

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

The screenshot shows the data.gov.sg website interface. At the top, there is a navigation bar with a Singapore Government Agency Website logo, a "How to identify" dropdown, a search bar containing "weather api", and links for "Datasets", "Help", "Feedback", and "Log in". Below the navigation, the URL path is shown as "Home > Datasets > Weather Forecast > 2-hour Weather Forecast". A breadcrumb trail indicates the current section: "Environment > 2-hour Weather Forecast".

The main content area features a title "2-hour Weather Forecast" with a subtitle "Data from Mar 2016 to Sep 2025" and "Update frequency: Every 30 minutes". It also mentions "NEA (National Environment Agency)". Below this, it says "Weather forecast for next 2 hours". There are three buttons at the bottom left: "Download API Specs (10.8 KB)", "Subscribe", and a link icon.

On the right side, there is a "See API uptime" button. The central part of the page is titled "Retrieve the latest two hour weather forecast" and contains a red box around the URL "https://api-open.data.gov.sg/v2/real-time/api/two-hr-forecast". Below the URL is a bulleted list of instructions:

- 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)

At the bottom right, there is a code editor window for a "GET /two-hr-forecast" request using Python Requests. The code is as follows:

```
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())
```

A "Test Request" button is located at the bottom right of the code editor.

Function

The screenshot shows a code editor with two tabs open, both titled "lambda_function.py 2".

Left Tab:

```
import os
import json
import logging
import requests
import boto3

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

# Create an S3 client
s3_client = boto3.client('s3')
s3_bucket = os.environ['S3_BUCKET'] # Make sure this environment variable is set

def fetch_and_store(api_url, s3_key):
    """Fetch data from an API and store it in S3.

    try:
        logger.info(f"Fetching data from API: {api_url}")
        response = requests.get(api_url, timeout=10)
        response.raise_for_status()
        data = response.json()

        # Convert the data to a JSON string
        data_string = json.dumps(data)

        # Write data to S3
        logger.info(f"Writing fetched data to S3: Bucket={s3_bucket}, Key={s3_key}")
        s3_client.put_object(
            Bucket=s3_bucket,
            Key=s3_key,
            Body=data_string
        )

        logger.info(f"Data stored in S3 under key: {s3_key}.")
    return {"status": "success", "data": data}
```

Right Tab:

```
logger.info("Writing fetched data to S3: Bucket={s3_bucket}, Key={s3_key}")
s3_client.put_object(
    Bucket=s3_bucket,
    Key=s3_key,
    Body=data_string
)

logger.info("Data stored in S3 under key: {s3_key}.")
return {"status": "success", "data": data}

except Exception as e:
    logger.error(f"Failed to fetch/store API data: {e}")
    return {"status": "error", "message": str(e)}

def lambda_handler(event, context):
    """
    Lambda entry point: Expects 'api_url' and 's3_key' in the event.
    """
    api_url = event.get("api_url")
    s3_key = event.get("s3_key")

    # Input validation
    if not api_url or not s3_key:
        logger.error("Missing parameters: 'api_url' or 's3_key'.")
        return {
            "statusCode": 400,
            "body": json.dumps({"error": "Missing parameters"})
        }

    # Fetch from API and store in S3
    result = fetch_and_store(api_url, s3_key)
    return {
        "statusCode": 200,
        "body": json.dumps(result)
    }
```

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 landing page. At the top, there's a navigation bar with the AWS logo, a search bar, and account information (Account ID: 0438-0902-7827, Region: Asia Pacific (Sydney)). Below the navigation, the word "Compute" is displayed. The main heading "AWS Lambda" is prominently shown, followed by the subtext "lets you run code without thinking about servers." A note below states: "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." To the right, a "Get started" box contains the text: "Author a Lambda function from scratch, or choose from one of many preconfigured examples." with a "Create a function" button. A red box highlights this "Create a function" button. Below this, a "How it works" section is visible, featuring a code editor with Node.js code:

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

Below the code editor, there are tabs for ".NET", "Java", "Node.js" (which is selected), "Python", "Ruby", and "Custom runtime". There are also "Run" and "Next: Lambda responds to events" buttons.

AWS Lambda

The screenshot shows the AWS Lambda 'Create function' wizard. The top navigation bar includes the AWS logo, a search bar, and account information (Account ID: 0438-0902-7827, Region: Asia Pacific (Sydney)). The main title is 'Create function' with an 'Info' link. Below it, a sub-header says 'Choose one of the following options to create your function.' with three radio button choices:

- 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.

The 'Basic information' section is highlighted with a red box. It contains fields for 'Function name' (demoLambdaFunction), 'Runtime' (Python 3.13), and 'Architecture' (x86_64). A note states: '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.' There are also sections for 'Change default execution role' and 'Additional configurations'.

At the bottom right are 'Cancel' and 'Create function' buttons.

AWS Lambda

The screenshot shows the AWS Lambda function configuration interface for a function named "demoLambdaFunction".

Function Overview: The top section displays the function name, a placeholder icon for the Lambda logo, and a note indicating 0 layers. It includes buttons for "Throttle", "Copy ARN", "Actions", "Export to Infrastructure Composer", and "Download".

Description: This section shows the last modified time (42 minutes ago) and the Function ARN (arn:aws:lambda:ap-southeast-2:043809027827:function:demoLambdaFunction). It also includes a "Function URL" link.

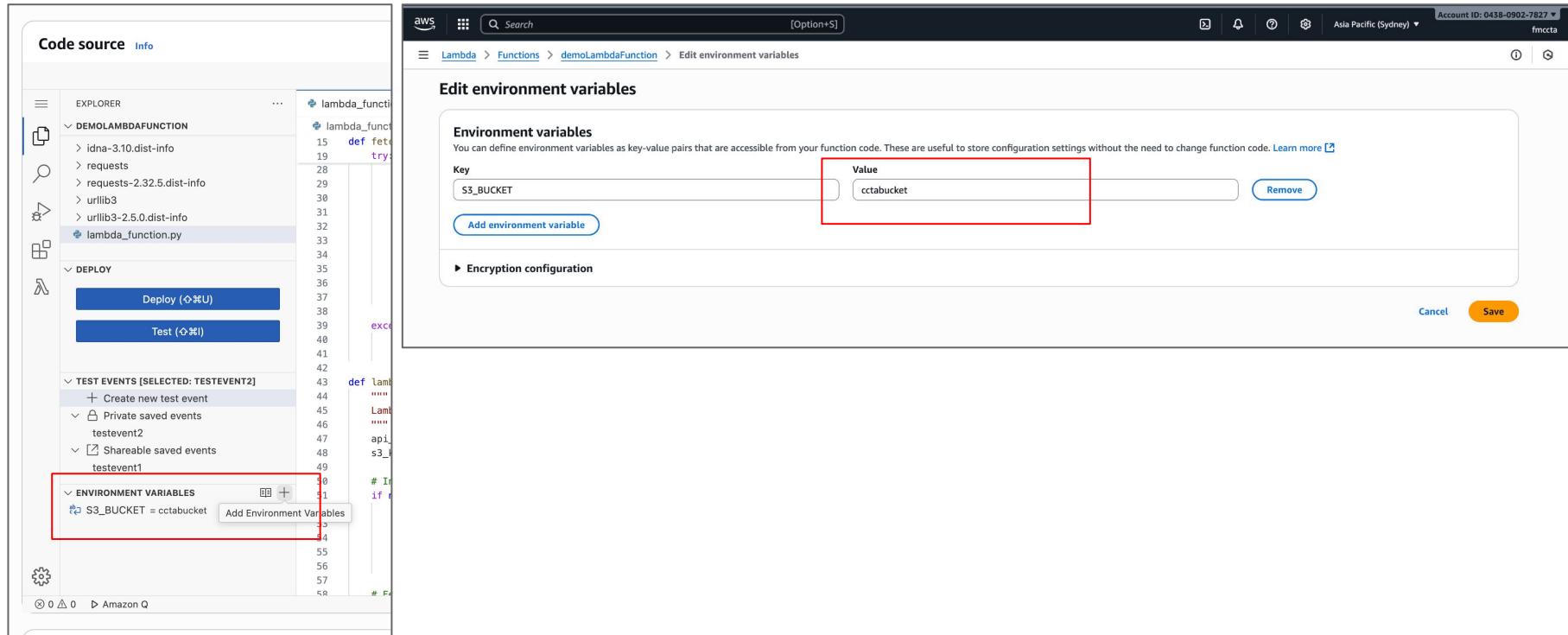
Code Tab: The "Code" tab is selected, showing the code source editor. The code editor displays the file "lambda_function.py" with the following content:

```
lambda_function.py
1 import os
2 import json
3 import logging
4 import requests
5 import boto3
6
```

Code Source: Below the code editor, there are options to "Open in Visual Studio Code", "Upload from", and "Amazon S3 location". A red box highlights the "Upload from" button and its dropdown menu, which contains ".zip file".

EXPLORER: On the left, the Explorer sidebar shows the project structure under "DEMOLAMBDAFUNCTION", including ".venv", "bin", "certifi", "certifi-2025.8.3.dist-info", and "charset_normalizer".

AWS Lambda - Environment Variables



AWS Lambda - Test Event

The screenshot shows the AWS Lambda console interface. On the left, the sidebar displays the project structure under 'DEMOLAMBDAFUNCTION' and 'TEST EVENTS (SELECTED: TESTEVENT1)'. The main area shows the code editor for 'lambda_function.py' with the following content:

```
15 def fetch_and_store(api_url, s3_key):
16     try:
17         response = requests.get(api_url)
18         data = response.json()
19
20         # Convert the data to a JSON string
21         data_string = json.dumps(data)
22
23         # Write data to S3
24         logger.info("Writing fetched data to S3: Bucket")
25         s3_client = boto3.client('s3')
26         Bucket=s3_bucket,
27         Key=s3_key,
28         Body=data_string
29
30         logger.info("Data stored in S3 under key: (%s)" % s3_key)
31         return {"status": "success", "data": data}
32
33     except Exception as e:
34         logger.error("Failed to fetch/store API data: %s" % str(e))
35         return {"status": "error", "message": str(e)}
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
```

On the right, a modal window titled 'Create new test event' is open, showing the configuration for 'testevent3'. The 'Event Name' field contains 'testevent3'. The 'Event sharing settings' section has 'Private' selected. The 'Template - optional' dropdown shows 'testevent1'. The 'event.json' content is highlighted with a red border and contains the following 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 }
```

At the bottom of the page, the 'Code properties' section shows:

- Package size: 6.4 MB
- SHA256 hash: 7+1m/MnTBdH15defT0hKtp9/KUBU/15qcMaeOBy7jxI=
- Last modified: 19 minutes ago
- Encryption with AWS KMS customer managed KMS key

Page footer: © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

AWS IAM Roles - Lambda Permissions

The screenshot shows the AWS IAM Roles page with the following details:

- Identity and Access Management (IAM)** is selected in the left sidebar.
- The role name is **demoLambdaFunction-role-uxeestmz**.
- Summary** section:
 - 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** tab is selected.
- Permissions policies (1/3)**:
 - You can attach up to 10 managed policies.
 - Filter by Type: All types.
 - Attached entities: 1.
 - Policies listed:
 - AmazonS3FullAccess** (selected, checked): AWS managed.
 - AWSLambdaBasicExecutionRole-33982f2f-9726-402d...**: Customer managed.
 - AWSLambdaVPCAccessExecutionRole-a3cdfb9e-4d77...**: Customer managed.
- Add permissions** button is highlighted with a red box.
- Actions** buttons: Simulate, Remove.

AWS Lambda - Success!

The screenshot shows the AWS Lambda console interface. The top navigation bar includes the AWS logo, a search bar, and account information (Account ID: 0438-0902-7827). The main navigation menu shows 'Lambda > Functions > demoLambdaFunction'. Below this, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code' tab is selected, displaying the code editor for 'lambda_function.py'. The code implements a function to fetch data from an API and store it in S3. A modal window titled 'Create new test event' is open, showing fields for 'Event Name' (set to 'testevent3') and 'Event sharing settings' (with options for 'Private' and 'Shareable'). The 'PROBLEMS', 'OUTPUT', 'CODE REFERENCE LOG', and 'TERMINAL' tabs are visible at the bottom. The 'TERMINAL' tab shows the execution logs for the Lambda function, indicating a successful deployment and execution. The logs include details about the environment variables, API requests, and data storage in S3.

Code source [Info](#)

EXPLORER

- DEMOLAMBDAFUNCTION
 - > idna-3.10.dist-info
 - > requests-2.32.5.dist-info
 - > urllib3
 - > urllib3-2.5.0.dist-info
 - lambda_function.py
- DEPLOY
 - Deploy (⚙️)
 - Test (⌘️)
- TEST EVENTS (SELECTED: TESTEVENT1)
 - + Create new test event
 - Private saved events
 - Shareable saved events
 - testevent2
 - testevent1
- ENVIRONMENT VARIABLES
 - S3_BUCKET = cctabucket

lambda_function.py

```
15 def fetch_and_store(api_url, s3_key):
16     try:
17         response = requests.get(api_url)
18         data = response.json()
19
20         # Convert the data to a JSON string
21         data_string = json.dumps(data)
22
23         # Write data to S3
24         logger.info(f"Writing fetched data to S3: Bucket={s3_client.put_object(Bucket=s3_bucket, Key=s3_key)}")
25
26         # Log the result
27         logger.info(f"Data written to S3 successfully!")
28
29     except Exception as e:
30         logger.error(f"An error occurred while fetching data: {e}")
31
32
33 if __name__ == "__main__":
34     fetch_and_store("https://api-open.data.gov.sg/v2/real-time/api/two-hr-forecast", "test-data.json")
```

Create new test event

Event Name: testevent3

Event sharing settings:

- Private: This event is only available in the Lambda Console and to the event creator. You can configure a total of ten. [Learn more](#)
- Shareable: This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Execution Results

```
[{"area": "Cloudy", "area": "Sungei Kadut", "forecast": "Cloudy", "area": "Tampines", "forecast": "Cloudy", "area": "Yangon", "forecast": "Cloudy", "area": "Tengah", "forecast": "Cloudy", "area": "Toa Payoh", "forecast": "Cloudy", "area": "Tuas", "forecast": "Cloudy", "area": "Western Islands", "forecast": "Cloudy", "area": "Western Water Catchment", "forecast": "Cloudy", "area": "Woodlands", "forecast": "Cloudy", "area": "Yishun", "forecast": "Cloudy"}]
```

Function Logs:

```
[INFO] 2025-09-07T12:29:44.313Z Found credentials in environment variables.
START RequestId: 99a446a6-3950-430b-b2e0-d2f5defb869 Version: $LATEST
[INFO] 2025-09-07T12:29:44.510Z 99a446a6-3950-430b-b2e0-d2f5defb869 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-d2f5defb869 Writing fetched data to S3: Bucket=cctabucket, Key=test-data.json
[INFO] 2025-09-07T12:29:46.458Z 99a446a6-3950-430b-b2e0-d2f5defb869 Data stored in S3 under key: test-data.json.
END RequestId: 99a446a6-3950-430b-b2e0-d2f5defb869
REPORT RequestId: 99a446a6-3950-430b-b2e0-d2f5defb869 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-d2f5defb869