

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.

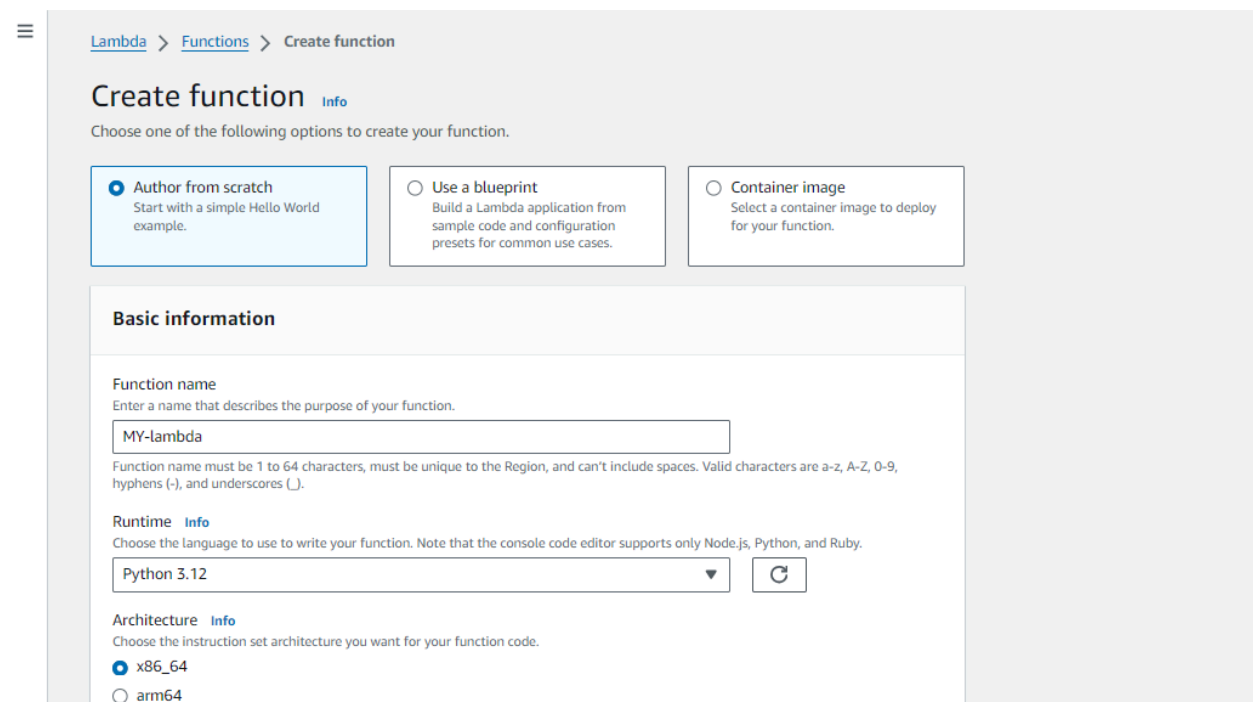
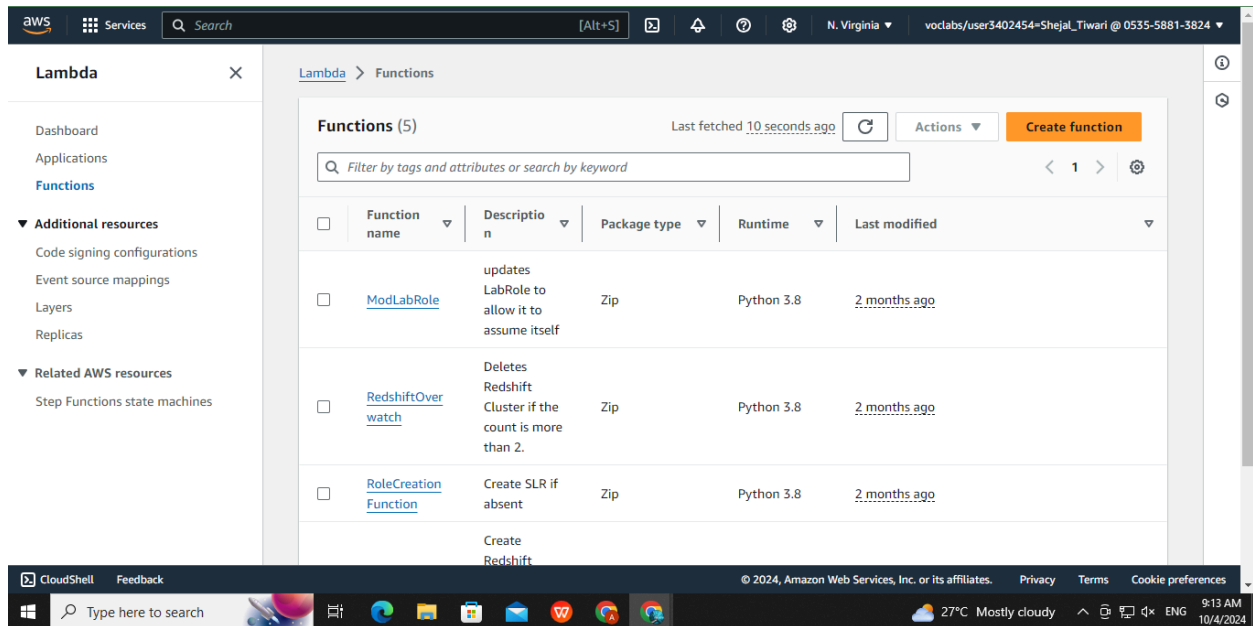
AWS Lambda Functions

- **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.



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

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 AWS Lambda console configuration page for a new function. The page is titled "Elastic Beanstalk" at the top. The configuration options are as follows:

- Memory:** 128 MB. Set memory to between 128 MB and 10240 MB.
- Ephemeral storage:** 512 MB. Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.
- SnapStart:** None. Supported runtimes: Java 11, Java 17, Java 21.
- Timeout:** 0 min, 1 sec.
- Execution role:** Use an existing role (selected). Create a new role from AWS policy templates (unselected).
- Existing role:** service-role/MY-lambda-role-en0dja4u. View the MY-lambda-role-en0dja4u role on the IAM console.

At the bottom right, there are "Cancel" and "Save" buttons. The "Save" button is highlighted in orange.

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.

Elastic Beanstalk

☰

☑ 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

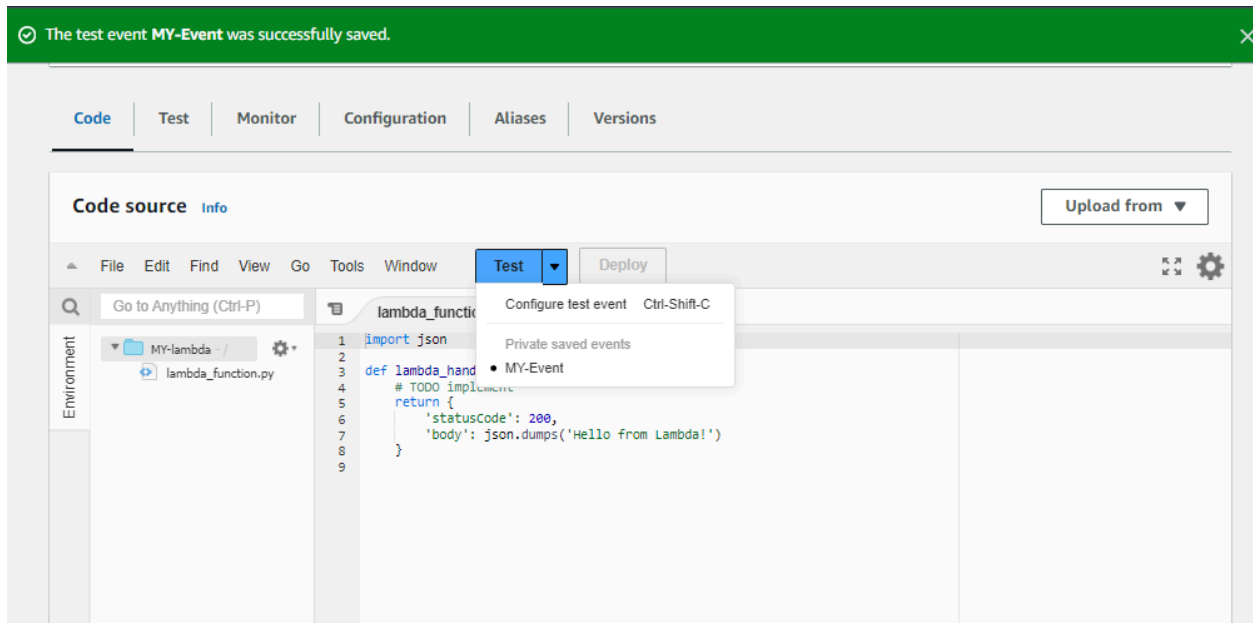
hello-world

Event JSON

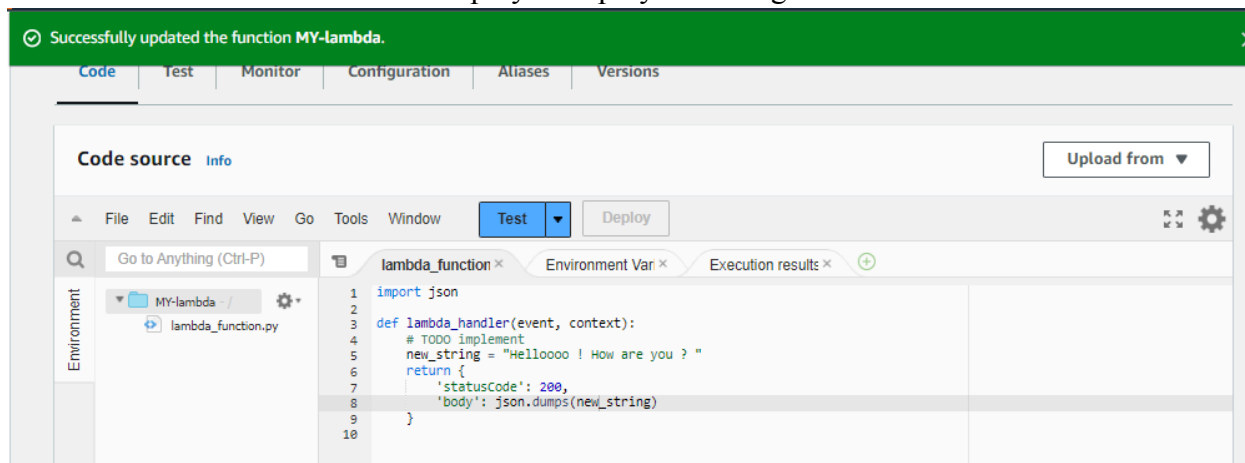
Format JSON

1

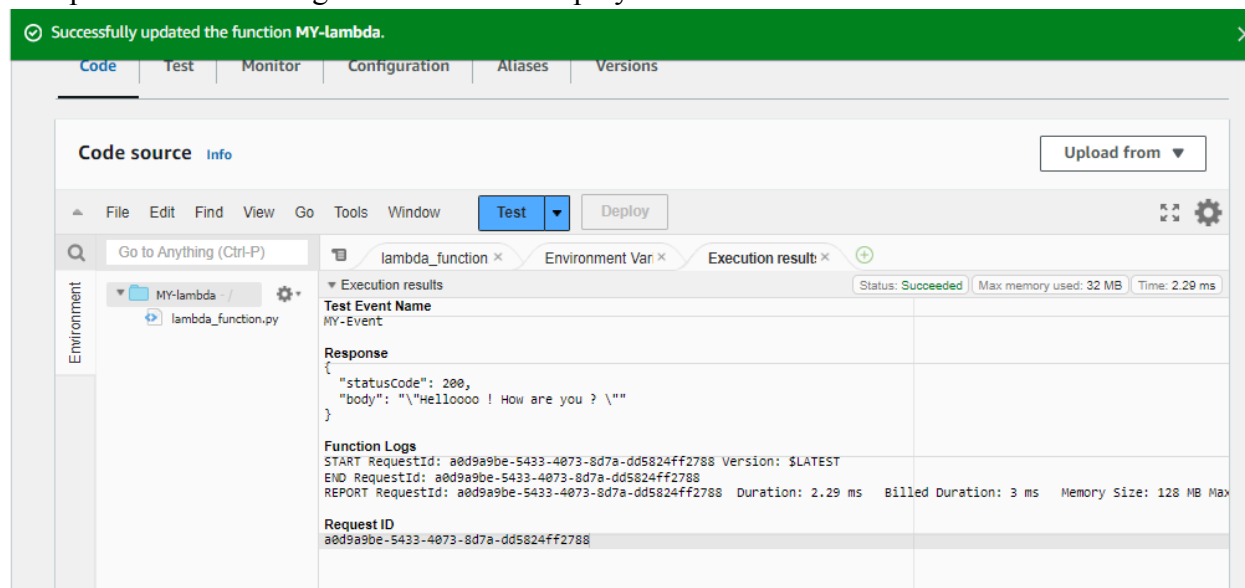
{} |



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.