

AWS: Lambda 101

CS5224 Cloud Computing AY 2025/2026 Sem 1

Walkthrough

- What is a Lambda Function/ FaaS?
- Demo
 - Getting an API from data.gov
 - Writing the lambda function
 - Preparation
 - Implementing the function on AWS
 - Checking your implementation

What is a Lambda Function?

Function as a Service (FaaS)

- Deploy small, focused functions
- Event-driven execution
- Automatic scaling (0 to thousands of instances)
- Pay per execution + compute time

Key Characteristics:

- **Event-driven:** Triggered by HTTP requests, file uploads, database changes, etc.
- **Stateless:** Each execution is independent
- **Time-limited:** Maximum of 15-minute execution (AWS)
- **Multi-language:** Python, Node.js, Java, C#, Go, Ruby, and more

Benefits:

- No server management
- Automatic scaling
- Cost-effective (pay per use)
- High availability built-in

AWS Free Tier

- One million free requests per month and
- 400,000 GB-seconds of compute time per month
- <https://aws.amazon.com/lambda/pricing/>

Lambda Function Recipe

Demo: Calling a data.gov API

1. API to call
2. Program to call the APIs
 - a. Zip file with requirements and **proper naming (gotcha #1)**
3. Set up S3 bucket (or DB to write your data to)
4. Function setup
 - a. Upload function
 - b. Update Lambda function permissions via IAM (**gotcha #2**)
 - c. Testing

API to call

A Singapore Government Agency Website [How to identify](#)

data.gov.sg

weather api

Datasets Help Feedback Log in

Home > Datasets > Weather Forecast > 2-hour Weather Forecast

Environment

2-hour Weather Forecast

See API uptime

Data from Mar 2016 to Sep 2025
Update frequency: Every 30 minutes
NEA (National Environment Agency)

Weather forecast for next 2 hours

[Download API Specs \(10.8 KB\)](#) [Subscribe](#) [Share](#)

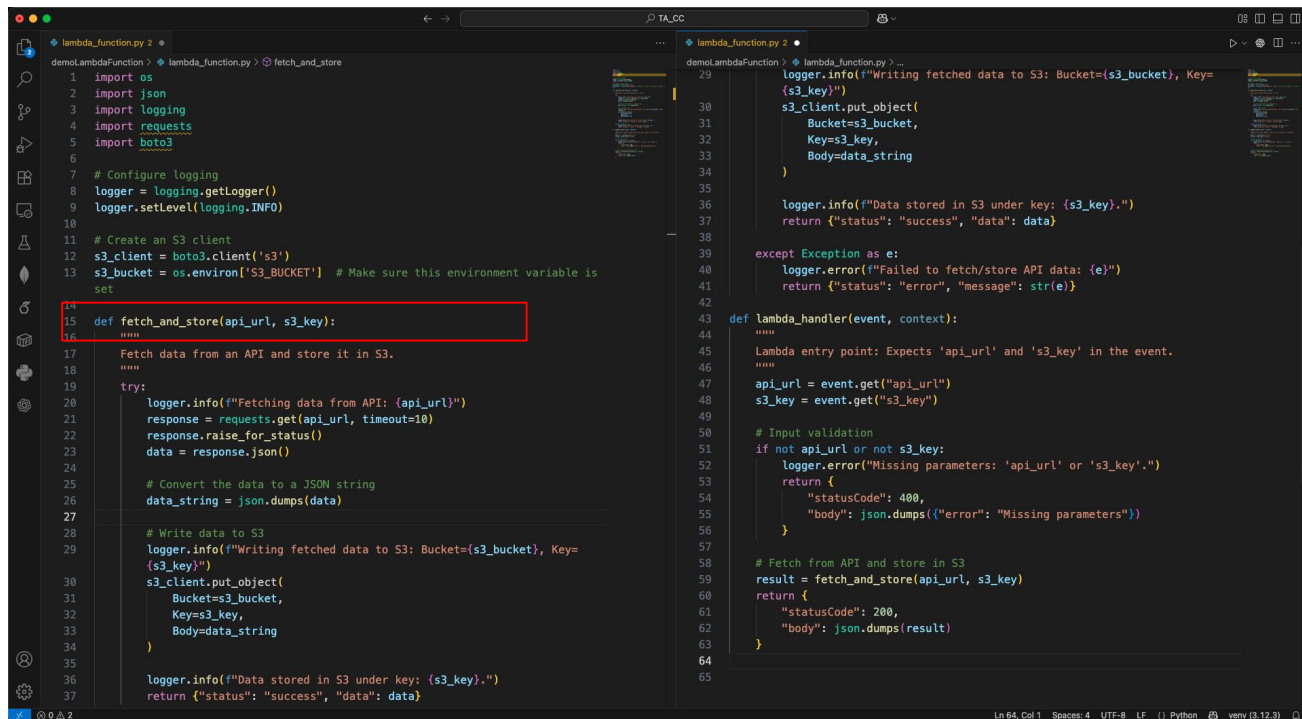
Retrieve the latest two hour weather forecast

<https://api-open.data.gov.sg/v2/real-time/api/two-hr-forecast>

- Updated half-hourly from NEA
- Forecasts are given for multiple areas in Singapore
- Filter for specific date or date-time by providing `date` in query parameter.
 - use YYYY-MM-DD format to retrieve all of the readings for that day
 - use YYYY-MM-DDTHH:mm:ss to retrieve the latest readings at that moment in time
 - example: `?date=2024-07-16` or `?date=2024-07-16T23:59:00`
- If `date` is not provided in query parameter, API will return the latest reading
- Possible values for forecast include:
 - Fair
 - Fair (Day)
 - Fair (Night)

```
GET /two-hr-forecast Python Requests v  
  
1 import requests  
2  
3 url = "https://api-open.data.gov.sg/v2/real-time/api/two-hr-forecast"  
4  
5 headers = {"X-API-Key": "YOUR_SECRET_TOKEN"}  
6  
7 response = requests.get(url, headers=headers)  
8  
9 print(response.json())  
  
Test Request
```

Function



```
lambda_function.py 2
demo.LambdaFunction > lambda_function.py > fetch_and_store
1 import os
2 import json
3 import logging
4 import requests
5 import boto3
6
7 # Configure logging
8 logger = logging.getLogger()
9 logger.setLevel(logging.INFO)
10
11 # Create an S3 client
12 s3_client = boto3.client('s3')
13 s3_bucket = os.environ['S3_BUCKET'] # Make sure this environment variable is set
14
15 def fetch_and_store(api_url, s3_key):
16     """
17     Fetch data from an API and store it in S3.
18     """
19     try:
20         logger.info(f"Fetching data from API: {api_url}")
21         response = requests.get(api_url, timeout=10)
22         response.raise_for_status()
23         data = response.json()
24
25         # Convert the data to a JSON string
26         data_string = json.dumps(data)
27
28         # Write data to S3
29         logger.info(f"Writing fetched data to S3: Bucket={s3_bucket}, Key={s3_key}")
30         s3_client.put_object(
31             Bucket=s3_bucket,
32             Key=s3_key,
33             Body=data_string
34         )
35
36         logger.info(f"Data stored in S3 under key: {s3_key}.")
37         return {"status": "success", "data": data}
38
39
40 lambda_function.py 2
demo.LambdaFunction > lambda_function.py >
29
30 logger.info(f"Writing fetched data to S3: Bucket={s3_bucket}, Key={s3_key}")
31 s3_client.put_object(
32     Bucket=s3_bucket,
33     Key=s3_key,
34     Body=data_string
35 )
36
37 logger.info(f"Data stored in S3 under key: {s3_key}.")
38 return {"status": "success", "data": data}
39
40 except Exception as e:
41     logger.error(f"Failed to fetch/store API data: {e}")
42     return {"status": "error", "message": str(e)}
43
44 def lambda_handler(event, context):
45     """
46     Lambda entry point: Expects 'api_url' and 's3_key' in the event.
47     """
48     api_url = event.get('api_url')
49     s3_key = event.get('s3_key')
50
51     # Input validation
52     if not api_url or not s3_key:
53         logger.error("Missing parameters: 'api_url' or 's3_key'.")
54         return {
55             "statusCode": 400,
56             "body": json.dumps({"error": "Missing parameters"})
57         }
58
59     # Fetch from API and store in S3
60     result = fetch_and_store(api_url, s3_key)
61     return {
62         "statusCode": 200,
63         "body": json.dumps(result)
64     }
65
```

Notes:

- Your file name **MUST** be `lambda_function.py`
- From within the folder with your function,
cd my_lambda
pip install requests -t .
- Zip the entire folder
zip -r ../my_lambda_deploy ment.zip .

AWS Lambda

The screenshot shows the AWS Lambda console interface. At the top, there's a navigation bar with the AWS logo, a search bar, and account information (Asia Pacific (Sydney), Account ID: 0438-0902-7827). The main heading is 'Compute' followed by 'AWS Lambda' and the tagline 'lets you run code without thinking about servers.' Below this, a 'Get started' box contains the text 'Author a Lambda function from scratch, or choose from one of many preconfigured examples.' and a 'Create a function' button. A red box highlights this 'Get started' section. Below the main heading, a paragraph explains the pay-per-use model. At the bottom, the 'How it works' section shows a code editor with a Node.js example. A red box highlights the 'Create a function' button and the 'How it works' section. The code editor shows a Node.js function handler that logs the event and returns a string.

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

How it works

Run Next: Lambda responds to events

.NET | Java | Node.js | Python | Ruby | Custom runtime

```
1 * exports.handler = async (event) => {
2   console.log(event);
3   return 'Hello from Lambda!';
4 };
5
```

AWS Lambda

aws

Search

[Option+S]

Account ID: 0438-0902-7827

fmccta

Asia Pacific (Sydney)

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 [Info](#)
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

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 3.13

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

☐ arm64

☒ x86_64

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

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

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

AWS Lambda

The screenshot displays the AWS Lambda console interface for a function named 'demoLambdaFunction'. The top navigation bar shows the AWS logo, a search bar, and the account ID '0438-0902-7827'. The main header includes the function name and buttons for 'Throttle', 'Copy ARN', and 'Actions'. Below this, the 'Function overview' section is active, showing a 'Diagram' tab and a 'Template' tab. A card for 'demoLambdaFunction' is visible, along with a '+ Add trigger' button and a '+ Add destination' button. The 'Description' section on the right provides details: 'Last modified 42 minutes ago', 'Function ARN: arn:aws:lambda:ap-southeast-2:043809027827:function:demoLambdaFunction', and 'Function URL'. The 'Code source' section at the bottom is highlighted with a red box, showing the 'Open in Visual Studio Code' button and the 'Upload from' dropdown menu, which is currently set to '.zip file' with an option for 'Amazon S3 location'.

demoLambdaFunction

Function overview

Diagram Template

demoLambdaFunction

Layers (0)

+ Add trigger

+ Add destination

Description

Last modified 42 minutes ago

Function ARN arn:aws:lambda:ap-southeast-2:043809027827:function:demoLambdaFunction

Function URL

Code Test Monitor Configuration Aliases Versions

Code source

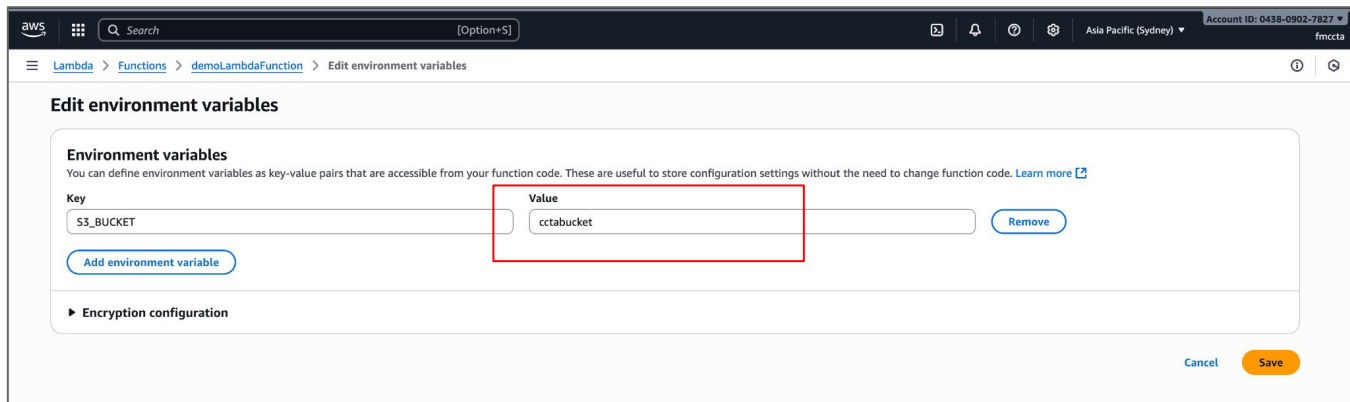
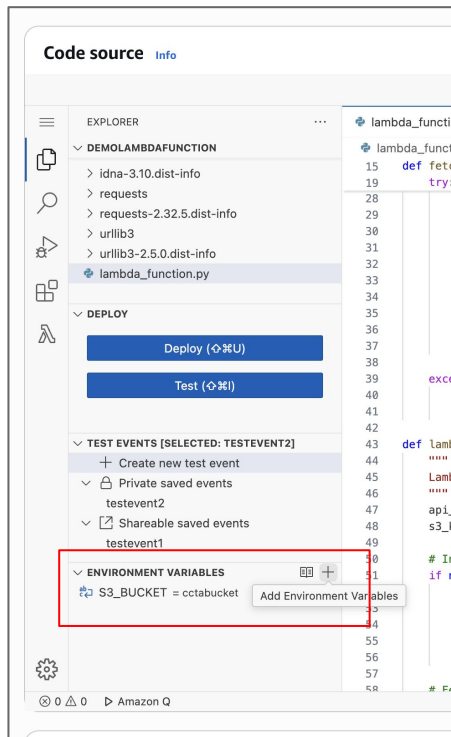
Open in Visual Studio Code

Upload from

.zip file

Amazon S3 location

AWS Lambda – Environment Variables



AWS Lambda – Test Event

The screenshot displays the AWS Lambda console interface for a function named 'demoLambdaFunction'. The 'Code source' tab is active, showing a Python file 'lambda_function.py' with the following code:

```
15 def fetch_and_store(api_url, s3_key):
16     try:
17         response = requests.get(api_url)
18         data = response.json()
19
20     except requests.exceptions.RequestException as e:
21         logger.error(f'Failed to fetch/store API data: {e}')
22         return {"status": "error", "message": str(e)}
23
24     # Convert the data to a JSON string
25     data_string = json.dumps(data)
26
27     # Write data to S3
28     logger.info(f'Writing fetched data to S3: Bucket={s3_bucket}, Key={s3_key}, Body={data_string}')
29     s3_client.put_object(Bucket=s3_bucket, Key=s3_key, Body=data_string)
30
31     logger.info(f'Data stored in S3 under key: {s3_key}')
32     return {"status": "success", "data": data}
33
34 except Exception as e:
35     logger.error(f'Failed to fetch/store API data: {e}')
36     return {"status": "error", "message": str(e)}
37
38 def lambda_handler(event, context):
39     """
40     Lambda entry point: Expects 'api_url' and 's3_key'
41     """
42     api_url = event.get("api_url")
43     s3_key = event.get("s3_key") # Instead of cache_key
44
45     # Input validation
46     if not api_url or not s3_key:
47         logger.error('Missing parameters: 'api_url' or 's3_key')
48         return {"status": "error", "message": "Missing parameters: 'api_url' or 's3_key'"}
49
50     return fetch_and_store(api_url, s3_key)
```

The 'Create new test event' dialog is open, showing the following details:

- Event Name:** testevent3
- Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.**
- Event sharing settings:** Private (selected)
- Template - optional:** testevent1
- Event JSON:**

```
1 {
2   "api_url": "https://api-open.data.gov.sg/v2/real-time/api/two-hr-forecast",
3   "s3_key": "test-data.json"
4 }
```

The 'Code properties' section at the bottom shows the package size (6.4 MB), SHA256 hash, and last modified time (19 minutes ago).

AWS IAM Roles – Lambda Permissions

The screenshot shows the AWS IAM console interface for the role **demoLambdaFunction-role-uxeestmz**. The left sidebar contains navigation links for Identity and Access Management (IAM), Access management, and Access reports. The main content area shows the role's summary and the **Permissions** tab, which lists attached policies.

Summary

- Creation date:** September 04, 2025, 22:15 (UTC+08:00)
- Last activity:** -
- ARN:** `arn:aws:iam::043809027827:role/service-role/demoLambdaFunction-role-uxeestmz`
- Maximum session duration:** 1 hour

Permissions policies (1/3)

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
<input checked="" type="checkbox"/> AmazonS3FullAccess	AWS managed	1
<input type="checkbox"/> AWSLambdaBasicExecutionRole-33982f2f-9726-402d...	Customer managed	1
<input type="checkbox"/> AWSLambdaVPAccessExecutionRole-a3cdfb9e-4d77...	Customer managed	1

Buttons and Actions:

- Simulate** (with external link icon)
- Remove** (with external link icon)
- Add permissions** (dropdown menu):
 - Attach policies
 - Create inline policy

AWS Lambda – Success!

The screenshot displays the AWS Lambda console for a function named `demoLambdaFunction`. The **Code source** tab is active, showing the Python code for `lambda_function.py`. The code fetches data from an API, converts it to JSON, and stores it in an S3 bucket. The **Test** tab is also visible, showing a successful execution of the `testevent1` test event. The **Execution Results** section at the bottom provides detailed logs of the function's execution, including the request ID, duration, and the data stored in S3.

Code source Info

Open in Visual Studio Code Upload from

lambda_function.py

```
15 def fetch_and_store(api_url, s3_key):
16     try:
17         response = requests.get(api_url)
23         data = response.json()
24
25         # Convert the data to a JSON string
26         data_string = json.dumps(data)
27
28         # Write data to S3
29         logger.info(f'Writing fetched data to S3: Bucket={s3_bucket}, Key={s3_key}')
30         s3_client.put_object(Bucket=s3_bucket, Key=s3_key, Body=data_string)
31
```

EXPLOSER

- DEMOLAMBDAFUNCTION
 - idna-3.10.dist-info
 - requests
 - requests-2.32.5.dist-info
 - urllib3
 - urllib3-2.5.0.dist-info
 - lambda_function.py

DEPLOY

Deploy (⇧U) Test (⇧H)

TEST EVENTS [SELECTED: TESTEVENT1]

- Create new test event
- Private saved events
 - testevent2
- Shareable saved events
 - testevent1

ENVIRONMENT VARIABLES

- S3_BUCKET = cctabucket

PROBLEMS OUTPUT CODE REFERENCE LOG TERMINAL

Execution Results

Function Logs:

```
[INFO] 2025-09-07T12:29:44.313Z Found credentials in environment variables.
START RequestId: 99a446a6-3950-430b-b2e0-d2f5deefb869 Version: $LATEST
[INFO] 2025-09-07T12:29:44.510Z 99a446a6-3950-430b-b2e0-d2f5deefb869 Fetching data from API: https://api-open.data.gov.sg/v2/real-time/api/two-hr-forecast
[INFO] 2025-09-07T12:29:46.145Z 99a446a6-3950-430b-b2e0-d2f5deefb869 Writing fetched data to S3: Bucket=cctabucket, Key=test-data.json
[INFO] 2025-09-07T12:29:46.458Z 99a446a6-3950-430b-b2e0-d2f5deefb869 Data stored in S3 under key: test-data.json.
END RequestId: 99a446a6-3950-430b-b2e0-d2f5deefb869
REPORT RequestId: 99a446a6-3950-430b-b2e0-d2f5deefb869 Duration: 1974.60 ms Billed Duration: 2667 ms Memory Size: 128 MB Max Memory Used: 97 MB Init Duration: 692.05 ms

Request ID: 99a446a6-3950-430b-b2e0-d2f5deefb869
```

Event Name: testevent3

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

- Private
- Shareable

Invoke Save