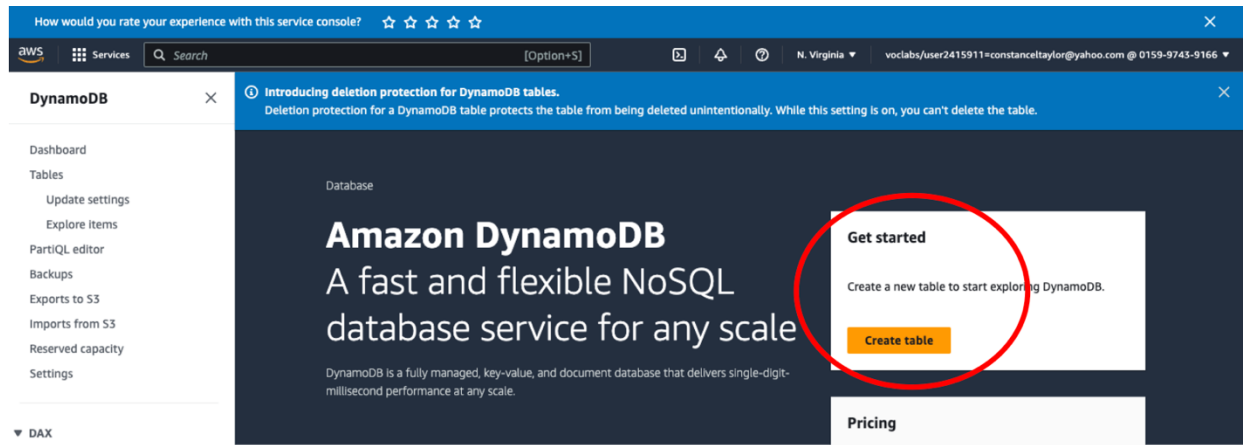


CPSC 3720: Creating an API in AWS Homework Assignment

This assignment should reinforce the concepts reviewed in class and assist you with Sprints 2- 4. For this homework you will create an example API in AWS using the following AWS Services provided in the AWS Console-→Services: 1) Lambda, 2) API Gateway, and 3) DynamoDB. You will test your API in Postman.



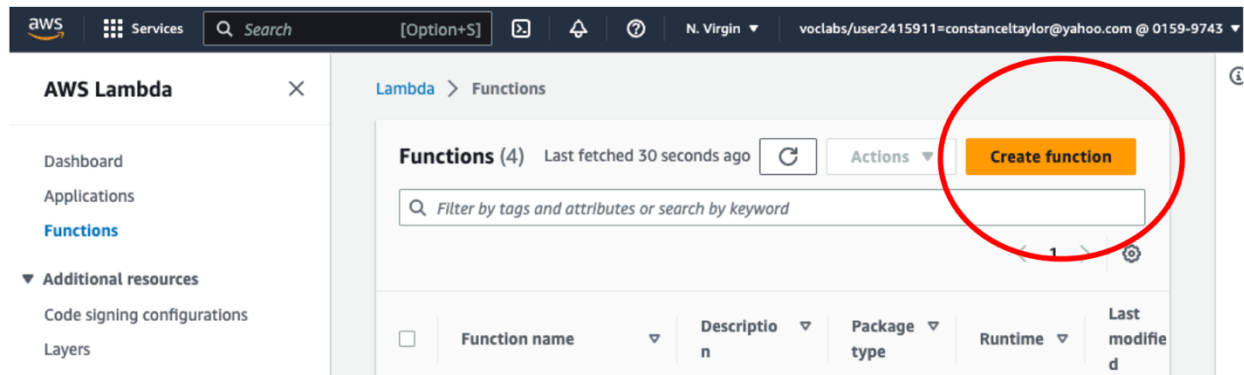
- Name it “demoapi” and give it a partition key called “id” with type Number

A screenshot of the AWS 'Create table' wizard. The 'Table name' field contains 'demoapi'. The 'Partition key' section shows 'id' as the key name and 'Number' as the data type. The 'Sort key - optional' section is empty. The 'Table settings' section has 'Default settings' selected. At the bottom, there are 'Cancel' and 'Create table' buttons, with the 'Create table' button circled in red.

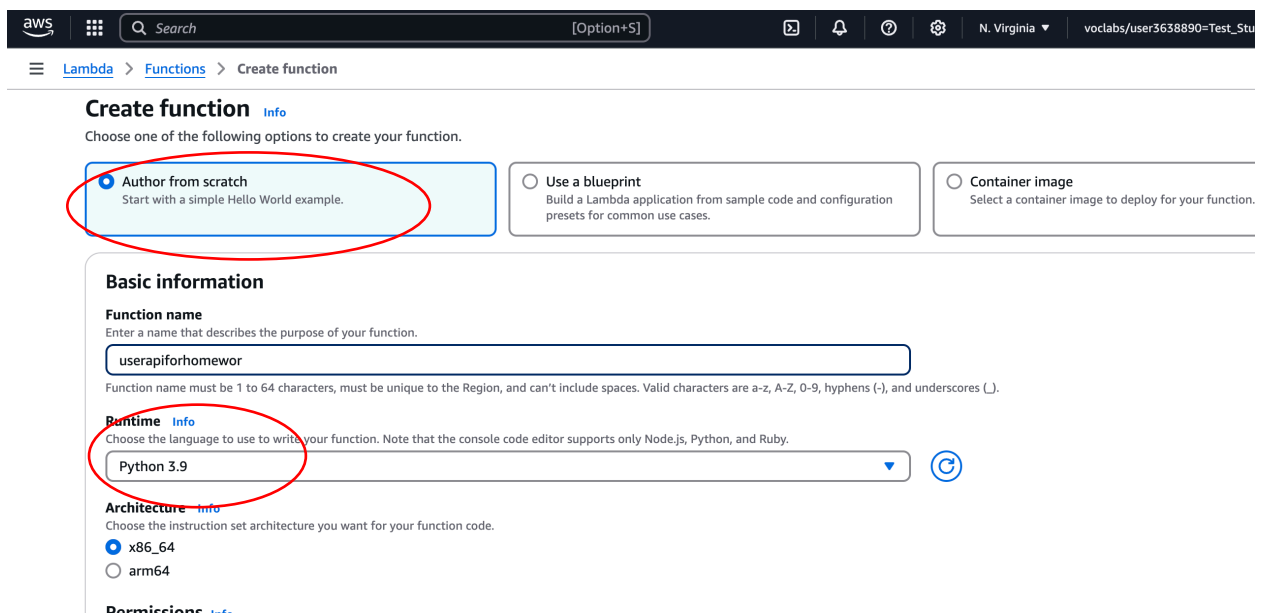
- No other changes need; just scroll to bottom and create table

2. Create a Lambda function (you can use either python or node to build the application)

Lambda function with Python code:



- Author for Scratch Python runtime 3.9x version!
- Select Use existing Role and type in LabRole



Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions
☒ Use an existing role
☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

LabRole [View the LabRole role](#) on the IAM console.

► **Additional Configurations**
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

[Cancel](#) [Create function](#)

(OR) Lambda function with Python code:

- Author for Scratch Node js runtime **18.x version!**
- Select Use existing Role and type in LabRole

[Lambda](#) > [Functions](#) > Create function

Create function [Info](#)
Choose one of the following options to create your function.

☒ **Author from scratch**
Start with a simple Hello World example.
 ☐ **Use a blueprint**
Build a Lambda application from sample code and configuration presets for common use cases.
 ☐ **Container image**
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.
userapiforhomework
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 18.x

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
☒ x86_64
☐ arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions
☒ Use an existing role
☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

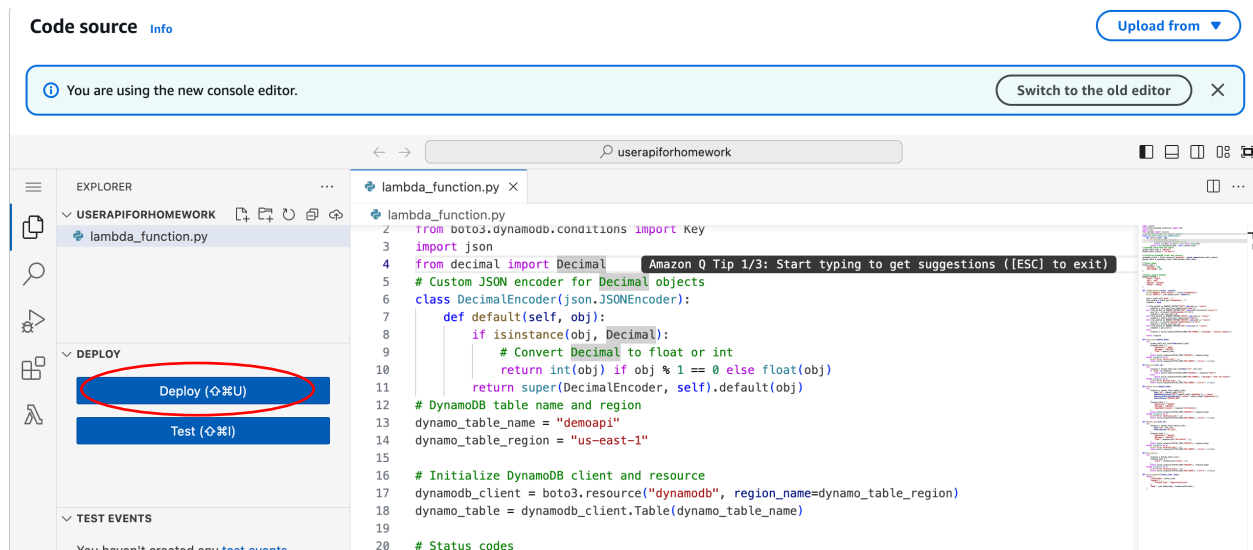
LabRole [View the LabRole role](#) on the IAM console.

► **Advanced settings**

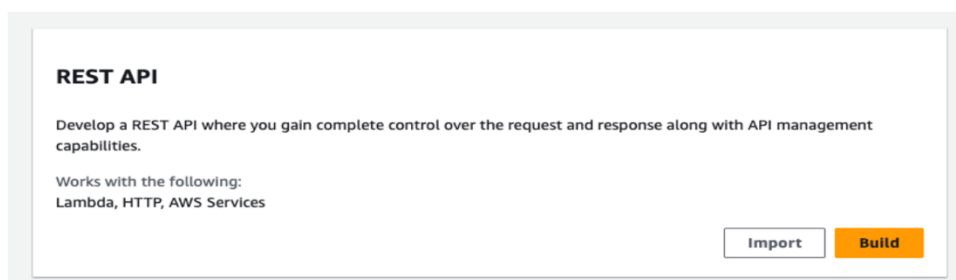
[Cancel](#) [Create function](#)

- Scroll to the bottom and select **Create Function**.

Copy in the code from index_python.py or indexv18node file provided with this assignment into the **code tab** in Lambda and then **deploy** the code. Anytime you change the code it will require a redeploy.



Create API with API Gateway a new REST API (not private)



- Select Import an API from a definition and use the AWSHomework.yaml file provided with the assignment to import.
- Also selection – Ignore Warnings on the bottom of the window

API Gateway > APIs > Create API > Create REST API

Create REST API

☐ New API
Create a new REST API.

☐ Clone existing API
Create a copy of an API in this AWS account.

☒ Import API
Import an API from an OpenAPI definition.

☐ Example API
Learn about API Gateway with an example API.

API definition
Upload an OpenAPI file, or paste the definition into the field below.

```

1 openapi: 3.0.0
2 info:
3   version: 1.0.0
4   title: 3720 Class- Users API
5   description: A users API for homework concepts
6
7 servers:
8   - url: https://users.api/v1
9
10 security:
11   - BasicAuth: []
12
13 paths:
14   /users:
15     get:
16       description: Returns all users
17       responses:
18         "200":
19           description: Successfully returned a list of
20             users
21           content:
22             application/json:
23               schema:
24                 type: array

```

Code editor supports JSON or YAML files

API endpoint type
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.

Warnings

☐ Fail on warnings
☒ Ignore warnings

- Create API on the bottom.

Now you have all the Resources and methods from the file ready to connect to the Lambda functions!

☑ Successfully created REST API '3720 Class- Users API (i2k2p41i1b)'

API Gateway > APIs > Resources - 3720 Class- Users API (i2k2p41i1b)

Resources

DELETE
GET
PATCH
POST
 GET

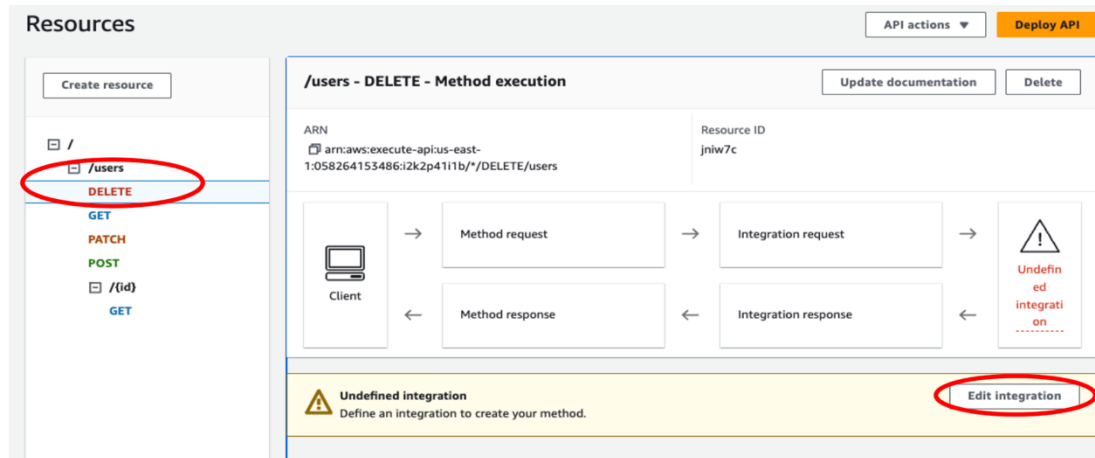
Resource details

Path: /users Resource ID: jniw7c

Methods (4)

	Method type	Integration type	Authorization	API key
<input type="radio"/>	DELETE	Not setup	None	Not required
<input type="radio"/>	GET	Not setup	None	Not required
<input type="radio"/>	PATCH	Not setup	None	Not required
<input type="radio"/>	POST	Not setup	None	Not required

- For each method of your two path (aka resources) you will need to connect them to the Lambda function. For example, to connect the /users Delete method to the Lambda function, you select Delete on the left Nav, and you will see the following screen:



- Select **Edit Integration** and you will see the following screen which you will choose Lambda Function Integration, Lambda Proxy Integration, and then choose the Lambda function you completed in Step 2 previously:

Edit integration request

Method details

Integration type

☒ **Lambda function**

Integrate your API with a Lambda function.



☐ **HTTP**

Integrate with an existing HTTP endpoint.



☐ **Mock**

Generate a response based on API Gateway mappings and transformations.



☐ **AWS service**

Integrate with an AWS Service.



☐ **VPC link**

Integrate with a resource that isn't accessible over the public internet.



☒ **Lambda proxy integration**

Send the request to your Lambda function as a structured event.

Lambda function

Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1

i Grant API Gateway permission to invoke your Lambda function. To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.

Execution role

arn:aws:iam::myAccount:role/myRole

Credential cache

Do not add caller credentials to cache key

☒ **Default timeout**

The default timeout is 29 seconds.

Cancel

Save

- Select Save when and you should be presented a screen similar to the following:

/users - DELETE - Method execution

Update documentation

Delete

ARN
arn:aws:execute-api:us-east-1:058264153486:i2k2p41i1b/*/DELETE/users

Resource ID
jniw7c

Client

Method request

Integration request

Method response

Integration response
Proxy integration

Lambda integration

Method request

Integration request

Integration response

Method response

Test

Integration request settings

Edit

Integration type [Info](#)
Lambda

Region
us-east-1

Lambda proxy integration [Info](#)
True

Lambda function
userapidemo18

Timeout
Default (29 seconds)

URL path parameters (0)

- Repeat the lambda integration for all methods for both paths: *users* and *users/{id}*
- Now it's time to deploy the API so you can test it out! Once you have completed your integration to the Lambda function for all 5 methods, go select the */users* root of your API and Deploy API.

Resources

Create resource

/

/users

DELETE

GET

PATCH

POST

/id

GET

Resource details

Delete Update documentation Enable CORS

Path /users Resource ID jniw7c

Methods (4)

Delete Create method

	Method type ▲	Integration type ▼	Authorization ▼	API key ▼
<input type="radio"/>	DELETE	Lambda	None	Not required
<input type="radio"/>	GET	Not setup	None	Not required
<input type="radio"/>	PATCH	Not setup	None	Not required
<input type="radio"/>	POST	Not setup	None	Not required

- Create a new Stage environment for your deployment and call it dev, then Deploy:

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage [New Stage] ▼

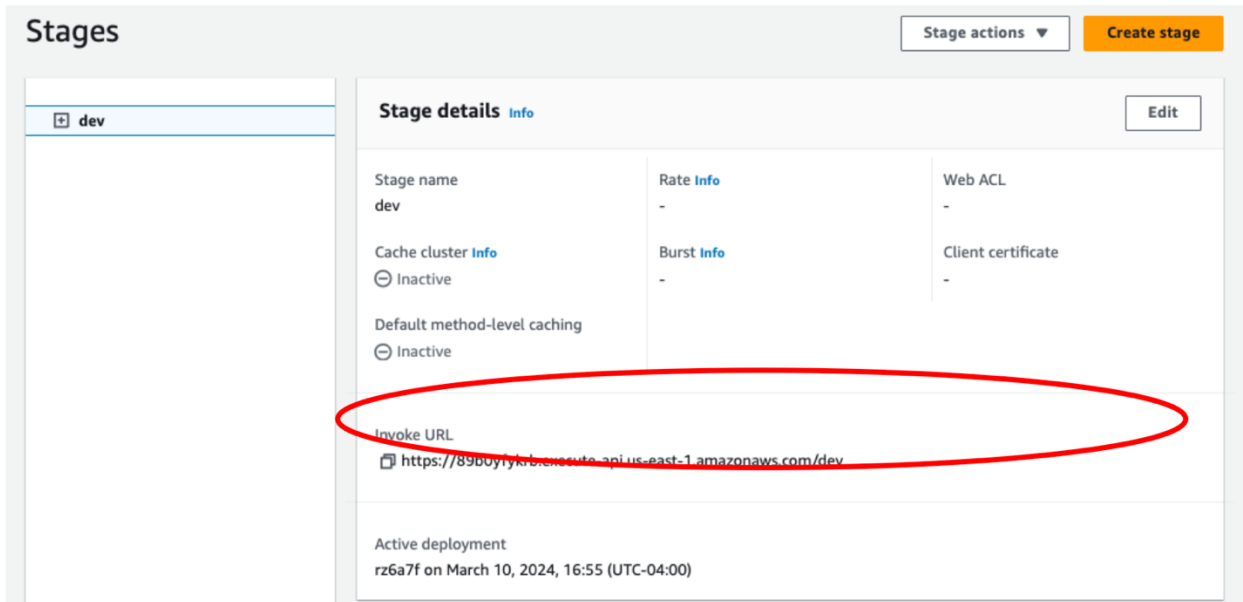
Stage name dev

Stage description

Deployment description

Cancel Deploy

- The next screen will show you the URL that was created for your API:



NOW YOU HAVE A WORKING API IN AWS = CONGRATULATIONS!!!!

4. Test your API in Postman

- Let's make sure your API actually works!
- Use the API definition that was imported into API Gateway to create a collection in Postman for testing your API
- You can use a variable to rename the URL to the actual URL provided by AWS.
- Here is an example running the /users GET to get all users (I am not using an environment variable in this case so you can see the full URL):

The screenshot shows a REST client interface with a GET request to `https://rj0ey8y171.execute-api.us-east-1.amazonaws.com/dev/users`. The response status is 200 OK, with a time of 169 ms and a size of 419 B. The response body is a JSON array of user objects.

```
1  {
2    "users": [
3      {
4        "id": "2000"
5      },
6      {
7        "firstname": "connie",
8        "lastname": "taylor",
9        "username": "ctaylor",
10       "id": "5001"
11      },
12      {
13        "id": "500"
14      },
15      {
16        "id": "100"
17      }
18    ]
19  }
```

- **You should run all five methods and show output.** For the POST and PATCH you will need a request Body. NOTE: You can add more data to the Dynamo DB by using the POST method on /users or you can directly add data in Dynamo DB.