI by clicking on the *Configuration* tab



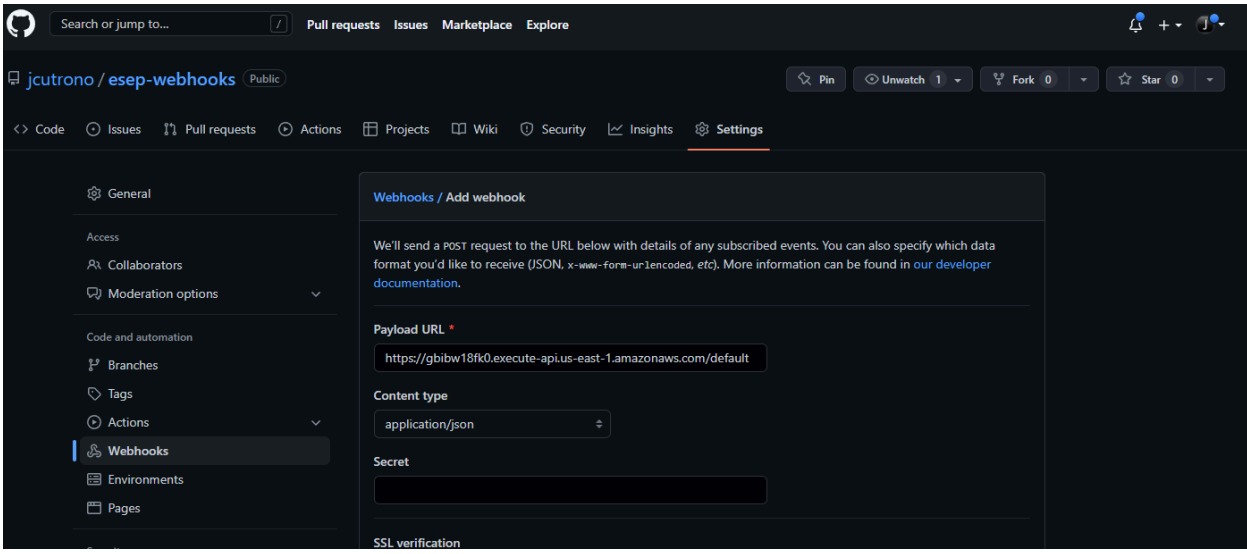
- Go to **API Gateway** in the AWS UI and create an API
- Select **REST API** and for Security select **Open**
- Leave all other default settings as is. The name should read *Esep-Webhook-Api* in the Additional Settings drop down.



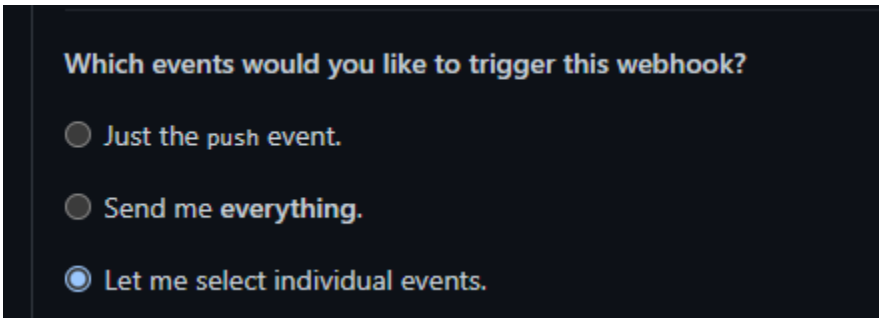
- Once created, click on the name of your API Gateway. This will take you to a new window. Click the Actions drop down and select **Create Method**
- In the new dropdown, select POST and click the checkmark icon to create it.
- In the Lambda function drop down, type the name of your lambda function and select it.
- Click save and then ok in the pop up
- In the actions dropdown, click **Deploy API**
- For stage select [New Stage]
- Name it **default** and then click deploy
- Leave default settings and click save at the bottom
- You should now get an Invoke URL you can test in Postman
- Make sure to add your webhook name after "/default" and click raw check box instead of none and type something



- Go to your GitHub repo and go to **Settings > Webhooks** to add the webhook. Paste your invoke URL in the Payload URL, select “application/json” for Content type, and keep the default SSL settings.



- Under the configuration, select “Let me select individual events” and make sure only the checkbox for **Issues** is checked



- Modify your Lambda function to interpret the input received from the github webhook and pull out the { issue: { html_url: “link to the issue created” }}

```
public string FunctionHandler(object input, ILambdaContext context)
{
    dynamic json = JsonConvert.DeserializeObject<dynamic>(input.ToString());

    string payload = $"{{'text':'Issue Created: {json.issue.html_url}'}}";
```

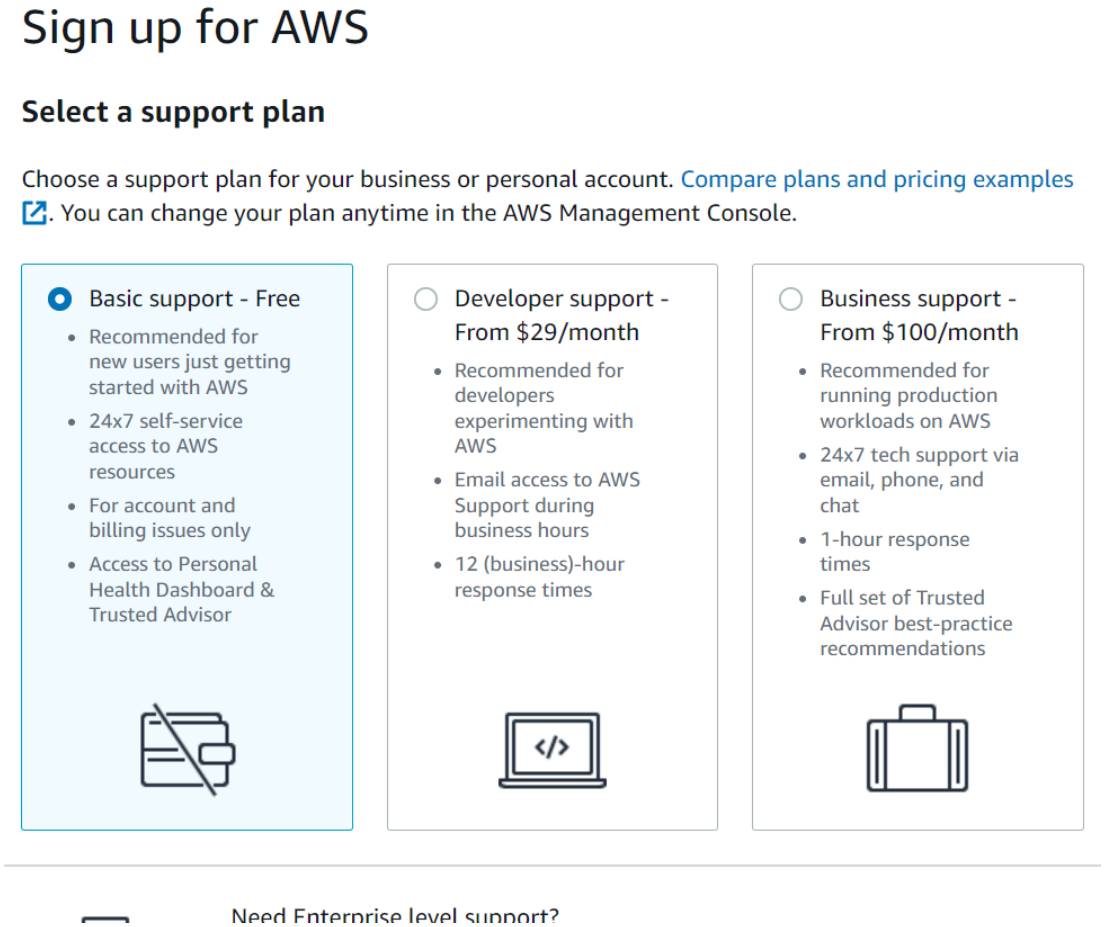
- Example code posted here: https://github.com/jcutrono/esep-webhooks/blob/main/EsepWebhook/src/EsepWebhook/Function.cs
- Commit and push your code to your public repository on Github
- **Important!! If you push the slack URL/link to Github it will invalidate the link for the entire class and YOU WILL LOSE 2 POINTS! Be sure to use the environment variable to avoid this.**
- Also run “dotnet add package Newtonsoft.Json”
- Deploy to AWS (using *dotnet lambda deploy-function EsepWebhook)*

```
public string FunctionHandler(object input, ILambdaContext context)
{
    dynamic json = JsonConvert.DeserializeObject<dynamic>(input.ToString());

    string payload = $"{{'text':'Issue Created: {json.issue.html_url}'}}";

    var client = new HttpClient();
    var webRequest = new HttpRequestMessage(HttpMethod.Post, Environment.GetEnvironmentVariable("SLACK_URL"))
    {
        Content = new StringContent(payload, Encoding.UTF8, "application/json")
    };

    var response = client.Send(webRequest);
    using var reader = new StreamReader(response.Content.ReadAsStream());

    return reader.ReadToEnd();
}
```

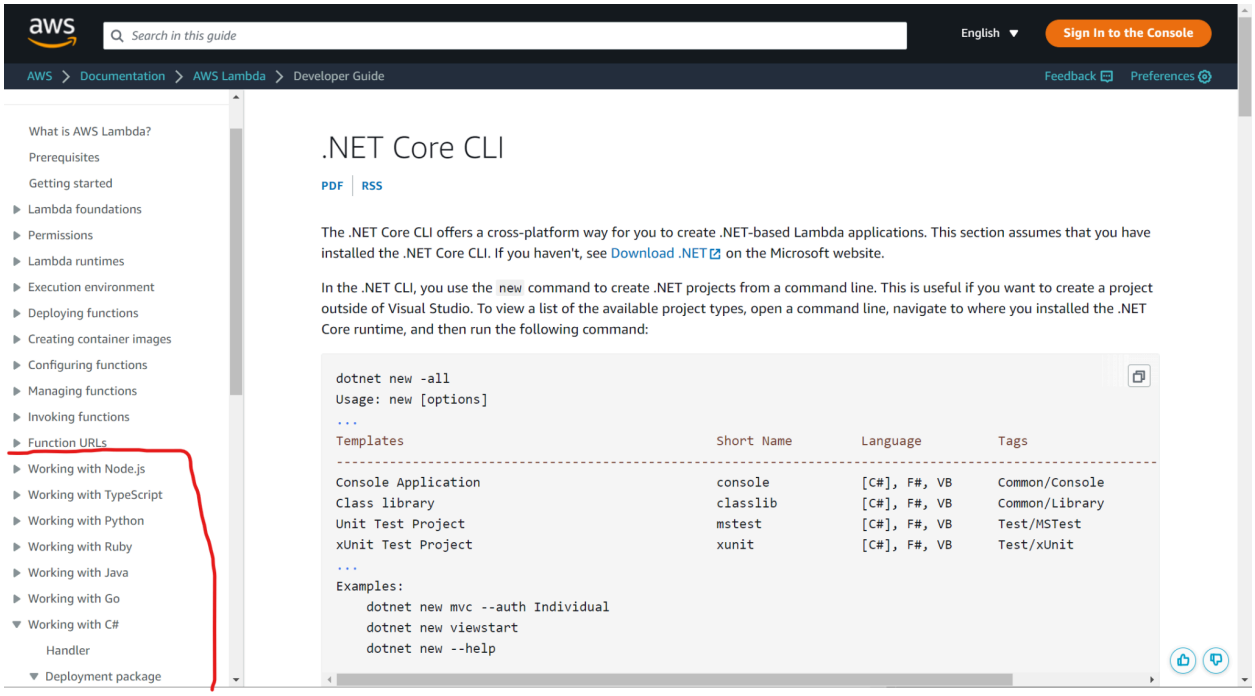
- Your program should post a message to our #eventing-webhook channel via the URL noted in the environment variable whenever your lambda function is triggered through creating an issue on your github repository

Prerequisites

- Create your AWS account
  - You will be required to enter a credit card, please note that nothing in this class will need anything other than the free tier of services
  - Select the free version of support



- Install the AWS Command Line Interface (CLI) https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
- For this application we are going to upload a zip file of artifacts from your machine. You can read through the setup and select your programming language of choice: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-package.html#gettingstarted-package-zip
- For help creating **IAM Security Credentials**: https://dev.to/hoangleitvn/how-to-build-test-and-deploy-lambda-function-to-aws-53cj
- Install the appropriate CLI for your programming language: https://docs.aws.amazon.com/lambda/latest/dg/csharp-package-cli.html



Resources

- If you do not already have a preferred IDE, please check out VS Code: https://code.visualstudio.com/download
- If you don't have a preferred way to hit web apis: https://www.postman.com/downloads/
- Testing Webhooks- give you the ability to see the payload going to your function http://webhookinbox.com/