😊 First Lambda Function

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS) that allows you to run your code in response to events without provisioning or managing servers. With AWS Lambda, you can write functions that automatically execute in response to various events, such as changes in data within Amazon S3, updates to DynamoDB tables, or HTTP requests via API Gateway.

Key Features:

1. **Event-driven**: Lambda functions can be triggered by events from other AWS services like S3, DynamoDB, SNS, and API Gateway. You can also invoke them directly via AWS SDKs or CLI.
2. **No Server Management**: Lambda takes care of server provisioning, scaling, and management. You only focus on writing your code.
3. **Automatic Scaling**: Lambda automatically scales depending on the number of events it receives. If your function gets a high volume of requests, Lambda will scale to handle them, and if the volume drops, it will scale down.
4. **Pay Only for Usage**: You are charged based on the number of requests to your Lambda functions and the duration it takes for your code to run. You don't pay for idle time.
5. **Support for Multiple Languages**: Lambda supports multiple programming languages including Node.js, Python, Java, Go, .NET, Ruby, and custom runtimes through AWS Lambda Layers.
6. **Integrated with Other AWS Services**: Lambda can be easily integrated with other AWS services such as S3, DynamoDB, API Gateway, CloudWatch, and more, making it ideal for building microservices and data processing workflows.

Benefits:

- **Cost-effective**: You only pay for the compute time used, not for idle resources.
- **Scalable**: Lambda automatically handles scaling without you needing to manually configure load balancing or provisioning servers.
- **Simple Deployment**: With Lambda, you can upload your code directly or via AWS CLI, AWS SDKs, or an integrated CI/CD pipeline, making deployment quick and easy.

Use Cases:

- **Real-time File Processing**: Trigger a Lambda function to process files in real time, for example, when a file is uploaded to an S3 bucket.
- **Backend for Web Applications**: You can use Lambda in conjunction with API Gateway to create a REST API for your web applications.
- **Data Processing**: Lambda can be used for processing data streams, such as those from Kinesis or DynamoDB streams.

- **Automation**: Automate tasks such as backups, sending notifications, or other administrative tasks.

How It Works:

1. **Create Lambda Function**: Write your function code and specify the runtime (e.g., Node.js, Python).
2. **Define Trigger**: Set up an event trigger, such as an S3 upload or an API Gateway HTTP request, that will invoke the function.
3. **Execution**: Once triggered, Lambda executes your function. The function code runs in a managed execution environment, with access to environment variables, AWS resources, and the internet.
4. **Response**: After execution, the function can return a result or perform an action, like writing data to DynamoDB, sending an email, or triggering another event.

Lambda Limits:

- **Timeout**: Each Lambda invocation has a maximum execution time of 15 minutes.
- **Memory Allocation**: You can allocate from 128 MB to 10 GB of memory to your Lambda function.
- **Payload Size**: The maximum event size that can be passed to Lambda is 6 MB for synchronous invocations and 256 KB for asynchronous invocations.

Conclusion:

AWS Lambda is a powerful service that simplifies the process of running code in the cloud, without worrying about server management. It's ideal for event-driven applications, real-time data processing, and cost-effective serverless architectures.

In this lab, you will learn how to create and test a basic AWS Lambda function. Here's a short summary of the steps:

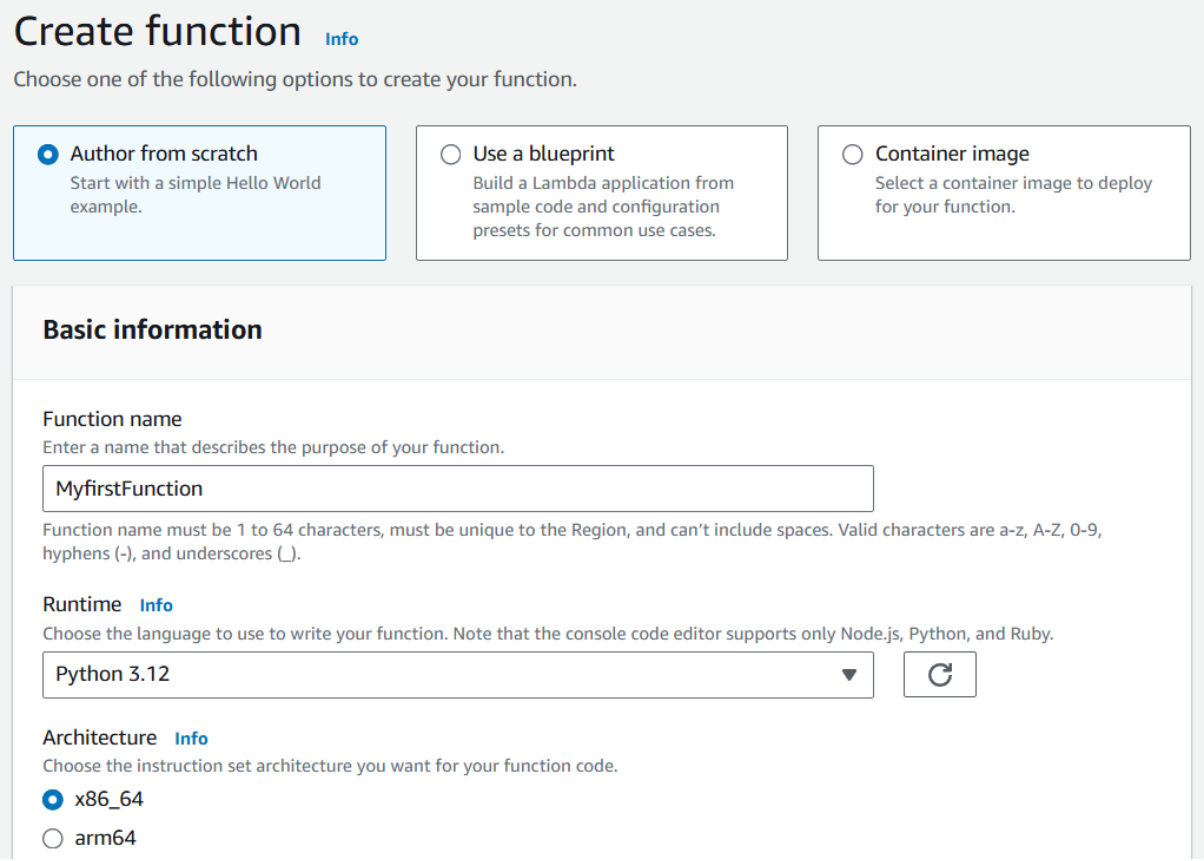
1. **Create Lambda Function**: You logged into AWS, created a Lambda function with Python as the runtime, and named it.
2. **Test the Function**: You wrote the initial code and tested it using an event trigger, which successfully executed.
3. **Monitor Logs**: You accessed CloudWatch to monitor the success of your Lambda function, identified errors in the logs when the code failed, and traced the issue to a syntax error in the code.
4. **Fix the Error**: After fixing the syntax error (removing extra dots), you redeployed the function and tested it again, resulting in the function working as expected.

End Goal:

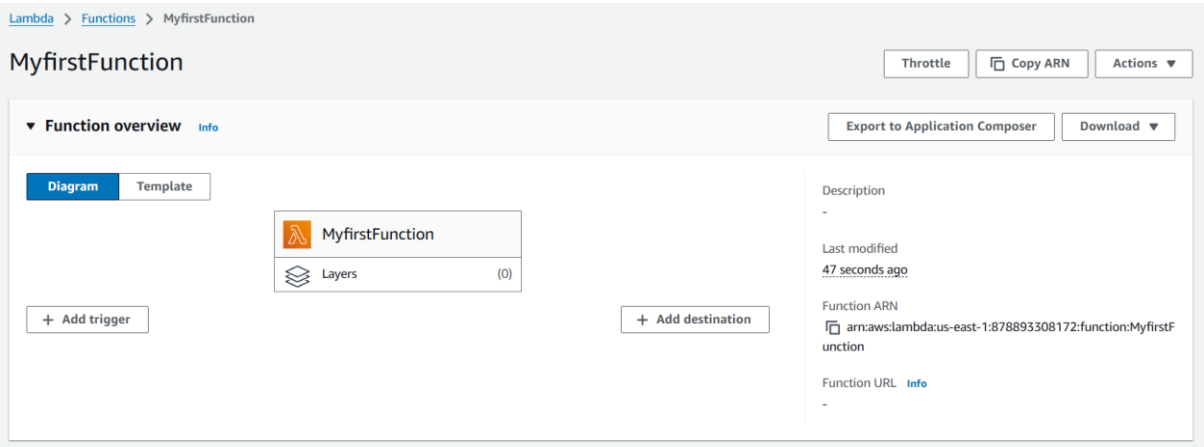
The end goal of this lab was to gain hands-on experience in creating a Lambda function, testing it, monitoring the function’s performance, identifying errors through CloudWatch logs, and fixing those errors to ensure successful execution.

😄 To begin with the Lab:

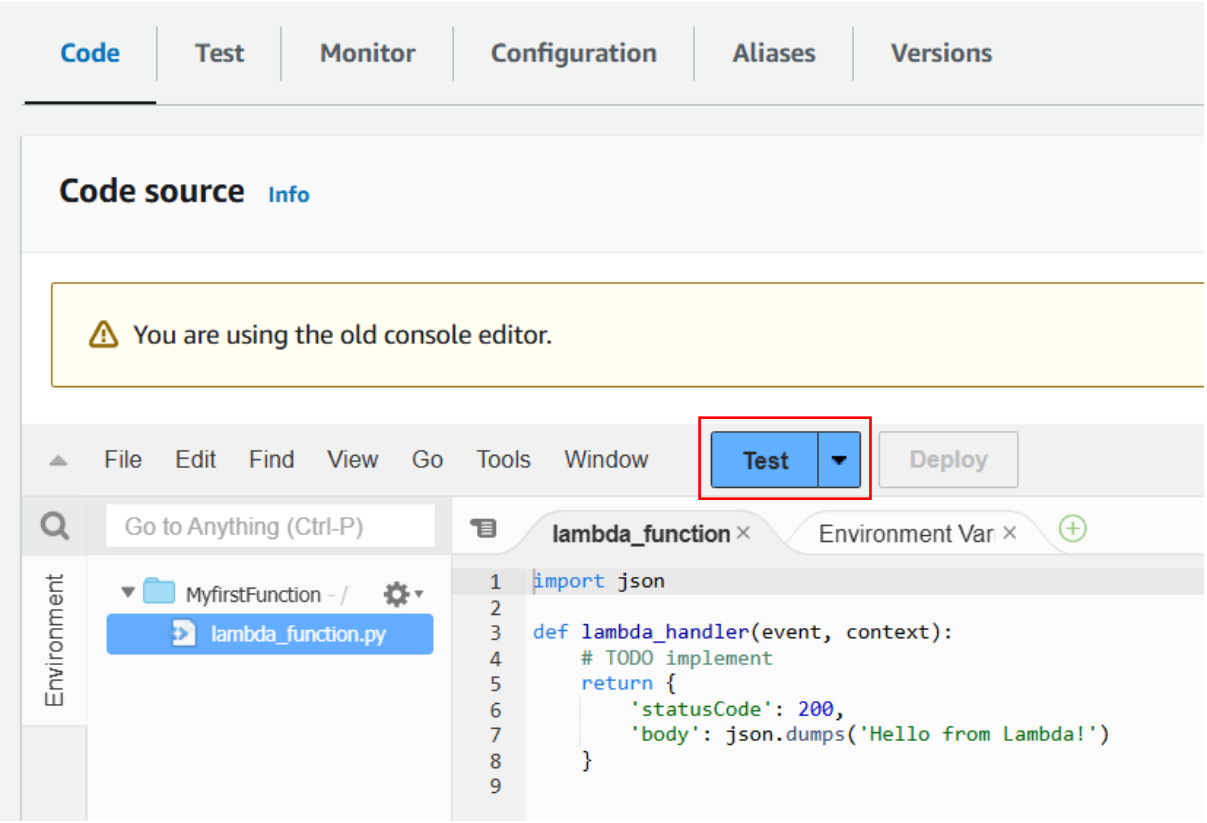
1. In this lab, we will create our first lambda function. To do so, log into your AWS Console, search for lambda, and navigate to it.
2. Then, you need to choose Author from scratch, give your function a name, and choose Python as the runtime. Click on the Create function button.



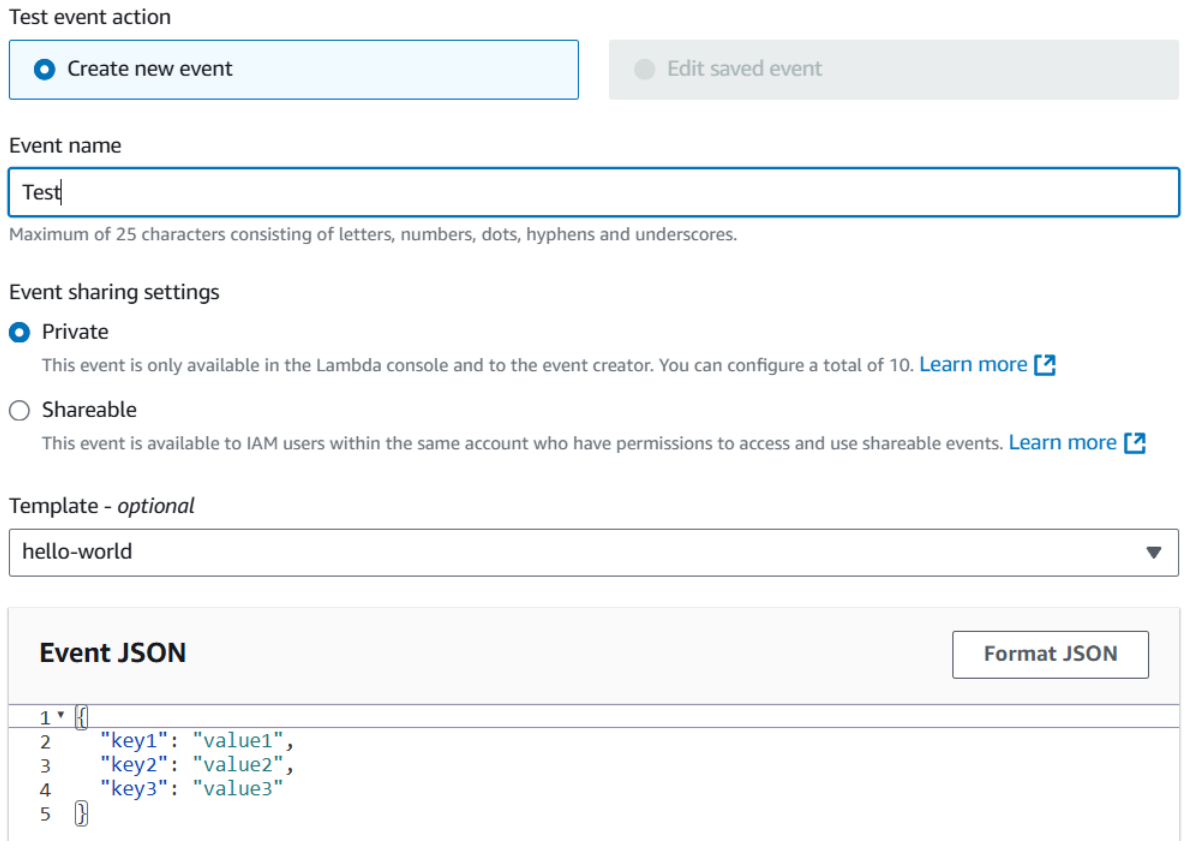
3. Here you can see that your function has been created and you are inside of your function.



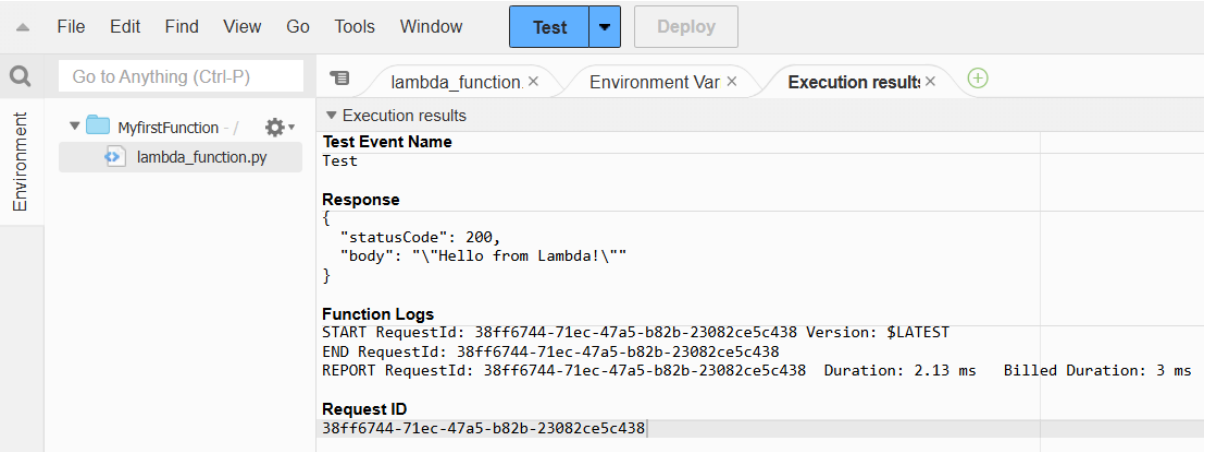
4. If you scroll down a little you will see that you have a code section. Read the code and click Test and this will create an event to test the code in your lambda function.



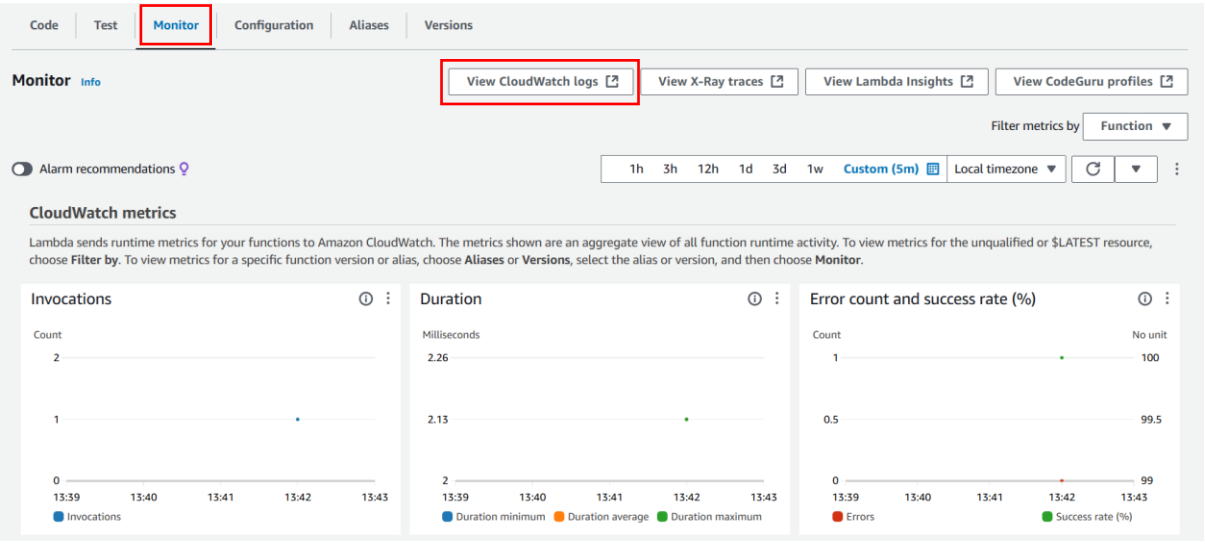
5. So, you need to give a name to your event and leave all the aspect as it is and click on save.



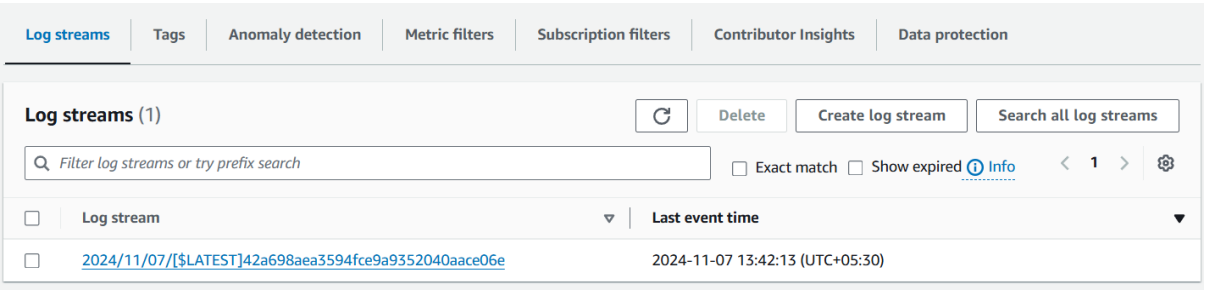
6. Once your test event is created then you need to click on Test. Here you can see that you get the response from lambda as expected.



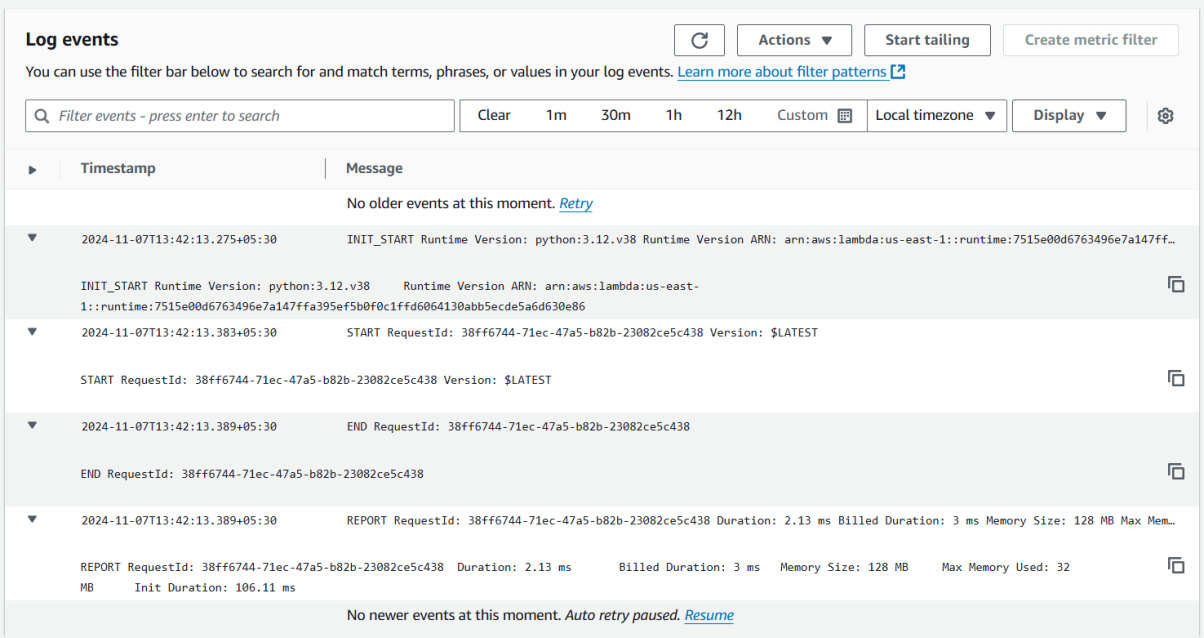
7. Then if you go to the monitor tab you will see that in the cloud watch metrics, you get the little dots in green color which means that your function was successful while testing.
8. Also, if you click on view Cloud Watch logs, you will be redirected to a new page for Cloud Watch.



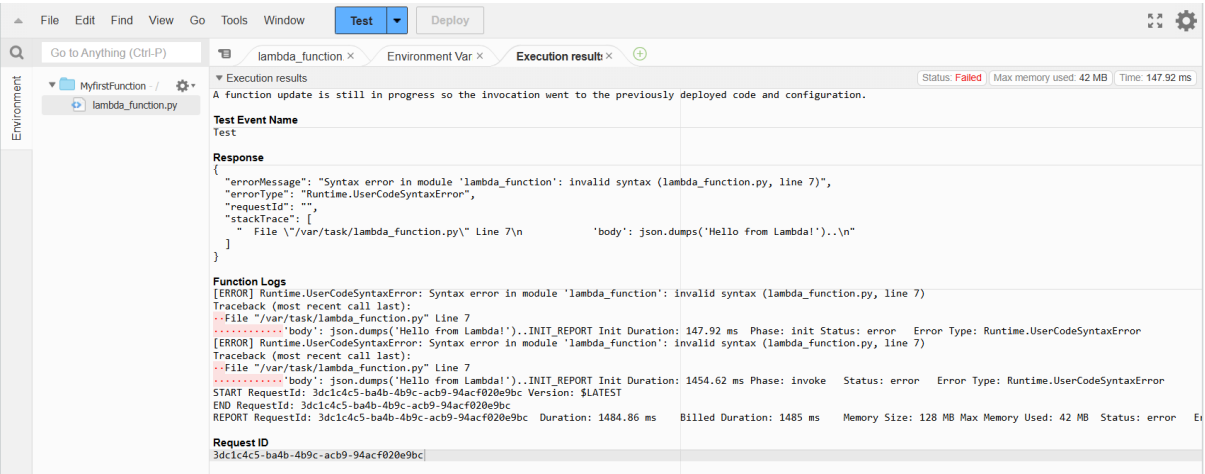
9. Here in the log groups, you need to click on your log stream.



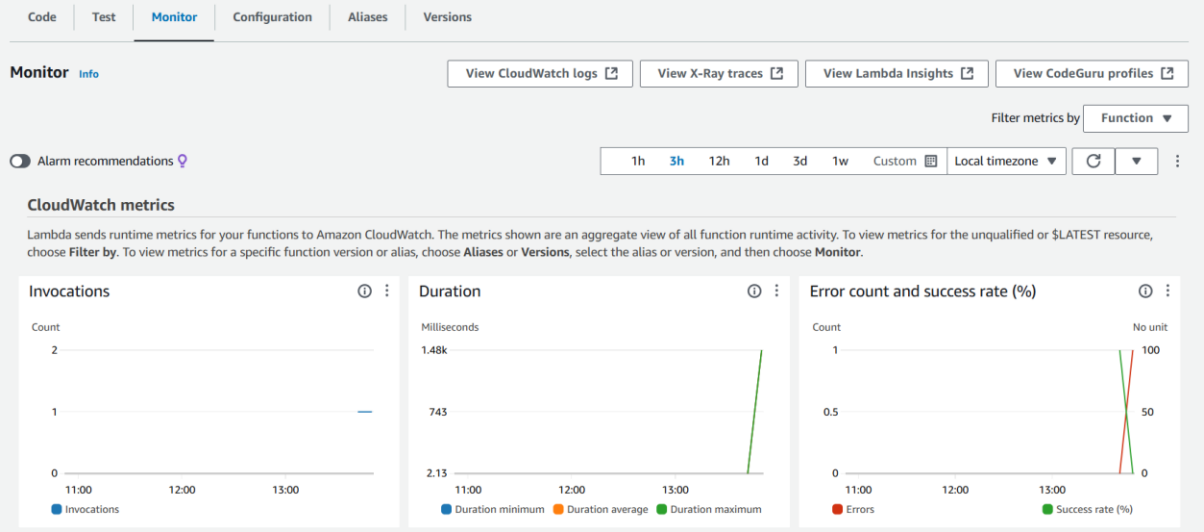
10. In the log events you will be able to track the process for your lambda function.



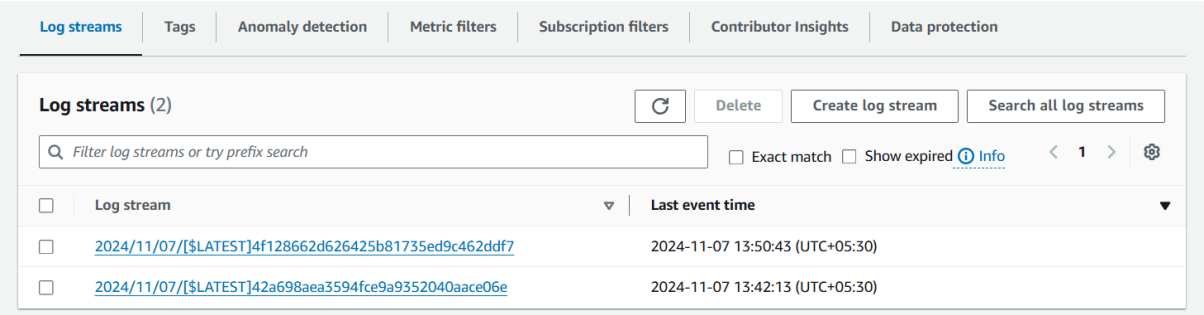
11. Now we made a syntactical error in our code then we deployed it and then we tested it here you can see that we got the error in our response.

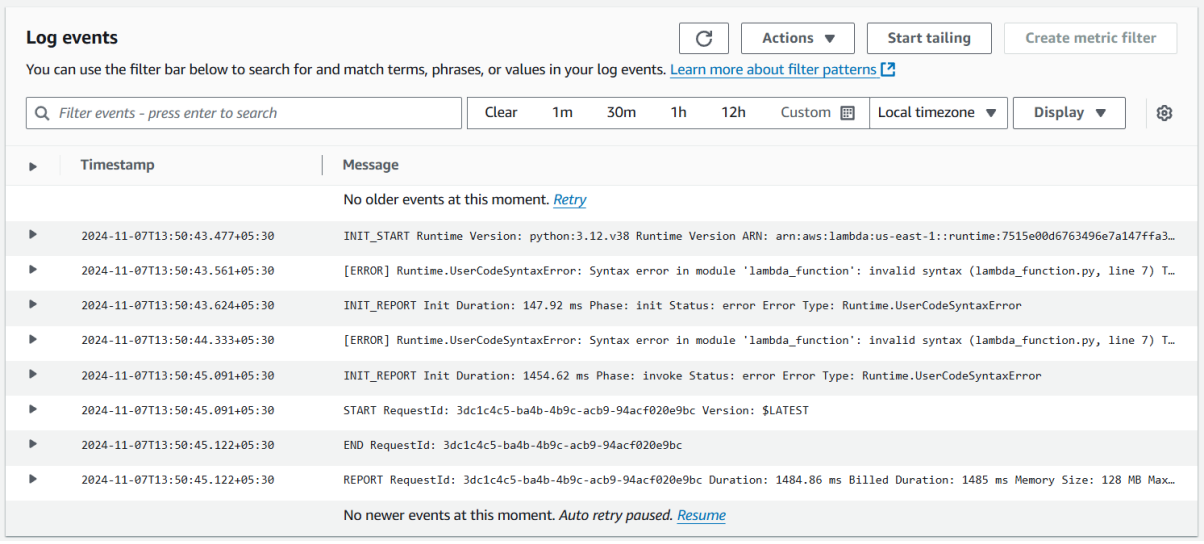


12. If we go to the monitor tab, we can see that in the error count and success rate graph area we have a red line which means that there is an error in the code of our lambda function.

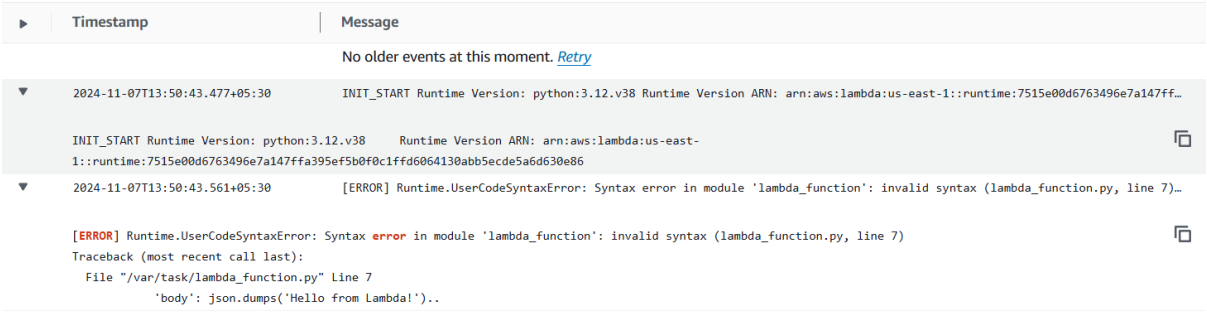


13. Also, if we go to the cloud watch logs and refresh the page for our log stream, we will see that we have a new log stream, and open it. Below you can see that we have the error log events which we can read and understand what the error is.





14. From the log events we can say that the error is in line number 7 some extra dots are present here.
15. We can clear them out and re-deploy the code. Then click on Test again.



16. And you will get the response as expected.