

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

Theory:

AWS Lambda

A fully managed, serverless computing service where you run code without provisioning or managing servers. Lambda automatically scales your application based on the number of incoming requests or events, ensuring efficient resource utilization. You are only charged for the time your code is running, with no upfront cost, making it cost-effective for on-demand workloads.

Lambda Workflow

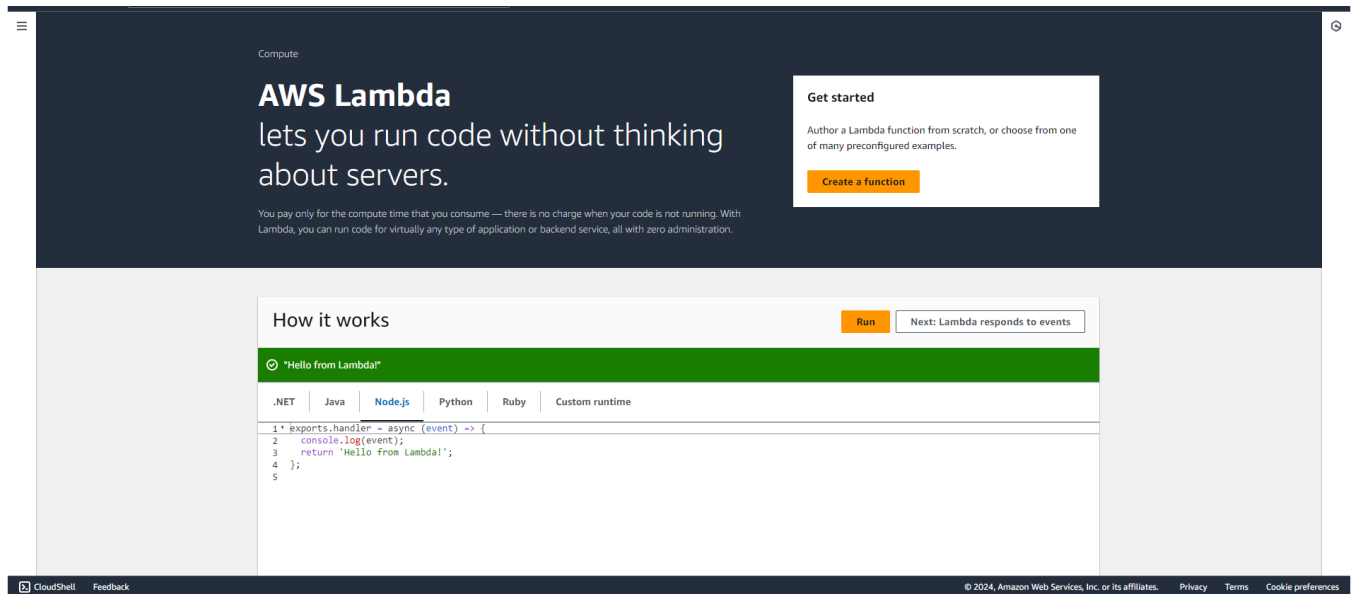
- **Create a Function:** Write the function code and define its handler (entry point). You can use the AWS Console, CLI, or upload a deployment package.
- **Set Event Sources:** Define how the function is triggered (e.g., when an object is uploaded to S3 or a DynamoDB table is updated).
- **Execution:** When triggered, Lambda runs your function, executes the logic, and automatically scales to handle the incoming event volume.
- **Scaling and Concurrency:** Lambda scales automatically by launching more instances of the function to handle simultaneous invocations. There are also options for configuring **reserved concurrency** to manage traffic.
- **Monitoring and Logging:** Lambda integrates with Amazon CloudWatch for logging and monitoring. Logs for each invocation are sent to CloudWatch, allowing you to track performance and troubleshoot errors.

AWS Lambda Functions

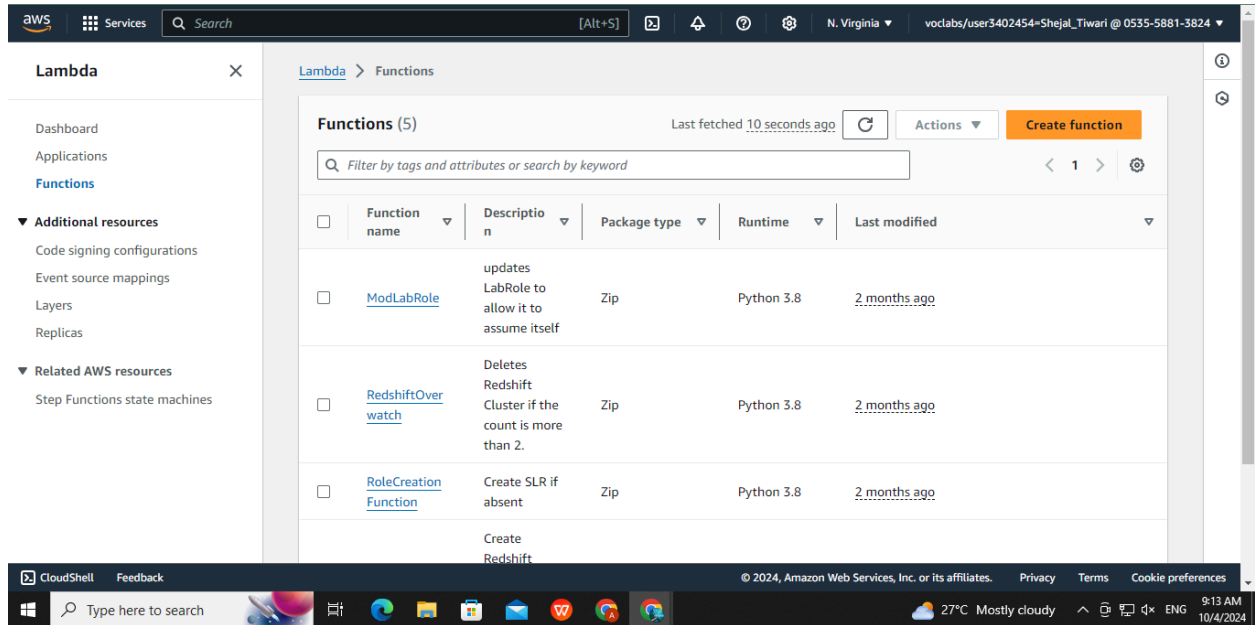
- **Python:** Great for quick development with its rich standard library and support for lightweight tasks.
- **Java:** Typically used for more complex, compute-intensive tasks. While it's robust, cold start times can be higher.
- **Node.js:** Excellent for I/O-bound tasks like handling APIs or streaming data, with fast startup times and efficient memory usage.

Steps To create the lambda function:

Step 1: Login to your AWS Personal/Academy Account. Open lambda and click on create function button.

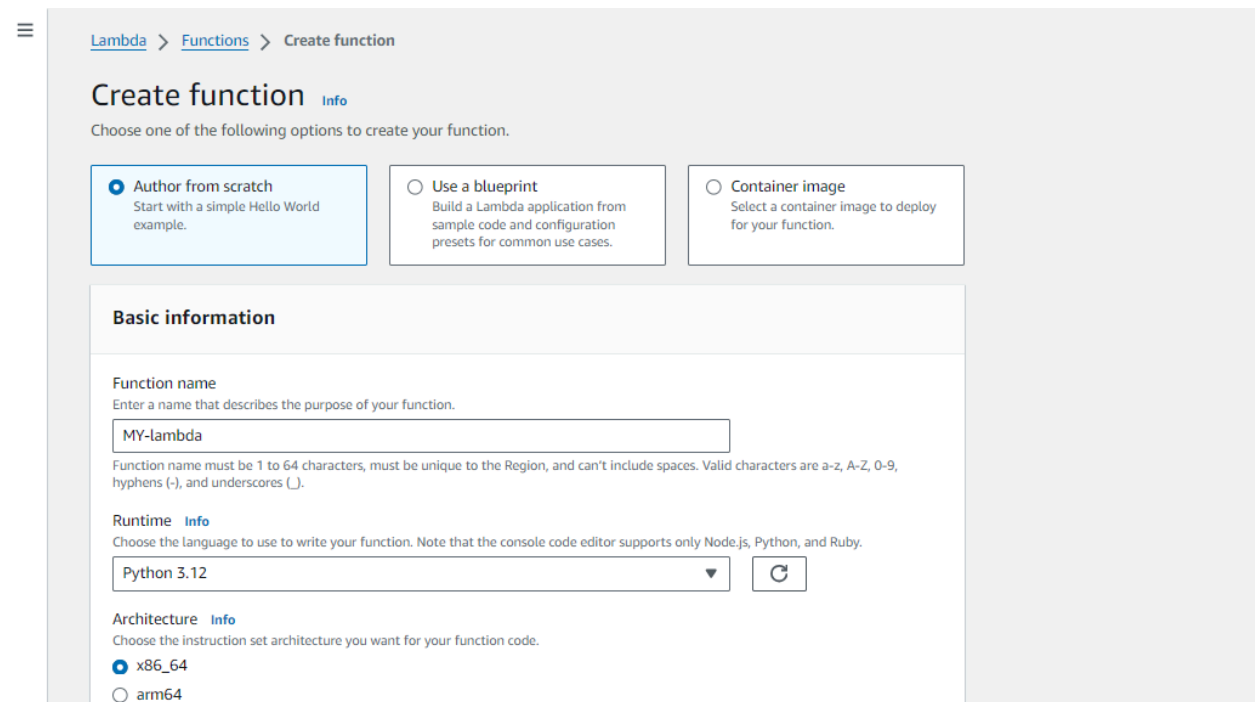


Step 2: Now Give a name to your Lambda function, Select the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby. So will select Python 3.12, Architecture as x86, and Execution role to Create a new role with basic Lambda permissions.



The screenshot shows the AWS Lambda console interface. The left sidebar contains navigation links for Dashboard, Applications, Functions, and Additional resources. The main content area displays a list of functions under the heading "Functions (5)". The list includes functions like "ModLabRole", "RedshiftOverwatch", and "RoleCreationFunction". Each function entry shows its name, description, package type (Zip), runtime (Python 3.8), and last modified time (2 months ago). A "Create function" button is visible in the top right corner.

Function name	Description	Package type	Runtime	Last modified
ModLabRole	updates LabRole to allow it to assume itself	Zip	Python 3.8	2 months ago
RedshiftOverwatch	Deletes Redshift Cluster if the count is more than 2.	Zip	Python 3.8	2 months ago
RoleCreationFunction	Create SLR if absent	Zip	Python 3.8	2 months ago



The screenshot shows the "Create function" page in the AWS Lambda console. It offers three options to create a function: "Author from scratch" (selected), "Use a blueprint", and "Container image". Below these options is the "Basic information" section, which includes fields for "Function name" (MY-lambda), "Runtime" (Python 3.12), and "Architecture" (x86_64).

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

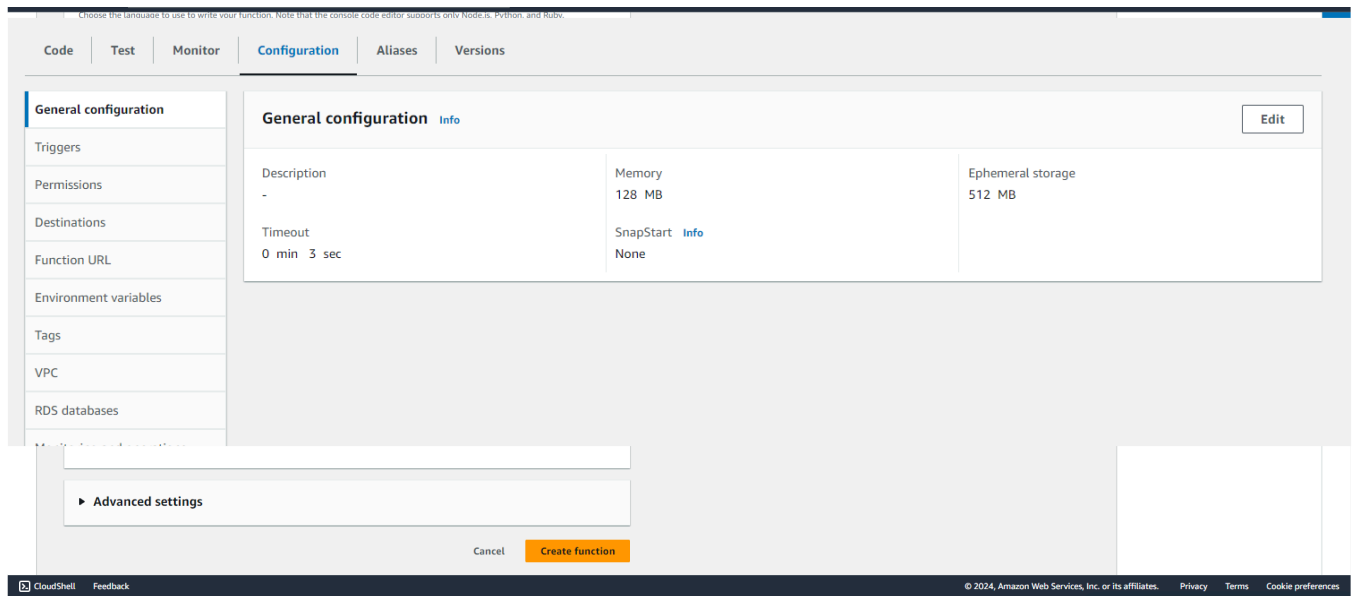
Function name
Enter a name that describes the purpose of your function.

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.
 [Refresh](#)

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
☒ **x86_64**
☐ **arm64**

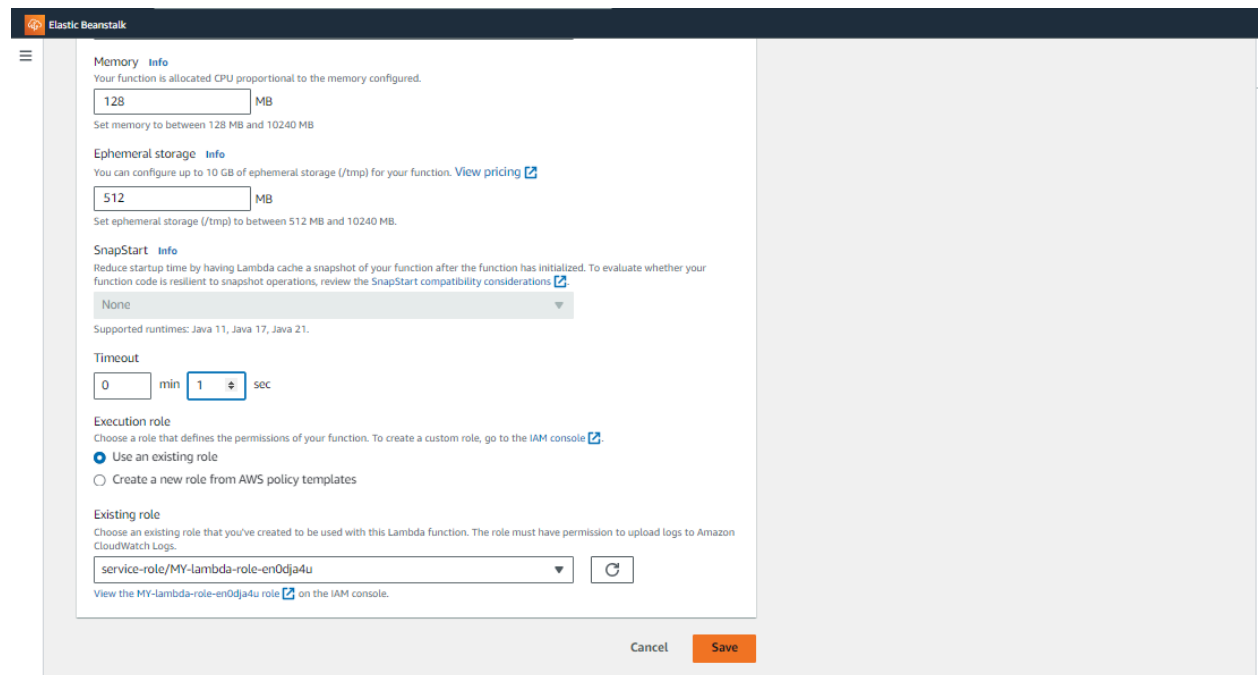
So See or Edit the basic settings go to configuration then click on edit general setting.



The screenshot shows the AWS Lambda Configuration page. The 'Configuration' tab is selected. On the left, a sidebar lists various configuration options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, and RDS databases. The 'General configuration' section is expanded, showing fields for Description, Memory (128 MB), Ephemeral storage (512 MB), Timeout (0 min 3 sec), and SnapStart (None). An 'Edit' button is in the top right. At the bottom, there are 'Cancel' and 'Create function' buttons.

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

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.



The screenshot shows the 'Advanced settings' page for an AWS Lambda function. It includes sections for Memory (128 MB), Ephemeral storage (512 MB), SnapStart (None), Timeout (0 min 1 sec), Execution role (Use an existing role), and Existing role (service-role/MY-lambda-role-en0dja4u). 'Cancel' and 'Save' buttons are at the bottom.

Memory [Info](#)
Your function is allocated CPU proportional to the memory configured.
128 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](#)
512 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](#).
None

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

Timeout
0 min 1 sec

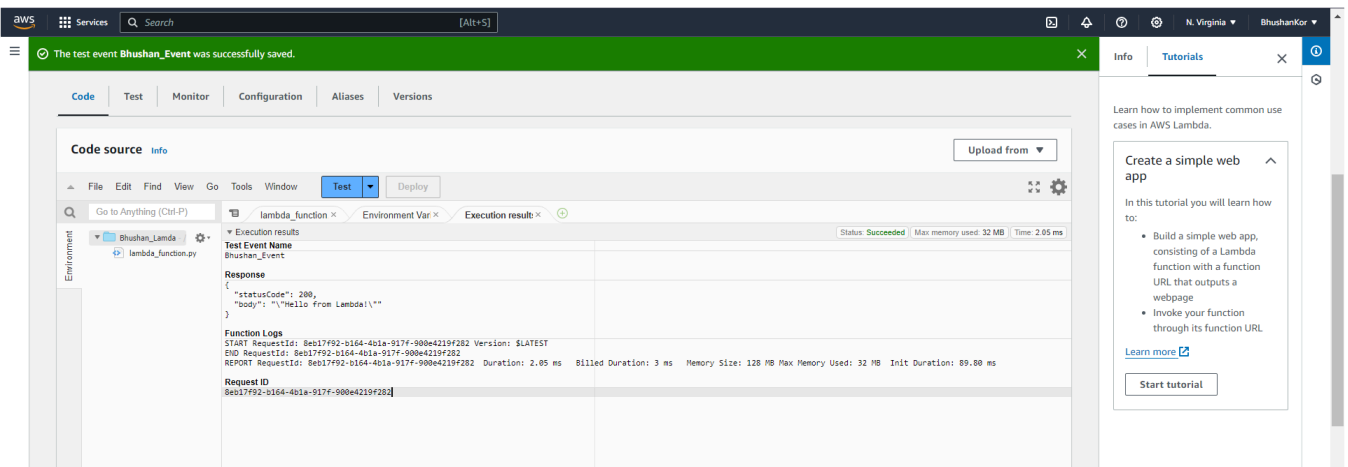
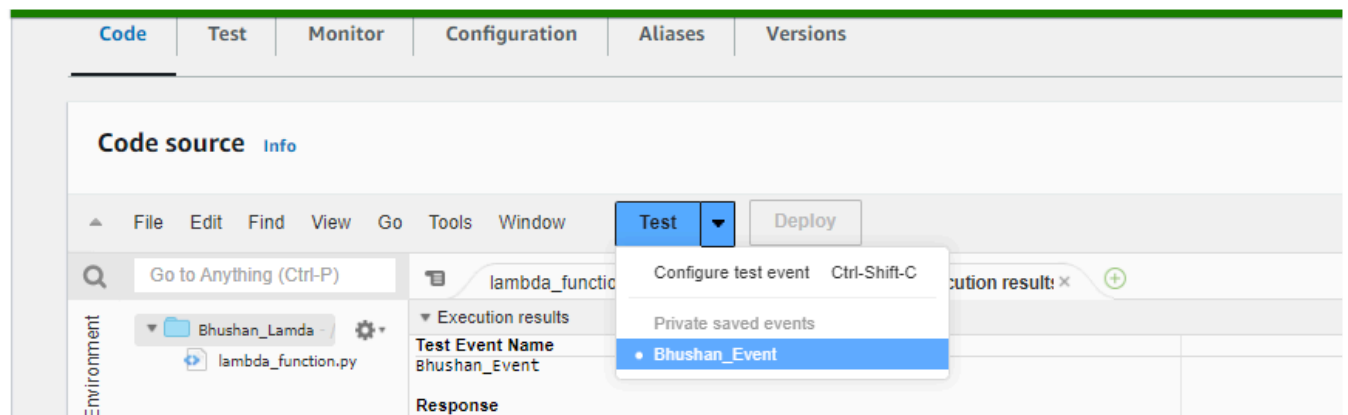
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).
☒ Use an existing role
☐ Create a new role from AWS policy templates

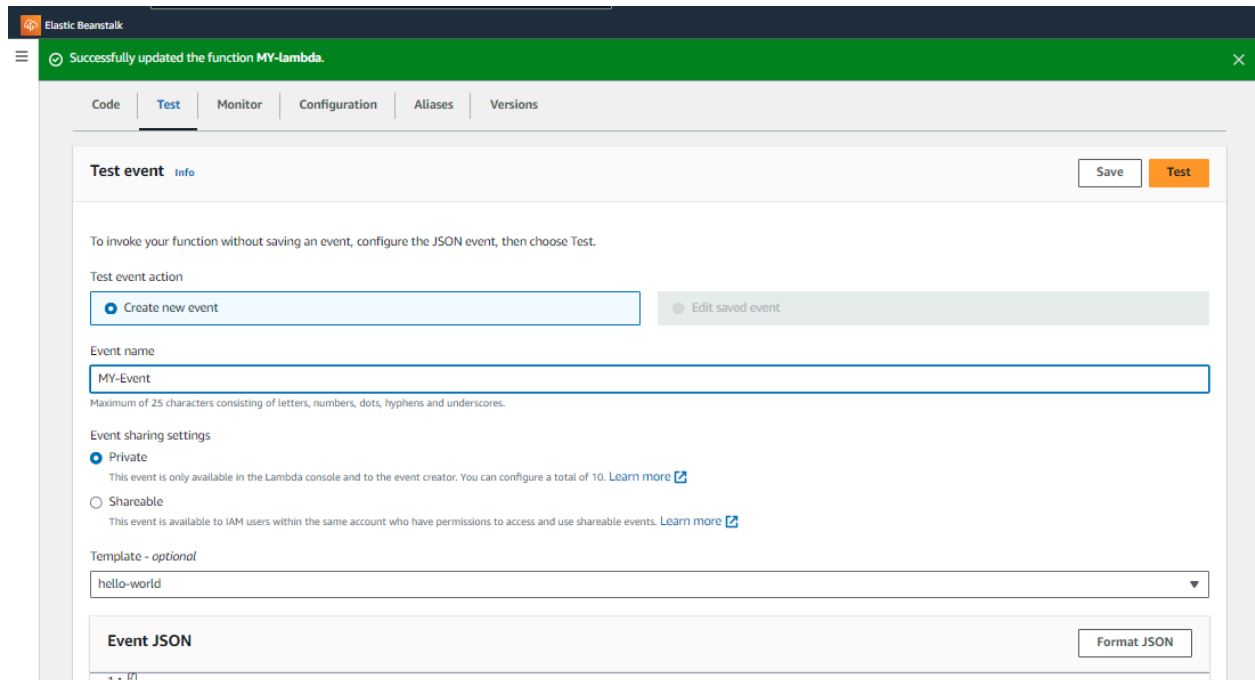
Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
service-role/MY-lambda-role-en0dja4u

[View the MY-lambda-role-en0dja4u role](#) on the IAM console.

Step 3: Now Click on the Test tab then select Create a new event, give a name to the event and select Event Sharing to private, and select hello-world template.

Step 4: Now In Code section select the created event from the dropdown of test then click on test . You will see the below output.





Successfully updated the function MY-lambda.

Code | **Test** | Monitor | Configuration | Aliases | Versions

Test event Info Save Test

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

MY-Event

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](#)

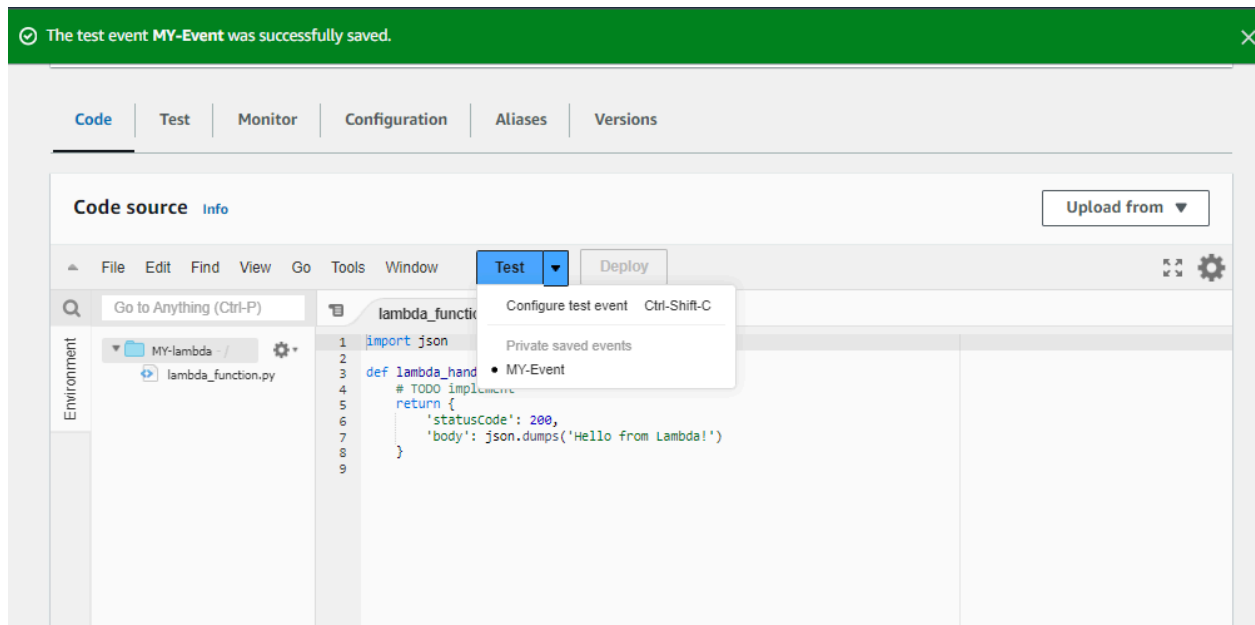
☐ Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

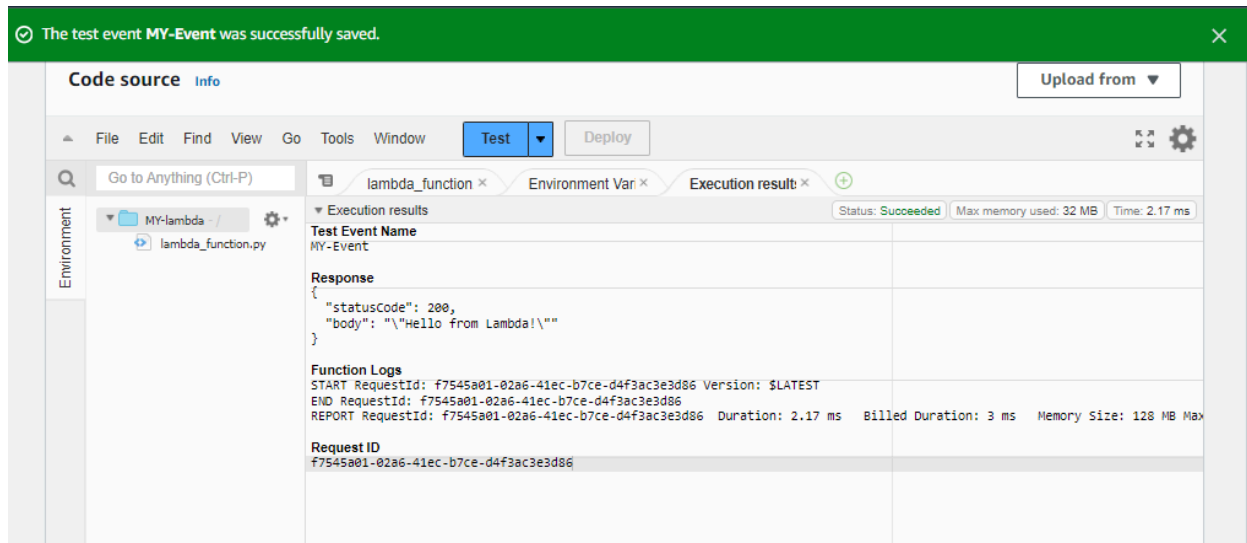
hello-world

Event JSON Format JSON

Step 5: You can edit your lambda function code. I have changed the code to display the new String. Now ctrl+s to save and click on deploy to deploy the changes.



Step 6: Now click on the test and observe the output. We can see the status code 200 and your string output and function logs. On successful deployment.

**Conclusion:**

This practical experience demonstrated the simplicity and flexibility of AWS Lambda in creating serverless applications, allowing you to focus on code while AWS manages the infrastructure and scaling.

Errors faced during the experiment :

After changing the code it's important to save the code then deploy it and then click on test to see successful output otherwise we might see errors after clicking on the test.