

AWS Lambda - AWS API Gateway Integration Project

First we will create a Function on AWS Lambda and we will test that function on AWS Lambda itself , then we will do api integration then we will test it on postman.

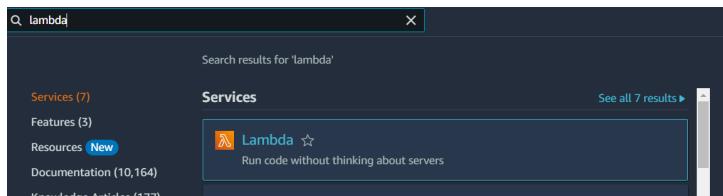
What is AWS Lambda ?

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers.

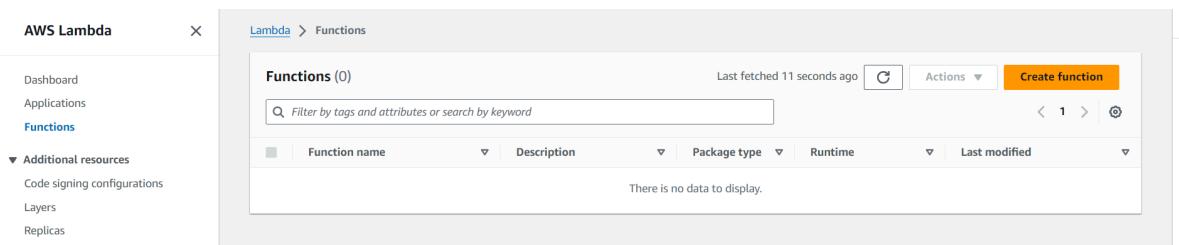
PART 1

STEPS

Step 1 :- Search Lambda -> Click on Lambda



Step 2 :- Click create function

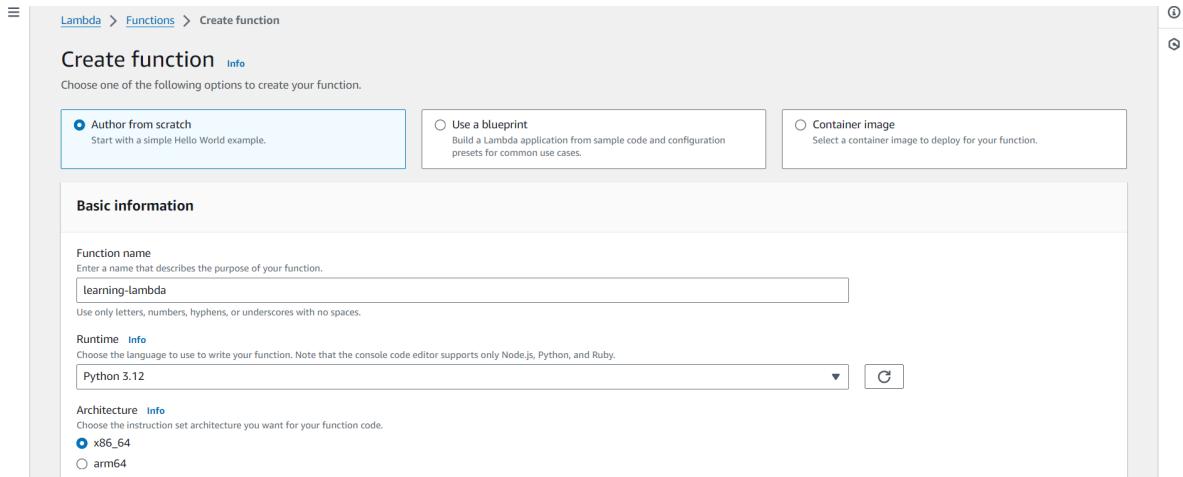


Step 3 :-

Give function name -> I'm giving learning-lambda

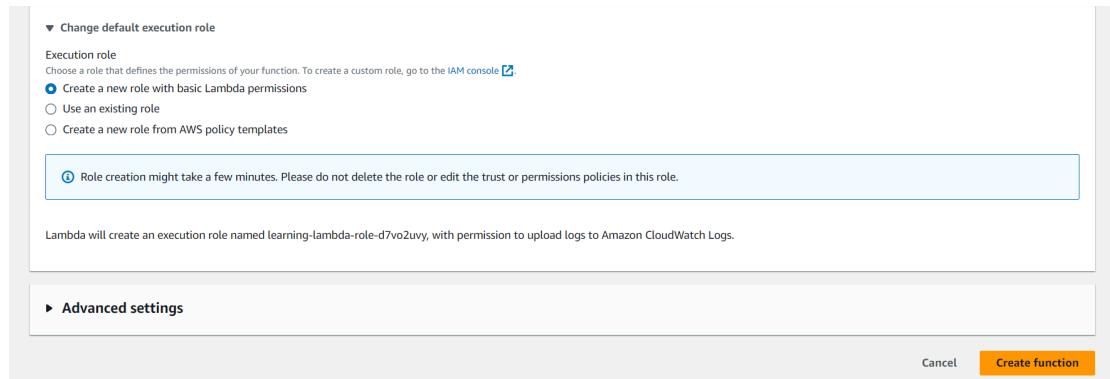
Select Runtime -> I'm selecting Python

Select Architecture -> I'm selecting x86_64

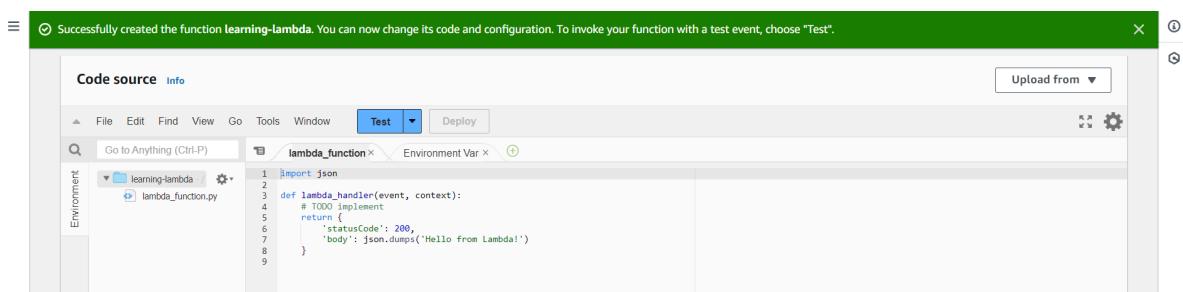


Permissions -> Select Create a new role with basic Lambda permissions

Press Create Function



The function created successfully



As you can see the function name is 'lambda_handler' which takes 2 parameters. One is the event and second is context.

event :- The data from the event that triggered the function.
 context : The data about the execution environment of the function.

Step 4 :- Testing the function

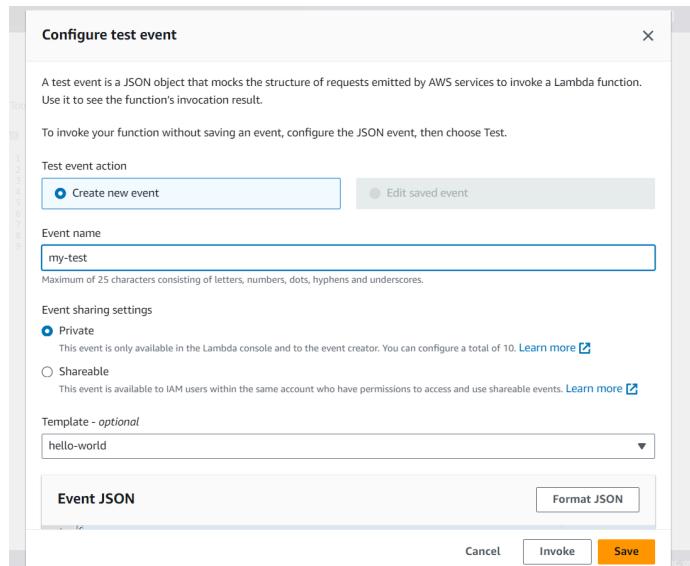
Click on the test button

```

1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9

```

Give name to your test -> I'm giving my-test
Click save



The status code is 200

```

1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9

```

PART 2

Go to
<https://jsonplaceholder.typicode.com/>
 Free fake API for testing and prototyping.

STEPS

Step 1 :- Create a new file -> Right click on folder -> Click new file

```

1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9

```

Give name to your file -> I'm giving config.py

```

1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9

```

Step 2 :- Copy the base url from jsonplaceholder website.

Try it

Run this code here, in a console or from any site:

```

fetch('https://jsonplaceholder.typicode.com/todos/1')
    .then(response => response.json())
    .then(json => console.log(json))

```

[Run script](#)

Open config.py and paste the base url.

```

1 base_url = "https://jsonplaceholder.typicode.com"

```

Open lambda_function.py -> Copy the below code

```

import json
from config import base_url
import requests

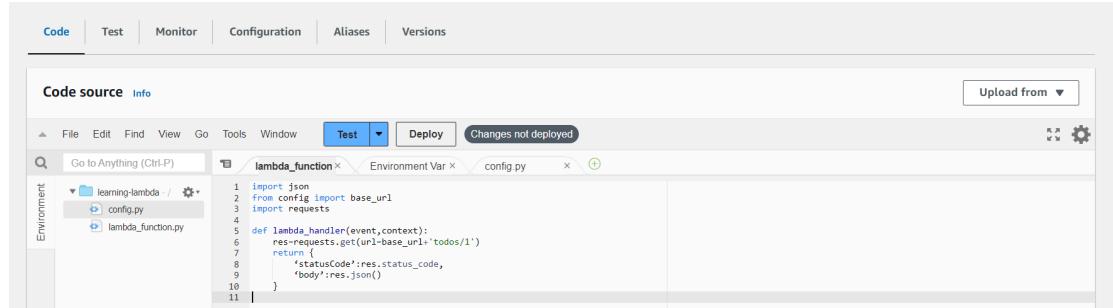
def lambda_handler(event,context):
    res=requests.get(url=base_url+'todos/1')

```

```

return {
    'statusCode':res.status_code,
    'body':res.json()
}

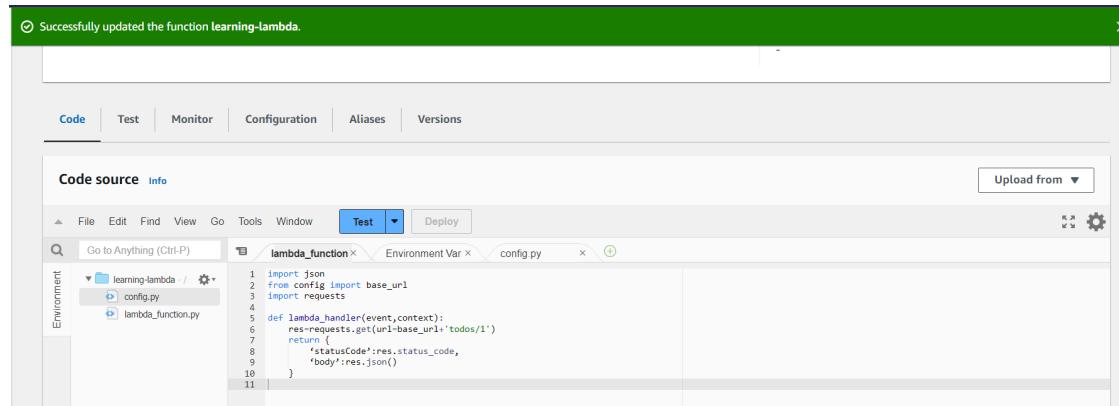
```



Step 3 :- Deploying the function

Now press Deploy button

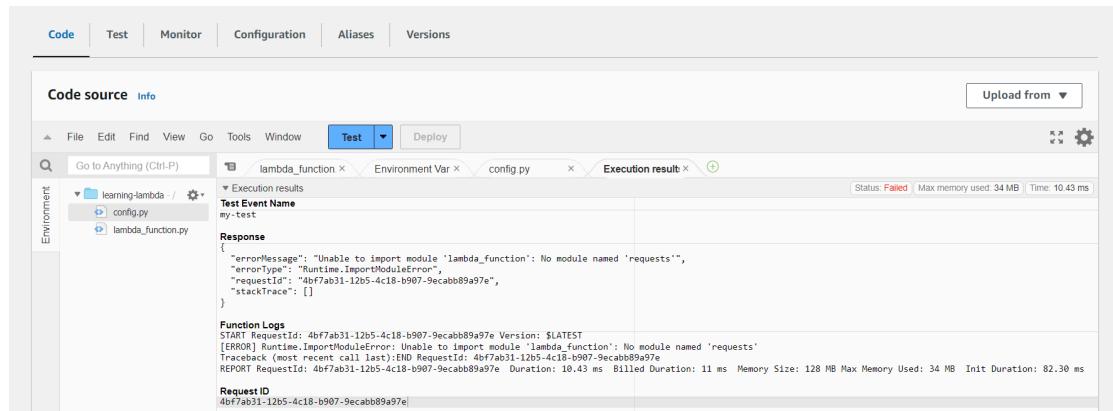
Successfully updated the function learning-lambda.



Step 4 :- Testing the function

Press Test button

Error :- "Unable to import module 'lambda_function': No module named 'requests'"



If the same error occurs on the server we would install the request library using pip. But as we are using lambda the serverless architecture so we are going to use the “Layers” concept.

A Lambda layer is a .zip file archive that contains supplementary code or data. Layers usually contain library dependencies, a custom runtime, or configuration files.

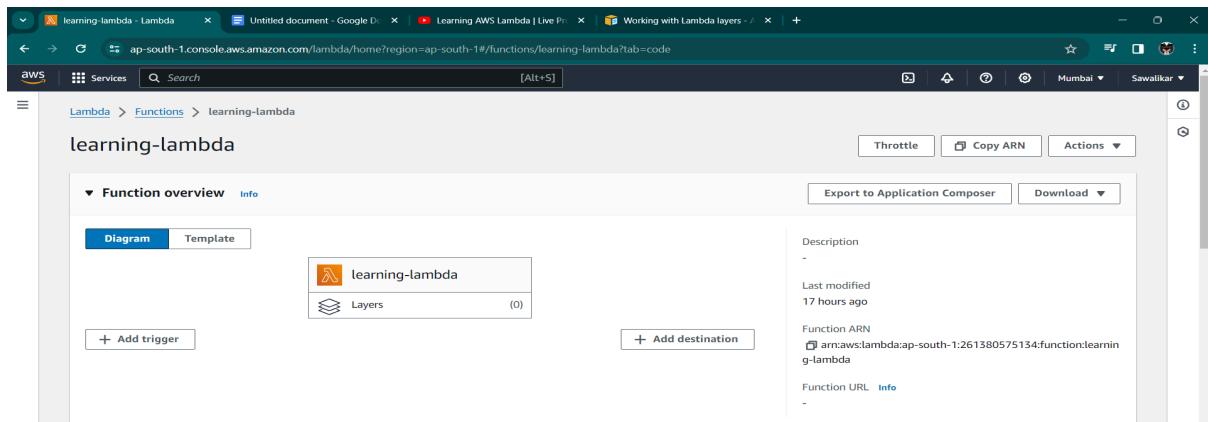
There are multiple reasons why you might consider using layers:

- To reduce the size of your deployment packages. Instead of including all of your function dependencies along with your function code in your deployment package, put them in a layer. This keeps deployment packages small and organised.
- To separate core function logic from dependencies. With layers, you can update your function dependencies independent of your function code, and vice versa. This promotes separation of concerns and helps you focus on your function logic.
- To share dependencies across multiple functions. After you create a layer, you can apply it to any number of functions in your account. Without layers, you need to include the same dependencies in each individual deployment package.
- To use the Lambda console code editor. The code editor is a useful tool for testing minor function code updates quickly. However, you can't use the editor if your deployment package size is too large. Using layers reduces your package size and can unlock usage of the code editor.

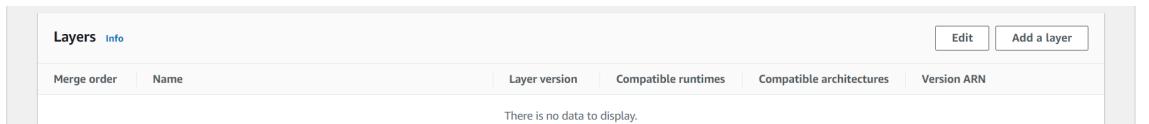
PART 3

Adding Layers

Step 1:- Click on Layers present below learning-lambda

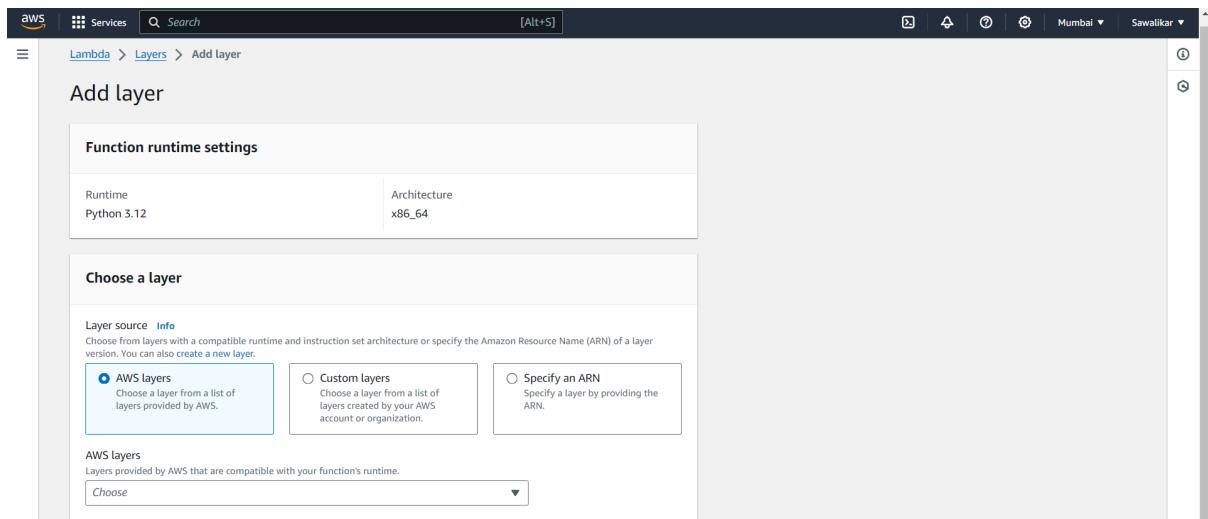


Step 2 :- click add a layer

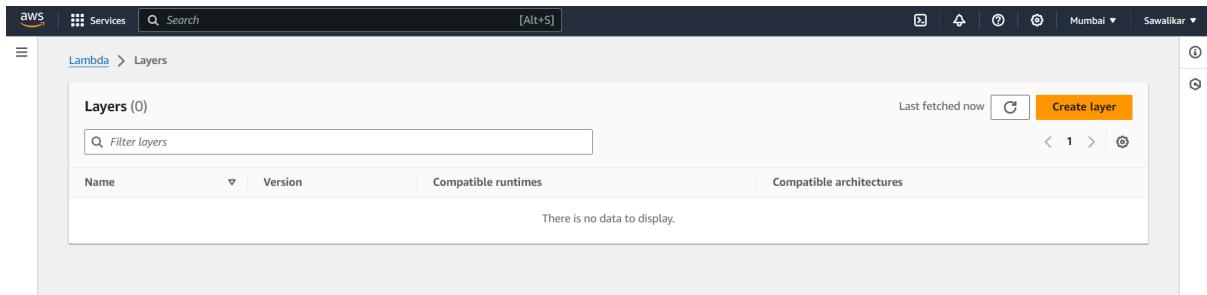


Right now we don't have layers so we will create a new layer first then we will add it.

Step 3 :- Click on Layers at the top



Press Create Layer



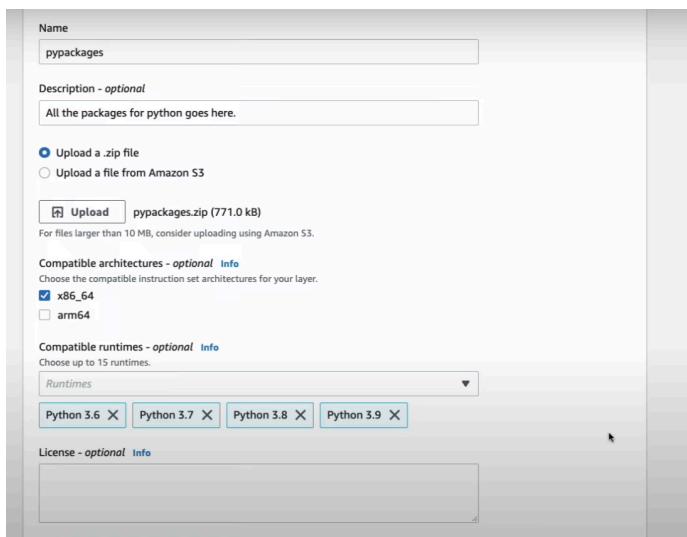
Now open a server and create a zip file (Use vs code to create this zip file by doing ssh)

```
$ mkdir pypackages
$ ls
$ cd pypackages
$ pwd
(Copy the path )
$ cd ..
$ pip3 install requests -t path_of_pypackages
pip3 install requests -t /home/ubuntu/pypackages
(In my case its /home/ubuntu/pypackages)
$ ls
```

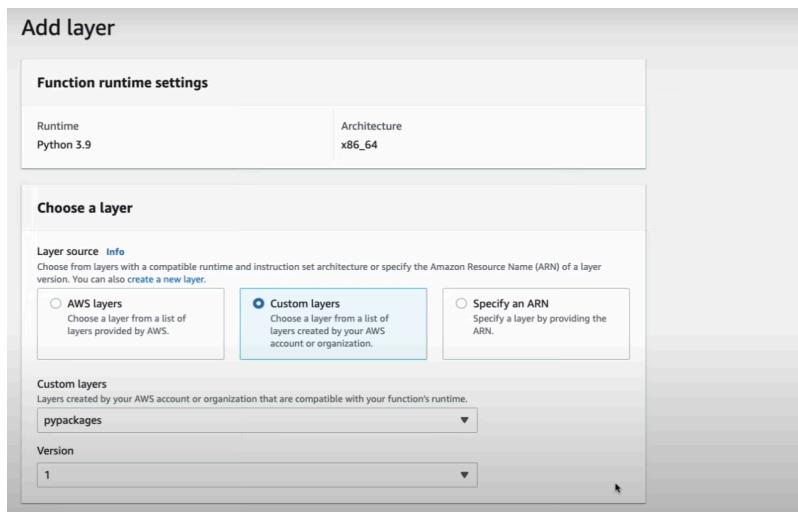
Now creating zip

```
$ sudo apt install zip
$ zip -r pypackages.zip pypackages/
$ ls
```

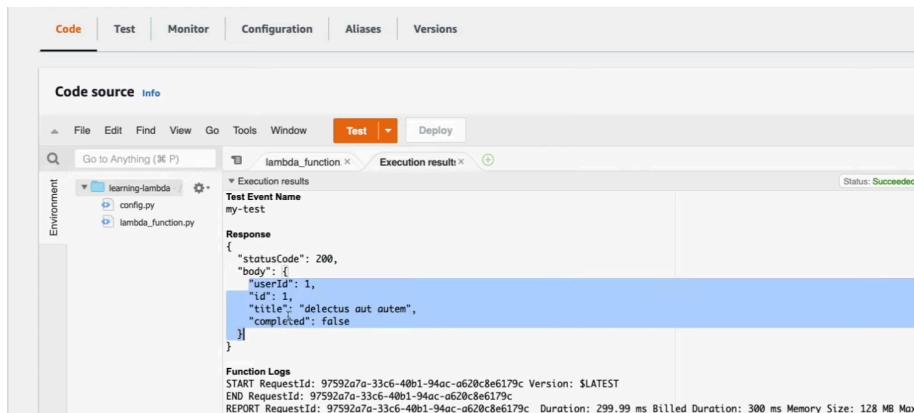
Step 4 :- Now Upload the zip file -> Select x86_64 ->Select all versions of Python



Step 5 :-Adding layer ->Choose custom layer (pypackages) ->Give version



Step 6 :- Now test the code again



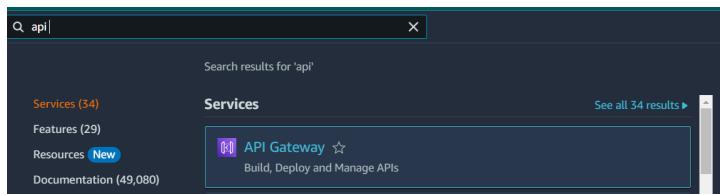
PART 4

Now we are creating a rest api that will power our function.

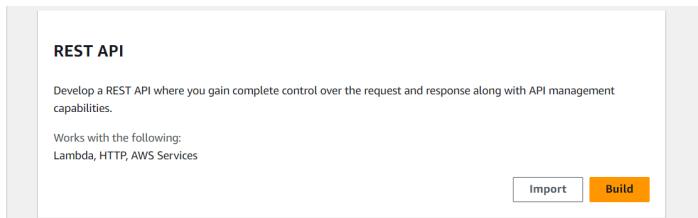
What is an API gateway ?

Amazon API Gateway helps developers to create and manage APIs to back-end systems running on Amazon EC2, AWS Lambda, or any publicly addressable web service. With Amazon API Gateway, you can generate custom client SDKs for your APIs, to connect your back-end systems to mobile, web, and server applications or services.

Step 1 :- Search api gateway -> Select api gateway



Step 2 :- Select REST Api -> Click Build



Step 3 :-

Create REST API

API details

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 name
get-user-data

Description - optional
gets the data from a lambda

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.
Edge-optimized

Cancel Create API

Now select Create Method

Successfully created REST API 'get-user-data (33f96yl4vb)'

API Gateway > APIs > Resources - get-user-data (33f96yl4vb)

Resources

Create resource

Resource details

Path / Resource ID u7yojjb52b

Methods (0)

Create method

No methods

No methods defined.

API actions Deploy API

Select get method

API Gateway > APIs > Resources - get-user-data (33f96y4vb) > Create method

Create method

Method details

Method type

Select a method type ▾

ANY

DELETE

GET

HEAD

OPTIONS

PATCH

POST

PUT

Mock
Generate a response based on API Gateway mappings and transformations.

Mock

Integration type -> Lambda function

API Gateway > APIs > Resources - get-user-data (33f96y4vb) > Create method

Create method

Method details

Method type

GET ▾

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.


Select above created function

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

ap-south-1 ▾
arn:aws:lambda:ap-south-1:261380575134:function:learning-lambda

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.

Default timeout
The default timeout is 29 seconds.

Cancel **Create method**

The screenshot shows the AWS API Gateway interface. On the left, the navigation pane is open with the 'APIs' section expanded, showing 'API: get-user-data' and its 'Resources' section. The main content area displays a success message: 'Successfully created method 'GET' in '/'. Below this, the 'Resources' section shows a single resource with the path '/'. The 'Method execution' tab is selected, showing the flow: Client → Method request → Integration request → Lambda integration. The ARN is listed as arn:aws:execute-api:ap-south-1:261380575134:33f96y4vb/*:GET/. The Resource ID is u7yojjb32b. There are buttons for 'API actions' and 'Deploy API'.

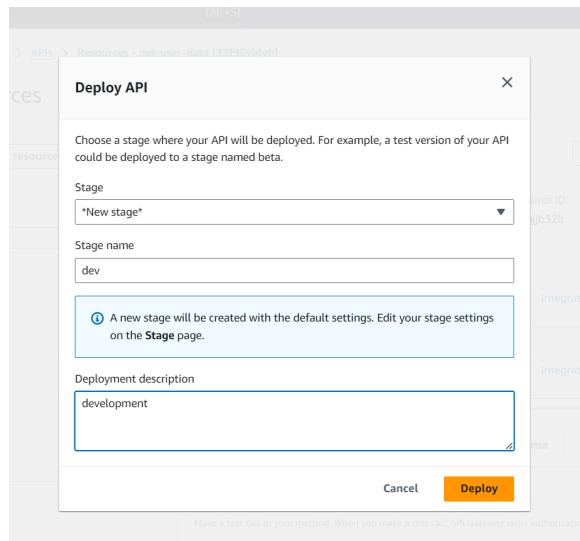
Testing the api
Click on Test

The screenshot shows the 'Method request' settings for the GET method of the '/get-user-data' resource. The 'Method request' tab is selected. The 'Method request settings' section includes fields for Authorization (NONE), Request validator (None), and API key required (False). The 'Request paths' section shows 'No request paths' and 'URL query string parameters' section shows 'No URL query string parameters'. There are tabs for 'Method request', 'Integration request', 'Integration response', 'Method response', and 'Test'.

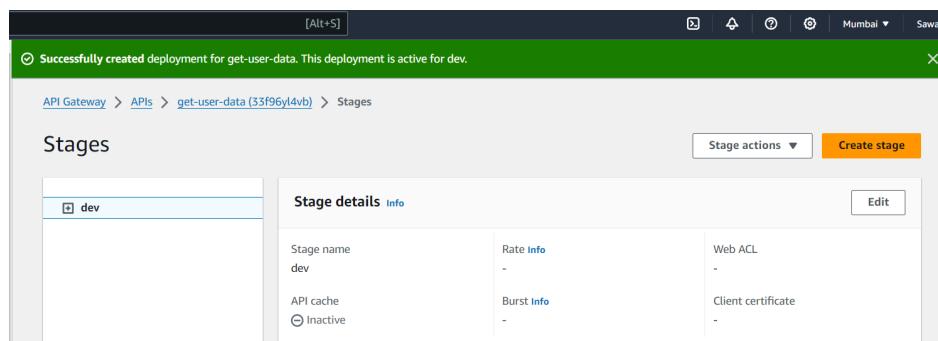
Deploying the api

Press Deploy API button

The screenshot shows the AWS API Gateway interface again. The 'API: get-user-data' resources page is displayed. The 'Method execution' tab is selected. The 'Method request' tab is currently active. The 'Test' tab is visible at the bottom. The 'Deploy API' button is located in the top right corner of the main content area.



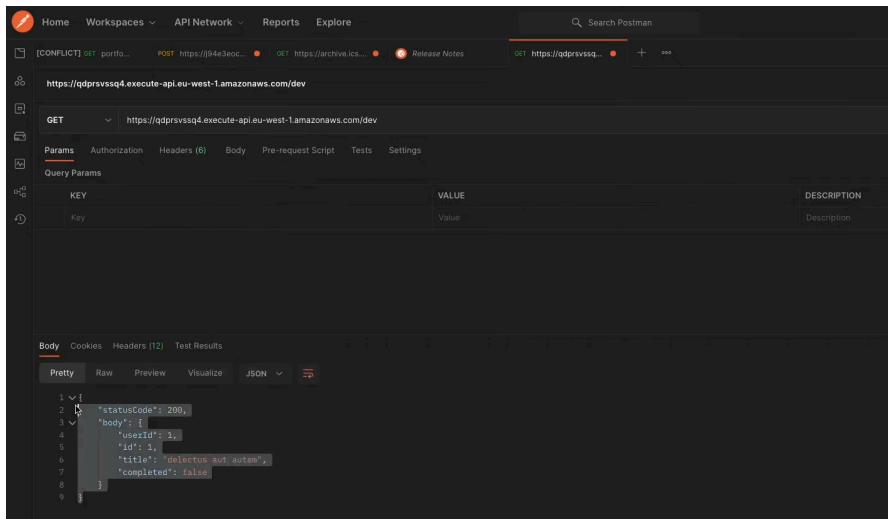
Successfully created deployment for get-user-data



PART 5

Open postman

Select get -> Copy the API url



The screenshot shows the Postman interface with a successful API call. The URL is <https://qdprsvssq4.execute-api.eu-west-1.amazonaws.com/dev>. The response body is a JSON object:

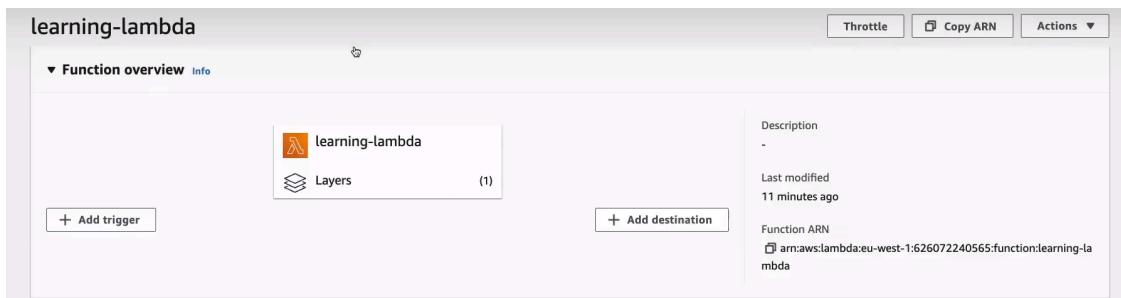
```
1 ↴ {  
2 ↴   "statusCode": 200,  
3 ↴   "body": [  
4 ↴     {  
5 ↴       "userId": 1,  
6 ↴       "id": 1,  
7 ↴       "title": "delectus aut autem",  
8 ↴       "completed": false  
9 ↴     }  
0 ↴   ]  
9 ↴ }
```

PART 6

Adding trigger

Means whenever my api will get trigger it will call my lambda function

Click on add trigger



The screenshot shows the AWS Lambda function details page for 'learning-lambda'. The function overview section shows the function name and a 'Layers' section with one layer named 'learning-lambda'. The 'Triggers' section is empty, and the 'Add trigger' button is highlighted.

Select api gateway -> select our created API ->Select deployment stage (dev) -> Select security (open) -> Click add

Add trigger

Trigger configuration

 API Gateway api application-services aws serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

API
Create a new API or attach an existing one.
get-user-data

Deployment stage
The name of your API's deployment stage.
dev

ⓘ When you connect your function to an existing API stage, Lambda deploys the API to that stage.

Security
Configure the security mechanism for your API endpoint.
Open

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add