REST API Lab

This lab has two parts:

1. Create a lambda function
2. Create a REST API with API Gateway

After completing this lab, you will understand the basics of creating a REST API and a Lambda function. That will be very useful in your group projects.

When creating AWS resources in the Learner Lab environment, ALWAYS select "Use an existing role" and select "LabRole." The Learner Lab does not allow for the creation of custom IAM roles.

(NOTE: Lab6.1 in the "Developing with AWS" course shows create a REST API using the Cloud9 IDE and the boto3 client. This lab below shows you an easier way to create resources in AWS by using the AWS web interface.)

Let's start!

Launch the Learner Lab

- In a browser, open the AWS Academy and selct the "Academy Learner Lab" course.
- Select "Modules" and then "Launch AWS Academy Learner Lab"
- Press the button labeled, "Load Launch AWS Academy Learner Lab in a new window"
- Click on "Start Lab"
- Once the "AWS" icon turns Green, click on it.

---

1. Create a Lambda function for Lambda non-proxy integration

In this step, you create a "Hello, World!"-like Lambda function for the Lambda custom integration. Throughout this walkthrough, the function is called `GetStartedLambdaIntegration`.

The Node.js implementation of this `GetStartedLambdaIntegration` Lambda function is as follows:

```
'use strict';

var days = ['Sun', 'Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat'];

var times = ['morning', 'afternoon', 'evening', 'night', 'day'];



console.log('Loading function');



export const handler = function(event, context, callback) {

  // Parse the input for the name, city, time and day property values

  let name = event.name === undefined ? 'you' : event.name;

  let city = event.city === undefined ? 'World' : event.city;

  let time = times.indexOf(event.time)<0 ? 'day' : event.time;

  let day = days.indexOf(event.day)<0 ? null : event.day;



  // Generate a greeting

  let greeting = 'Good ' + time + ', ' + name + ' of ' + city + '. ';

  if (day) greeting += 'Happy ' + day + '!';



  // Log the greeting to CloudWatch

  console.log('Hello: ', greeting);
  // Return a greeting to the caller

  callback(null, {

      "greeting": greeting

  });

};
```

For the Lambda custom integration, API Gateway passes the input to the Lambda function from the client as the integration request body. The `event` object of the Lambda function handler is the input.

Our Lambda function is simple. It parses the input `event` object for the `name`, `city`, `time`, and `day` properties. It then returns a greeting, as a JSON object of `{"message":greeting}`, to the caller. The message is in the `"Good [morning|afternoon|day], [`*`name`*`|you] in [`*`city`*`|World]. Happy `*`day`*`!"` pattern. It is assumed that the input to the Lambda function is of the following JSON object:

```
{

  "city": "...",

  "time": "...",

  "day": "...",

  "name" : "..."

}
```

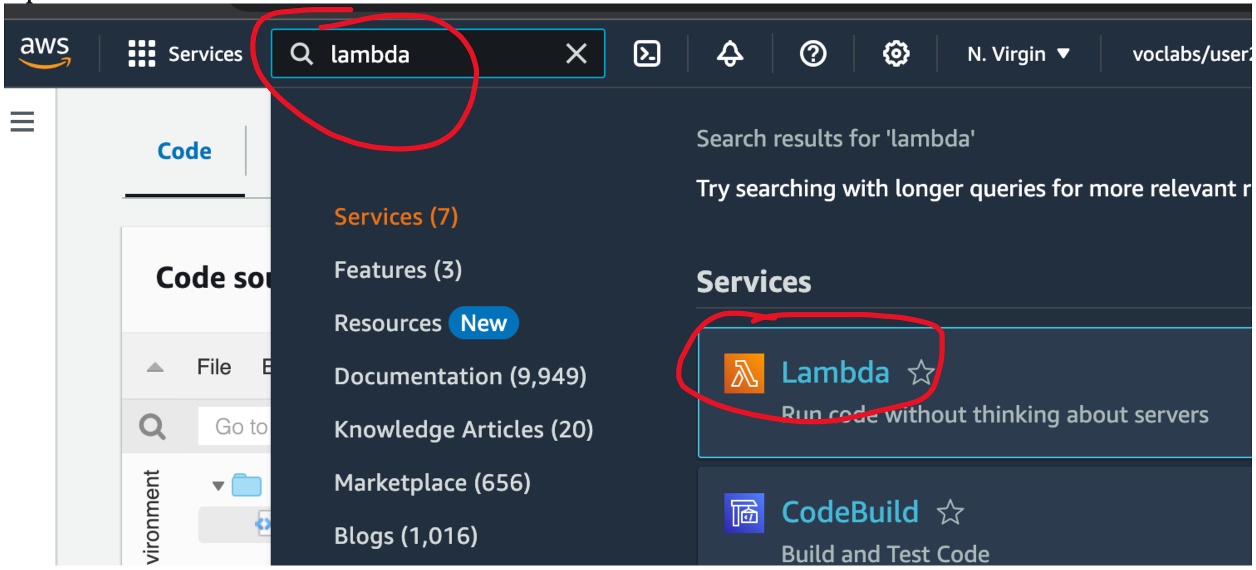
For more information, see the AWS Lambda Developer Guide.

In addition, the function logs its execution to Amazon CloudWatch by calling `console.log(...)`. This is helpful for tracing calls when debugging the function. (At the end of this lab, you will see how this message is written to CloudWatch.)

Now, create the `GetStartedLambdaIntegration` Lambda function.

To create the GetStartedLambdaIntegration Lambda function for Lambda custom integration

1. Open "Lambda" from the Services menu.



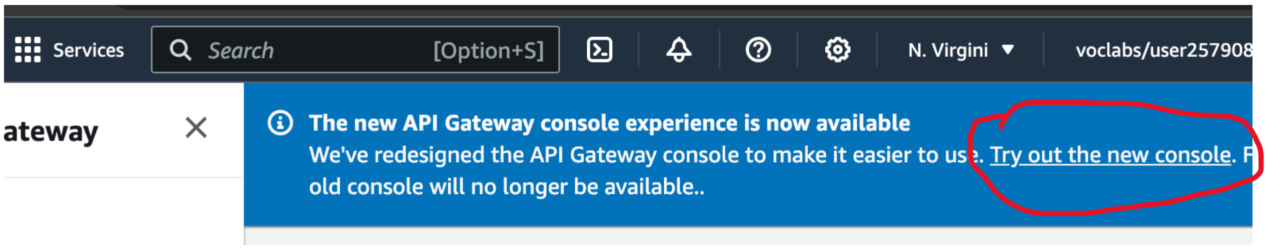
2. Do one of the following:
   - If the welcome page appears, choose **Get Started Now** and then choose **Create function**.
   - If the **Lambda > Functions** list page appears, choose **Create function**.
3. Choose **Author from scratch**.
4. In the **Author from scratch** pane, do the following:
   - For **Name**, enter `GetStartedLambdaIntegration` as the Lambda function name.
   - For **Execution role**, "Use an existing role" and select "LabRole.".
   - Choose **Create function**.
5. In the **Configure function** pane, under **Function code** do the following:
   - Copy the Lambda function code listed in the beginning of this section and paste it in the inline code editor.
   - Leave the default choices for all other fields in this section.
   - Choose **Deploy**.
6. To test the newly created function, choose **Configure test events** from **Select a test event...**.
   - For **Create new event**, replace any default code statements with the following, enter `HelloWorldTest` for the event name, and choose **Create**.

```
{
  "name": "Jonny",

  "city": "Seattle",

  "time": "morning",

  "day": "Wednesday"

}
```

- Choose **Test** to invoke the function. The **Execution result: succeeded** section is shown. Expand **Detail** and you see the following output.

```
{ "greeting": "Good morning, Jonny of Seattle. Happy
Wednesday! }
```

Note: switch the "new console". The screenshots from API Gateway below are from the "new console."



The new API Gateway console experience is now available
We've redesigned the API Gateway console to make it easier to us . Try out the new console. F
old console will no longer be available..

---

2. Create an API with Lambda non-proxy integration

With the Lambda function (`GetStartedLambdaIntegration`) created and tested, you are ready to expose the function through an API Gateway API. For illustration purposes, we expose the Lambda function with a generic HTTP method. We use the request body, a URL path variable, a query string, and a header to receive required input data from the client. We turn on the API Gateway request validator for the API to ensure that all of the required data is properly defined and specified. We configure a mapping template for API Gateway to transform the client-supplied request data into the valid format as required by the backend Lambda function.

NOTE: The screenshots below show the "new"

To create an API with Lambda custom integration with a Lambda function

1. Select "API Gateway" from the Services menu.
2. If this is your first time using API Gateway, you see a page that introduces you to the features of the service. Under **REST API**, choose **Build.** When the **Create Example API** popup appears, choose **OK**.

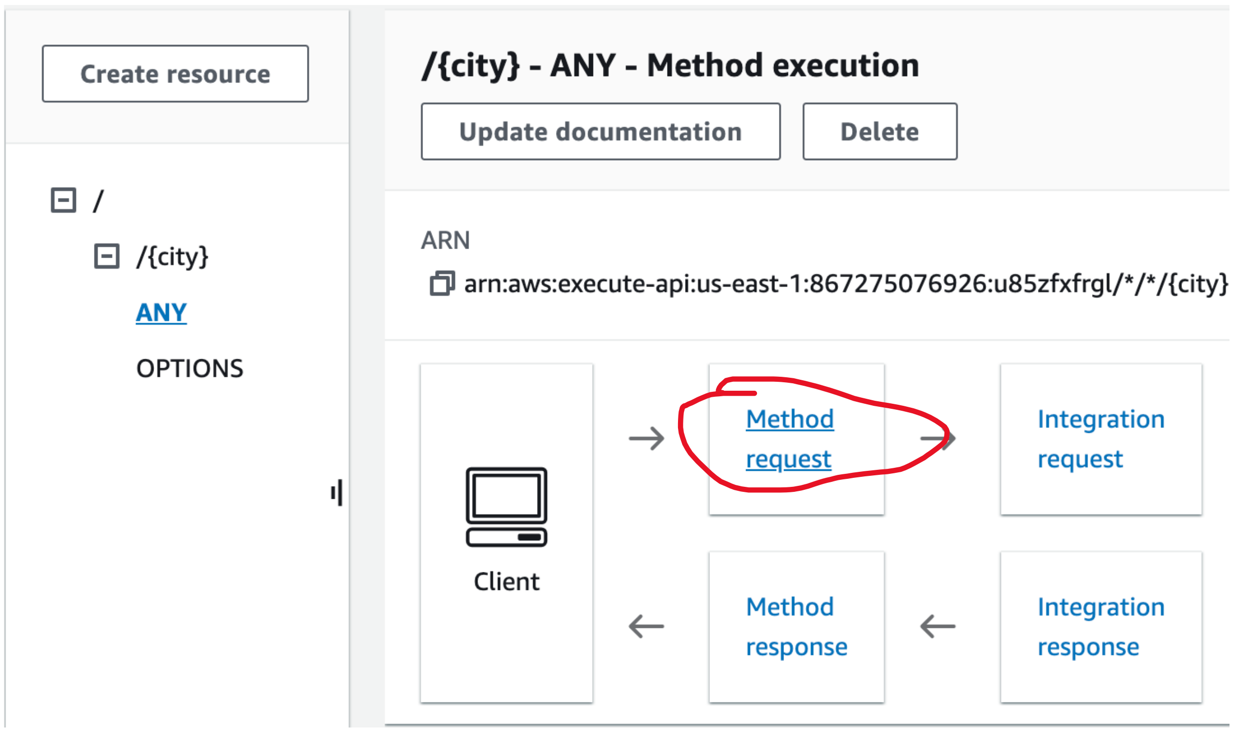
   If this is not your first time using API Gateway, choose **Create API**. Under **REST API**, choose **Build**. (Do NOT choose "REST API Private.")
   a. Choose **New API**
   b. Enter a name in **API Name**.
   c. Optionally, add a brief description in **Description**.
   d. Choose **Create API**.
3. Choose the root resource (`/`) under **Resources**. From the **Actions** menu, choose **Create Resource**.
   a. Type `city` for **Resource Name**.
   b. Replace **Resource Path** with `{city}`. This is an example of the templated path variable used to take input from the client. Later, we show how to map this path variable into the Lambda function input using a mapping template.
   c. Select the **Enable API Gateway Cors** option.
   d. Choose **Create Resource**.
4. With the newly created `/{city}` resource highlighted, choose **Create Method** from **Actions**.
   a. Choose `ANY` from the HTTP method drop-down menu. The `ANY` HTTP verb is a placeholder for a valid HTTP method that a client submits at run time. This example shows that `ANY` method can be used for Lambda custom integration as well as for Lambda proxy integration.
   b. To save the setting, choose the check mark.
5. In **Method Execution**, for the `/{city} ANY` method, do the following:
   a. Choose **Lambda Function** for `Integration type`.
   b. Leave the **Use Lambda Proxy integration** box unchecked.
   c. Choose the region where you created the Lambda function; for example, `us-west-2`.
   d. Type the name of your Lambda function in **Lambda Function**; for example, `GetStartedLambdaIntegration`.
   e. Leave the **Use Default timeout** box checked.

f. Choose **Save**.
g. Choose **OK** in the **Add Permission to Lambda Function** popup to have API Gateway set up the required access permissions for the API to invoke the integrated Lambda function.

6. In this step you'll configure the following:
   a. A query string parameter (`time`)
   b. A header parameter (`day`)
   c. A payload property (`callerName`)

   At run time, the client can use these request parameters and the request body to provide time of the day, the day of the week, and the name of the caller. You already configured the /{city} path variable.

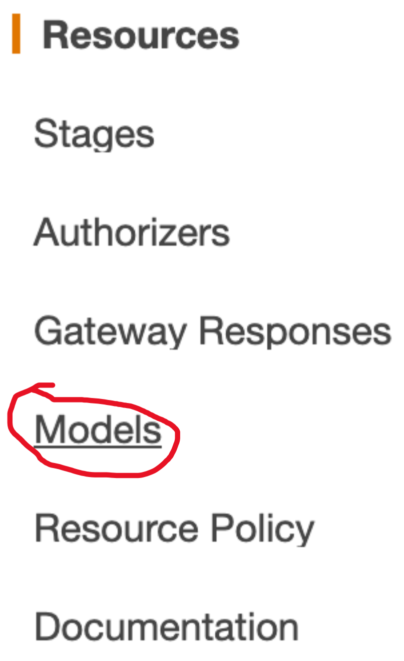
   d. In the **Method Execution** pane, choose **Method Request**.



   e. Expand the **URL Query String Parameters** section. Choose **Add query string**. Type `time` for **Name**. Click the checkmark. Select the **Required** option. You can ignore the warning message to have a request validator. Leave **Caching** cleared to avoid an unnecessary charge for this exercise.
   f. Expand the **HTTP Request Headers** section. Choose **Add header**. Type `day` for **Name**. Click the checkmark. Select the **Required** option. You

can ignore the warning message to have a request validator.
Leave **Caching** cleared to avoid an unnecessary charge for this exercise.

g. To define the method request payload, do the following:
   i. To define a model, choose **Models** under the API from the API Gateway primary navigation pane, and then choose **Create**.



   ii. Type `GetStartedLambdaIntegrationUserInput` for **Model name**.
   iii. Type `application/json` for **Content type**.
   iv. Leave the **Model description** blank.
   v. Copy the following schema definition into the **Model schema** editor:

```
{

  "$schema": "http://json-schema.org/draft-04/schema#",

  "title": "GetStartedLambdaIntegrationUserInput",

  "type": "object",

  "properties": {

    "callerName": { "type": "string" }

  }
}
```

vi. Choose **Create model** to finish defining the input model.

vii. Choose **Resources**, choose the `/{city} ANY` method, choose **Method Request**, and expand **Request body**. Choose **Add model**. Type `application/json` for **Content type**. Choose `GetStartedLambdaIntegrationUserInput` for **Model name**. Choose the check-mark icon to save the setting.

7. Choose the `/{city} ANY` method and choose **Integration Request** to set up a body-mapping template. In this step you'll map the previously configured method request parameter of `nameQuery` or `nameHeader` to the JSON payload, as required by the backend Lambda function:
    a. Expand the **Mapping Templates** section. Choose **Add mapping template**. Type `application/json` for **Content-Type**. Choose the check-mark icon to save the setting.
    b. In the pop-up that appears, choose **Yes, secure this integration**.
    c. Check the recommended `When there are no templates defined` for **Request body passthrough**.
    d. Choose `GetStartedLambdaIntegrationUserInput` from **Generate template** to generate an initial mapping template. This option is available because you defined a model schema, without which you would need to write the mapping template from scratch.
    e. Replace the generated mapping script in the mapping template editor with the following:

```
#set($inputRoot = $input.path('$'))

{

  "city": "$input.params('city')",

  "time": "$input.params('time')",

  "day":  "$input.params('day')",

  "name": "$inputRoot.callerName"

}
```

f. Choose **Save**.

---

3. Test invoking the API method

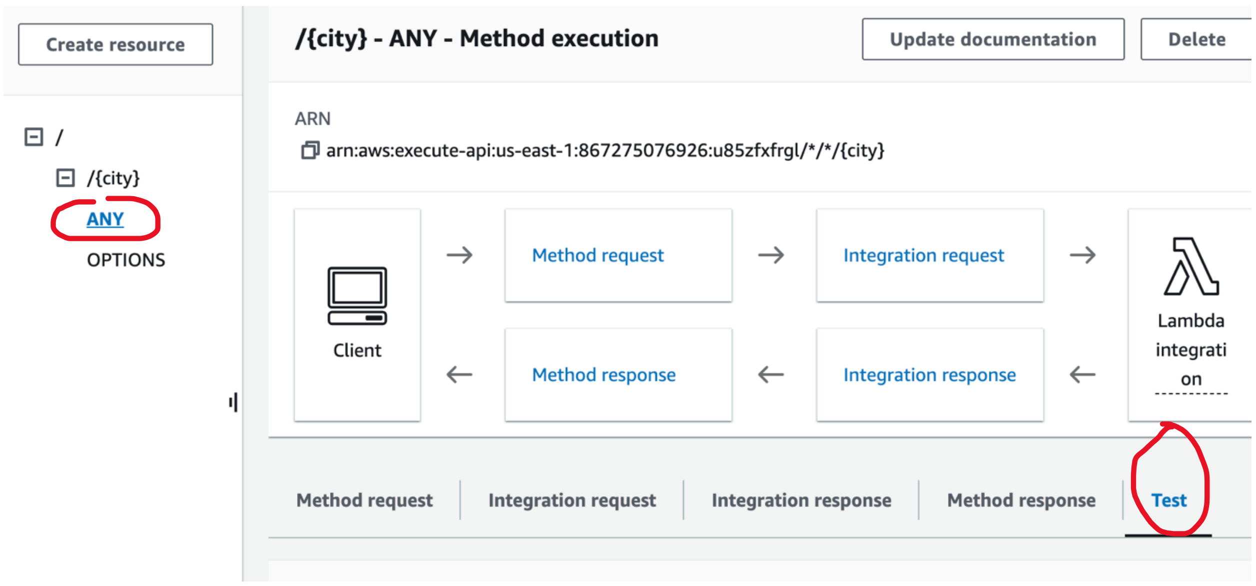
The API Gateway console provides a testing facility for you to test invoking the API before it is deployed. You use the Test feature of the console to test the API by submitting the following request:

```
POST /Seattle?time=morning
day:Wednesday


{
    "callerName": "John"
}
```

In this test request, you'll set `ANY` to `POST`, set `{city}` to `Seattle`, assign `Wednesday` as the `day` header value, and assign `"John"` as the `callerName` value.

To test-invoke the `ANY /{city}` method

1. In **Method Execution**, choose **Test**.



ARN
arn:aws:execute-api:us-east-1:867275076926:u85zfxfrgl/*/*/{city}

2. Choose `POST` from the **Method** drop-down list.
3. In **Path**, city field, type `Seattle`.
4. In **Query Strings**, type `time=morning`.
5. In **Headers**, type `day:Wed`.
6. In **Request Body**, type `{ "callerName":"John" }`
7. Choose **Test**. (Note: you may have to scroll down to see the resulting output.)
8. Verify that the returned response payload is as follows:

```
{ "greeting": "Good morning, John of Seattle. Happy
Wednesday!" }
```

9. You can also view the logs to examine how API Gateway processes the request and response. The logs output should look similar to the following:

```
Execution log for request test-request

Thu Aug 31 01:07:25 UTC 2017 : Starting execution for request:
test-invoke-request

Thu Aug 31 01:07:25 UTC 2017 : HTTP Method: POST, Resource
Path: /Seattle

Thu Aug 31 01:07:25 UTC 2017 : Method request path:
{city=Seattle}

Thu Aug 31 01:07:25 UTC 2017 : Method request query string:
{time=morning}

Thu Aug 31 01:07:25 UTC 2017 : Method request headers:
{day=Wednesday}

Thu Aug 31 01:07:25 UTC 2017 : Method request body before
transformations: { "callerName": "John" }

Thu Aug 31 01:07:25 UTC 2017 : Request validation succeeded
for content type application/json
```

```
Thu Aug 31 01:07:25 UTC 2017 : Endpoint request URI:
https://lambda.us-west-2.amazonaws.com/2015-03-
31/functions/arn:aws:lambda:us-west-
2:123456789012:function:GetStartedLambdaIntegration/invocation
s

Thu Aug 31 01:07:25 UTC 2017 : Endpoint request headers: {x-
amzn-lambda-integration-tag=test-request,
Authorization=************************************************
**************************************************************
**************************************************************
**************************************************************
**************************************************************
********************************338c72, X-Amz-
Date=20170831T010725Z, x-amzn-apigateway-api-id=beags1mnid, X-
Amz-Source-Arn=arn:aws:execute-api:us-west-
2:123456789012:beags1mnid/null/POST/{city},
Accept=application/json, User-
Agent=AmazonAPIGateway_beags1mnid, X-Amz-Security-
Token=FQoDYXdzELL//////////wEaDMHGzEdEOT/VvGhabiK3AzgKrJw+3zLq
JZG4PhOq12K6W21+QotY2rrZyOzqhLoiuRg3CAYNQ2eqgL5D54+63ey9bIdtwH
GoyBdq8ecWxJK/YUnT2Rau0L9HCG5p7FC05h3IvwlFfvcidQNXeYvsKJTLXI05
/yEnY3ttIAnpNYLOezD9Es8rBfyruHfJfOqextKlsC8DymCcqlGkig8qLKcZ0h
WJWVwiPJiFgL7laabXs++ZhCa4hdZo4iqlG729DE4gaV1mJVdoAagIUwLMo+y4
NxFDu0r7I0/EO5nYcCrppGVVBYiGk7H4T6sXuhTkbNNqVmXtV3ch5bOlh7
[TRUNCATED]

Thu Aug 31 01:07:25 UTC 2017 : Endpoint request body after
transformations: {

  "city": "Seattle",

  "time": "morning",

  "day": "Wednesday",

  "name" : "John"

}
```

```
Thu Aug 31 01:07:25 UTC 2017 : Sending request to
https://lambda.us-west-2.amazonaws.com/2015-03-
31/functions/arn:aws:lambda:us-west-
2:123456789012:function:GetStartedLambdaIntegration/invocation
s

Thu Aug 31 01:07:25 UTC 2017 : Received response. Integration
latency: 328 ms

Thu Aug 31 01:07:25 UTC 2017 : Endpoint response body before
transformations: {"greeting":"Good morning, John of Seattle.
Happy Wednesday!"}

Thu Aug 31 01:07:25 UTC 2017 : Endpoint response headers: {x-
amzn-Remapped-Content-Length=0, x-amzn-RequestId=c0475a28-
8de8-11e7-8d3f-4183da788f0f, Connection=keep-alive, Content-
Length=62, Date=Thu, 31 Aug 2017 01:07:25 GMT, X-Amzn-Trace-
Id=root=1-59a7614d-373151b01b0713127e646635;sampled=0,
Content-Type=application/json}

Thu Aug 31 01:07:25 UTC 2017 : Method response body after
transformations: {"greeting":"Good morning, John of Seattle.
Happy Wednesday!"}

Thu Aug 31 01:07:25 UTC 2017 : Method response headers: {X-
Amzn-Trace-Id=sampled=0;root=1-59a7614d-
373151b01b0713127e646635, Content-Type=application/json}

Thu Aug 31 01:07:25 UTC 2017 : Successfully completed
execution

Thu Aug 31 01:07:25 UTC 2017 : Method completed with status:
200
```

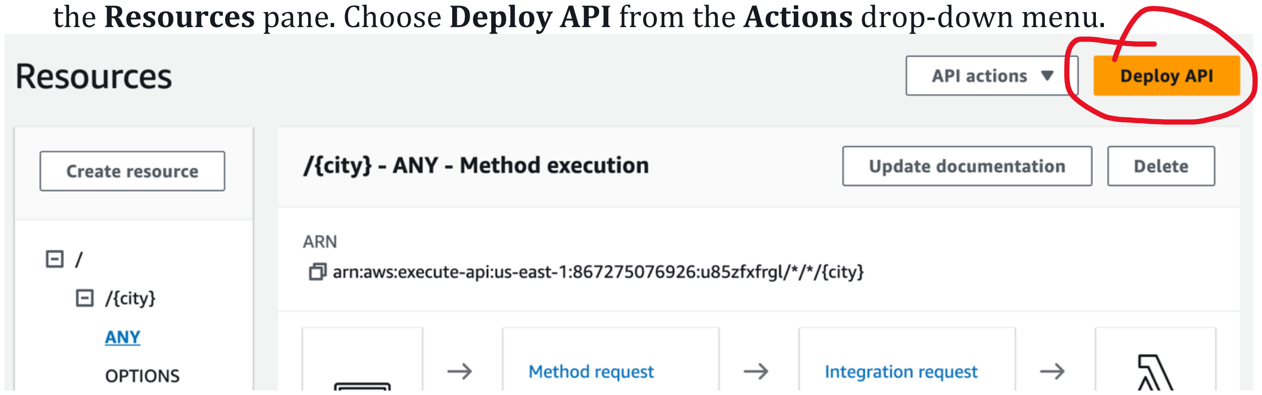
The logs show the incoming request before the mapping and the integration request after the mapping. When a test fails, the logs are useful for evaluating whether the original input is correct or the mapping template works correctly.

4. Deploy the API

The test invocation is a simulation and has limitations. For example, it bypasses any authorization mechanism enacted on the API. To test the API execution in real time, you must deploy the API first. To deploy an API, you create a stage to create a snapshot of the API at that time. The stage name also defines the base path after the API's default host name. The API's root resource is appended after the stage name. When you modify the API, you must redeploy it to a new or existing stage before the changes take effect.

To deploy the API to a stage

1. Choose the API from the **APIs** pane or choose a resource or method from the **Resources** pane. Choose **Deploy API** from the **Actions** drop-down menu.



ARN
arn:aws:execute-api:us-east-1:867275076926:u85zfxfrgl/*/*/{city}

2. For **Deployment stage**, choose **New Stage**.
3. For **Stage name**, type a name; for example, `test`.

   Note

   The input must be UTF-8 encoded (i.e., unlocalized) text.
4. For **Stage description**, type a description or leave it blank.
5. For **Deployment description**, type a description or leave it blank.
6. Choose **Deploy**. After the API is successfully deployed, you see the API's base URL (the default host name plus the stage name) displayed as **Invoke URL** at the top of the **Stage Editor**. The general pattern of this base URL is `https://`*`api-id`*`.`*`region`*`.amazonaws.com/`*`stageName`*. For example, the base URL of the API (`beags1mnid`) created in the `us-west-2` region and deployed to the `test` stage is `https://beags1mnid.execute-api.us-west-2.amazonaws.com/test`.

5. Test the API in a deployment stage

There are several ways you can test a deployed API. For GET requests using only URL path variables or query string parameters, you can type the API resource URL in a browser. For other methods, you must use more advanced REST API testing utilities, such as POSTMAN or cURL.

To test the API using cURL

1. Open a terminal window on your local computer connected to the internet.
2. To test `POST /Seattle?time=evening`:

   Copy the following cURL command and paste it into a text editor (NOT a word processor. In other words, do not use MS Word.) (Note: Windows users may not have the curl command installed if you are using Windows 10, version 1803 or earlier. Windows may need to install curl or use git bash.)

   Replace the path highlited below with your Invoke URI.

   Unix / Mac:

   ```
   curl -v -X POST 'https://u7xf5zfral.execute-api.us-east-1.amazonaws.com/test/Seattle?time=evening' \
     -H 'content-type: application/json' \
     -H 'day: Thursday' \
     -H 'x-amz-docs-region: us-east-1' \
     -d '{
        "callerName": "John"
   }'
   ```

   Windows

   ```
   curl -v -X POST 'https://u7xf5zfral.execute-api.us-east-1.amazonaws.com/test/Seattle?time=evening' ^
     -H 'content-type: application/json' ^
     -H 'day: Thursday' ^
     -H 'x-amz-docs-region: us-east-1' ^
     -d '{
        "callerName": "John"
   ```

```
}'
```

You should get a successful response with the following payload:

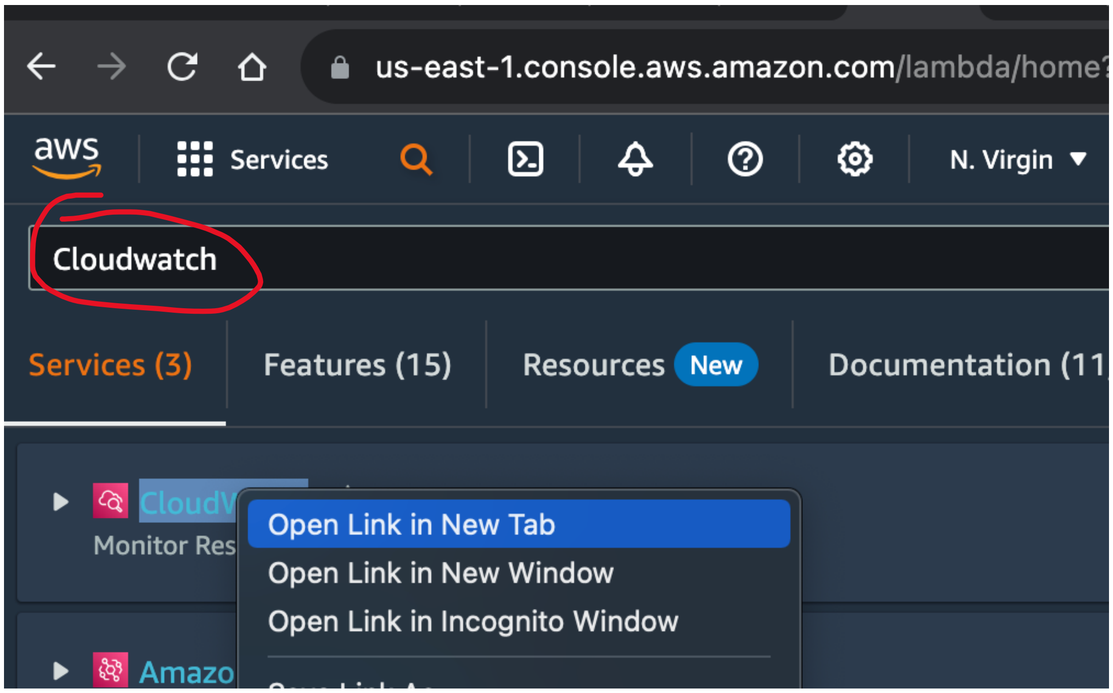
```
{"greeting":"Good evening, John of Seattle. Happy
Thursday!"}
```

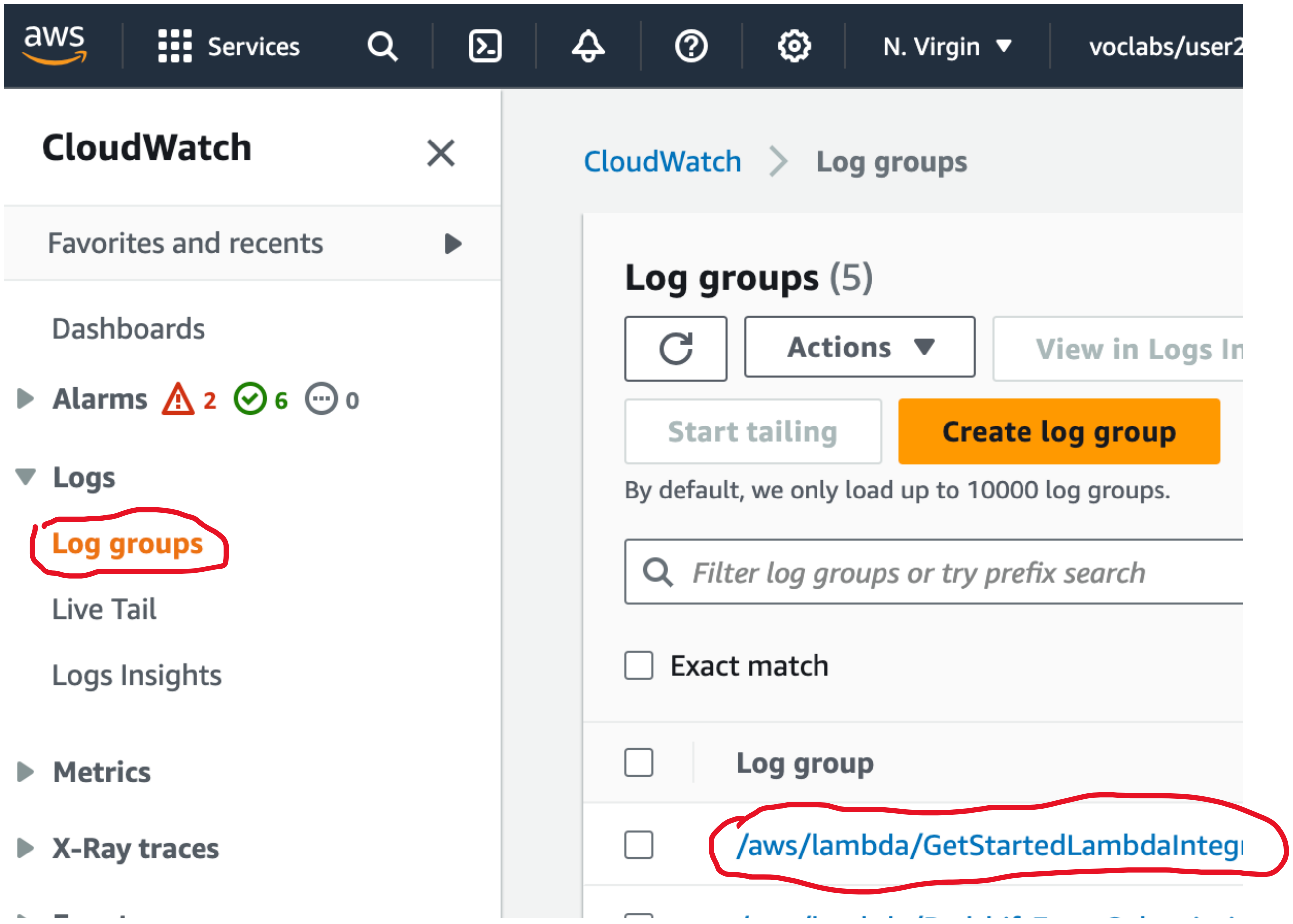
If you change `POST` to `PUT` in this method request, you get the same response.

---

6. CloudWatch logs

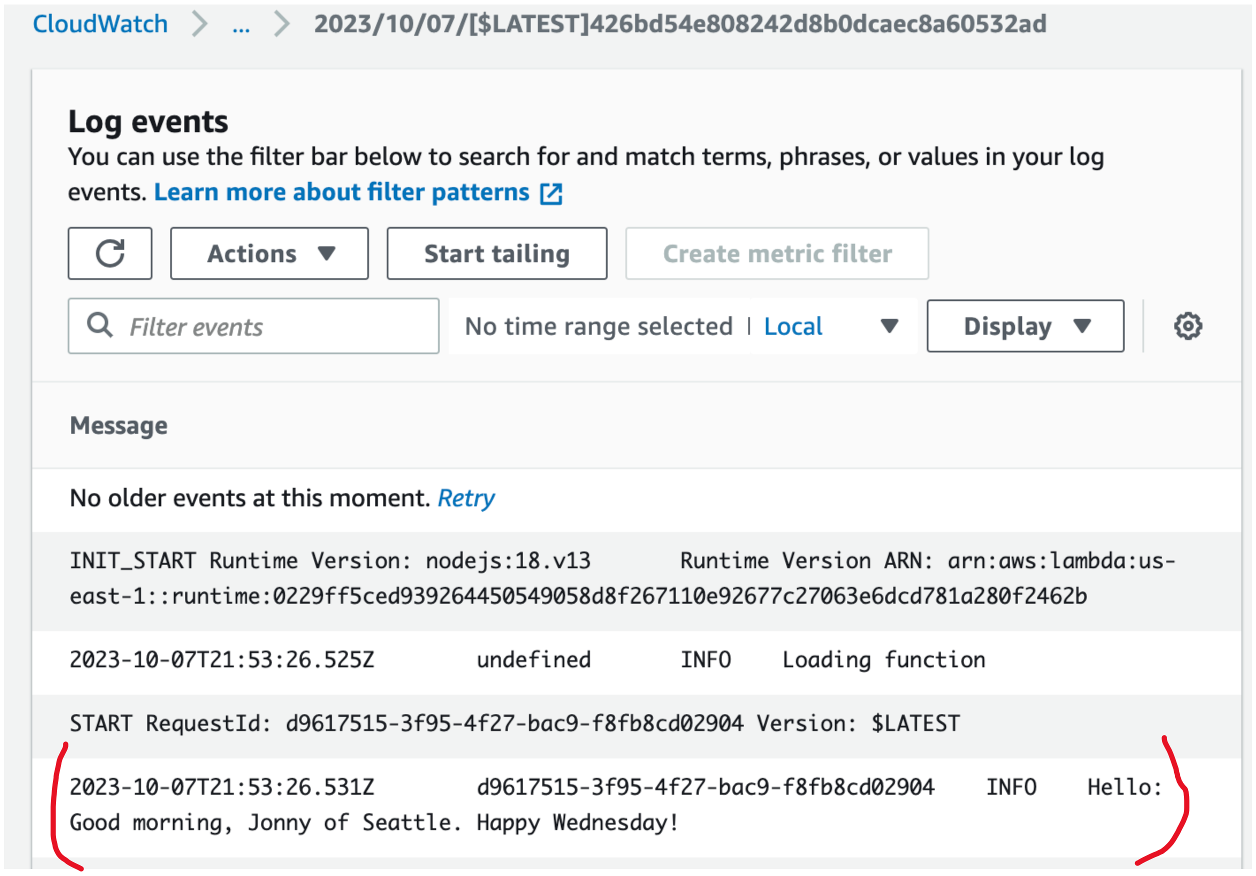
The output from the lambda code is written to CloudWatch Logs.





By default, we only load up to 10000 log groups.

Click on the "GetStartedLambdaIntegration" (as shown above). Scroll to the bottom and click on the most recent "Log Stream" listed. Can you find your console.log message in the logs?


CloudWatch > ... > 2023/10/07/[$LATEST]426bd54e808242d8b0dcaec8a60532ad
Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
Actions
Start tailing
Create metric filter
Filter events
No time range selected | Local
Display
Message
No older events at this moment. Retry

```
INIT_START Runtime Version: nodejs:18.v13       Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:0229ff5ced939264450549058d8f267110e92677c27063e6dcd781a280f2462b

2023-10-07T21:53:26.525Z        undefined       INFO    Loading function

START RequestId: d9617515-3f95-4f27-bac9-f8fb8cd02904 Version: $LATEST

2023-10-07T21:53:26.531Z        d9617515-3f95-4f27-bac9-f8fb8cd02904    INFO    Hello:
Good morning, Jonny of Seattle. Happy Wednesday!
```

You can use console.log message and CloudWatch to debug your lambda functions.

Optional: Try creating a lambda function with the same functionality above, but implement it in Python instead of NodeJS. Trying creating a new API Gateway REST API and connect it to your Python Lambda function.

Note: These instructions are taken from an AWS Tutotial. However, that tutorial discusses IAM Roles, which are not available to the Learner Lab. For that reason, I have copied/pasted/edited the tutorial above.