Exercise: API Gateway

In this exercise you will:

1. Create a web API for the Lambda function you wrote to send email in the Lambda/IAM exercise
2. Learn to pass inputs to your Lambda function through the HTTP request body
3. Learning to pass inputs to your Lambda function through the HTTP request URL and headers
4. Test your web API internally by calling it from within the AWS environment
5. Deploy your web API
6. Test your web API externally by calling it from outside the AWS environment using Curl or Postman
7. Generate an SDK that can be used to call your web API from a client program (alternatives are Java, Android, iOS, Javascript, and Ruby)

Assumptions

You have already completed the Lambda/IAM exercise.

Steps

1. Login to the AWS API Gateway Console
    a. https://us-west-2.console.aws.amazon.com/apigateway

2. Create a new web API
    a. Click the Create API button
    b. Select REST
    c. Select New API
    d. Specify API name and description
    e. Select Regional for Endpoint Type

3. Create model definitions for the HTTP request and response bodies your web API will receive and send.
    a. Select “Models” on the left side
    b. Create an EmailRequest model
        i. For “Model name” enter “EmailRequest”
        ii. For “Content type” enter “application/json”
        iii. For “Model description” enter “Email request model”
        iv. For “Model schema” enter the following text. This text is a “JSON schema” that defines the format of an email request JSON object.

            {

```
 "title": "EmailRequest",
 "type": "object",
 "properties": {
  "to": {
   "type": "string",
   "description": "The to email address"
  },
  "from": {
   "type": "string",
   "description": "The from email address"
  },
  "subject": {
   "type": "string",
   "description": "The email's subject"
  },
  "textBody": {
   "type": "string",
   "description": "The plain text email content"
  },
  "htmlBody": {
   "type": "string",
   "description": "The HTML email content"
  }
 }
}
```

v. Click the Create button

c. Create an EmailResult model
   i. For “Model name” enter “EmailResult”
   ii. For “Content type” enter “application/json”
   iii. For “Model description” enter “Email result model”
   iv. For “Model schema” enter the following text. This text is a “JSON schema” that defines the format of an email result JSON object.

```
{
 "title": "EmailResult",
 "type": "object",
 "properties": {
  "message": {
   "type": "string",
   "description": "Operation status message"
  },
  "timestamp": {
```

```
        "type": "string",
        "description": "Operation timestamp"
      }
    }
  }
```

v. Click the Create button

4. Next, we will add some resources to your web API. Resources define the URLs that clients will use when calling your web API operations.
    a. Select “Resources” on the left side.

5. Add the /sendemail resource to your web API.
    a. Select the root resource
    b. In the Actions menu, select Create Resource
    c. Configure as proxy resource: No
    d. Fill in Resource Name: sendemail
    e. Fill in Resource Path: sendemail
    f. Enable API Gateway CORS: Yes
    g. Click the Create Resource button

6. Add a POST method to the /sendemail resource. (This means that clients will use HTTP POST requests to call the /sendemail endpoint.)
    a. Select the /sendemail resource
    b. In the Actions menu, select Create Method
    c. Select POST as the new method type
    d. Click the checkmark to create the method
    e. Integration type: Lambda Function
    f. Use Lambda Proxy Integration: No
    g. Lambda Region: accept default
    h. Lambda Function: select the Lambda function you created in the Lambda/IAM exercise (e.g., send_email)
    i. Use Default Timeout: Yes
    j. Click "OK" when asked if you want to give API Gateway permission to call your Lambda function
    k. You have now created a web API endpoint for calling your send_email Lambda function. This endpoint can be called by using an HTTP POST request that has /sendemail as the URL path and an appropriate JSON object in the HTTP request body (whatever JSON is expected by your send_email Lambda function).

7. Next, configure the models for the HTTP request and response bodies used by the /sendemail POST method.
    a. Click on the “Method Request” link.
    b. Click on “Request Body”.

c. Click “Add model”.
d. For the model’s “Content type”, enter “application/json”
e. For the model’s “Model name”, select “EmailRequest”
f. This tells API Gateway that the request body for the /sendmail POST method should have the format defined by the EmailRequest model you created earlier.
g. Click the “<- Method Execution” link in the top-left corner to go back to the previous screen.
h. Click the “Method Response” link.
i. Open the “200” section by clicking the icon to the left of “200”.
j. In the “Response Body for 200” section, add a response model with a “Content type” of “application/json” and “Models” value of “EmailResult”. (If there is already a response model of type “application/json”, you don’t need to create a new one; just modify its “Models” value to be “EmailResult” by clicking the edit icon.)
k. This tells API Gateway that the response body for the /sendemail POST method should have the format defined by the EmailResult model you created earlier.
l. Click the “<- Method Execution” link in the top-left corner to go back to the previous screen.

8. Do an internal test of your /sendemail endpoint.
    a. Click Test (lightning bolt)
    b. In the "Request Body" field, enter a JSON object containing:

```
{
        "to": "INSERT A TO EMAIL ADDRESS HERE",
        "from": "INSERT A FROM EMAIL ADDRESS HERE",
        "subject": "Test Message",
        "textBody": "This is a test ...",
        "htmlBody": "This is a test ..."
}
```

    c. Click the Test (lightning bolt) button
    d. Look in the "Logs" field to see the log output for the test request.
    e. If all went well, an email should have been sent.

6. Deploy your API. This will make it callable from outside the AWS environment by anyone on the Web.
    a. In the Actions menu, select Deploy API
    b. Select [New Stage]
    c. Give your stage a name (e.g., dev) and description
    d. Click the Deploy button

7. Do an external test of your /sendemail endpoint.
    a. Using Curl, Postman, or an equivalent tool, call your web API from outside the AWS environment. Curl is a command-line program that lets you construct and

send HTTP requests. Postman is a GUI-based tool that lets you do the same thing. You may choose whichever you prefer: command-line or GUI

b. If you want to use Curl, do the following:
    i. Try running the "curl" command in a shell. If you don't already have it installed, you can download it from https://curl.haxx.se/download.html
    ii. Put the request body you want to send into a text file, e.g., data.txt
    iii. Run the following curl command to call your web API:
    curl -d @data.txt -X POST <WEB-API-URL>
    For example,
    curl -d @data.txt -X POST https://gqv3z38u0i.execute-api.us-west-2.amazonaws.com/dev/sendemail

c. If you want to use Postman, do the following:
    i. If you don't have it, you can download Postman here:
    ii. https://www.getpostman.com/downloads/
    iii. In Postman, create a request, select POST as the request type, specify your web API's URL, and in the Body tab select "raw" and enter the request JSON object
    iv. Click the Send button

8. So far we have been specifying the email parameters in a JSON object contained in the HTTP request body. Next, you will learn how to send parameters in the URL and HTTP headers instead of the request body. Specifically, the "to" and "from" email addresses will be specified in the URL, and the subject and body text will be specified in HTTP headers named "EmailSubject" and "EmailText". Note that because we are not passing parameters in through the HTTP request body, we will not define a “model” for the request body. Instead, we will define the URL parameters and HTTP headers used to pass in the email parameters.

9. In the API Gateway console, underneath the /sendemail resource create a sub-resource with the following values:
    a. Configure as proxy resource: No
    b. Resource Name: to
    c. Resource Path: {to}
    d. Enable API Gateway CORS: Yes
    e. The curly braces in the path mean that this part of the URL is variable, not fixed.

10. Underneath the /sendemail/{to} resource create a sub-resource with the following values:
    a. Configure as proxy resource: No
    b. Resource Name: from
    c. Resource Path: {from}
    d. Enable API Gateway CORS: Yes

e. Again, the curly braces in the path mean that this part of the URL is variable, not fixed.

11. Add a POST method to the /sendemail/{to}/{from} resource. (This means that clients will use HTTP POST requests to call the /sendemail/{to}/{from} endpoint.)
    a. Select the /sendemail/{to}/{from} resource
    b. In the Actions menu, select Create Method
    c. Select POST as the new method type
    d. Click the checkmark to create the method
    e. Integration type: Lambda Function
    f. Use Lambda Proxy Integration: No
    g. Lambda Region: accept default
    h. Lambda Function: select the Lambda function you created in the Lambda/IAM exercise (e.g., send_email)
    i. Use Default Timeout: Yes
    j. Click "OK" when asked if you want to give API Gateway permission to call your Lambda function

12. Next, we need to define the HTTP headers that callers should send to the /sendemail/{to}/{from} POST method. We will define a header named “EmailSubject” that will be used to pass in the email’s subject, and a header named “EmailText” that will be used to pass in the email’s body text.
    a. Click the “Method Request” link.
    b. Click on “HTTP Request Headers”.
    c. Click “Add Header” to create a header named “EmailSubject”, and make it required by clicking the Required checkbox.
    d. Click “Add Header” to create a header named “EmailText”, and make it required by clicking the Required checkbox.
    e. This tells API Gateway that calls to the /sendemail/{to}/{from} POST method should contain these two HTTP headers that define the email’s subject and body text.

13. Next, we need to map the {to} and {from} URL parameters, as well as the "EmailSubject" and "EmailText" HTTP headers to a JSON object that will be sent to the Lambda function. This will be accomplished by creating a “mapping template”. Mapping templates describe how to map the parts of an HTTP request (URL parameters, HTTP headers, etc.) to a JSON object that will be passed to the Lambda function.
    a. Click "Integration Request"
    b. Keep the default values for all of the settings
    c. Click on the Mapping Templates section
    d. For "Request body passthrough", select: Never
    e. Click "Add mapping template"

f. For the Content-Type, enter "application/json" (it looks like it's already there, but it isn't). This setting means that this mapping template will only be applied to HTTP requests that contain “application/json” in the HTTP Content-Type header.
g. Enter the following text for the "application/json" template:
h. {
"to": "$input.params('to')",
"from": "$input.params('from')",
"subject": "$input.params('EmailSubject')",
"textBody": "$input.params('EmailText')",
"htmlBody": "$input.params('EmailText')"
}
i. Click the "Save" button
j. Go back to the Method configuration screen by clicking “<- Method Execution” in the top-left corner)

14. Next, configure the models for the HTTP response body used by the /sendemail/{to}/{from} POST method.
    a. Click the “Method Response” link.
    b. Open the “200” section by clicking the icon to the left of “200”.
    c. In the “Response Body for 200” section, add a response model with a “Content type” of “application/json” and “Models” value of “EmailResult”. (If there is already a response model of type “application/json”, you don’t need to create a new one; just modify its “Models” value to be “EmailResult” by clicking the edit icon.)
    d. This tells API Gateway that the response body for the /sendemail/{to}/{from} POST method should have the format defined by the EmailResult model you created earlier.
    e. Click the “<- Method Execution” link in the top-left corner to go back to the previous screen.

15. Do an internal test of your /sendemail/{to}/{from} endpoint.
    a. Click Test (lightning bolt)
    b. In the Path section, enter values for the {from} and {to} email addresses
    c. In the Headers text area, enter the following text:
    EmailSubject: My Email Subject
    EmailText: My Email Text

    d. Click the Test (lightning bolt) button
    f. Look in the "Logs" field to see the log output for the test request.
    g. If all went well, an email should have been sent.

13. Re-deploy your API
    a. Select the stage you created earlier
    b. Click the Deploy button

14. Externally Test Your Web API
    a. Using Curl, Postman, or an equivalent tool, call your web API from outside the AWS environment
    b. If you want to use Curl, do the following:
        i. Run the following curl command to call your web API:
        curl -X POST -H "EmailSubject: My Email Subject" -H "EmailText: My Email Text" <WEB-API-URL>/<TO-EMAIL-ADDRESS>/<FROM-EMAIL-ADDRESS>
        For example,
        curl -X POST -H "EmailSubject: My Email Subject" -H "EmailText: My Email Text" https://gqv3z38u0i.execute-api.us-west-2.amazonaws.com/dev/sendemail/bob@uvnets.com/bob@gmail.com
    c. If you want to use Postman, do the following:
        i. In Postman, create a request, select POST as the request type, specify your web API's URL.
        ii. Include the TO and FROM emails addresses in the URL.
        iii. In the Headers tab, create the following HTTP headers
        KEY: EmailSubject VALUE: My Email Subject
        KEY: EmailText VALUE: My Email Text
        iv. Click the Send button

15. To call your web API from a client program, you need to generate a client SDK for your environment of choice, and incorporate it into your client project. Client SDKs for the following environments can be generated: Java, Android, iOS, Javascript, Ruby
    a. Follow the instructions at the following link to generate and download a client SDK for your preferred environment.
    https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-generate-sdk-console.html
    b. Submit your generated SDK through Canvas to demonstrate that you have completed this exercise.
    c. If you have time, you might want to write a program that uses your client SDK to call the web APIs created in the previous steps.
    d. If you use Typescript, refer to the Generated SDK Help document for better solutions (like actual Typescript support) than the Javascript SDK for your final project.