

# Experiment 11

**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.

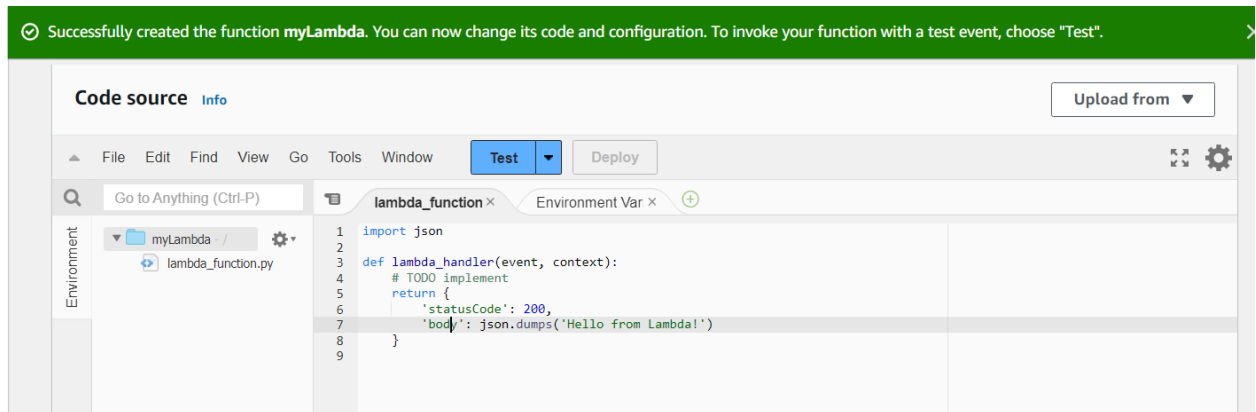
Data processing. Lambda functions are optimized for event-based data processing. It is easy to integrate AWS Lambda with data sources like Amazon DynamoDB and trigger a Lambda function for specific kinds of data events. For example, you could employ Lambda to do some work every time an item in DynamoDB is created or updated, thus making it a good fit for things like notifications, counters and analytics.

## Steps to create an AWS Lambda function

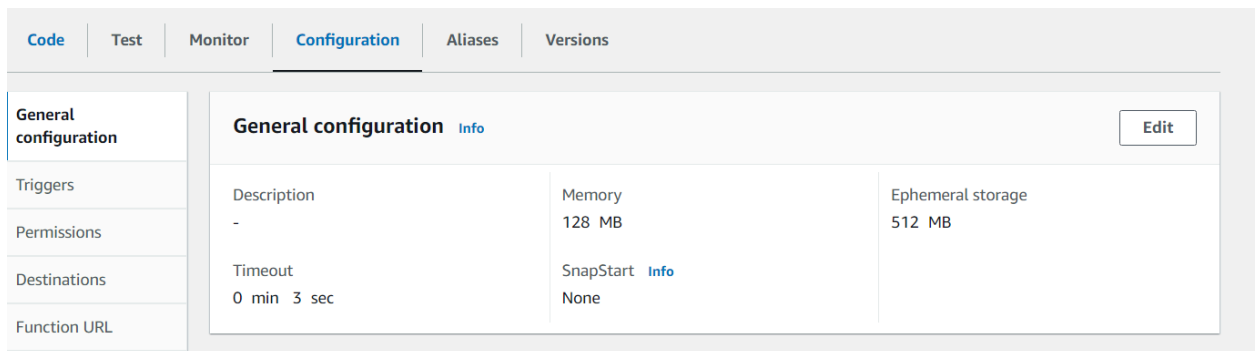
1. Open up the Lambda Console and click on the Create button.
2. Choose to create a function from scratch or use a blueprint, i.e templates defined by AWS for you with all configuration presets required for the most common use cases.

The screenshot shows the 'Create function' page in the AWS Lambda console. At the top, there's a breadcrumb trail: 'Lambda > Functions > Create function'. Below this is the title 'Create function' with an 'Info' link. A subtitle reads: 'Choose one of the following options to create your function.' There are three main options, each in a box with a radio button: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. The 'Author from scratch' box contains the text: 'Start with a simple Hello World example.' Below these options is a section titled 'Basic information'. It contains three fields: 'Function name' with a text input containing 'mylambda' and a description 'Enter a name that describes the purpose of your function.'; 'Runtime' with a dropdown menu set to 'Python 3.12' and a description 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.'; and 'Architecture' with radio buttons for 'x86\_64' (selected) and 'arm64', and a description 'Choose the instruction set architecture you want for your function code.'

3. This process will take a while to finish and after that, you'll get a message that your function was successfully created.



4. To change the configuration, open up the Configuration tab and under General Configuration, choose Edit.
- Here, you can enter a description and change Memory and Timeout. I've changed the Timeout period to 1 sec since that is sufficient for now.



**Basic settings** [Info](#)

Description - optional

**Memory** [Info](#)  
Your function is allocated CPU proportional to the memory configured.  
 MB  
Set memory to between 128 MB and 10240 MB

**Ephemeral storage** [Info](#)  
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#)  
 MB  
Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

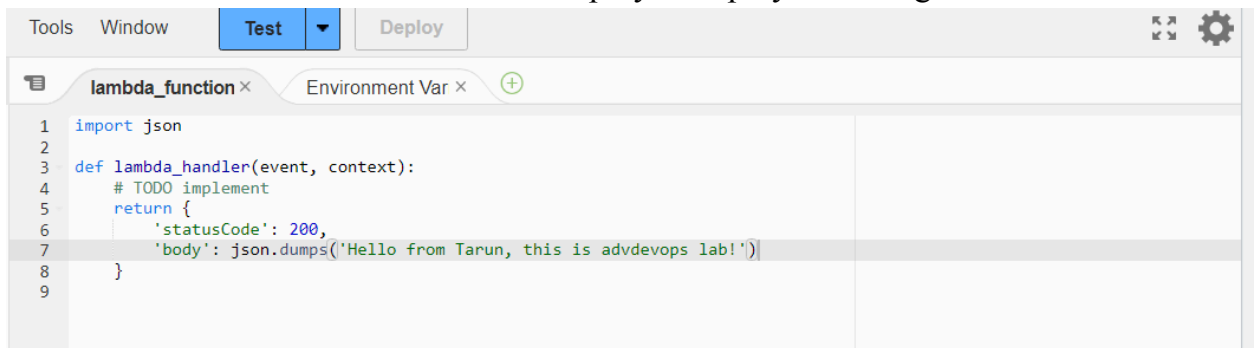
**SnapStart** [Info](#)  
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the [SnapStart compatibility considerations](#).

Supported runtimes: Java 11, Java 17, Java 21.

**Timeout**  
 min  sec

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
☒ Use an existing role

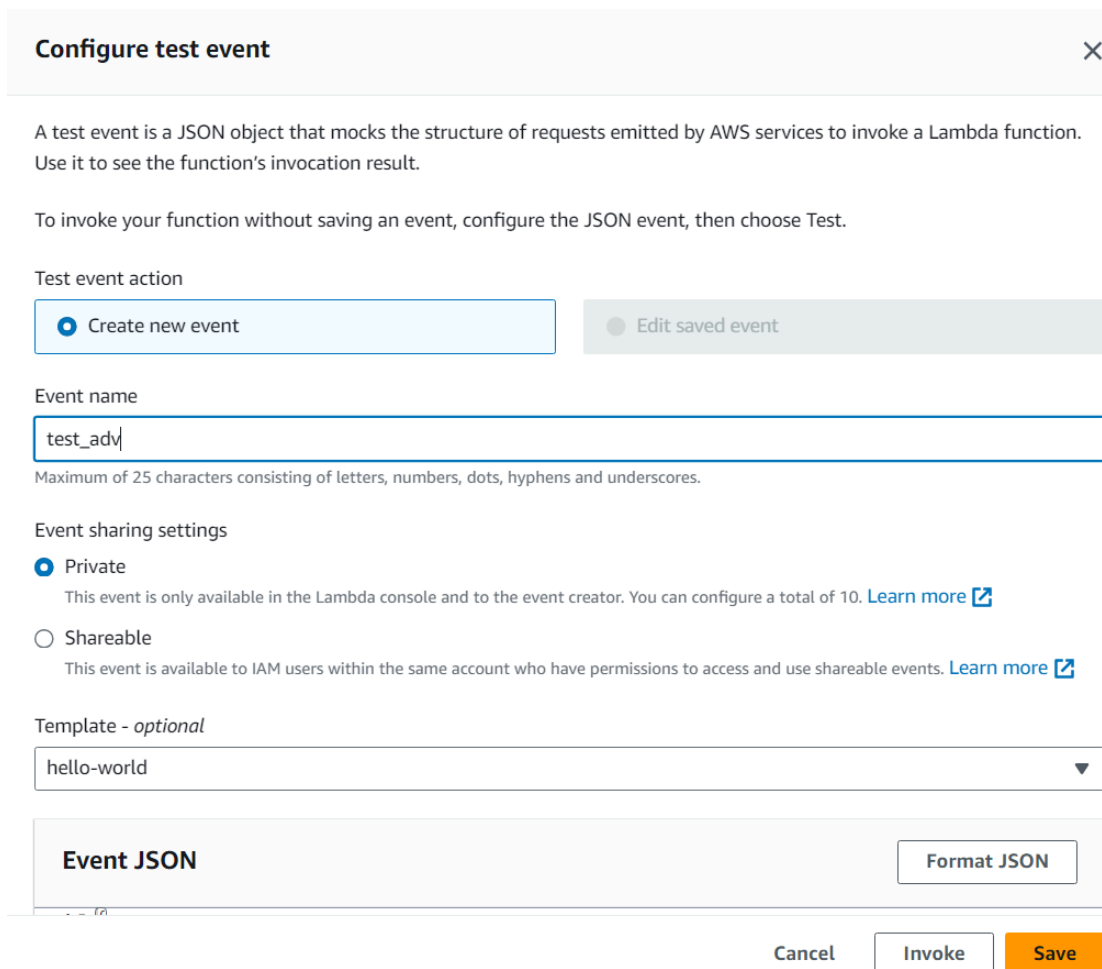
5. You can make changes to your function inside the code editor. You can also upload a zip file of your function or upload one from an S3 bucket if needed. Press Ctrl + S to save the file and click Deploy to deploy the changes.



The screenshot shows the AWS Lambda console's code editor. At the top, there are tabs for 'Tools', 'Window', 'Test', and 'Deploy'. Below these, there are tabs for 'lambda\_function' and 'Environment Var'. The code editor displays a Python function named 'lambda\_handler' that takes 'event' and 'context' as arguments. The function returns a JSON object with 'statusCode': 200 and 'body': 'Hello from Tarun, this is advdevops lab!'. 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 Tarun, this is advdevops lab!')}
8
9
```

6. Click on Test and you can change the configuration, like so. If you do not have anything in the request body, it is important to specify two curly braces as valid JSON, so make sure they are there.



The screenshot shows the 'Configure test event' dialog in the AWS Lambda console. The dialog has a title bar with a close button (X). Below the title bar, there is a description: '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.' Below this, there is a note: 'To invoke your function without saving an event, configure the JSON event, then choose Test.'

The 'Test event action' section has two radio buttons: 'Create new event' (selected) and 'Edit saved event'.

The 'Event name' section has a text input field containing 'test\_adv'. Below the input field, there is a note: 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.'

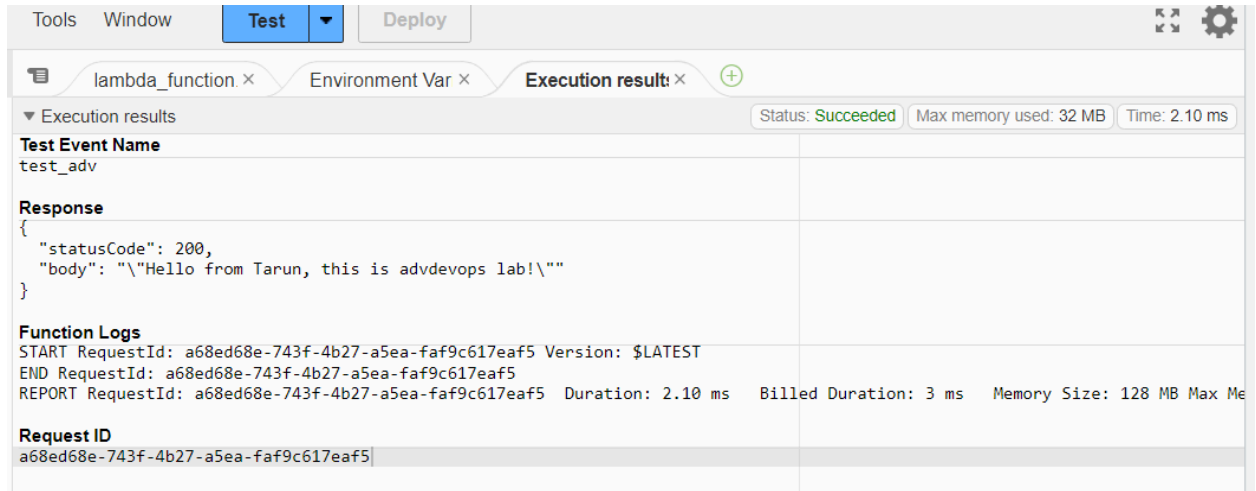
The 'Event sharing settings' section has two radio buttons: 'Private' (selected) and 'Shareable'. Below the 'Private' radio button, there is a note: 'This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)'. Below the 'Shareable' radio button, there is a note: 'This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)'.

The 'Template - optional' section has a dropdown menu with 'hello-world' selected.

The 'Event JSON' section has a text area for entering the JSON event. To the right of the text area is a 'Format JSON' button.

At the bottom of the dialog, there are three buttons: 'Cancel', 'Invoke', and 'Save'.

7. Now click on Test and you should be able to see the results.



The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'Tools', 'Window', 'Test', and 'Deploy'. The 'Test' tab is selected. Below the tabs, there are three sub-tabs: 'lambda\_function', 'Environment Var', and 'Execution result'. The 'Execution result' tab is active, showing the execution details for a test event named 'test\_adv'. The status is 'Succeeded', with a max memory used of 32 MB and a time of 2.10 ms. The response is a JSON object: 

```
{  "statusCode": 200,  "body": "\"Hello from Tarun, this is advdevops lab!\""}
```

. The function logs show the start and end of the request, with a duration of 2.10 ms, billed duration of 3 ms, and memory size of 128 MB. The request ID is 'a68ed68e-743f-4b27-a5ea-faf9c617eaf5'.

| Test Event Name | Response  | Function Logs   | Request ID                           |
|-----------------|---|---|--------------------------------------|
| test_adv        | <pre>{  "statusCode": 200,  "body": "\"Hello from Tarun, this is advdevops lab!\""}</pre> | START RequestId: a68ed68e-743f-4b27-a5ea-faf9c617eaf5 Version: \$LATEST<br>END RequestId: a68ed68e-743f-4b27-a5ea-faf9c617eaf5<br>REPORT RequestId: a68ed68e-743f-4b27-a5ea-faf9c617eaf5 Duration: 2.10 ms Billed Duration: 3 ms Memory Size: 128 MB Max Me | a68ed68e-743f-4b27-a5ea-faf9c617eaf5 |

## Conclusion:

In this experiment, we learned about AWS Lambda and using it to create, deploy and test serverless functions in the Cloud.