



# AutOps

[Dashboard](#)

## AWS Serverless with Lambda & API Gateway Hands-On

AWS Lambda & Amazon API Gateway



AutOps

Feb 18, 2024 · 4 min read



End User



Amazon API Gateway



AWS Lambda



**Author: Ujwal Pachghare**

# Step 1



Go to lambda console → Click on the function → Click on Create function

Compute

## AWS Lambda

lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

### Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

Create a function

*Create a Lambda function with the following configurations:*

**Function name:** *api-testing*

**Runtime:** *Python 3.10*

**Architecture:** *x86\_64*

**Permissions** → **Execution role:** *Create a new role with basic lambda permissions*

## Create function Info

Choose one of the following options to create a new function:

☒ Author from scratch

Start with a simple Hello World example.

Build a Lambda application from sample code and configuration presets for common use cases.

### Basic information

#### Function name

Enter a name that describes the purpose of your function.

api-testing

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.

Python 3.10

#### 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

## Step 2



Open Visual Studio Code/other IDEs → Create file  
**lambda\_function.py** → Paste below code in that file

► **AWSGI**

► **FIASK**

COPY

```
import awsgi
from flask import (
    Flask,
    jsonify,
)

app = Flask(__name__)
```

```
@app.route("/")
def index():
    return jsonify(status=200, message="OK")

@app.route("/version")
def get_version():
    return {"version": "1.0.0"}

@app.route("/profile")
def get_name():
    return {"name": "AWS Dev"}

def lambda_handler(event, context):
    return awsgi.response(app, event, context, base64_content_types=
```

👉 Open terminal → Run the following commands one by one  
(Install **Python 3.10** if not)

```
# Install pip on ubuntu(if not):
sudo apt install python3-pip -y

# Install zip on ubuntu(if not):
sudo apt install zip -y

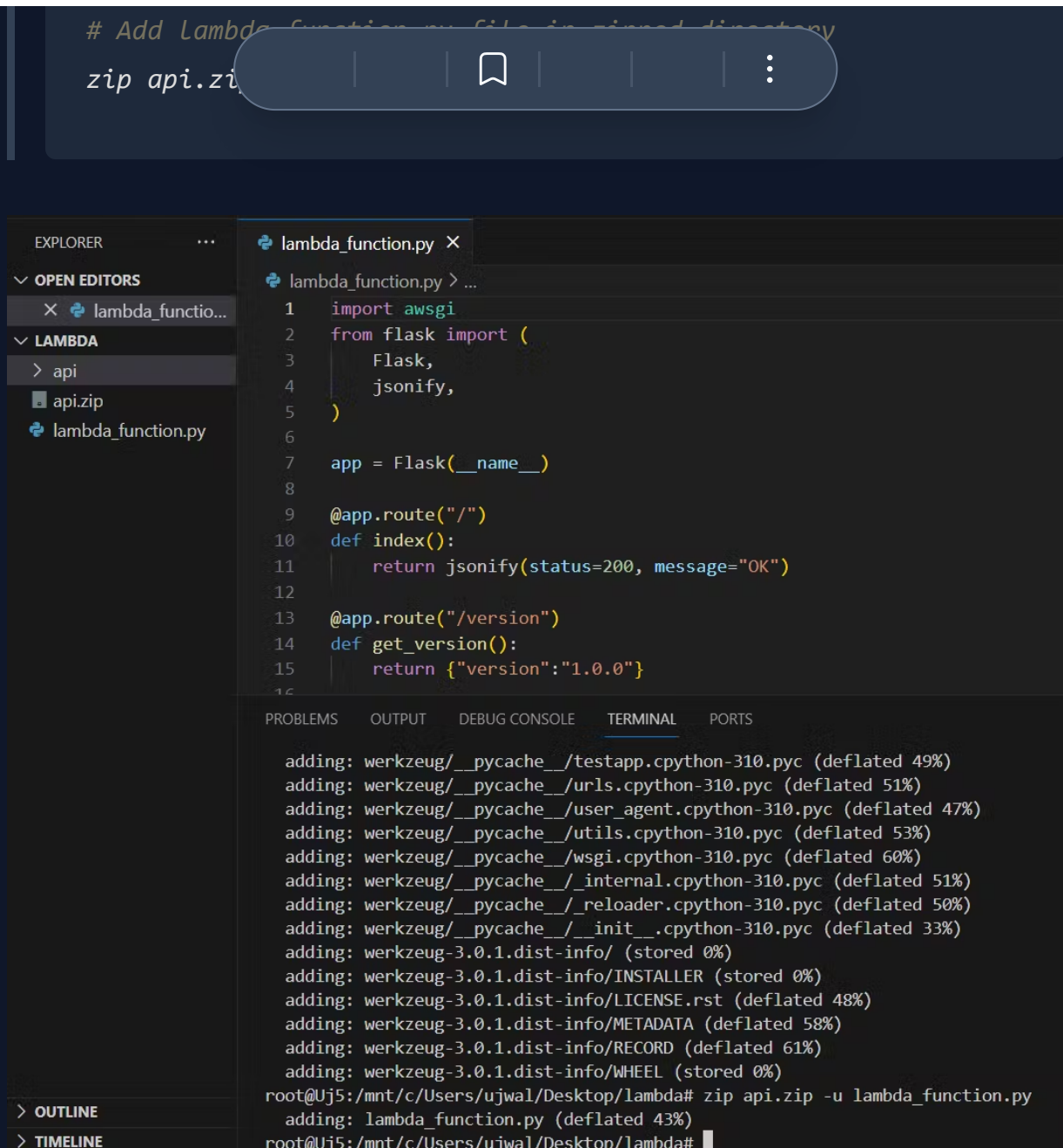
# Install awsgi and flask in api directory
pip install aws-wsgi flask -t api

# zip the code of api directory
(cd api; zip ../api.zip -r .)
```

COPY 

# Add Lambda Function by File in minio directory

zip api.zip



The screenshot shows a VS Code interface with the following components:

- EXPLORER:** Shows the project structure with 'api' folder containing 'api.zip' and 'lambda\_function.py'.
- lambda\_function.py:** Contains the following code:

```
1 import awsgi
2 from flask import (
3     Flask,
4     jsonify,
5 )
6
7 app = Flask(__name__)
8
9 @app.route("/")
10 def index():
11     return jsonify(status=200, message="OK")
12
13 @app.route("/version")
14 def get_version():
15     return {"version": "1.0.0"}
16
```
- TERMINAL:** Shows the command to create the zip file and its output:

```
root@Uj5:/mnt/c/Users/ujwal/Desktop/lambda# zip api.zip -u lambda_function.py
adding: lambda_function.py (deflated 43%)
root@Uj5:/mnt/c/Users/ujwal/Desktop/lambda#
```

## Step 3



Go to our Lambda function → Click on the **Upload from** button → Select the **.zip file** → Search for **api.zip** file that we have just created and upload

```

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

```

Successfully updated the function api-testing.

```

1 import awsgl
2 from flask import (
3     Flask,
4     jsonify,
5 )
6
7 app = Flask(__name__)
8
9 @app.route("/")
10 def index():
11     return jsonify(status=200, message="OK")
12
13 @app.route("/version")
14 def get_version():
15     return {"version": "1.0.0"}
16
17 @app.route("/profile")
18 def get_name():
19     return {"name": "AWS Dev"}
20
21 def lambda_handler(event, context):
22     return awsgl.response(app, event, context, base64_content_types={"image/png"})

```

## Step 4



**Creating Amazon API Gateway to call the function with the help of APIs** Go to **Amazon API Gateway Console** → Scroll down and Select **REST API** by clicking on **Build** button

Networking & Content Delivery

## Amazon API Gateway

create, maintain, and secure APIs at any scale

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.

## REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:

Lambda, HTTP, AWS Services

[Import](#)[Build](#)

## Select the following API Details:

Choose **New API**

**API name:** *lambda-api*

**API endpoint type:** *Edge-optimized (it can be accesible from browser)*

### 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

lambda-api


Description - optional

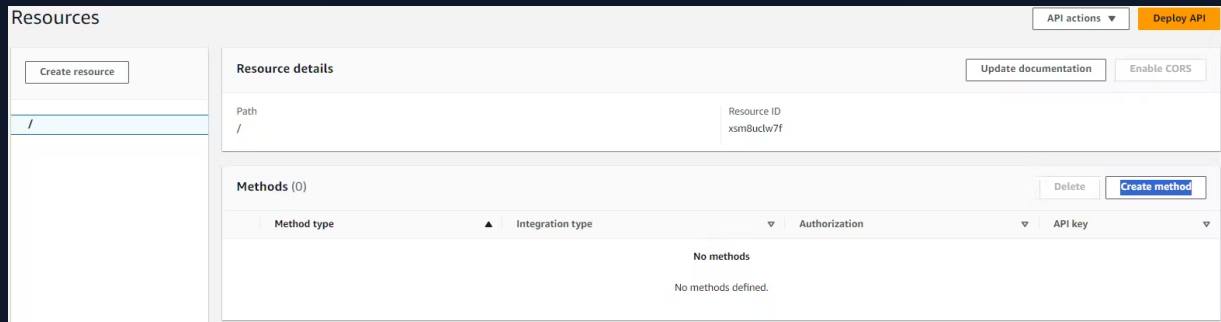
API endpoint type

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](#)

 Click on



The screenshot shows the AWS API Gateway console. On the left, there's a 'Resources' sidebar with a 'Create resource' button and a list containing a single resource with path '/'. The main panel is titled 'Resource details' and shows the path '/' and resource ID 'xsm8uctw7f'. Below this, the 'Methods (0)' section is empty, with a 'Create method' button. At the top right of the console, there are buttons for 'API actions' and 'Deploy API'. In the top left of the console area, there are buttons for 'Update documentation' and 'Enable CORS'.

Fill in the following method details:

**Method type:** *GET (To get the value)*

**Integration type:** *Lambda function (API integration with lambda)*

**Lambda proxy integration:** *ON (For sending request to lambda)*

**Lambda function:** *choose region and lambda function ARN*

**Default timeout:** *default (29 sec)*



## 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.

☒ 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.

ap-south-1

Choose a Lambda function or enter its ARN

arn:aws:lambda:ap-south-1:814495875142:function:api-testing

- 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.



**Creating Resource for version api** Click on the **Create Resource** button → Give Resource name as **version**

## Resource details

☒ Proxy resource [Info](#)

Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path

/

Resource name

version

☐ CORS (Cross Origin Resource Sharing) [Info](#)

Create an OPTIONS method that allows all origins, all methods, and several common headers.



Follow the **method** that we followed in the previous method creation. Ex: Click on the Create method button, then.....



*Creating Resource for profile api* Click on the **Create Resource** button → Give Resource name as **profile** (make sure it is in "/" Resource Path)



Now Create the same **GET method** that you created for the **version resource**

# Step 5



Click on the **Deploy API** button

The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with a 'Create resource' button and a list of resources: '/', '/profile', and '/version', each with a 'GET' method. The main area is titled 'Resources' and shows details for the '/' resource. It includes a 'Path' field with '/' and a 'Resource ID' field with 'xsm8uctw7f'. Below this, there's a table for 'Methods (1)' with one entry: a 'GET' method using 'Lambda' integration, 'None' authorization, and 'Not required' API key. In the top right corner, there's a 'Deploy API' button highlighted in orange.

Fill in the following Deploy API configurations:

**Stage:** *|New stage\*\**

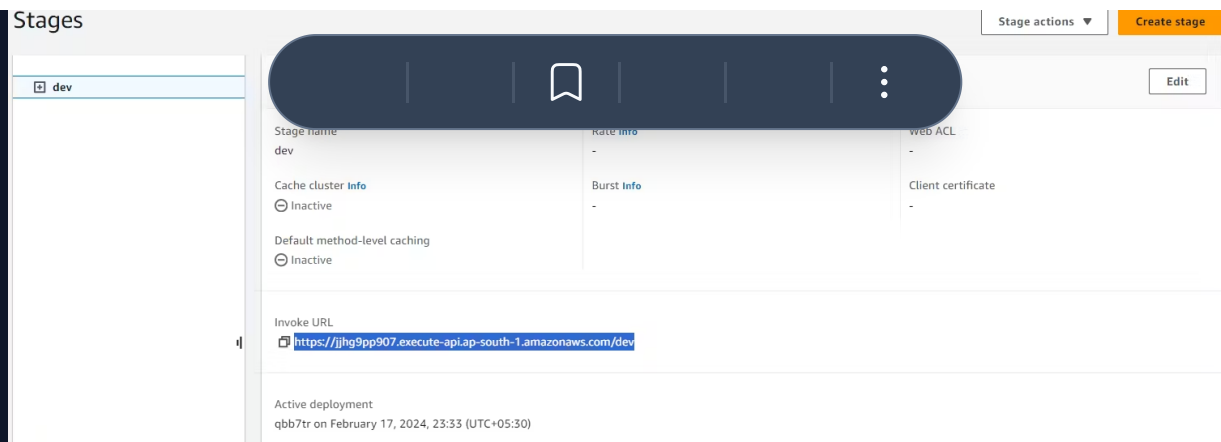
**Stage name:** *dev*

**Description:** *optional*

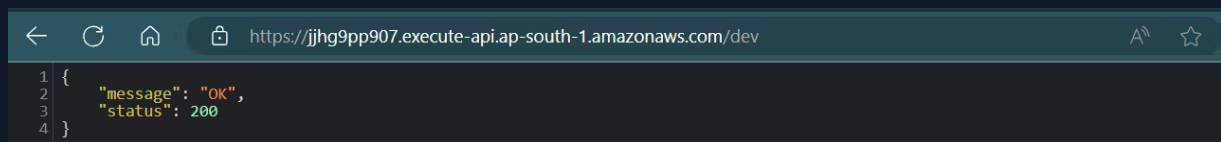
The screenshot shows the 'Deploy API' dialog box. It has a title bar with 'Deploy API' and a close button. The main text says: 'Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.' Below this, there are two input fields: 'Stage' with a dropdown menu showing '\*New stage\*' and a downward arrow, and 'Stage name' with a text input field containing 'dev'.



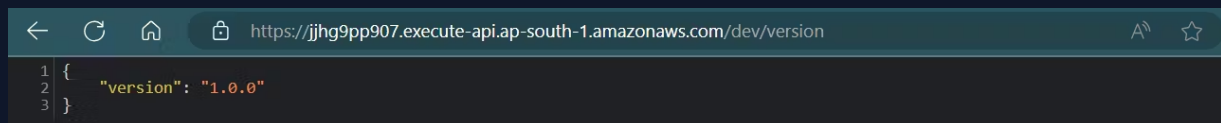
After deploying the API, you will get the following URL



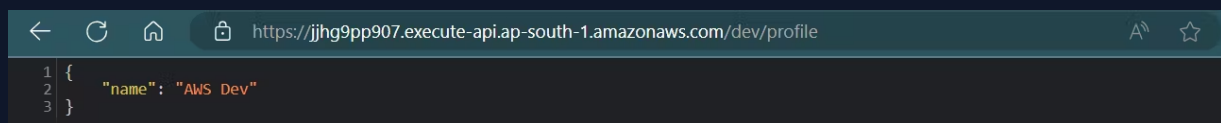
Copy that URL and paste it into the browser; you will get the following result for a **GET** API of the root.



Testing version API: you will get result as follows:



Testing profile API: you will get result as follows:



## Conclusion:

We have successfully created an automation task with the help of AWS Lambda and the Amazon API Gateway, and it will perform the following actions:

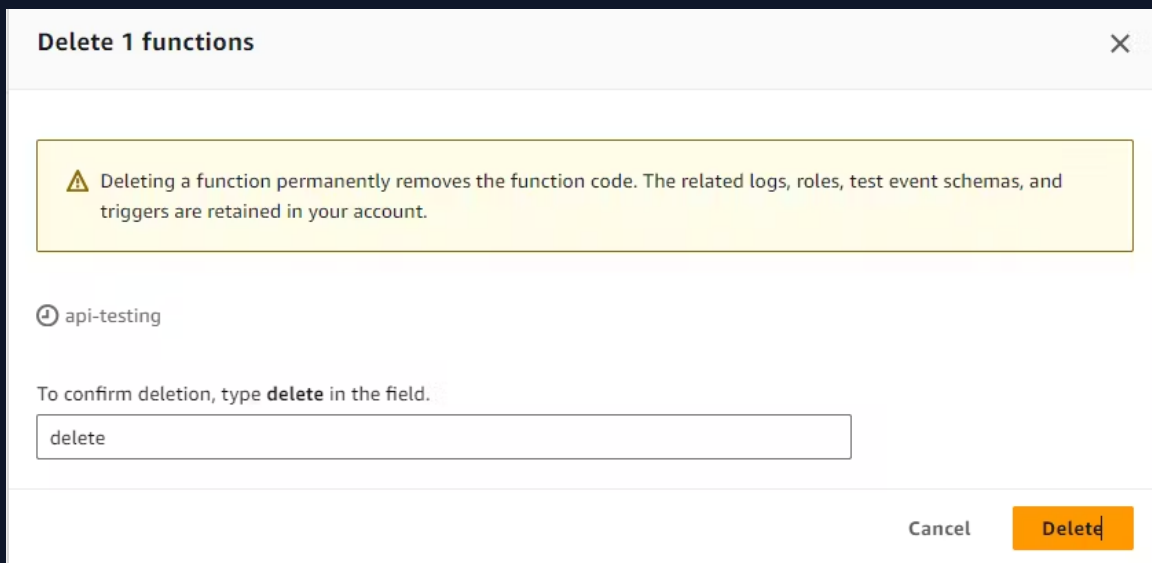
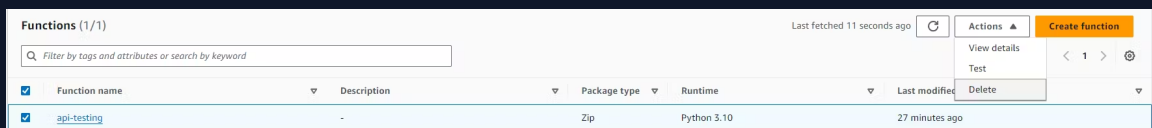
→ Whenever any of the above APIs are called by any user, it will trigger the Lambda function.

# Final Step

## Cleaning Up

### 1. Delete Lambda function

✕ Go to the Lambda console → Select function to delete -  
> Click on Delete → Type delete in pop-up → Delete



### Delete API Gateway

✕ Go to API Gateway console → Select API Gateway to Delete  
→ click on Delete button → type confirm in pop-up → Delete

