

**AdvanceDevOps Experiment No: 11**

**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**

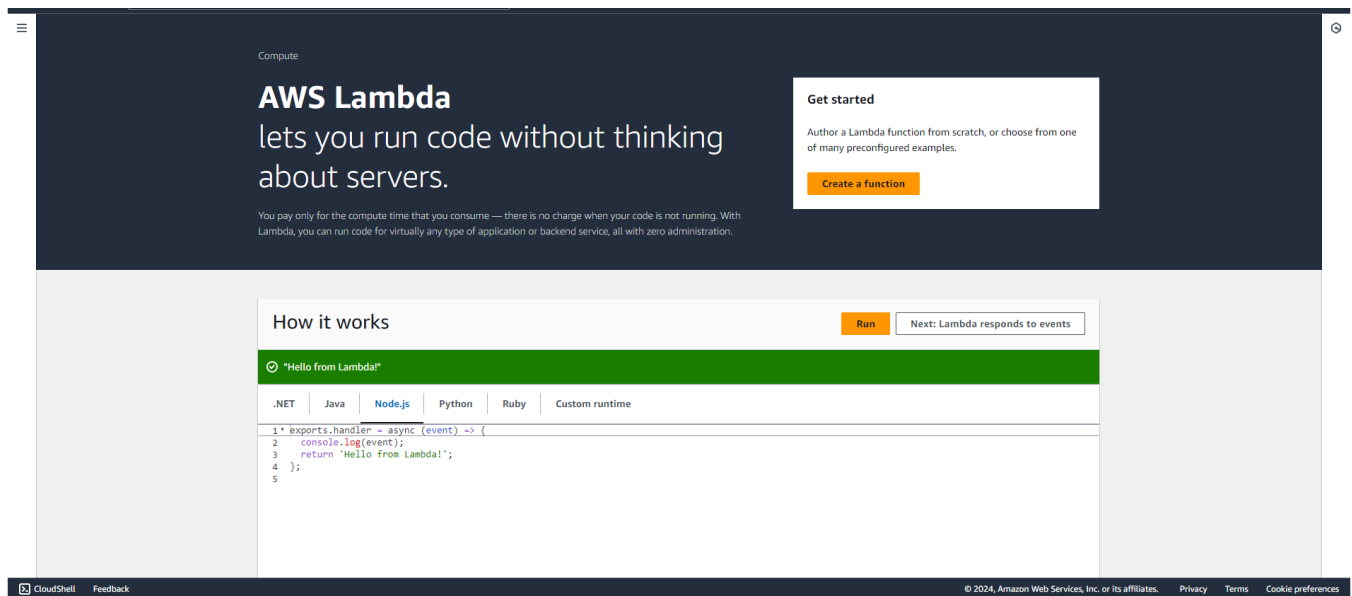
- **Create a Function:** Write the function code and define its handler (entry point). You can use the AWS Console, CLI, or upload a deployment package.
- **Set Event Sources:** Define how the function is triggered (e.g., when an object is uploaded to S3 or a DynamoDB table is updated).
- **Execution:** 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**

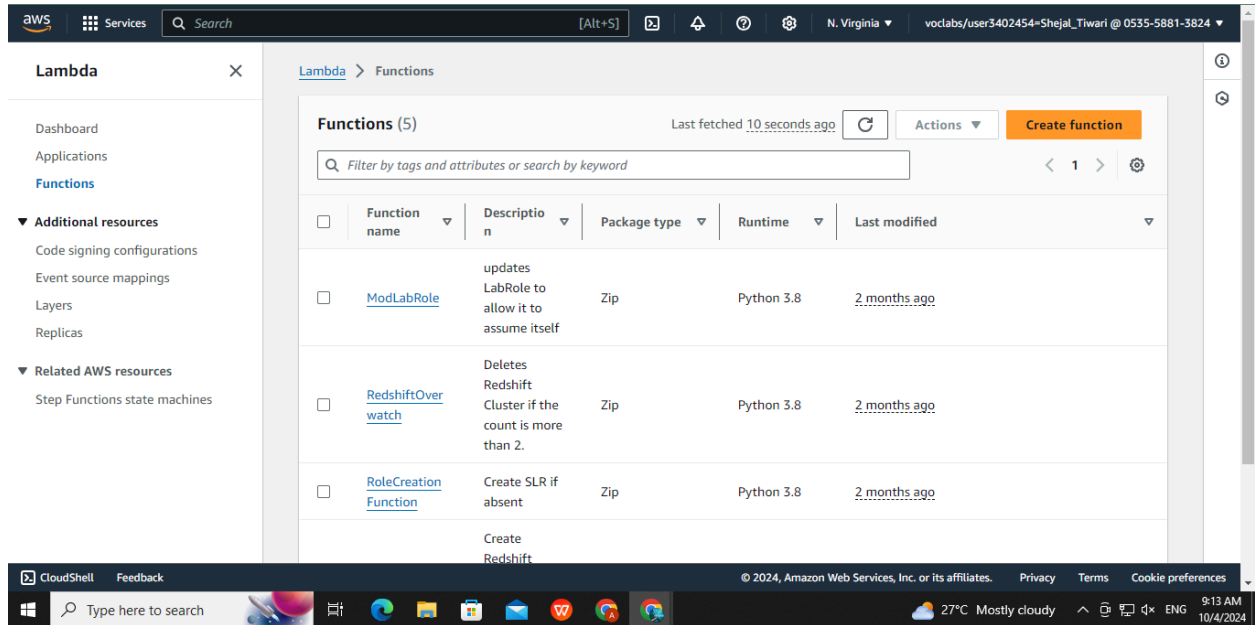
- **Python:** Great for quick development with its rich standard library and support for lightweight tasks.
- **Java:** Typically used for more complex, compute-intensive tasks. While it's robust, cold start times can be higher.
- **Node.js:** Excellent for I/O-bound tasks like handling APIs or streaming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

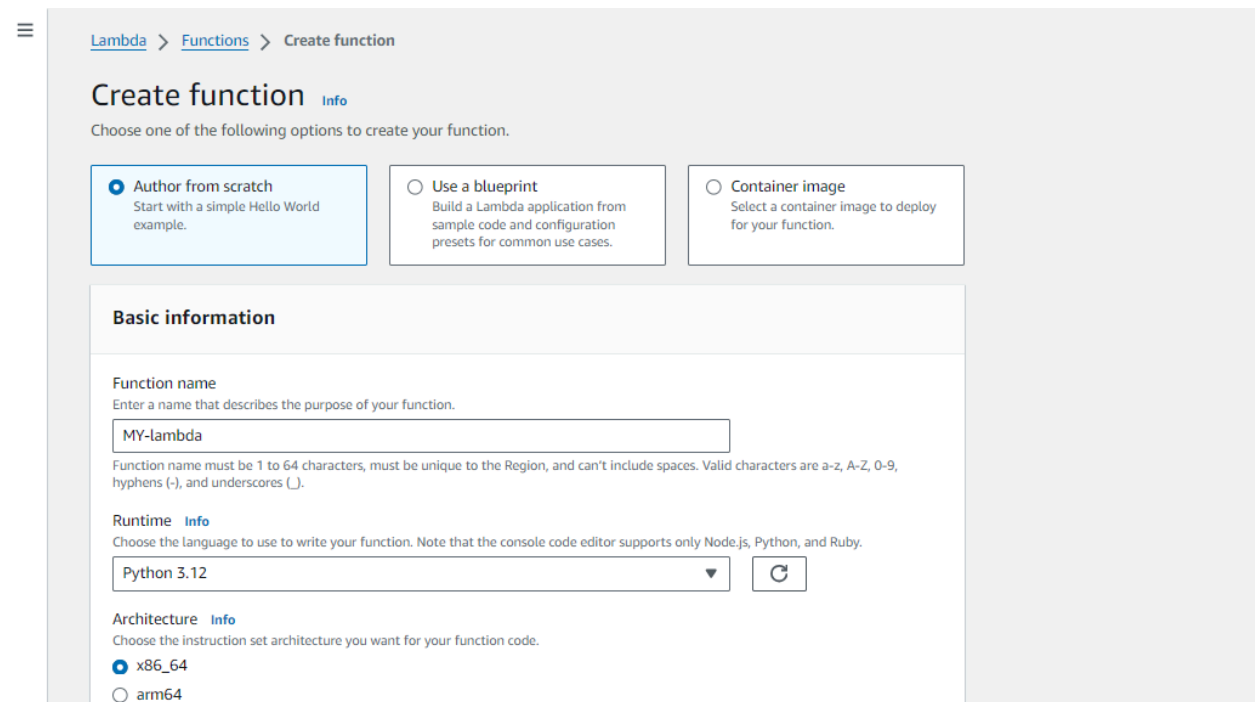


**Step 2:** Now Give a name to your Lambda function, Select the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby. So will select Python 3.12, Architecture as x86, and Execution role to Create a new role with basic Lambda permissions.



The screenshot shows the AWS Lambda console interface. The top navigation bar includes the AWS logo, a search bar, and the user's profile. 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 columns for Function name, Description, Package type, Runtime, and Last modified. Three functions are visible: ModLabRole, RedshiftOverwatch, and RoleCreationFunction. Each function has a checkbox to its left and a link to its configuration page.

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



The screenshot shows the "Create function" wizard in the AWS Lambda console. The wizard is divided into three main sections: "Author from scratch", "Use a blueprint", and "Container image". The "Author from scratch" section is selected. Below this, the "Basic information" section is visible, containing fields for "Function name", "Runtime", and "Architecture".

**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.

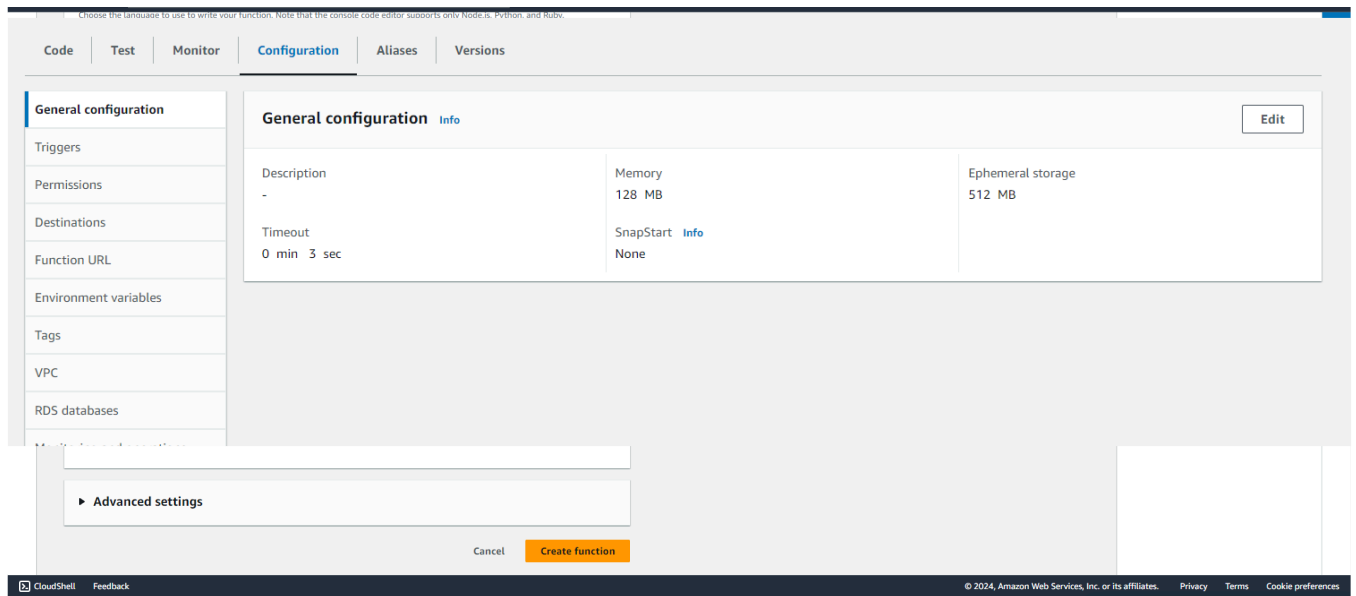
**Basic information**

**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.

**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.



Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

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

**General configuration** [Info](#) [Edit](#)

Description	Memory 128 MB	Ephemeral storage 512 MB
Timeout 0 min 3 sec	SnapStart <a href="#">Info</a> None	

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

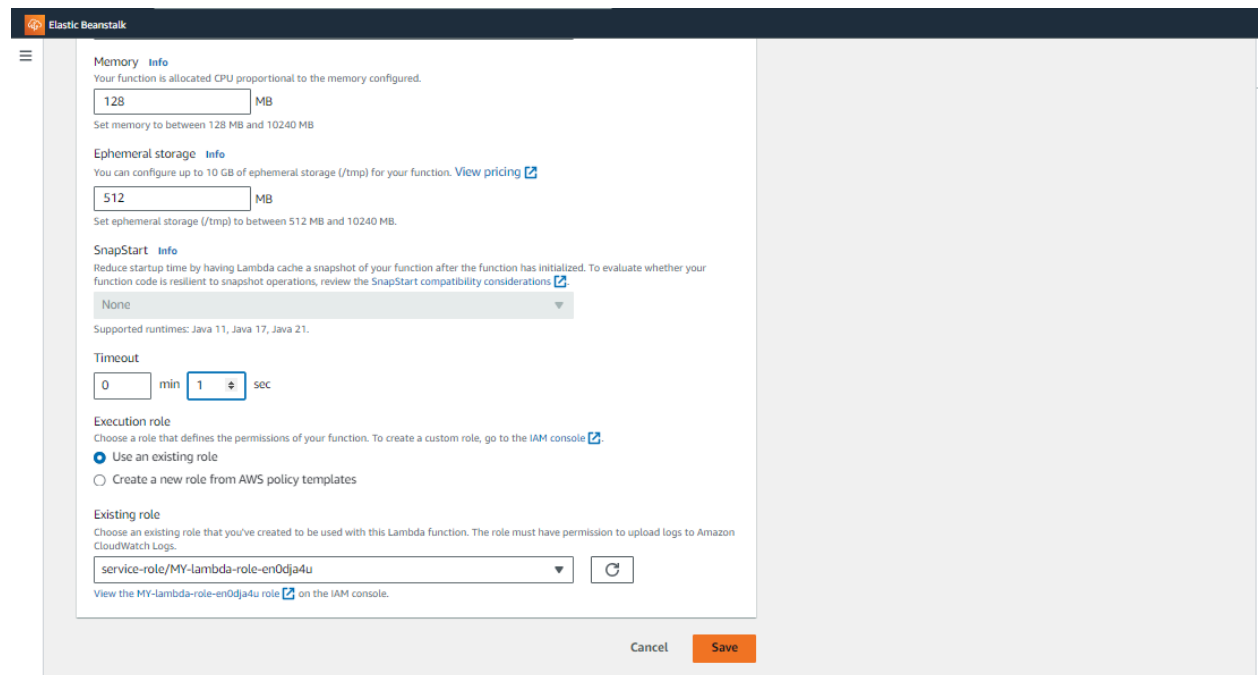
RDS databases

► Advanced settings

Cancel [Create function](#)

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

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.



**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

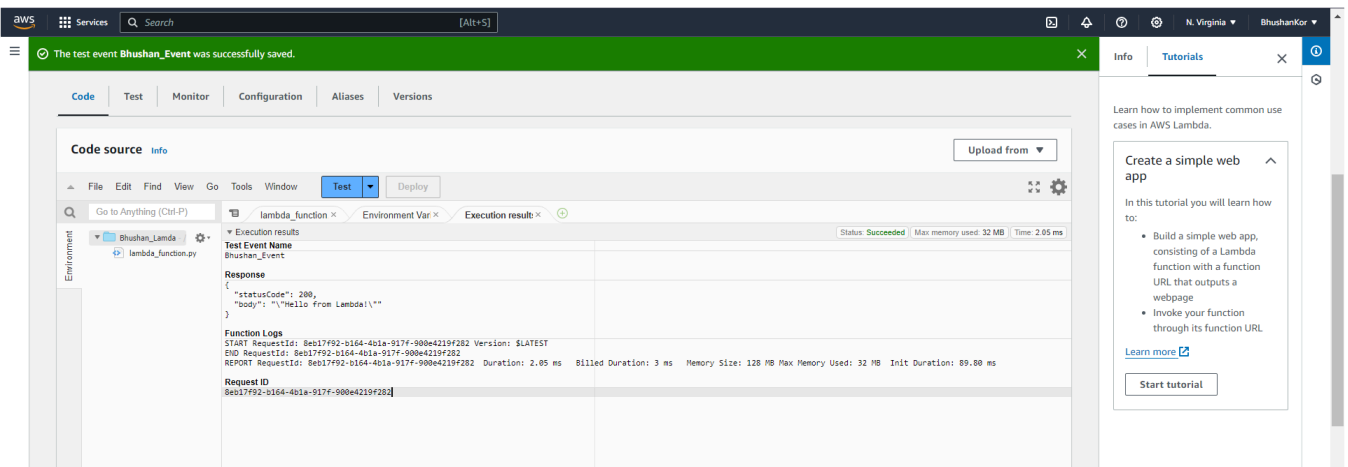
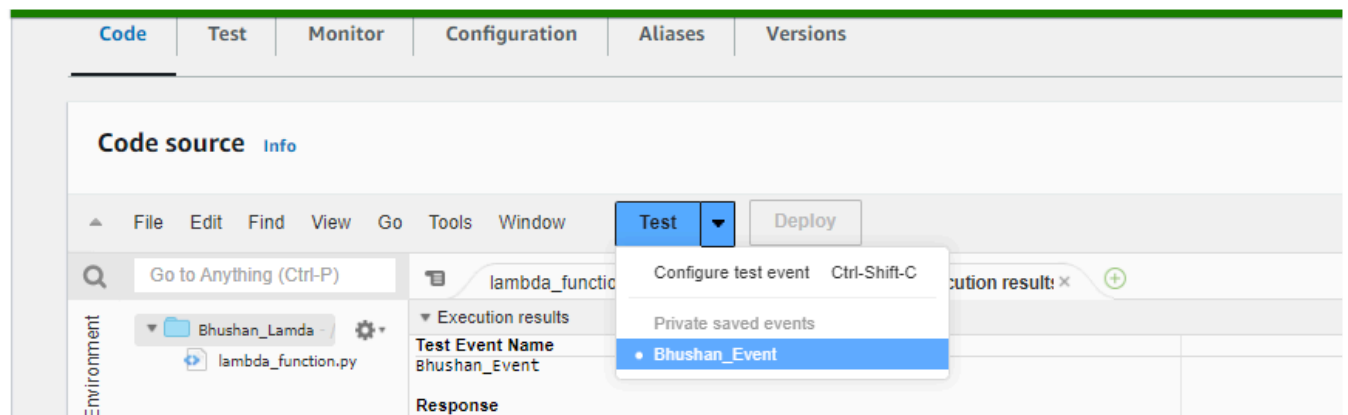
**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
☒ Use an existing role  
☐ Create a new role from AWS policy templates

**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.

Cancel [Save](#)

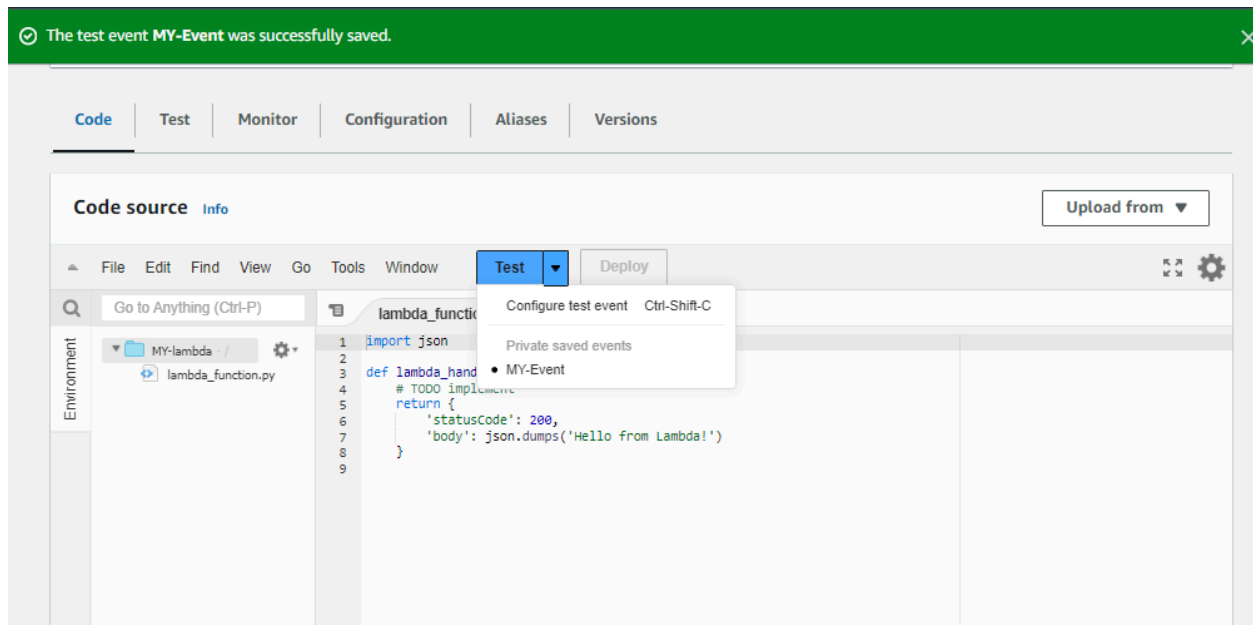
**Step 3:** Now Click on the Test tab then select Create a new event, give a name to the event and select Event Sharing to private, and select hello-world template.

**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.

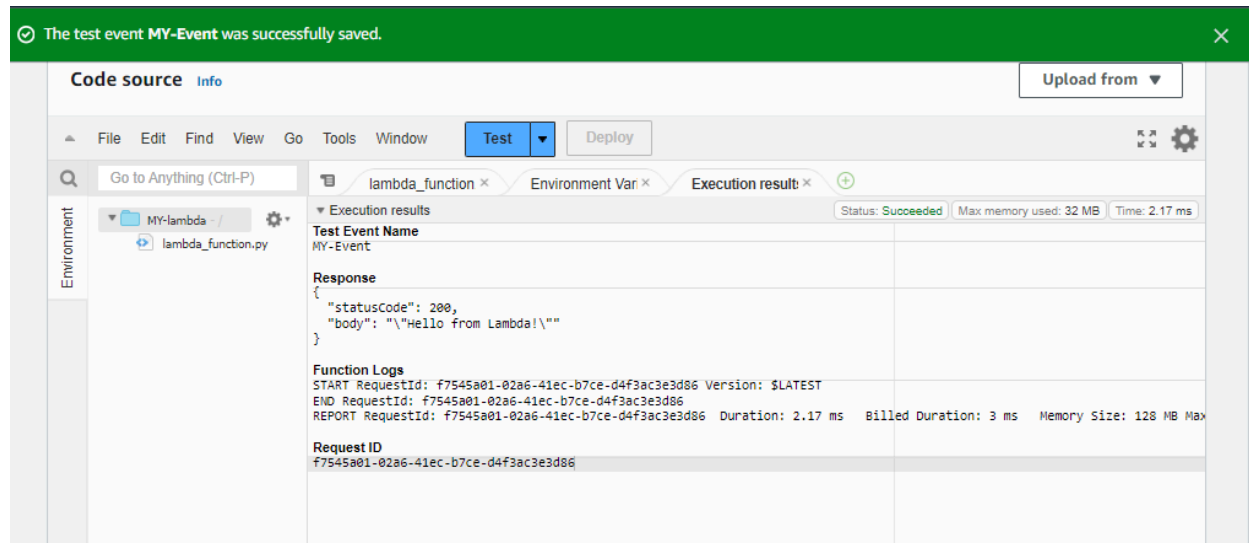


The screenshot shows the 'Test event' configuration page in the AWS Lambda console. At the top, a green banner indicates 'Successfully updated the function MY-lambda.' Below this, tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions' are visible. The 'Test' tab is active, showing a 'Test event' section with an 'Info' icon and 'Save' and 'Test' buttons. The instructions state: 'To invoke your function without saving an event, configure the JSON event, then choose Test.' Under 'Test event action', 'Create new event' is selected. The 'Event name' field contains 'MY-Event'. Under 'Event sharing settings', 'Private' is selected. A 'Template - optional' dropdown is set to 'hello-world'. At the bottom, there is an 'Event JSON' section with a 'Format JSON' button.

**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:

In this experiment, we successfully created and deployed our first AWS Lambda function using Python, gaining an understanding of its workflow and capabilities. The function was executed and tested, allowing us to observe the output and logs. While the overall process was smooth, there were several challenges that we encountered:

- **Role and Permissions Configuration:** Setting up the correct execution role with the necessary permissions was critical, as misconfigurations could prevent the function from running or accessing other AWS services. Debugging permission issues was time-consuming, especially when using new roles.
- **Timeout Issues:** Initially, the default timeout setting was insufficient for some operations. Adjusting the timeout to a value that suits the function's workload was necessary, especially for more complex operations beyond basic tasks.
- **Event Testing:** Configuring and testing events within Lambda required careful attention. Choosing the wrong template or incorrect event parameters often led to failures during the test phase, requiring adjustments to the event settings.