Exercise: Lambda / IAM - Typescript

In this exercise you will:

1. Write an AWS Lambda function that sends email messages using the AWS Simple Email Service (SES)
2. Create a ZIP file containing your compiled Lambda function
3. Create an AWS IAM role that provides your Lambda function the security permissions it requires.
4. Deploy your Lambda function using your ZIP file and IAM role
5. Test your Lambda function using a test event

Assumptions

You have already configured your AWS credentials on your computer. This should have been done when you installed the AWS CLI on your computer.

Node.js and ts-node are installed on your computer.

Steps

1. By default, to send emails using the Simple Email Service (SES), you must first "verify" the email addresses you intend to send emails to and from (verifying an email address means proving that you actually own the addresses).
    a. Login to the AWS console
    b. Navigate to the Simple Email Service
    c. Select "Email Addresses" on the left side of the SES screen
    d. Verify each of the email addresses you want to send SES emails to and from. Do this by clicking the "Verify a New Email Address" button.

2. When you deploy an AWS Lambda, you must assign it an IAM role that specifies what security permissions the Lambda should have when it runs. For this purpose, create an IAM role that defines the security permissions your Lambda function should have when it executes.
    a. Login to the AWS console
    b. Navigate to the IAM service
    c. On the left of the IAM service screen, select "Roles"
    d. Click the "Create role" button
    e. When asked to "Select type of trusted entity", select "Lambda" and click the Next button
    f. On the "Attach permissions policies" screen, attach the AmazonSESFullAccess policy to your role. This will allow your Lambda function to send emails using the

SES service. You can find the AmazonSESFullAccess role by entering “SES” in the search field.

   g. Also attach the CloudWatchLogsFullAccess policy to your role. This will allow your Lambda function to log messages using the AWS CloudWatch service. After doing that, click the Next button.
   h. Skip the “Add tags” screen by clicking the Next button
   i. On the “Review” screen, enter a name for your role (e.g., cs340lambda or whatever you like), and click the “Create role” button

3. Create a directory that will contain the code for this exercise

4. Create and initialize a new Node.js Typescript project

5. Install the AWS SDK

   ```
   > npm install --save aws-sdk
   ```

6. In a file named EmailRequest.ts, create a Typescript class named “EmailRequest”. In this class, create public properties for the different parts of an email message, including: “to”, “from”, “subject”, “textBody”, “htmlBody”. All of these should be strings.

7. In a file named EmailResult.ts, create a Typescript class named “EmailResult”. In this class, create a “message” property for returning the status of an email send operation, and a “timestamp” property for returning the date/time at which the email was sent. Both of these should be strings.

8. Create a file named index.ts and add the following code to this file. You might need to change the AWS region value if you’re using a region other than us-west-2.

   ```
   import { SES } from 'aws-sdk';
   import { EmailRequest } from './EmailRequest';
   import { EmailResult } from './EmailResult';

   export const handler = async (event: EmailRequest): EmailResult => {
           console.log('Entering send_email');

           let ses = new SES({ region: 'us-west-2' });

           // TODO:
           // Use the SES object to send an email message using
           // the values in the EmailRequest parameter object

           console.log('Leaving send_email');

           // TODO:
   ```

```
        // Return EmailResult
}
```

9. Fill in the code that actually creates and sends an email address using the values contained in the EmailRequest parameter object. Return an EmailResult containing a message indicating the *result* (e.g. “Message was sent”) of the operation and the date/time at which the email was sent. The online documentation for the AWS Javascript SDK can be found here:
    a. https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/

10. Compile and bundle your Lambda code:
    a. Compile your code by running the “tsc” command. The output Javascript files should be in the dist/ directory. If they are not, add "outDir": "./dist/" to your tsconfig.json.
    ```
    > cd <project-directory>
    > tsc
    ```
    b. Copy your project’s node_modules/ directory into the dist/ directory
    ```
    > cp -r node_modules/ dist/
    ```
    c. Create a ZIP file containing the contents of the dist/ directory. (It is important that the ZIP file contain the contents of the dist/ directory, but not the dist/ directory itself):
    ```
    > cd dist/
    > zip -r dist.zip *
    ```

11. Next, deploy your Lambda function.
    a. Login to the AWS console
    b. Navigate to the Lambda service
    c. Select “Functions” on the left side of the Lambda screen
    d. Click the “Create Function” button in the top-right corner
    e. Give your function a name (e.g., send_email)
    f. As your function’s Runtime select “Node.js 8.10”
    g. Under Permissions, click “Choose or create an execution role”
    h. In the “Execution role” drop-down, select “Use an existing role”
    i. In the “Existing role” drop-down, select the name of the IAM role that you created in a previous step (e.g., cs340lambda)
    j. Click the “Create function” button. This will actually create the function, and take you to a screen that lets you further configure the function.
    k. Scroll down to the “Function code” section of the function configuration screen.
    l. In the “Handler” field, specify the name of the file and function that contain your Lambda function handler (e.g., index.handler)
    m. Click the Function package “Upload” button and select the ZIP file created in the previous step (e.g., dist.zip)
    n. Scroll to the top, and click the “Save” button. This will deploy your Lambda function, which means it can now be called

12. Now, test your Lambda function. This can be done as follows:
    a. Scroll to the top and open the “Select a test event” drop-down
    b. Select “Configure test events”. A test event is simply a JSON object that you will send to your Lambda function to see if it works.
    c. Specify a name for your test event (e.g., SendEmail)
    d. Fill in the test JSON object. Its attribute names should match the attribute names in the Typescript EmailRequest interface created in a previous step.
    e. Click the “Create” button at the bottom. This will take you back to the Lambda function configuration screen.
    f. Execute your test event. Scroll to the top, select the name of your test event in the drop-down next to the “Test” button, and click the “Test” button. This will execute your Lambda function with the test event JSON object.
    g. The output of your function will be displayed in the console. For more details, you can click the “logs” link to view the complete CloudWatch logs for your Lambda function

13. Submit your Typescript code through Canvas.