Assignment 10

New Attempt

- Due Nov 26, 2024 by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip

Please submit your assignment on Canvas as a ZIP with your report and any code you may have created. If your code is minimal and does not have many lines you can provide just the PDF with the code inside it.

AWS Lambda & Serverless Architecture

In this assignment, you will get introduced to serverless architecture. You are going to build a small API using AWS Lambda (for serverless execution) and AWS API Gateway (to create and link the API to Lambda). You will also be introduced to AWS Cognito, a user management and identity tool.

AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second.

In this set of tasks, you are going to create your first Lambda function. Head to AWS Lambda and click Create a Function, for the Runtime choose Node.js 18.x or Node.js 20.x.

For execution role you can create a new Role with the LambdaExecution Policy, but if you have logged in through Educate account you will not be able to do so. In that case, choose Use an existing role > robomaker_students.

For the name you may use `cybersecurityComments`.

After configuring the role click Create Function.

If you scroll down you can see that a sample function code has been generated.

Task 1: For this task, you should fill in the function template code. Create a constant that is an array of Javascript objects and call it `comments`. The constant `comments` will store comments submitted by users. Each element of `comments` must have a title and a body e.g `[{title: “a”, body: “b”}, {...}]`. The returned value of the function should be a Javascript object. One of the fields in that object should be called `body`. Create the object having the body field and assign the `comments` constant to the `body` field. Report the completed function code.

In order to test this function we need to create a test event. To do so, go to the dropdown box and select Test > Create new event > give a proper name and Event JSON `{}` > Click Save.

Then, select the newly created event and click Test.

Verify that the response you get (from the green box that appears on top of your page) does not have any errors and corresponds to the comments you created earlier.

NOTE: If you still get the "Hello from Lambda" response confirm that you saved the index.js file and that you also clicked the Deploy button.

This Lambda you created will serve as the "backend" of the API. Each time your API is called it will invoke the Lambda function you created. This setup is usually referred to as serverless because there is no need to create a server from your side.

Task 2: Discuss the differences, pros and cons between creating a serverless API using Lambda vs creating a server on Amazon's EC2 you learned in the previous assignments.

API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

In the following tasks, you are going to create a simple HTTP (GET/POST) API that uses the Lambda function you created previously. The API will be used to GET the set of comments in your Lambda function or create a new comment (via POST). In order to do so go to API Gateway > REST API (not the Private one) > Build > Fill in `cybersecurity_api` for API name and click Create API.

Now we need to create an API Resource for comments. Click Actions > Create Resource > Fill in `comments` for the name and path and check CORS > Click Create Resource.

After creating the API resource we need to create a method and link it with the Lambda we have created. Select the resource you have created and click Create method, and from the dropdown select GET.

The GET method is usually used in APIs to fetch data. The POST function is used to change data in an API (in our case it will add a new comment).

Next, check Use Lambda function and for the function name, start typing the Lambda function name you created, and you should get it autocompleted. Select the name of the Lambda function and click Create method.

In the next step click again on Deploy API > Select New Stage. You may use `dev` to fill in the Name textbox. Next, click Create.

Now you should be able to see your API's public link. You can test your Lambda by following the link similar to:

`https://65xmsec2sj.execute-api.us-east-1.amazonaws.com/dev/comments`

Task 3: In order for our Lambda to be able to work with GET and POST requests we need to modify it. Modify the Lambda function you created in order to be able to serve a GET and a POST method request. For a GET request, you should return the set of comments. For a POST request, read the body of the request, parse it using JSON.parse() and add it to the comments set. Please consider doing the bare minimum validation to the POST body on your Lambda so that you don't add null values or that it does not crash when provided with a null POST body. Report the code.

Now repeat the same procedure as before in which you created a GET method, to create a POST method and use the same Lambda function for integration. The Lambda function should be able to handle both GET and POST requests if you have implemented it correctly in Task 3.

You need to redeploy the API using the same steps as before. Now your API should have a POST method. You can use software for REST API calls like [Postman (https://www.getpostman.com/)](https://www.getpostman.com/) or Advanced REST Client ([Program (https://install.advancedrestclient.com/install)](https://install.advancedrestclient.com/install)) to test your newly created API. Try using GET to get all the comments and POST to create a new comment.

Task 4: If you create a new comment and then do a GET, you should be able to get all previously inserted comments including your newly created comment. If you wait for a while (e.g. 30 minutes) and do a GET again you will realize that your newly created comment is not returned (only the default `comments` constant you created in your Lambda is returned). Explain the reason. What AWS component do you need to create in order for your comments to persist in time?

Cognito

Amazon Cognito is a simple user identity and data synchronization service that helps you securely manage and synchronize App data for your users across their mobile devices. You can create unique identities for App users through a number of public login providers (Amazon, Facebook, and Google) and also support unauthenticated guests.

We will use AWS Cognito to provide authorized access to your API.

In order to create an AWS Cognito User Pool go to Cognito > Create User Pool > Choose one Cognito user pool sign-in option > Next > Choose an MFA method (and configure it) > Next > Send email with Cognito > Add an IAM role name > Give a name of cybersecurity_cognito > Review Defaults > Create User Pool (fill only mandatory values)

Task 5: Explain the steps needed to integrate your newly created AWS Cognito to the POST method you developed in Task 3, so that your POST methods are authorized.

Make sure to test your configuration and verify that it works by doing a POST request on your API and checking that you are not authorized to access the API (Don't forget to redeploy the API).

Furthermore, report a short demonstration of the steps in order to invoke your Cognito Authorized endpoint of your API (POST method).

NOTE: Here is some information that you may find helpful:

Amazon Cognito uses JWT for authentication [https://jwt.io (https://jwt.io/)](https://jwt.io/). JWT is a token encoded in base 64 and one of the fields included in the token is the expiration time.

In order to be able to do a POST request on the Lambda you created in the previous Tasks you need to obtain a JWT. To provide some help here are three basic ways that you can get the JWT required in order to access the Authenticated POST endpoint of your Lambda (Don't forget to create a user for testing purposes):

Way 1: Consider following **this (https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-app-integration.html#cognito-user-pools-create-an-app-integration)** guide to obtain a "code" for a user. Also, you might need to perform a request to get the "id_token" from the OAuth endpoint. For this task consider reading **this (https://docs.aws.amazon.com/cognito/latest/developerguide/token-endpoint.html)** .

Way 2: Alternatively you can test the authenticated method by downloading **this (https://github.com/awslabs/aws-serverless-workshops/archive/master.zip)** repository and experimenting with WebApplication/1_StaticWebHosting/website to acquire a new token and test it against Postman or any REST API client of your choice.

Way 3: Following is a script that implements **Way 1** and automates the requests needed to be done in AWS in order for you to get the JWT required. You can download the file from **here (https://canvas.sfu.ca/courses/85026/files/24007536?wrap=1)** **(https://canvas.sfu.ca/courses/85026/files/24007536/download?download_frd=1)** . Inside you will find 2 `package*.json` files. The `index.js` and a `config.env` file to provide some configuration. Open the `config.env` file and provide the following configuration:

```
CALLBACK_URL (required)
APP_CLIENT_ID (required)
APP_CLIENT_SECRET (only if you have enabled it)
AWS_COGNITO_DOMAIN (required)
```

There are instructions in the file itself (`config.env`) on how you fill these, however, you need to find out how to configure AWS properly. The provided script is used only to automate the process of getting the JWT without you having to make subsequent requests manually to AWS; in other words, it does not create the configuration on AWS for you. If you have misconfigured something on AWS or you misconfigure the script with the wrong input you will get a 400 HTTP error.

Task 6: A user or a potential attacker can manipulate that token to increase the expiration time. However, this wouldn’t work as the team that created JWT has taken measures against such attacks. Now, suppose that as an attack you could change the expiration time or the authenticated user type of the JWT token from your browser. Explain why this attack would not work.