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.

- **Open** config.js file in your favourite text editor:
  lesson-3/lambda/user-profile/config.js

  You'll need to set your **AUTH0_DOMAIN**.

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

  cd lesson-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

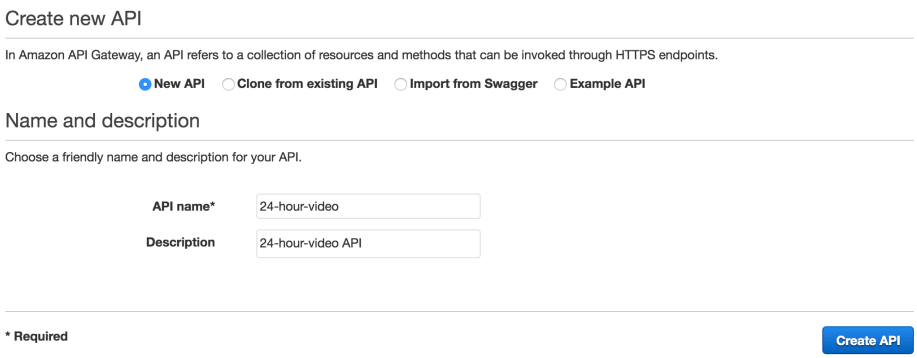
  You will need to **zip up all the files** in the **lesson-3/lambda/user-profile** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the user-profile folder. Zip up the files inside of it**).

- In the AWS console, click **Lambda**, and then click ***Create a Lambda Function***.
- Skip over the blueprint and configure triggers.
- **Name** the function **user-profile** and make sure that **Node.js 4.3** is selected in the ***Runtime*** dropdown.
- Select **Upload a ZIP file.** Choose the zip file you just created:
  /lesson-3/lambda/user-profile/Lambda-Deployment.zip
- Under Role select ***lambda-s3-execution-role***.
- Click ***Next*** to go the Review screen and from there click ***Create function*** to finish.

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:

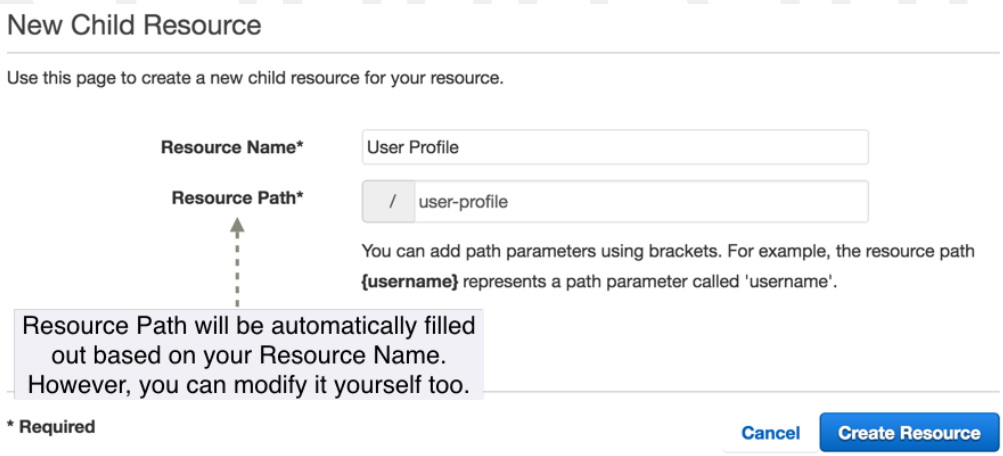
- Click on **API Gateway**
- Type in a name for your API such as **24-Hour-Video** and, optionally, a description
- Click **Create API** to create your first API



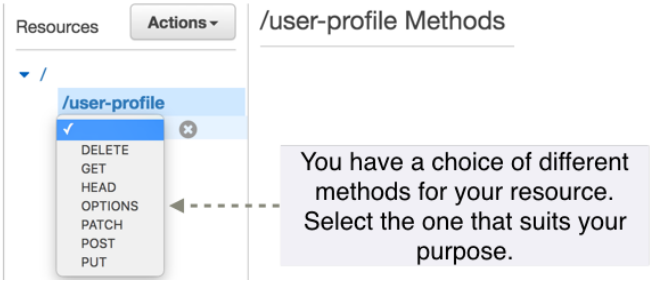
3. CREATE RESOURCE AND METHOD

API's in the Gateway are built around resources. We are going to create a resource called *user-profile* and combine it with a GET method.

- Click **Actions** and select **Create Resource** and type **User Profile** in the Resource Name. The *Resource Path* should be automatically filled in.
- Click *Create Resource* button to create and save the resource.

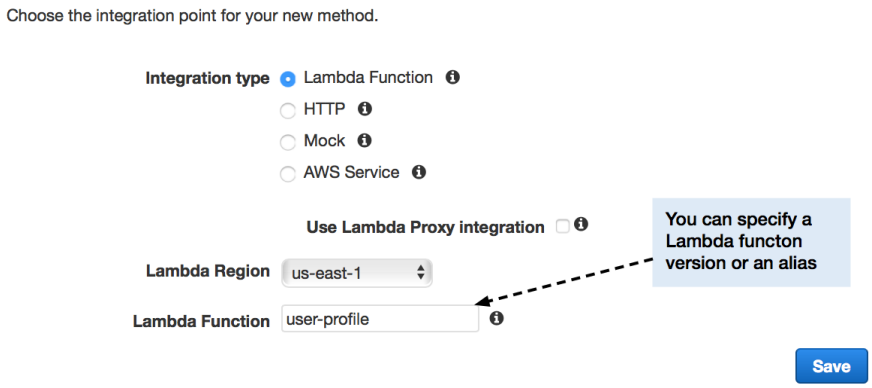


- The left-hand side list should now show /user-profile. Click it and then click **Actions** and select **Create Method** to reveal a small dropdown under /user-profile.
- From the dropdown select **GET** and click the button with the tick on it to confirm.



Having created the GET method, we need to configure the Integration Request. This is the screen you should be looking at right now. (If you are not on it, click **Integration Request** in the **Method Execution** screen of your GET function). An Integration Request specifies what Lambda function (or HTTP endpoint) the API Gateway should invoke.

- Click the **Lambda Function** radio button.
- Select your region (for example, **us-east-1**) from the Lambda Region dropdown.
- Type **user-profile** in the Lambda Function text box.

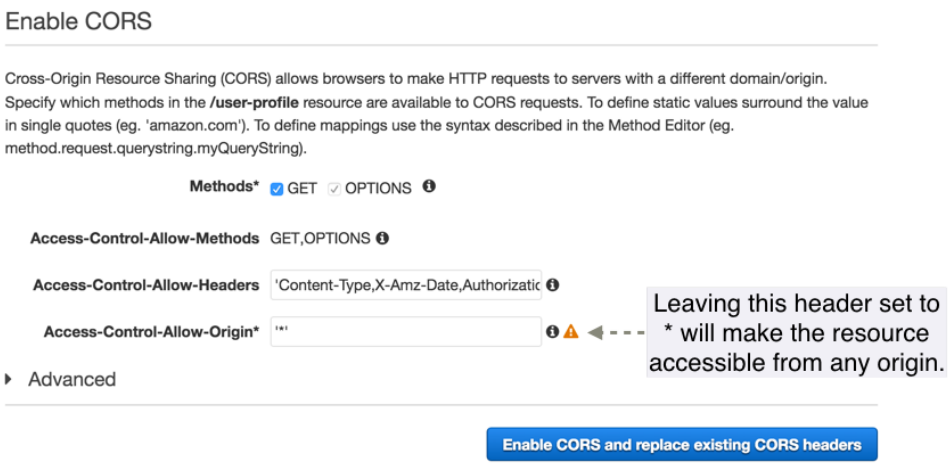


- Click **Save**.
- Click **OK** if you are asked if it's ok to add permission to the Lambda function.

4. ENABLING CORS

Next we need to enable CORS to be able to access our API Gateway endpoint.

- Click the **/user-profile** resource.
- Click **Actions**.
- Select **Enable CORS**.
- Click **Enable CORS and replace exists CORS headers** to save the configuration.
- Click **Yes, replace existing values** in the confirmation box that pops up.

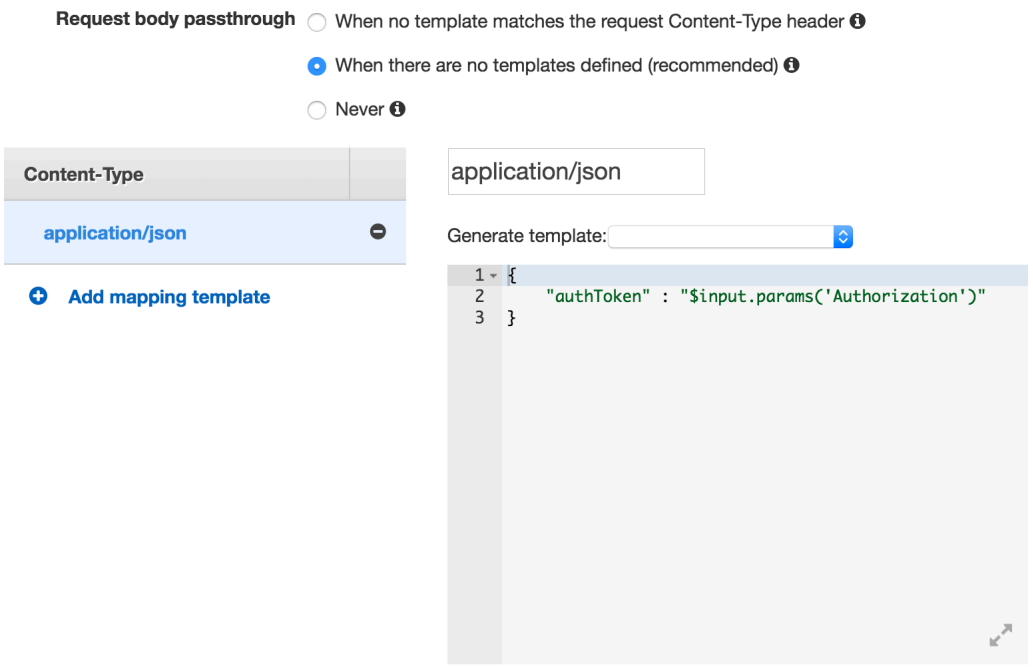


5. MAPPING

We need to create a mapping to pass the bearer token from the request in to our Lambda function.

- Click the **GET** method under the /user-profile resource.
- Click **Integration Request**.
- Expand **Body Mapping Templates**.
- Click **Add mapping template**.
- Type in **application/json** and click the tick button.
- Select **Yes, secure this integration** if you see a dialog box titled **Change passthrough behavior**.
- In the template box type in the following code.

```
{
    "authToken" : "$input.params('Authorization')"
}
```

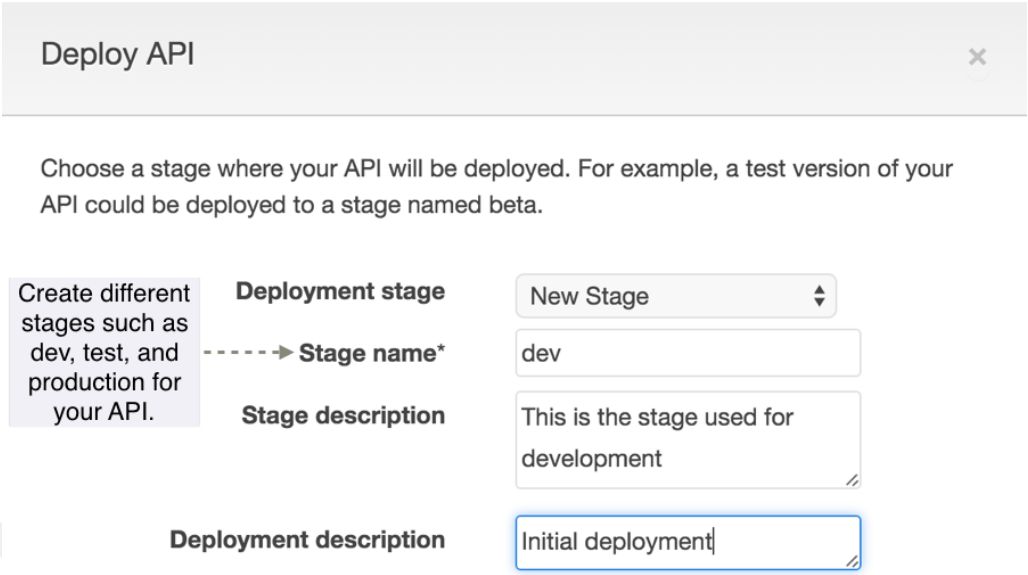


- Click **Save** once you have finished

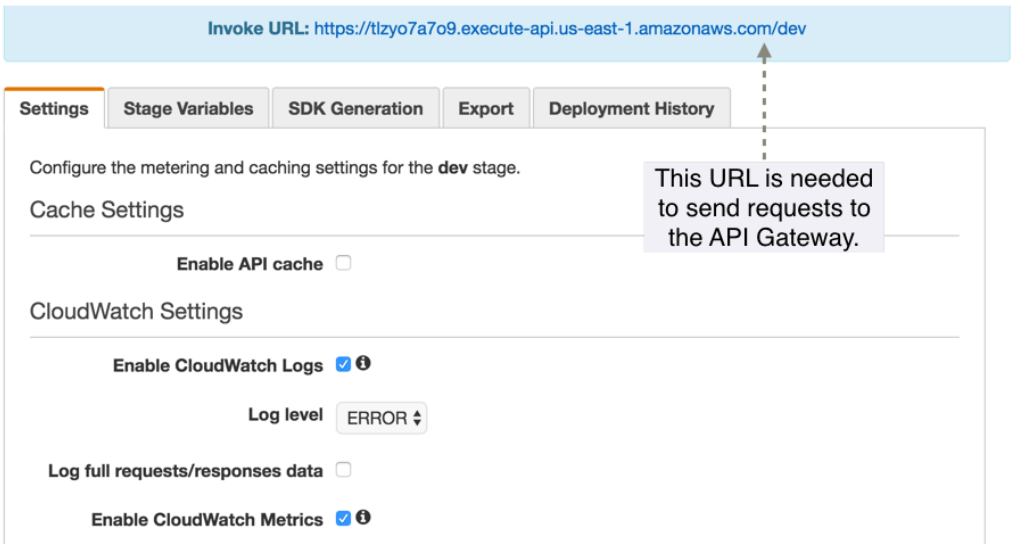
6. DEPLOY

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

- In the API Gateway make sure that your API is selected
- Click **Actions**
- Select **Deploy API**
- In the popup select **[New Stage]**
- Type **dev** as the Stage Name
- Click **Deploy** to provision the API



- The next page you will see will show the API URL and a number of options
- Copy the **Invoke URL** as you will need it later on



7. 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 lesson-2/website/js/config.js to lesson-3/website/js/config.js

- Now edit the copied config file to add the following line:
  apiBaseUrl: 'API GATEWAY INVOKE URL FROM STEP 6'

```
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.

8. 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, click **Identity & Access Management** and then click **Roles**.
- Click **Create New Role** and name it **api-gateway-lambda-exec-role**
- In step 2 of the role creation process select **AWS Lambda**
- From the list of policies select **AWSLambdaBasicExecutionRole**
- Click **Next Step**
- Click **Create Role** to save and exit

9. CUSTOM AUTHORIZER

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

- **Open** the config.js file in your favourite text editor:
  lesson-3/lambda/custom-authorizer/config.js

  You'll need to set your **AUTH0_SECRET**.

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

  ```
  cd lesson-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

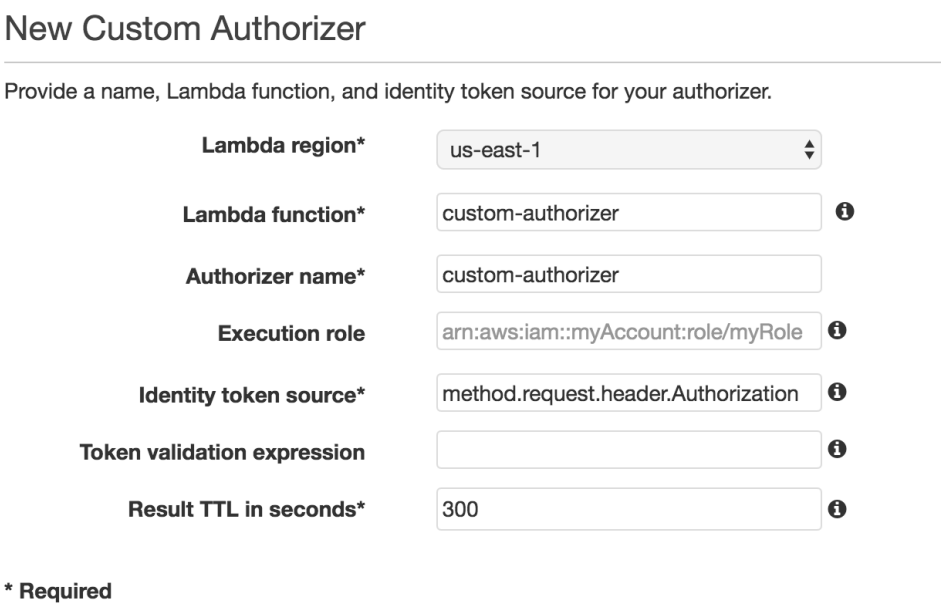
You will need to **zip up all the files** in the **lesson-3/lambda/custom-authorizer** folder via the Windows Explorer GUI, or using a utility such as 7zip. (**Note: don't zip the custom-authorizer folder. Zip up the files inside of it**).

- In the AWS console, click **Lambda**, and then click ***Create a Lambda Function***.
- Skip over the blueprint and configure triggers.
- **Name** the function **custom-authorizer** and make sure that **Node.js 4.3** is selected in the ***Runtime*** dropdown.
- Select **Upload a ZIP file.** Choose the zip file you just created: **/lesson-3/lambda/custom-authorizer/Lambda-Deployment.zip**
- Under Role select **api-gateway-lambda-exec-role**.
- Click ***Next*** to go the Review screen and from there click ***Create function*** to finish.

10. ASSIGN CUSTOM AUTHORIZER

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

- In the API Gateway open the **24 Hour Video** API.
- Click **Authorizers** on the left.
- Click the **Create** button.
- Select **Custom Authorizer**.
- Fill out the **New Custom Authorizer** form
    - Select region (us-east-1)
    - Type in **custom-authorizer** as the name of the Lambda function.
    - Set the name as **custom-authorizer**
    - Set the **Identity token source** to **method.request.header.Authorization**
- Click **Create** to create the custom authorizer.
- Confirm that you want to allow API Gateway to invoke the custom-authorizer function.



To make the custom authorizer invoke on the GET method, follow these steps:

- Click **Resources** under 24-hour-video
- Click **GET** under /user-profile
- Click **Method Request**
- Click the pencil next to **Authorization**.
- From the dropdown select **custom authorizer** and save.
- Deploy the API again.
    - Click **Actions**
    - Click **Deploy API**
    - Select **dev** as the **Deployment Stage**
    - Click **Deploy**

11. TEST THE SYSTEM

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

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

```
lesson-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:

- Log in to the website by clicking on Login button.
- Click the profile button (it'll have your nickname and, possible, your picture). After a short wait you will see a modal box with your user information.

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.