

# Event-Driven Architecture Assignment

## Table of Contents

[Overview](#)

[Grading \(6 points total\)](#)

[Instructions](#)

[Prerequisites](#)

[Resources](#)

## 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*

Add permissions to aws-cli

Grant permissions

Use IAM policies to grant permissions. You can assign an existing policy or create a new one.

Add user to group
 Copy permissions from existing user
 Attach existing policies directly

[Create policy](#)

Filter policies		Q iamfullaccess	Showing 1 result	
	Policy name		Type	Used as
<input checked="" type="checkbox"/>	IAMFullAccess		AWS managed	<i>None</i>

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

The screenshot shows the AWS Lambda console. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, a search bar ('Search for services, features, blogs, docs, and more'), and a keyboard shortcut '[Alt+S]'. Below the navigation, the path 'Lambda > Functions > EsepWebhook' is shown. The main title 'EsepWebhook' is displayed above a 'Function overview' section. This section includes a small Lambda icon, the function name 'EsepWebhook', and a 'Layers' section indicating '(0)' layers. At the bottom of this section are two buttons: '+ Add trigger' and '+ Add destination'.

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

The screenshot shows the 'Configuration' tab of the AWS Lambda function 'EsepWebhook'. On the left, a sidebar lists various configuration sections: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables (which is currently selected), Tags, VPC, and Monitoring and operations tools. The main content area displays the 'Environment variables' section, which contains one entry: 'SLACK\_URL' with the value 'https://hooks.slack.com/services/T04FG3BS457/B04RFBCPHNV/zCxvlGCaoy5VaSh0mPa1CII'. There is an 'Edit' button in the top right corner of this section.

- **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.

The screenshot shows the AWS Lambda 'Test' interface. At the top, there are tabs: Code, Test (which is selected), Monitor, Configuration, Aliases, and Versions. Below the tabs, a section titled 'Test event' contains a 'Test event action' dropdown set to 'Create new event'. To its right is a 'Edit saved event' button. A 'Event name' input field contains 'MyEventName'. Under 'Event sharing settings', the 'Private' radio button is selected. There are two options: 'This event is only available in the Lambda console and to the event creator. You can configure a total of 10.' and 'Shareable'. The 'Shareable' option is described as 'This event is available to IAM users within the same account who have permissions to access and use shareable events.' A 'Template - optional' input field contains 'hello-world'. At the bottom, an 'Event JSON' code editor shows the following JSON:

```

1 "test this"

```

With a 'Format JSON' button to its right.

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

The screenshot shows the AWS Lambda 'Configuration' tab. The left sidebar has navigation links: General configuration, Triggers (which is selected and highlighted in orange), Permissions, Destinations, Function URL, Environment variables, and Trace. The main area is titled 'Triggers (0)' and contains a search bar with 'Find triggers' placeholder text, a refresh button, and buttons for 'Fix errors', 'Edit', 'Delete', and 'Add trigger'. Below the search bar, there's a table header with columns for 'Trigger' and 'No triggers'. A message states 'No triggers are configured.' with a 'Add trigger' button at the bottom.

- 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.

Lambda > Add trigger

## Add trigger

### Trigger configuration

 API Gateway  
api application-services aws serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

**Intent**  
Use an existing api or have us create one for you.

Create a new API  
 Use existing API

**API type**

**HTTP API**  
Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

**REST API**  
Develop a REST API where you gain complete control over the request and response along with API management capabilities.

**Security**  
Configure the security mechanism for your API endpoint.

[Open](#)

[▶ Additional settings](#)

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

[Cancel](#) [Add](#)

- 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**

The screenshot shows the Postman interface with a successful API call. The URL is `https://gbibw18fk0.execute-api.us-east-1.amazonaws.com/default`. The request method is POST. The Body tab shows the payload: `1 "TEST"`. The response status is 200 OK, and the body is `1 "TEST"`.

- 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.

The screenshot shows the GitHub repository settings for `jcutrono / esep-webhooks`. Under the **Webhooks** tab, the configuration includes a Payload URL of `https://gbibw18fk0.execute-api.us-east-1.amazonaws.com/default`, a Content type of `application/json`, and the Secret field is empty. The SSL verification is set to `SSL verification`.

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

The screenshot shows the event selection dialog for a GitHub webhook. It lists three options: `Just the push event.`, `Send me everything.`, and `Let me select individual events.`. The third option is selected.

- 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

# Sign up for AWS

## Select a support plan

Choose a support plan for your business or personal account. [Compare plans and pricing examples](#). You can change your plan anytime in the AWS Management Console.

### Basic support - Free

- Recommended for new users just getting started with AWS
- 24x7 self-service access to AWS resources
- For account and billing issues only
- Access to Personal Health Dashboard & Trusted Advisor



### Developer support - From \$29/month

- Recommended for developers experimenting with AWS
- Email access to AWS Support during business hours
- 12 (business)-hour response times



### Business support - From \$100/month

- Recommended for running production workloads on AWS
- 24x7 tech support via email, phone, and chat
- 1-hour response times
- Full set of Trusted Advisor best-practice recommendations



Need Enterprise level 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-53c>
- Install the appropriate CLI for your programming language:  
<https://docs.aws.amazon.com/lambda/latest/dg/csharp-package-cli.html>

The screenshot shows the AWS Lambda Developer Guide for the .NET Core CLI. The sidebar on the left lists various topics, with 'Function URLs' highlighted by a red box. The main content area displays the .NET Core CLI documentation, specifically the 'dotnet new -all' command, which creates .NET-based Lambda applications. This command includes usage instructions, examples, and a table of available templates with their short names, languages, and tags.

Templates	Short Name	Language	Tags
Console Application	console	[C#], F#, VB	Common/Console
Class library	classlib	[C#], F#, VB	Common/Library
Unit Test Project	mstest	[C#], F#, VB	Test/MSTest
xUnit Test Project	xunit	[C#], F#, VB	Test/xUnit

## 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/>