Guided Lab: Creating an AWS Lambda function

Description

AWS Lambda is a serverless computing service that lets you run code without provisioning or managing servers. Unlike traditional server-based setups like EC2, where you're charged as long as your server is up, AWS Lambda matches costs directly with your actual usage. You pay only when your code is running, which is a big plus for event-based processes that don't need to run 24/7. AWS Lambda automatically scales out, making it a great fit for workloads that have unpredictable or varying demands.

In this lab, you will learn how to create a Lambda function, which is a piece of code that runs in response to events and automatically manages the computing resources required by that code. The lab will guide you through the process of setting up a Lambda function in the AWS Console and testing the function to ensure it works as expected.

Objectives

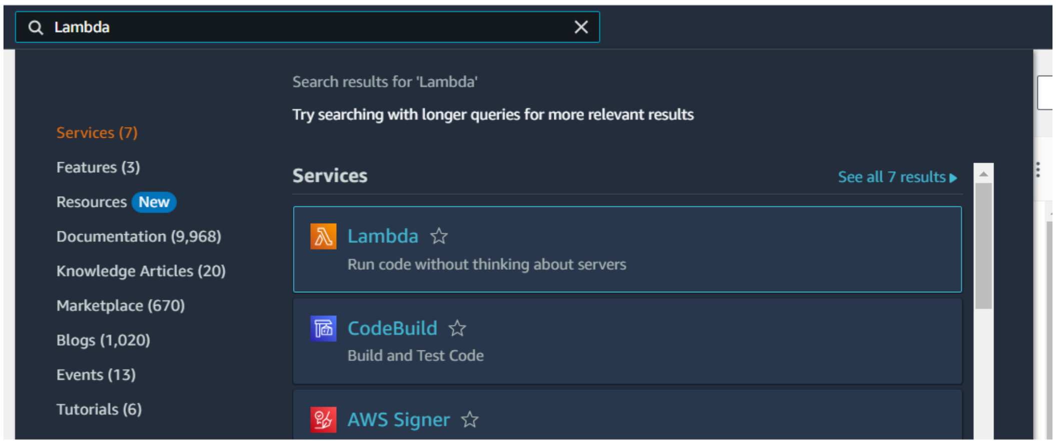
In this lab, you will:

- Have a basic understanding of Lambda functions
- Understand the cost model of AWS Lambda
- Create and run Lambda functions in the AWS Console using Python
- Understand the general configurations of a Lambda function

Lab Steps

Create and run Lambda functions in the AWS Console using Python

1. Navigate to the search bar, type “Lambda”, and click to open the Lambda.



Lambda

Search results for 'Lambda'

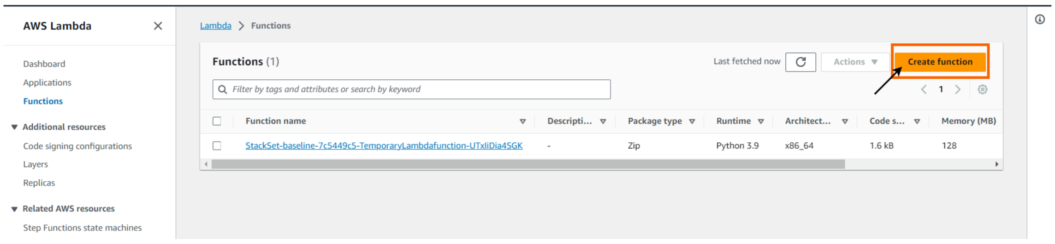
Try searching with longer queries for more relevant results

Lambda — Run code without thinking about servers

CodeBuild — Build and Test Code

AWS Signer

2. Click on the “Create function” button. You will be prompted to configure your function.



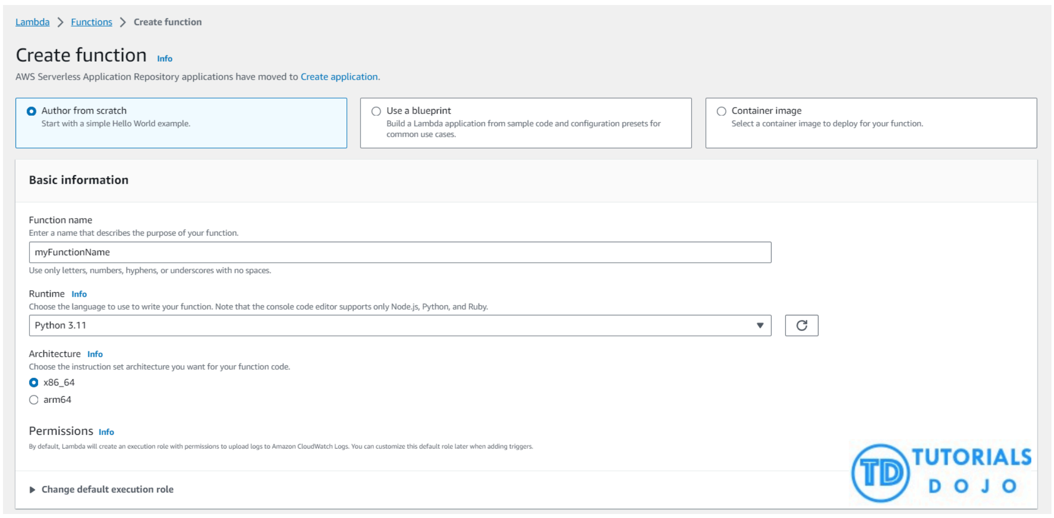
Functions (1)

Filter by tags and attributes or search by keyword

Function name	Descripti...	Package type	Runtime	Architect...	Code s...	Memory (MB)
StackSet-baseline-7c5449c5-TemporaryLambdafunction-UTxIiDia4SGK	-	Zip	Python 3.9	x86_64	1.6 kB	128

3. Configure your function:

- **Function name:** Give your function a unique name.
- **Runtime:** Select “Python 3.11” from the dropdown menu.



Lambda > Functions > Create function

Create function Info

AWS Serverless Application Repository applications have moved to Create application.

[✓ 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
Enter a name that describes the purpose of your function.
myFunctionName
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 3.11

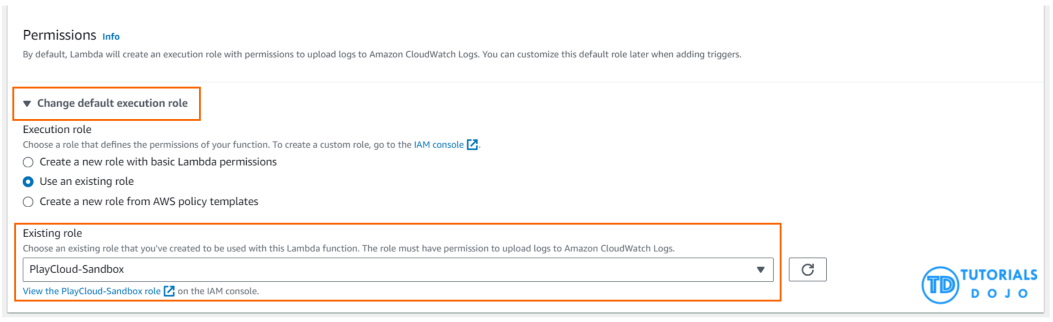
Architecture Info
Choose the instruction set architecture you want for your function code.
[✓ x86_64] arm64

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

4.To change the default execution role, please follow these steps:

- Go to the "Permissions" tab.
- Navigate to the section where you can edit the execution role.
- Modify the default execution role to "Use an existing role".
- Choose the existing role named "PlayCloud-Sandbox".



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

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

PlayCloud-Sandbox

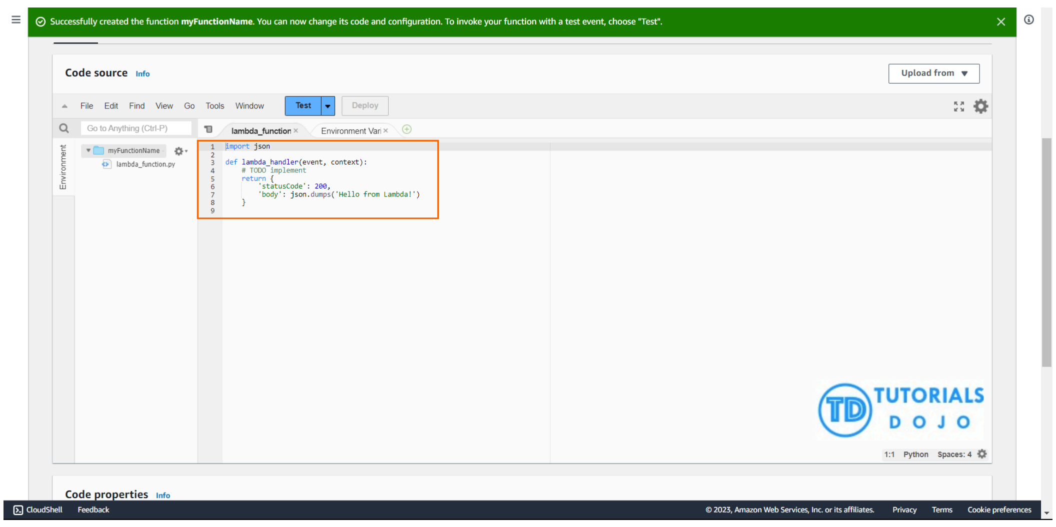
View the PlayCloud-Sandbox role on the IAM console.

An execution role grants the Lambda function permissions to access AWS services and resources. For example, if you want your Lambda function to make API calls to access an S3 bucket, its execution role would need to include policies allowing it to read and write to that specific bucket. In this lab, an execution role with the necessary permissions has already been created for you.

5. Click "Create Function"

6. In the Function code section, you will see an inline code editor where you can write your Python code. AWS provides a simple "Hello from Lambda!" example to get you started.

***Note**: We are using the **old console editor** for this lab. You can switch to the **new editor** as you desire; the process remains the same, but the interface may look slightly different.*



Successfully created the function **myFunctionName**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

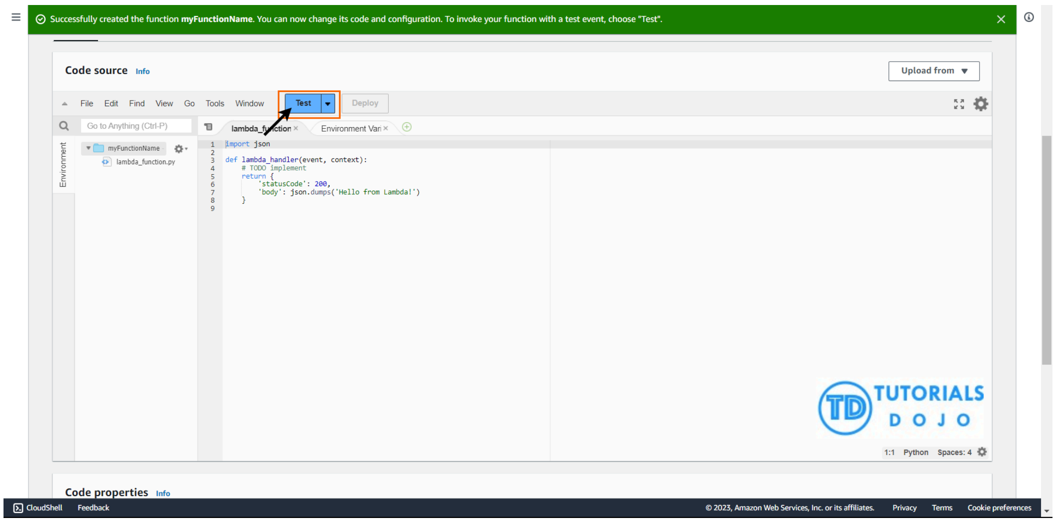
Code source Info

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

7. If you have changes to the code, click on the “Deploy” button at the top to save your function.

8. Test your function:

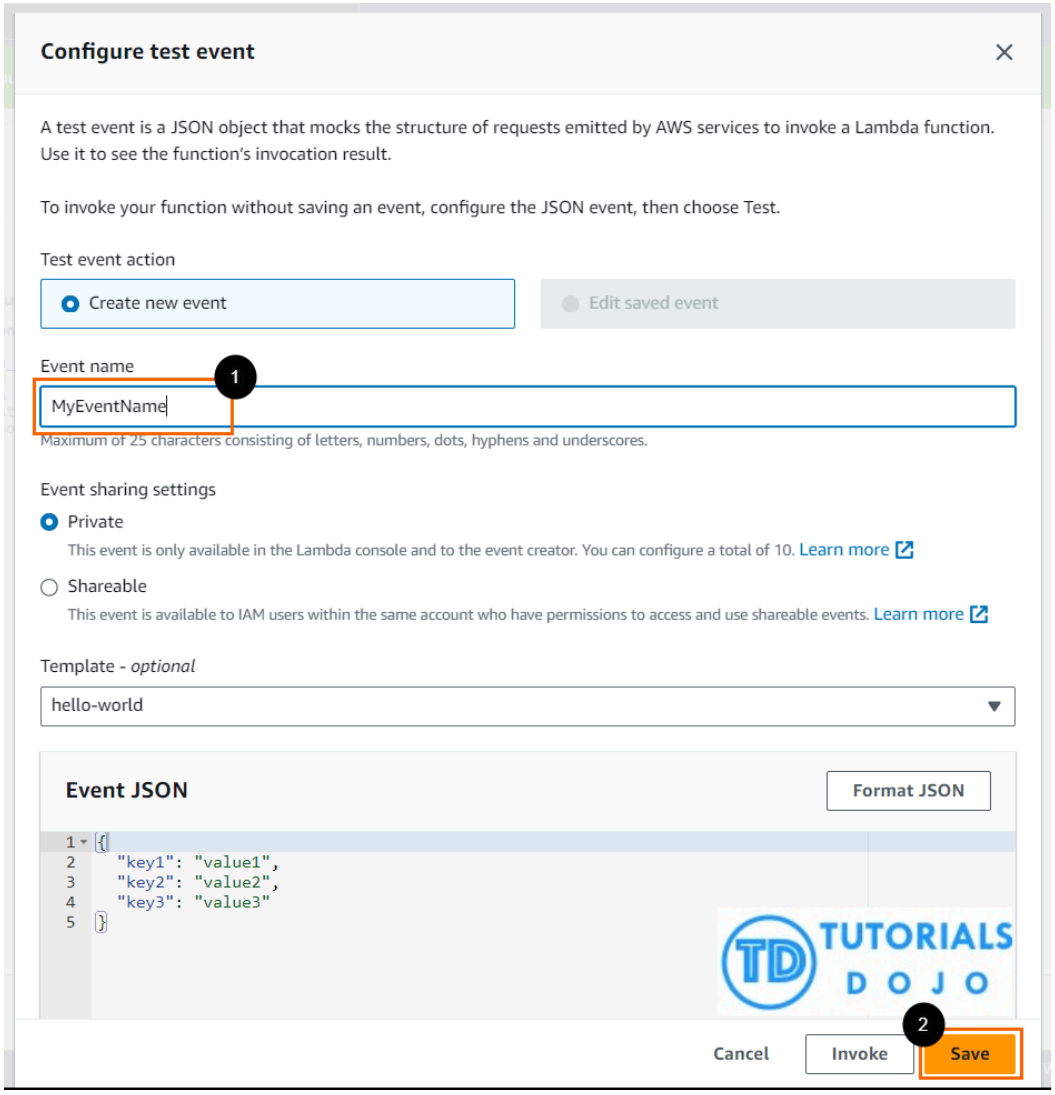
- Click on the “Test” button at the top of the page.



Successfully created the function **myFunctionName**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

- In the Configure test event dialog, give your test event a name, and provide a JSON-formatted input for your function. Then, click on “Save” to close the dialog and create your test event.



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: MyEventName

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

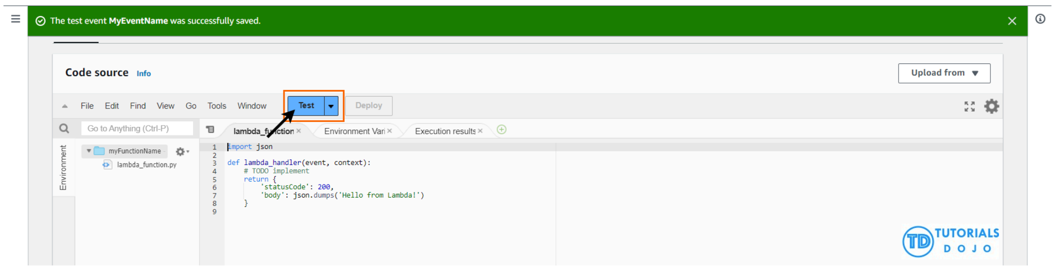
Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*: hello-world

Event JSON

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

- Back in the main console, click on "Test" again to run your function with your test event.

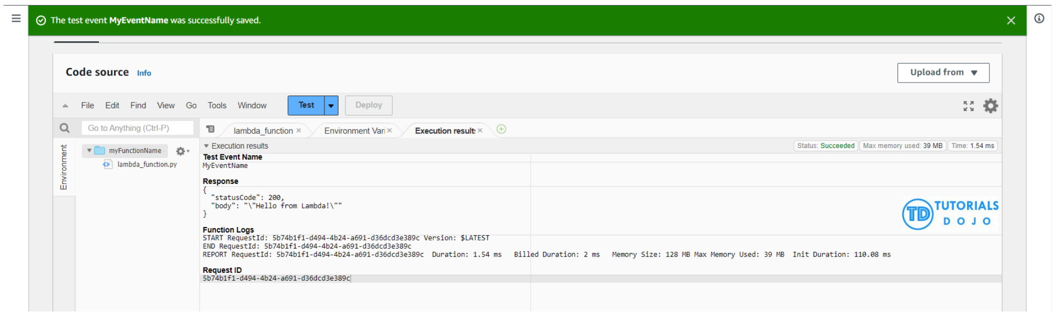


The test event **MyEventName** was successfully saved.

Code source Info

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

9. After running your function, you will see the execution results at the top of the page. This includes Test Event Name, Response, Function Logs, Duration of execution, and Request ID.



The test event **MyEventName** was successfully saved.

Code source Info

Execution results — Status: Succeeded | Max memory used: 39 MB | Time: 1.54 ms

Test Event Name

```
MyEventName
```

Response

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

Function Logs

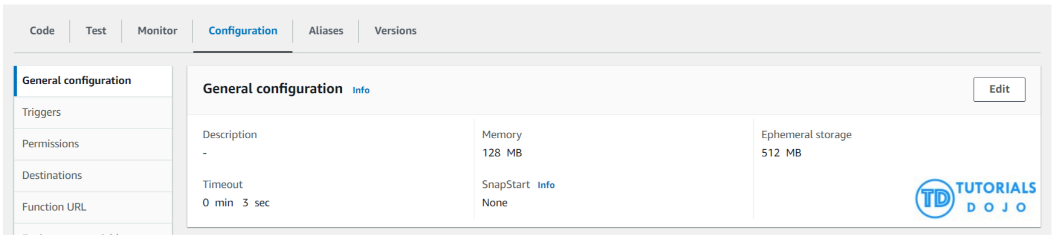
```
START RequestId: 5b74b1f1-d494-4b24-a691-d36dcd3e389c Version: $LATEST
END RequestId: 5b74b1f1-d494-4b24-a691-d36dcd3e389c
REPORT RequestId: 5b74b1f1-d494-4b24-a691-d36dcd3e389c  Duration: 1.54 ms   Billed Duration: 2 ms   Memory Size: 128 MB Max Memory Used: 39 MB  Init Duration: 110.08 ms
```

Request ID

```
5b74b1f1-d494-4b24-a691-d36dcd3e389c
```

General configuration

1. Go to the “Configuration” tab.



General configuration

Description
-

Memory
128 MB

Ephemeral storage
512 MB

Timeout
0 min 3 sec

SnapStart
None

2. **Ephemeral Storage –** This refers to the temporary storage space that a Lambda function has for reading or writing data. By default, AWS Lambda allocates 512 MB for a function’s /tmp directory. You can increase or decrease this amount using the “Ephemeral storage (MB)” setting1. To configure the size of a function’s /tmp directory, you can set a whole number value between 512 MB and 10,240 MB, in 1-MB increments. This feature allows you to control the amount of ephemeral storage a function gets, enabling you to use AWS Lambda for ETL jobs, ML inference, or other data-intensive workloads.

3. **SnapStart –** is a performance optimization feature developed by AWS that can significantly improve the startup time for applications. Lambda SnapStart for Java can improve startup performance for latency-sensitive applications by up to 10x at no extra cost, typically with no changes to your function code. The largest contributor to startup latency (often referred to as cold start time) is the time that Lambda spends initializing the function, which includes loading the function’s code, starting the runtime, and initializing the function code. With SnapStart, Lambda initializes your function when you publish a function version. Lambda takes a Firecracker microVM snapshot of the memory and disk state of the initialized execution environment, encrypts the snapshot, and caches it for low-latency access. When you invoke the function version for the first time, and as the invocations scale up, Lambda resumes new execution environments from the cached snapshot instead of initializing them from scratch, improving startup latency.

4. **Memory –** refers to the amount of memory available to your Lambda function at runtime.

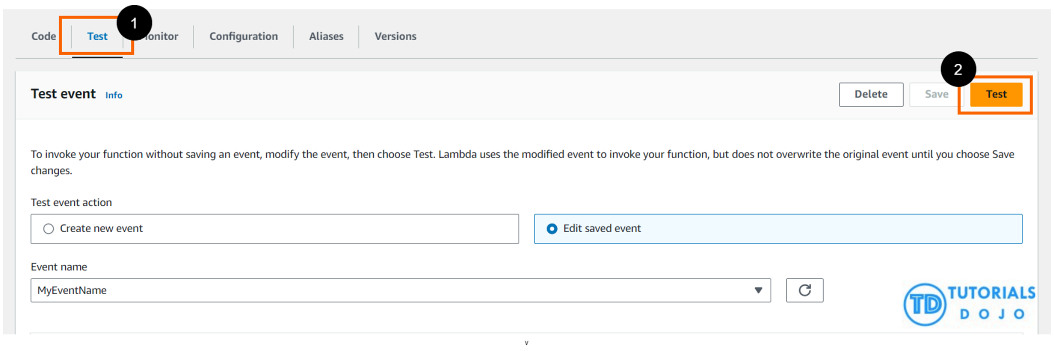
You can increase or decrease the memory and CPU power allocated to your function using the “Memory (MB)” setting. To configure the memory for your function, you can set a value between 128 MB and 10,240 MB in 1-MB increments. Lambda allocates CPU power in proportion to the amount of memory configured. Thus, increasing the memory also increases the CPU power, thereby increasing the overall computational power available.

To configure the memory for your function, you can follow these steps:

- Assume that we’re executing this particular Lambda Function. This code finds all prime numbers up to a certain limit.

```
import json

def lambda_handler(event, context):
    upper_limit = int(event.get('upper_limit', 1000000))
    primes = find_prime(upper_limit)
    return {
        'statusCode': 200,
        'body': json.dumps({'message': f'Found 
{len(primes)} primes up to {upper_limit}.'})
    }

def find_prime(n):
    sieve = [True] * (n + 1)
    sieve[0:2] = [False, False]
    for current in range(2, int(n**0.5) + 1):
        if sieve[current]:
            sieve[current*current::current] = [False] * 
len(range(current*current, n+1, current))
    return [i for i, v in enumerate(sieve) if v]
```

- To save the changes made in the code, click on the “Deploy” button.
- To test your code, head over to the “Test” tab and click on the “Test” button.



Test event Info

To invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but does not overwrite the original event until you choose Save changes.

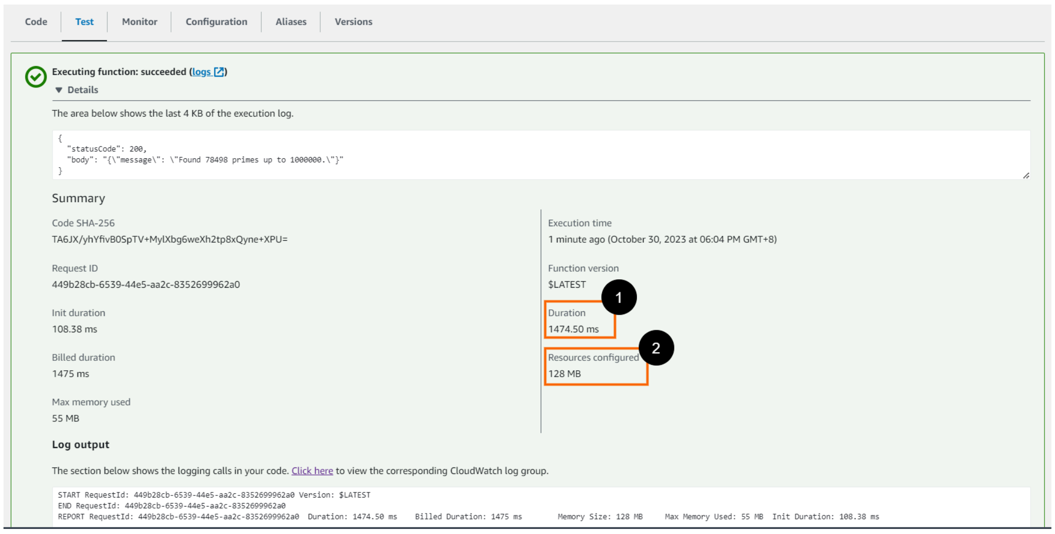
Test event action

Create new event [✓ Edit saved event]

Event name

MyEventName

- Take note of the result.



Executing function: succeeded (logs)

Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Found 78498 primes up to 1000000.\"}"
}
```

Summary

Code SHA-256
TA6JX/yhYfivB0SpTV+MylXbg6weXh2tp8xQyne+XPU=

Request ID
449b28cb-6539-44e5-aa2c-8352699962a0

Init duration
108.38 ms

Billed duration
1475 ms

Max memory used
55 MB

Execution time
1 minute ago (October 30, 2023 at 06:04 PM GMT+8)

Function version
$LATEST

Duration
1474.50 ms

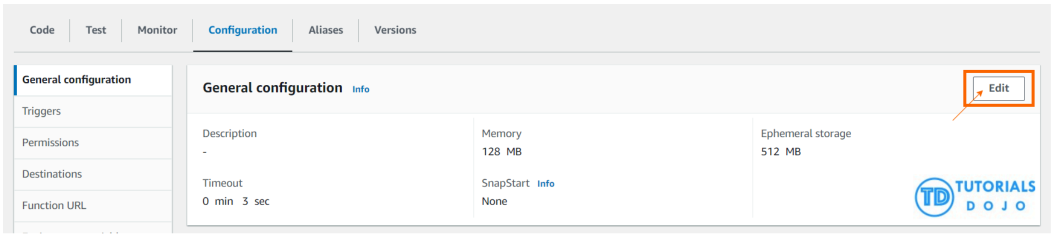
Resources configured
128 MB

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 449b28cb-6539-44e5-aa2c-8352699962a0 Version: $LATEST
END RequestId: 449b28cb-6539-44e5-aa2c-8352699962a0
REPORT RequestId: 449b28cb-6539-44e5-aa2c-8352699962a0  Duration: 1474.50 ms    Billed Duration: 1475 ms       Memory Size: 128 MB     Max Memory Used: 55 MB  Init Duration: 108.38 ms
```

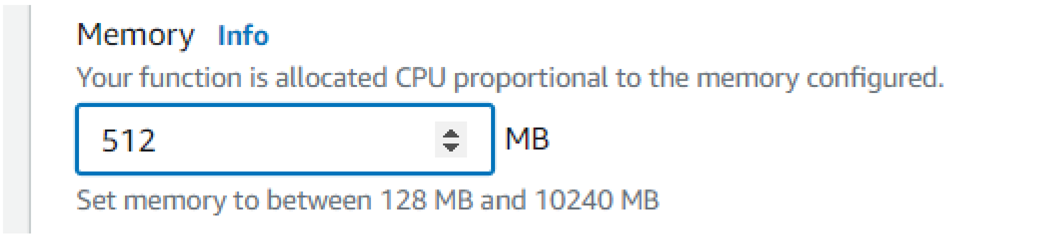
- On the function configuration page, on the General configuration pane, choose Edit.



General configuration Info

Description	Memory	Ephemeral storage
-	128 MB	512 MB
Timeout	SnapStart Info	
0 min 3 sec	None	

- In the “Memory (MB)” field, increase the memory from 128 MB to 512 MB.



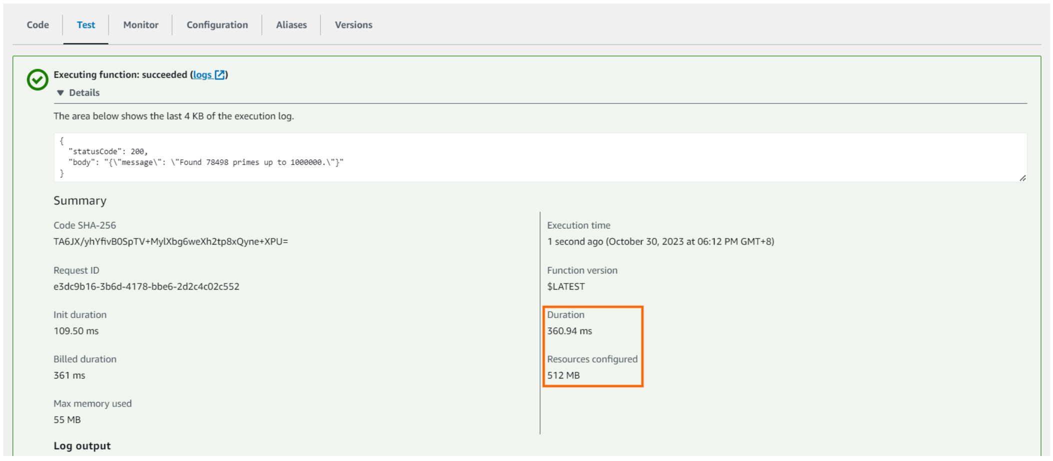
Memory Info

Your function is allocated CPU proportional to the memory configured.

512 MB

Set memory to between 128 MB and 10240 MB

- Click on the “Save” button to apply your changes.
- Head over to the “Test” tab again and click on the “Test” button.
- You will notice that the function executed faster than the first time we ran it, as indicated by the shorter duration.



Executing function: succeeded (logs)

Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Found 78498 primes up to 1000000.\"}"
}
```

Summary

Code SHA-256
TA6JX/yhYfivB0SpTV+MylXbg6weXh2tp8xQyne+XPU=

Request ID
e3dc9b16-3b6d-4178-bbe6-2d2c4c02c552

Init duration
109.50 ms

Billed duration
361 ms

Max memory used
55 MB

Execution time
1 second ago (October 30, 2023 at 06:12 PM GMT+8)

Function version
$LATEST

Duration
360.94 ms

Resources configured
512 MB

Log output

5. **Timeout –** refers to the maximum amount of time that your Lambda function can run before it is stopped by the Lambda services.

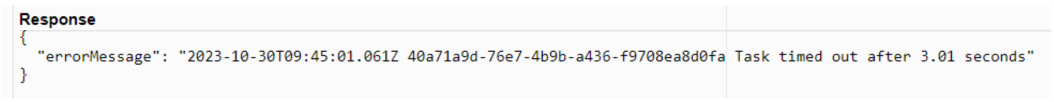
You can set the timeout value between 1 and 900 seconds (15 minutes). The default timeout value in the Lambda console is 3 seconds. This timeout value acts as a safety buffer that prevents functions that never exit from running indefinitely.

To set the timeout for an AWS Lambda function, you can follow these steps:

- Copy and paste the following code into your Lambda function. This code simulates a function performing actual processing by pausing the execution for 5 seconds. Since the default timeout is 3 seconds, the function will return a timeout error.

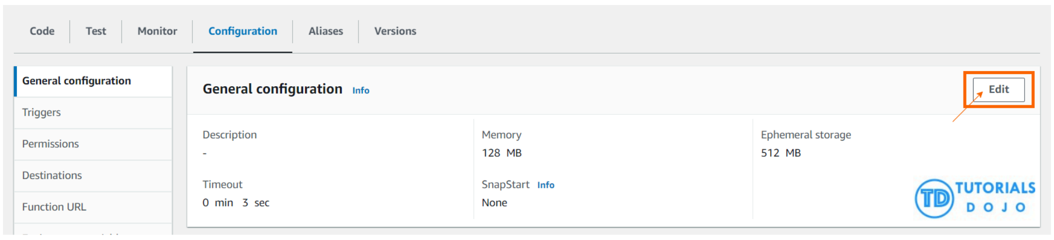
```
import json
import time

def lambda_handler(event, context):
    # TODO implement
    time.sleep(5);
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

- To save the changes made in the code, click on the “Deploy” button.
- Upon execution, the function will display an error message similar to the screenshot below:



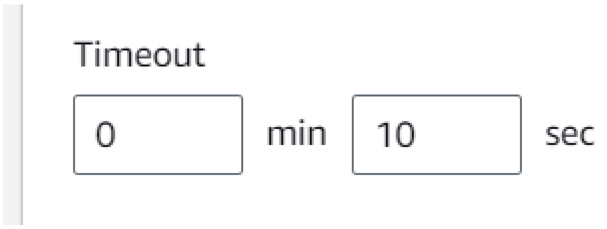
```
Response
{
  "errorMessage": "2023-10-30T09:45:01.061Z 40a71a9d-76e7-4b9b-a436-f9708ea8d0fa Task timed out after 3.01 seconds"
}
```

- On the General configuration pane, choose Edit.



Description
-
Memory
128 MB
Ephemeral storage
512 MB
Timeout
0 min 3 sec
SnapStart Info
None

- For Timeout, set a value from 3 seconds to 10 seconds.



Timeout
0 min 10 sec

- Click on the “Save” button to apply your changes.
- Go to the “Code” tab and execute the function again.
- Your function should be able to run now after configuring the timeout for your AWS Lambda function.

It’s worth noting that setting a timeout value close to the average duration of a function can increase the risk of unexpected timeouts. This is because the duration of a function can vary depending on factors such as the amount of data to be processed, the latency of the services it interacts with, and the amount of processing required.