LESSON 3

In this lesson, we will create a User Profile Lambda function. This function will talk to Auth0 and retrieve information about the user. We will also set up an API Gateway. The API Gateway will allow our website to invoke the function.

Lastly, we will create a custom authorizer. A custom authorizer is a special Lambda function that the API Gateway executes to decide whether to allow or reject a request. We will use this custom authorizer to make sure that only authenticated users have access to the User Profile Lambda function.

NOTE: PLEASE CREATE ALL YOUR RESOURCES IN THE N. VIRGINIA REGION (US-EAST-1)

1. SET UP THE USER PROFILE LAMBDA FUNCTION

Let's get our User Profile Lambda function organized first.

- **Install npm packages**
  In the terminal / command-prompt, change to the directory of the function:

  ```
  cd lab-3/lambda/user-profile
  ```

  Install npm packages by typing:

  ```
  npm install
  ```


- **Zip Lambda function**

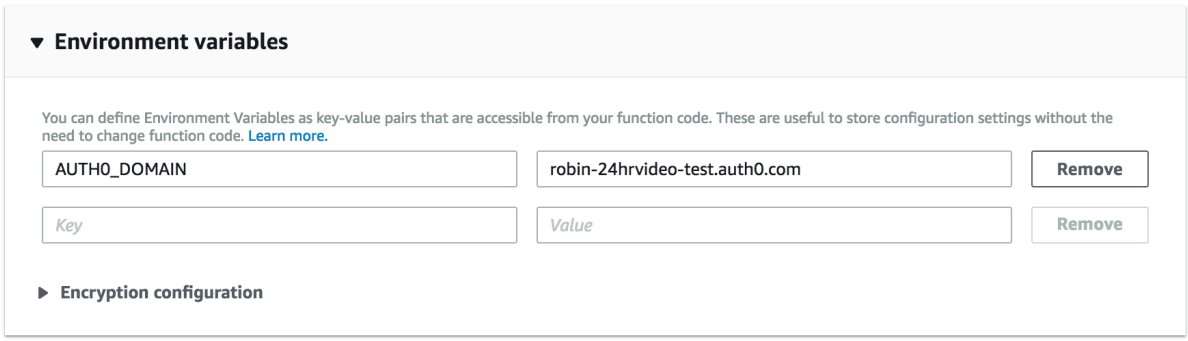
  For OS X / Linux Users
  Now create create a ZIP file of the function, by typing:

  ```
  npm run predeploy
  ```

  For Windows

  You will need to **zip up all the files** in the **lab-3/lambda/user-profile** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the user-profile folder. Zip up the files inside of it**).

- In the AWS console's **Services**, click **Lambda** under **Compute**, and then click the **Create function** button.
- Click the **Author from scratch** button.
- **Name** the function **user-profile.**
- Set the **Runtime** to **Node.js 4.3**.
- Under **Role** select **Choose an existing** role and ***lambda-s3-execution-role***.
- Click **Create function**.
- Once the function is created, scroll down to the **Basic settings** section, set the **Timeout** to 0 minutes, 30 seconds.
- For **Code entry type,** select **Upload a .ZIP file.**
- Click **Upload**, and choose the zip file you just created:
  /lab-3/lambda/user-profile/Lambda-Deployment.zip
- Expand the **Environment variables** section, and create an environment variable with the key **AUTH0_DOMAIN** and set its value to the Auth0 domain from the last lesson.

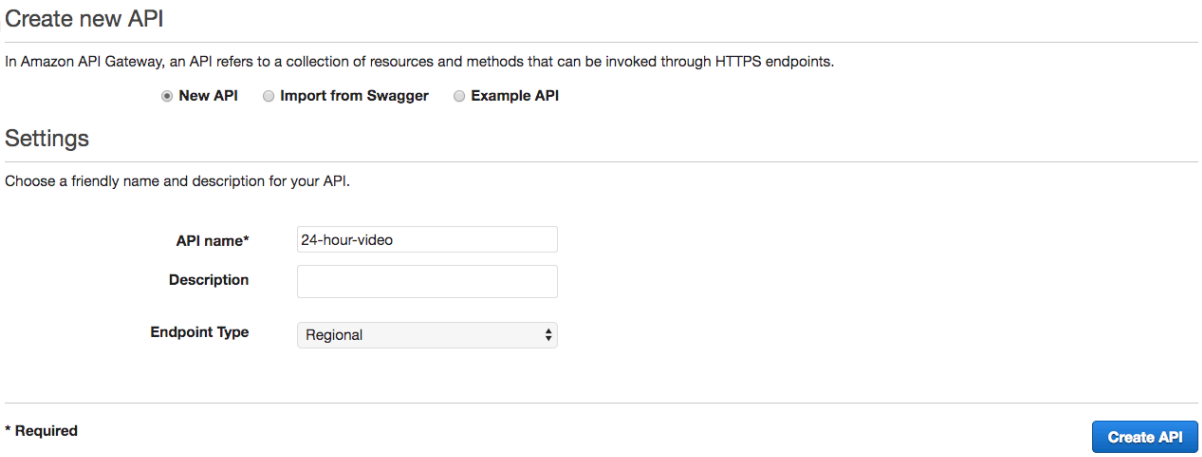


- Click the **Save** button at the top of the page.

2. CREATE THE API GATEWAY

The API Gateway needs to be set up to accept requests from our website. We need to create a resource, add support for a GET method, and enable Cross-Origin Resource Sharing (CORS). In the AWS console follow these steps:

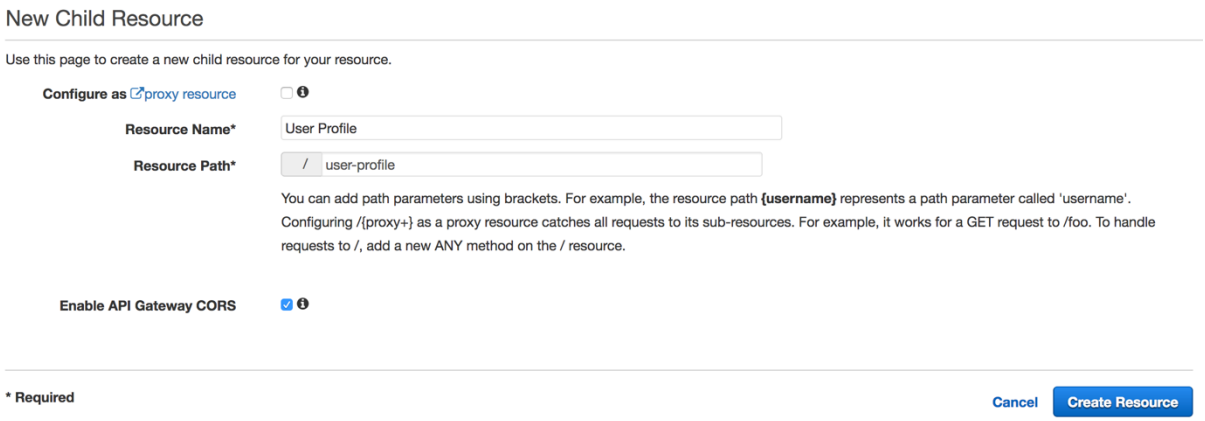
- In the AWS Console's **Services** tab, click on **API Gateway** under **Networking & Content Delivery**.
- If this is your first API Gateway, click the **Get Started** button. Otherwise, click the **Create API** button.
- Select the **New API** radio button.
- Type in a name for your API, such as **24-Hour-Video** and optionally, a description.
- Set **Endpoint Type** to **Regional**
- Click **Create API** to create your API:



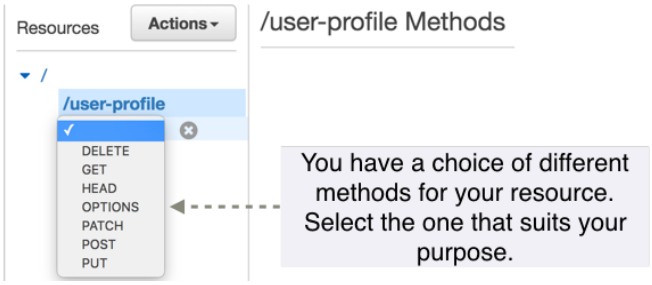
3. CREATE RESOURCE AND METHOD

APIs in the Gateway are built around *Resources*. We are going to create a Resource called *user-profile* and combine it with a GET method.

- In the **Resources** tab in the left navigation menu, select the **Actions** dropdown and click **Create Resource**.
- Type "User Profile" in the **Resource Name**. The **Resource Path** should be automatically filled in.
    - Note: Since the website will connect to the resource path */user-profile*, this value must match exactly.
- Tick the **Enable API Gateway CORS** box.
- Click the **Create Resource** button.

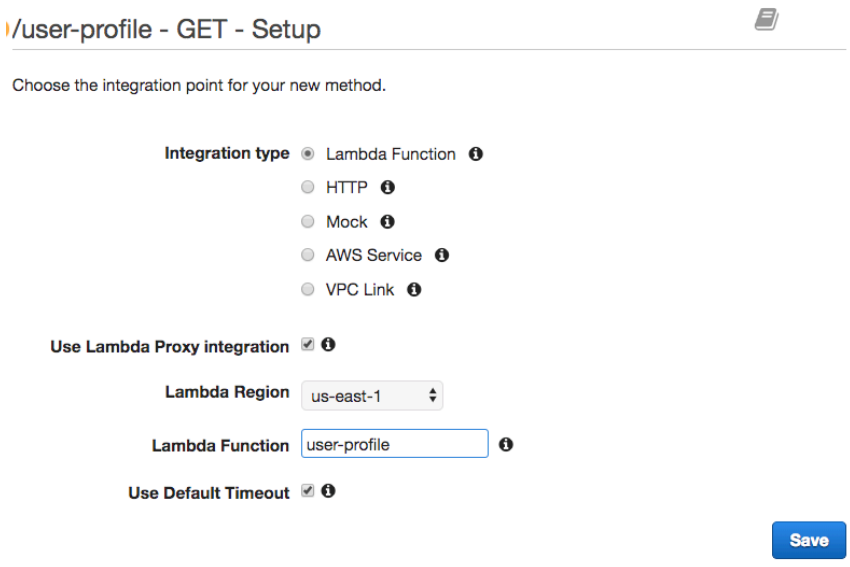


- The left-hand side list should now show **/user-profile**. Click it and then select the **Actions** dropdown again, and click the **Create Method** button to see a small dropdown under **/user-profile**.
- From that dropdown select **GET** and click the button with the **tick/check mark** on it to confirm.



Having created the GET method, we need to configure the *Integration Point*. This is the screen you should be looking at right now. (If you are not on it, click **GET** under **/user-profile).** An Integration Point specifies which Lambda function (or HTTP endpoint) the API Gateway should invoke.

- Click the **Lambda Function** radio button.
- Tick the checkbox labeled **Use Lambda Proxy Integration**.
- Select your region (**us-east-1**) from the **Lambda Region** dropdown.
- Type ***user-profile*** in the **Lambda Function** text box.



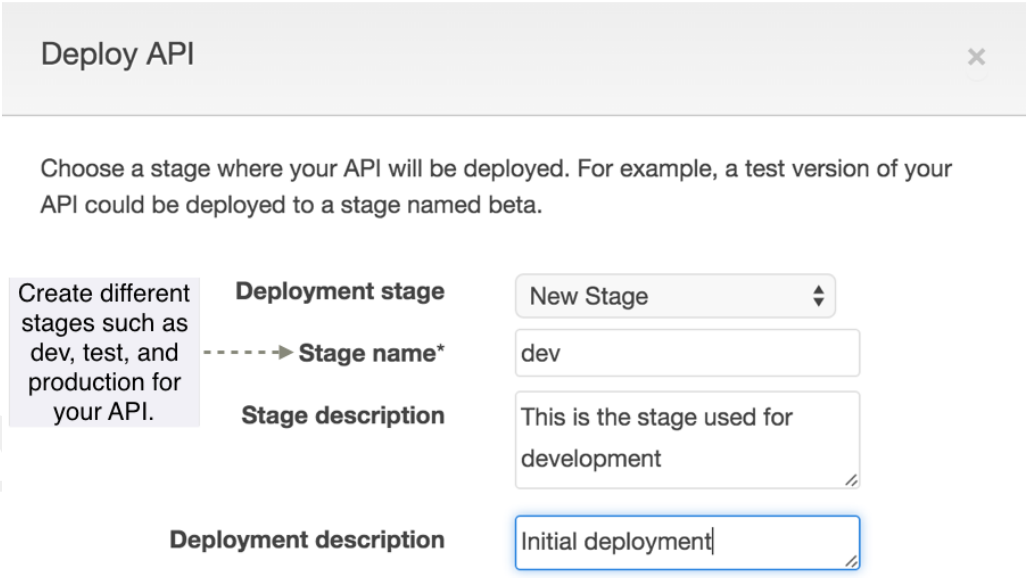
- Click **Save**.

- Click **OK** if you are asked if it's ok to add permission to the Lambda function.

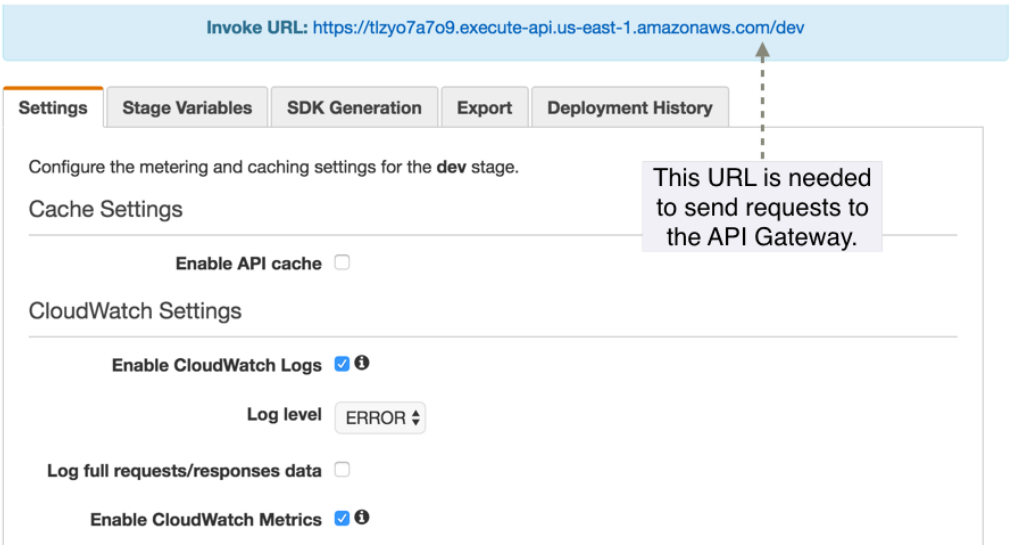
4. DEPLOY

Next, we need to deploy the API and get a URL to invoke it from the website.

- You should still be on your new API within API Gateway.
- Select the **Actions** dropdown again.
- Click **Deploy API**.
- In the **Deploy API** popup select **[New Stage]**.
- Type **dev** as the Stage Name.
- Click **Deploy** to deploy the API.



- The next page you will see will show the **Invoke URL** and a number of settings.
- Copy the **Invoke URL** as you will need it in the next step.



5. UPDATE THE WEBSITE

We need to update the website to invoke the right API Gateway URL.

- Copy the config.js file containing your account specific settings, from the last lesson:
  Copy lab-2/website/js/config.js to lab-3/website/js/config.js

- Now edit the config.js file you just copied (lab-3/website/js/config.js) to add the following line after the **auth0** section:
  apiBaseUrl: 'API GATEWAY INVOKE URL FROM STEP 4'

```
var configConstants = {
    auth0: {
        domain: 'serverless.auth0.com',
        clientId: 'ab1Qdr91xU3KTGQ01e598bwee8MQr'
    },
    apiBaseUrl: 'https://tlzyo7a7o9.execute-api.us-east-1.amazonaws.com/dev'
};
```

Don't forget to save **config.js** when you are done.

6. A NEW ROLE

API Gateway supports custom request authorizers. These are Lambda functions that the API Gateways uses to authorize requests. Custom authorizers can validate a token and return an IAM policy to authorize the request. However, before we begin using custom authorizers we are going to create a different role for it.

- In the AWS console's **Services** tab, click **IAM** under **Security, Identity & Compliance** and then click **Roles** from the left navigation menu.
- Click **Create role**.
- From the **Select role type** step, select **Lambda**, and click the **Next: Permissions** button.
- From the list of policies check the box next to **AWSLambdaBasicExecutionRole**.
- Click **Next: Review**.
- Name the role **api-gateway-lambda-exec-role**.
- Click **Create role** to save and exit.

7. CUSTOM AUTHORIZER

Having created a new IAM role we can begin work on the custom authorizer now.

- **Install npm packages**
  In the terminal / command-prompt, change to the directory of the function:

  cd lab-3/lambda/custom-authorizer

  Install npm packages by typing:

  npm install

- **Zip Lambda function**

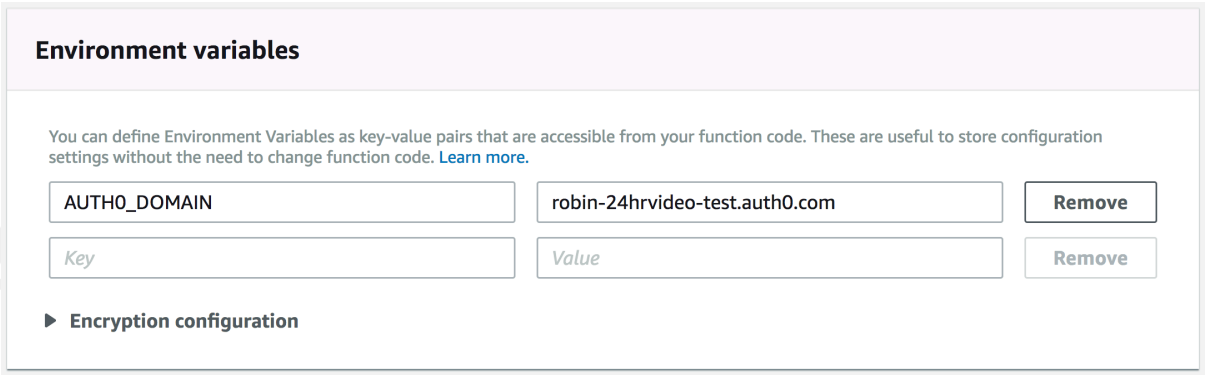
  For OS X / Linux Users
  Now create create a ZIP file of the function, by typing:

npm run predeploy

For Windows

You will need to **zip up all the files** in the **lab-3/lambda/custom-authorizer** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the custom-authorizer folder. Zip up the files inside of it**).

- In the AWS console's **Services** tab, click **Lambda** under **Compute**, and then click **Create function**.
- Click the **Author from scratch** button.
- **Name** the function **custom-authorizer**.
- Under **Runtime**, select **Node.js 6.10**.
- Under **Role**, select **Choose an existing role** and your new role: *api-gateway-lambda-exec-role*.
- Click **Create function**.
- For **Code entry type,** select **Upload a .ZIP file.**
- Click **Upload**, and choose the zip file you just created: /lab-3/lambda/custom-authorizer/Lambda-Deployment.zip
- Click **Environment variables** to expand it, and create an environment variable with the key **AUTH0_DOMAIN** and set its value to the Auth0 domain from the last lesson.



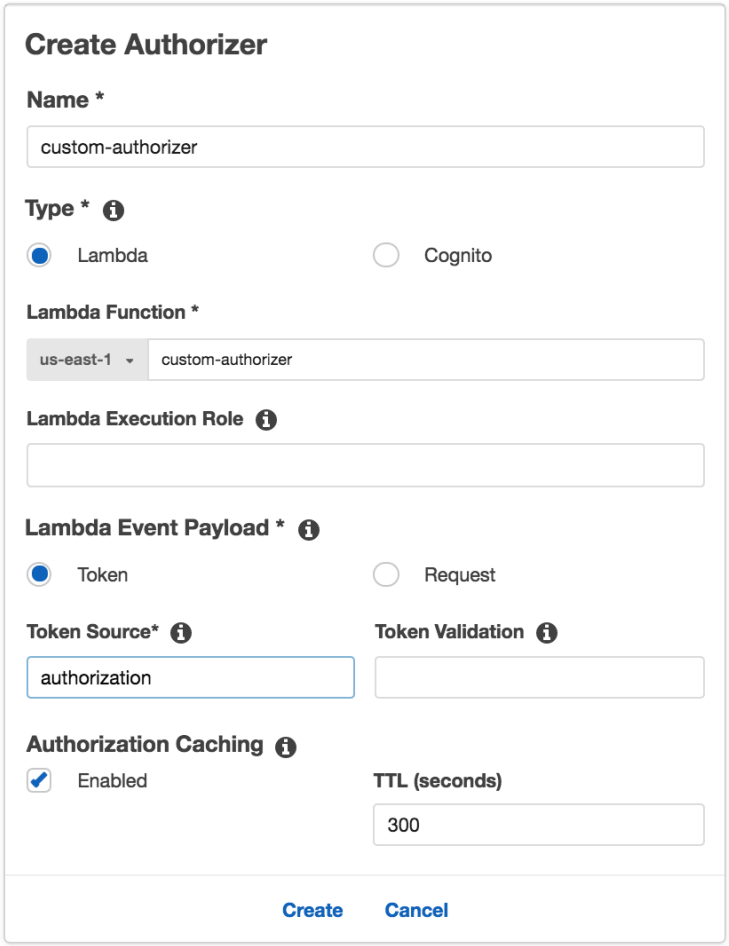
- Under **Basic settings**:
  - To improve the performance of your function, increase the **Memory (MB)** slider to *1536 MB*. This will also allocate more CPU to your function.
  - Set the **Timeout** to 0 minutes, 30 seconds.
- Click the **Save** button at the top of the page.

8. ASSIGN CUSTOM AUTHORIZER

Having deployed our custom authorizer function, we need to configure it so that it runs before our User Profile function.

- In API Gateway open the **24 Hour Video** API again.
- Click **Authorizers** from the left navigation menu.
- Click the **Create New Authorizer** button.
- Fill out the **New Custom Authorizer** form:
  - Type in **custom-authorizer** as the name of the Lambda function.
  - Set the region under **Lambda Function** to **us-east-1**
  - Set the **Lambda Event Payload** to **Token**.
  - In the **Token Source** box, type "authorization"
- Click **Create** to create the custom authorizer.

- In the **Add permission** popup, click **Grant & Create** to confirm that you want to allow API Gateway to invoke the custom-authorizer function.



To make the custom authorizer invoke when the GET method is called, follow these steps:

- Click **Resources** under 24-hour-video in the left navigation menu.
- Click **GET** under **/user-profile**.
- Click **Method Request**.
- Click the pencil next to **Authorization**.
- From the dropdown select **custom authorizer** and click the tick/check mark icon to save.

  Note: if custom-authorizer isn't in the list, refresh the page
- Click on the **HTTP Request Headers** section to expand it.
- Click on the **Add Header** link, put "authorization" for the name, and click the tick/check mark icon to the right and save it.
- Click on the **URL Query String Parameters** section to expand it.
- Click on the **Add query string** link, put "accessToken" for the name, and click the tick/check mark icon to the right and save it.
- Deploy the API again.
    - Select the **Actions** dropdown.
    - Click **Deploy API**.
    - Select **dev** as the **Deployment Stage**.
    - Click **Deploy**.

9. TEST THE SYSTEM

Lesson 3 is complete! Now it's time to test.

- In your terminal or command-prompt, change to the following folder:

  lab-3/website

- Run the following command to make sure that required npm components are installed:

  npm install

- Now run:

  npm start

- Open the web-site in your browser:
  http://localhost:8100

To test whether everything has worked:

- If you're still logged in to your *24 hour video* website, click the **Sign Out** button in the upper right.
- Refresh the page
- Log in to the website by clicking on Sign In button.
- Click the profile button (it'll have your nickname and, possibly, your picture). After a short wait you will see a modal box with your user information.

When you're done with this lab, exit the "npm start" command in your terminal by pressing <Control>-c.

Isn't this fun!? There is actually more goodness to come ☺. See you in the next lesson.

Optional Exercises

Try to do the following exercises to confirm your understanding of concepts presented in this lesson.

1. Create a Lambda function (*user-profile-update*) for updating a user's personal profile. Assume that you can access the first name, last name, email address and the userId on the event object. Because we don't have a database yet, this function doesn't need to persist this information; however, you can log it to CloudWatch.
2. Create a POST method for the /user-profile resource in the API Gateway. This method should invoke the *user-profile-update* function and pass in the user's information. It should use the custom authorizer developed in this lesson.
3. Create a page in the *24 Hour Video* website to allow signed-in users to update their first name, last name, and email. This information should be submitted to the *user-profile-update* function via the API Gateway.
4. Modify the User Profile Lambda function to no longer validate the JSON Web Token. This validation isn't needed due to the custom authorizer. The function should still request user information from the Auth0 *tokeninfo* endpoint.