

Adv. DevOps Experiment 11

Name : Rohan Lalchandani

Class : D15A Roll no : 25

Aim: To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs. **Theory:**

AWS Lambda

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). Users of AWS Lambda create functions, self-contained applications written in one of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner. The Lambda functions can perform any kind of computing task, from serving web pages and processing streams of data to calling APIs and integrating with other AWS services.

The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so “serverless” doesn’t mean that there are no servers involved: it just means that the servers, the operating systems, the network layer and the rest of the infrastructure have already been taken care of so that you can focus on writing application code.

Features of AWS Lambda

- AWS Lambda easily scales the infrastructure without any additional configuration. It reduces the operational work involved.
 - It offers multiple options like AWS S3, CloudWatch, DynamoDB, API Gateway, Kinesis, CodeCommit, and many more to trigger an event.
 - You don’t need to invest upfront. You pay only for the memory used by the lambda function and minimal cost on the number of requests hence cost-efficient.
 - AWS Lambda is secure. It uses AWS IAM to define all the roles and security policies.
 - It offers fault tolerance for both services running the code and the function. You do not have to worry about the application down.
- Packaging Functions**

Lambda functions need to be packaged and sent to AWS. This is usually a process of compressing the function and all its dependencies and uploading it to an S3 bucket.

And letting AWS know that you want to use this package when a specific event takes place. To help us with this process we use the Serverless Stack Framework (SST). We’ll go over this in detail later on in this guide.

Execution Model

The container (and the resources used by it) that runs our function is managed completely by AWS. It is brought up when an event takes place and is turned off if it is not being used. If additional requests are made while the original event is being served, a new container is brought up to serve a request. This means that if we are undergoing a usage spike, the cloud provider simply creates multiple instances of the container with our function to serve those requests.

This has some interesting implications. Firstly, our functions are effectively stateless. Secondly, each request (or event) is served by a single instance of a Lambda function. This means that you are not going to be handling concurrent requests in your code.

AWS brings up a container whenever there is a new request. It does make some optimizations here. It will hang on to the container for a few minutes (5 - 15mins depending on the load) so it can respond to subsequent requests without a cold start.

Stateless Functions

The above execution model makes Lambda functions effectively stateless. This means that every time your Lambda function is triggered by an event it is invoked in a completely new environment. You don't have access to the execution context of the previous event.

However, due to the optimization noted above, the actual Lambda function is invoked only once per container instantiation. Recall that our functions are run inside containers. So when a function is first invoked, all the code in our handler function gets executed and the handler function gets invoked. If the container is still available for subsequent requests, your function will get invoked and not the code around it.

For example, the `createNewDbConnection` method below is called once per container instantiation and not every time the Lambda function is invoked. The `myHandler` function on the other hand is called on every invocation.

Common Use Cases for Lambda

Due to Lambda's architecture, it can deliver great benefits over traditional cloud computing setups for applications where:

1. Individual tasks run for a short time;
 2. Each task is generally self-contained;
 3. There is a large difference between the lowest and highest levels in the workload of the application.
- Some of the most common use cases for AWS Lambda that fit these criteria are: Scalable APIs. When building APIs using AWS Lambda, one execution of a Lambda function can serve a single HTTP request.

Different parts of the API can be routed to different Lambda functions via Amazon API Gateway. AWS Lambda automatically scales individual functions according to

the demand for them, so different parts of your API can scale differently according to current usage levels. This allows for cost-effective and flexible API setups.

Data processing. Lambda functions are optimized for event-based data processing. It is easy to integrate AWS Lambda with data sources like Amazon DynamoDB and trigger a Lambda function for specific kinds of data

events. For example, you could employ Lambda to do some work every time an item in DynamoDB is created or updated, thus making it a good fit for things like notifications, counters and analytics.

Steps to create an AWS Lambda function

Step 1: Create a Lambda Function

1. Choose a Function Creation Method: Select Author from scratch.

2. Configure the Function:

Function name: Enter a name for your function (e.g., MyFirstLambda).

Runtime: Choose Python 3.x (the latest available version).

Permissions: Choose Create a new role with basic Lambda permissions (this creates a role with the necessary permissions).

3. Click on Create function.

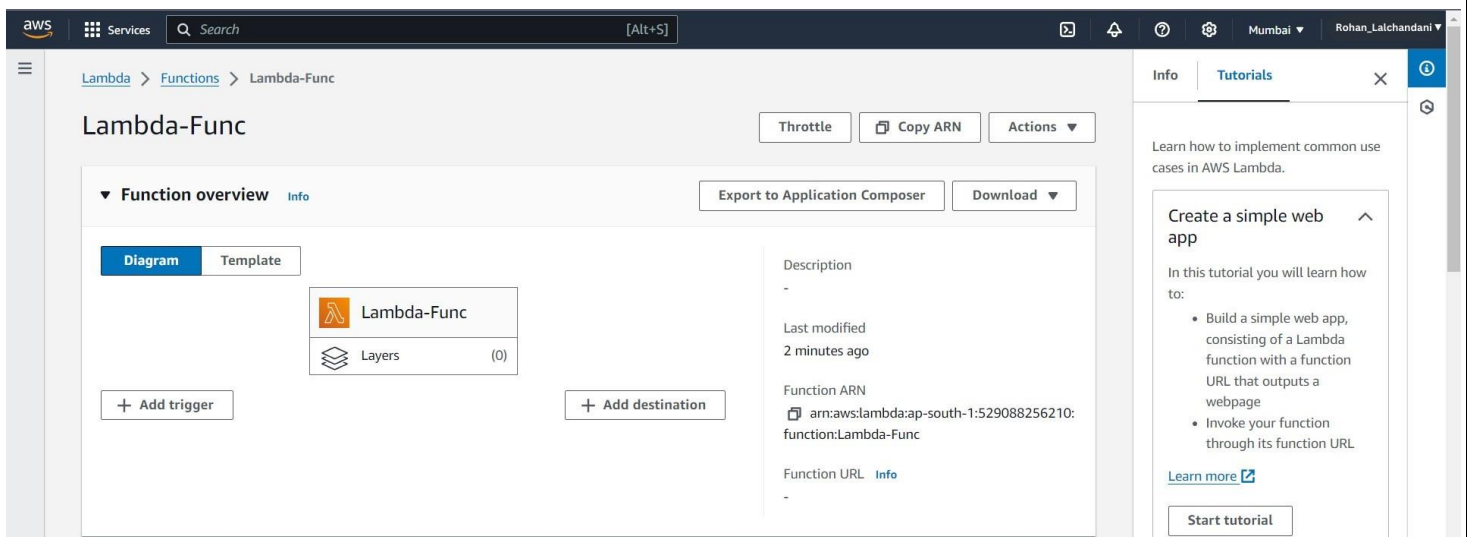
The screenshot shows the AWS Lambda 'Create function' page in a web browser. The page has a dark header with the AWS logo, 'Services' menu, a search bar, and user information 'Mumbai' and 'Rohan_Lalchandani'. The main content area is titled 'Create function' with a sub-header 'Choose one of the following options to create your function.' There are three tabs: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below these is the 'Basic information' section with fields for 'Function name' (containing 'Lambda-Func'), 'Runtime' (set to 'Python 3.12'), and 'Architecture' (set to 'x86_64'). A 'Permissions' section is partially visible at the bottom. On the right, there is a 'Tutorials' sidebar with a 'Create a simple web app' tutorial card that includes a 'Start tutorial' button. The footer contains 'CloudShell', 'Feedback', and copyright information for Amazon Web Services, Inc. or its affiliates.

Step 2: Write Your Lambda Function Code

In the Function code section, you will see a code editor. Replace the default code with the following Python code:

```
python Copy code def lambda_handler(event, context): #
This function returns a greeting message name =
event.get('name', 'World') return {
    'statusCode': 200, 'body': f'Hello, {name}!' }
```

This function reads a name from the event and returns a greeting message. If no name is provided, it defaults to "World".



Step3: 1. Configure a Test Event:

Click on the Test button.

In the Configure test event dialog, give your event a name (e.g., TestEvent). Replace the default JSON with the following:

```
{
  "name": "Lambda User"
}
```

2. Run the Test:

Click on the Test button again to execute your Lambda function.

You should see the execution results below the code editor, including the response: json

Copy code

```
{
  "statusCode": 200,
  "body": "Hello, Lambda User!"
}
```

Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

Create new event

Edit saved event

Event name

TestEvent

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

Event sharing settings

Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [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](#)

Template - optional

hello-world

Event JSON

Format JSON

1 {

2 "name": "Lambda User"

3 }

4

Cancel

Invoke

Save

The screenshot displays the AWS Lambda console interface. At the top, there's a navigation bar with the AWS logo, 'Services' link, a search bar, and a user profile 'Rohan_Latchandani'. The main content area is titled 'Code source' and includes tabs for 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test', and 'Deploy'. The 'Test' tab is active, showing 'Execution results' for a test event named 'soham_test'. The response is a JSON object: `{ "statusCode": 200, "body": "\nHello from Lambda!\n" }`. Below this, 'Function Logs' show the execution details: `START RequestId: fb1457e5-f5cd-4787-bc2b-141c111b7271 Version: $LATEST`, `END RequestId: fb1457e5-f5cd-4787-bc2b-141c111b7271`, and `REPORT RequestId: fb1457e5-f5cd-4787-bc2b-141c111b7271 Duration: 1.55 ms Billed Duration: 2 ms Memory Size: 128 MB Max Me`. The 'Request ID' is `fb1457e5-f5cd-4787-bc2b-141c111b7271`. On the right, a 'Tutorials' sidebar offers a guide on 'Create a simple web app' with a 'Start tutorial' button.

This screenshot shows the 'Code source' tab in the AWS Lambda console for a function named 'Lambda-Func'. The code is a Python lambda function that returns a JSON response with a status code of 200 and a body of 'Hello from Lambda!'. The code is as follows:

```
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
```

Conclusion:

AWS Lambda is a serverless computing service that allows you to run code without managing servers, making it highly scalable, cost-effective, and easy to use. It automatically manages the compute resources, executes your code in response to specific events such as API calls, file uploads, or database updates, and scales based on the demand.