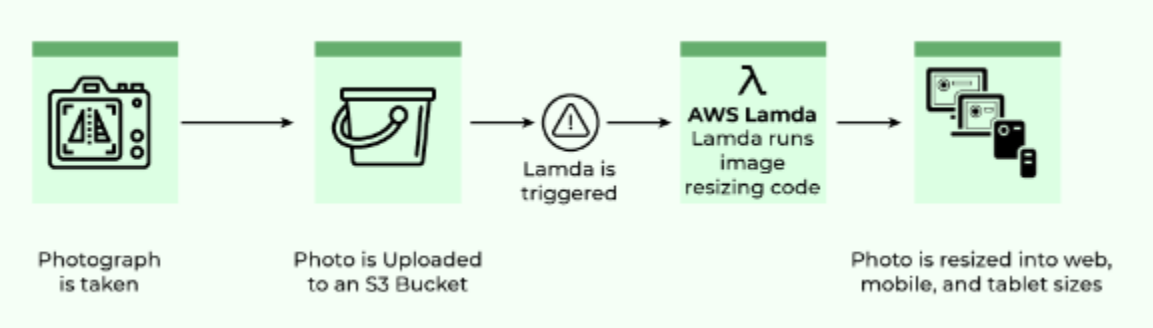
Contents

INTRODUCTION

AWS Lambda is an Amazon serverless computing system that runs code and automatically manages the underlying computing resources. It is an event-driven computing service. It lets a person automatically run code in response to many types of events, such as HTTP requests from the Amazon API gateway, table updates in Amazon DynamoDB, and state transitions. It also enables the person to extend to other AWS services with custom logic and even creates its own back-end services. For example, just write the code and then upload it as a .zip file or any container image. The service works by running code on high-availability computer infrastructure. It then performs all the administrative duties of that compute resource such as: providing maintenance on the server and operating system, auto-scaling and managing the person's capacity provisions, handling security patch deployment lambda extensions code, monitoring, logging, and concurrency function blueprints.

SAMPLE WORKFLOW DIAGRAM:



HISTORY OF AWS LAMBDA

AWS Lambda, introduced by Amazon Web Services (AWS) in November 2014, revolutionized the way developers build and deploy serverless applications. Serverless computing, as the name suggests, abstracts away the need to manage server infrastructure, allowing developers to focus solely on writing code to implement specific functions or tasks. Here's a brief history of AWS Lambda:

1. Launch and Early Adoption (2014-2016):

- AWS Lambda was officially announced at AWS in November 2014. It was one of the first widely adopted serverless computing platforms.

- Initially, it supported execution of code written in Node.js, followed by Java and Python.

- Developers quickly embraced Lambda for building event-driven and microservices-based applications.

2. Language Expansions and Features (2016-2017):

- AWS expanded the programming language support of Lambda to include .NET Core, Go, and custom runtime APIs. This allowed developers to use additional languages and runtimes.

- Features like VPC (Virtual Private Cloud) support were added, enabling Lambda functions to access resources in a VPC.

- The introduction of AWS API Gateway made it easier to create APIs that trigger Lambda functions, allowing for more versatile applications.

3. Wider Adoption and Service Integration (2018-2019):

- Adoption of AWS Lambda continued to grow as it became a crucial part of the serverless ecosystem.

- AWS introduced features like Lambda Layers, which enabled sharing of code and dependencies across functions.

- Integration with Amazon DynamoDB Streams, Amazon Kinesis, and other AWS services allowed for seamless event-driven architectures.

4. Enhancements and Event Sources (2020-Present):

- AWS announced support for custom container images as Lambda runtimes. This allowed developers to use any programming language or dependencies.

- Event Bridge (formerly known as CloudWatch Events) integration improved event-driven workflows, making it easier to build complex serverless applications.

- The introduction of Provisioned Concurrency improved the performance and latency of Lambda functions by keeping a set number of instances warm and ready to execute.

5. Serverless Application Model (SAM) and Beyond:

- AWS introduced the Serverless Application Model (SAM), a framework for building and deploying serverless applications, simplifying the development process.

- The serverless landscape continued to evolve with more advanced observability, monitoring, and debugging tools.

COMPUTE SERVICES IN AWS

- Amazon EC2
- Amazon ECS
- Amazon EKS
- **AWS Lambda**
- AWS Lightsail
- AWS Batch
- AWS Fargate
- Amazon SageMaker
- AWS Ouposts
- AWS Elastic Beanstalk

AWS SERVERLESS SERVICES

Amazon Web Services (AWS) offers a variety of serverless services and tools that enable developers to build and deploy applications without the need to manage servers or infrastructure.

These serverless offerings include:

AWS Lambda:

AWS Lambda is a compute service that allows you to run code in response to events or triggers. It's the cornerstone of serverless computing on AWS. You can write functions in various programming languages and have them automatically scaled in response to events from AWS services, APIs, or custom events.

Amazon API Gateway:

This service enables you to create, publish, and manage RESTful APIs for your serverless applications. You can connect AWS Lambda functions to API Gateway endpoints to build serverless API backends.

Amazon DynamoDB:

DynamoDB is a fully managed NoSQL database service. It can be used as a serverless data store for your applications, providing automatic scaling and high availability without the need for server management.

Amazon S3 (Simple Storage Service):

S3 is an object storage service that can be used to store and serve large amounts of data in a serverless manner. It's often used for static website hosting, data storage, and content distribution.

Amazon SNS (Simple Notification Service) and Amazon SQS (Simple Queue Service):

These services provide messaging and event-driven communication capabilities for your serverless applications. They allow you to decouple components and scale independently.

Amazon Event Bridge:

This service enables event-driven architectures by connecting various AWS services and custom applications through events. It can route events to AWS Lambda functions, among other targets.

AWS Step Functions:

Step Functions allow you to build serverless workflows by coordinating the execution of AWS services, including Lambda functions, in a visual, state-machine-based approach.

Amazon Cognito:

Cognito provides authentication, authorization, and user management for serverless applications. It makes it easier to secure your APIs and applications.

AWS App Runner:

While not strictly serverless, App Runner simplifies the process of deploying and scaling containerized applications. It abstracts the underlying infrastructure, making it easier to deploy microservices and APIs.

AWS Glue:

AWS Glue is a fully managed ETL (Extract, Transform, Load) service. It's often used in serverless data processing workflows for tasks like data transformation and integration.

AWS Amplify:

Amplify is a set of tools and services that simplifies the development of web and mobile applications. It includes features for authentication, API management, and frontend development in a serverless context.

AWS Fargate:

While not strictly serverless, AWS Fargate is a serverless compute engine for containers. It allows you to run containers without having to manage the underlying infrastructure, like serverless functions.

WHAT IS LAMBDA?

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). It allows you to run code in response to various events and triggers without having to manage the underlying infrastructure. Lambda is a key component of serverless computing, which abstracts away the traditional server management tasks, such as provisioning, scaling, and patching, making it easier for developers to focus on writing and deploying code.

AWS Lambda is commonly used for a wide range of use cases, including real-time data processing, image and video manipulation, web application backends, IoT device communication, and more. It is a fundamental building block of serverless and event-driven architectures within AWS, enabling developers to create scalable and cost-effective applications without the burden of server management.

WHY DO WE USE LAMBDA?

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). AWS Lambda is distinct from other AWS services in that it offers serverless, event-driven, pay-per-use compute capabilities with automatic scaling and minimal operational overhead.

It is different from other AWS services in several keyways:

Serverless Computing:

Lambda is a serverless service, which means you don't need to provision or manage servers. With other services like EC2 (Elastic Compute Cloud), you must provision and manage virtual machines, but Lambda abstracts this underlying infrastructure away.

Event-Driven:

Lambda is designed to be event driven. You can trigger Lambda functions in response to events from various AWS services (e.g., S3 object uploads, API Gateway requests, DynamoDB changes) or even from custom events using the AWS SDK. Other services may require you to set up polling or use more traditional methods to react to events.

Pay-Per-Use Billing:

Lambda charges you based on the number of requests and the time your code executes. You don't pay for idle resources, which is different from EC2 instances, where you pay for the entire uptime of the instance, regardless of whether it's actively processing requests.

Scaling:

Lambda scales automatically. When an event occurs, AWS provisions the necessary resources to run your function, and if there are many events, it can scale

out to run multiple instances of your function in parallel. With other services, you often need to configure and manage scaling settings manually.

Limited Execution Environment:

Lambda has some limitations regarding the runtime and execution environment. For example, it imposes a maximum runtime of 15 minutes per function execution and enforces resource limits such as memory allocation. In contrast, services like EC2 allow more control over the runtime environment and have no inherent execution time limits.

Stateless:

Lambda functions are stateless, meaning they don't retain data between invocations. Each invocation of a Lambda function is independent of previous invocations. Other services, such as databases, may maintain state and have persistence mechanisms.

Use Cases:

Lambda is particularly well-suited for event-driven and short-lived compute tasks, like processing files when they're uploaded to an S3 bucket or responding to API requests. Other AWS services like EC2 or ECS (Elastic Container Service) are better suited for long running or stateful applications.

Operational Overhead:

With Lambda, AWS manages most operational tasks, such as patching the underlying infrastructure, monitoring, and scaling. In contrast, services like EC2 require more manual management and maintenance.

Security:

You can configure Lambda functions to run within a Virtual Private Cloud (VPC) for network isolation and control access to resources us.

WHEN TO USE AWS LAMBDA

AWS lambda is very helpful when you know how to write code, but you don’t know how to provision the underlying infrastructure in AWS. AWS lambda will scale up the applications rapidly when there is sudden incoming traffic and scale down to zero when the incoming traffic is reduced.

Let’s you run code without provisioning or managing servers. You pay only for the compute time you consume—there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service—all with zero administration. Just upload your code, and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services, or you can call it directly from any web or mobile app.

WHERE TO USE AWS LAMBDA?

1. **Complex Logic:** Lambda functions are best suited for short, simple operations. If your function requires complex logic or multiple lines of code, using a lambda can make the code less readable and harder to maintain. In such cases, it's better to use a named function with a clear, descriptive name.

2. Lack of Descriptive Names: Lambda functions have no inherent name, so they can make code less self-explanatory if not used judiciously. If the purpose of the function is not immediately obvious from the lambda expression, it's better to use a named function with a descriptive name.

3. Reuse: If you need to use the same function in multiple places within your code or across different parts of your application, it's more appropriate to define a named function that can be easily reused. Repeated use of lambda functions can lead to duplicated code and decreased maintainability.

4. Debugging: Debugging lambda functions can be more challenging than debugging named functions because they don't have clear names or stack traces.

When debugging becomes necessary, it's often easier to work with named functions that can be identified in error messages and debug output.

5. Team Consistency: In team-based development, consistency in coding style and practices is essential. If your team has established guidelines or coding standards that discourage or limit the use of lambda functions, it's important to follow those guidelines to ensure uniformity in the codebase.

6. Complex Function Signatures: When working with functions that require a significant number of parameters or complex type annotations, lambda functions can make code harder to read due to their concise syntax. In such cases, using a named function with explicit parameter names and types is preferable.

7. Long Expressions: If your lambda expression becomes excessively long, it can negatively impact code readability. In such situations, it's better to refactor the logic into a named function with clear documentation.

8. Immutability and Functional Purity: In functional programming, adhering to principles of immutability and functional purity is important. Lambda functions can sometimes make it harder to ensure these principles are maintained, especially if they involve modifying external variables.

9. Incompatible Syntax: Not all programming languages support lambda expressions, so if you're working in an environment where lambda functions are not available or are not idiomatic, it's better to use the appropriate language constructs.

KEY FEATURES OF AWS LAMBDA

1. **Autoscaling and High Availability:** AWS lambda will make sure that your application was highly available to the end users when there is sudden incoming traffic. High availability can be achieved by scaling the application.

2. **Serverless Execution:** There is no need for provisioning the servers manually in AWS. AWS lambda will provision the underlying infrastructure based on the triggers you are mentioned whenever a new file uploaded to a

particular then AWS lambda will automatically trigger and takes care of the infrastructure.

3. **Pay-per-use-pricing:** AWS will charge you only for the time that time compute engine was active. AWS bills you based on the time taken to execute the code.

4. **Supports different programming languages:** AWS lambda function will support different programming languages. You can build the function with the language at your convenience. Following are some languages supported by AWS lambda:

1. Python.
2. Node.js.
3. Java.
4. C#.
5. PowerShell.
6. Go.

5. **Integrates with other AWS Services:** AWS lambda can be integrated with different AWS services like the following:

1. API Gateway
2. DynamoDB
3. S3
4. Step Functions
5. SNS
6. SQS.

6. **Versioning and Deployment:** AWS lambda function will maintain the different kinds of versions of the code by which you can change between the versions without any disruptions y based on the application performances.

7. **Security and Identity Management:** AWS lambda will leverage AWS Identity and Access Management (IAM) to control the access to the functions which are built by using lambda You can define fine-grained permissions and policies to secure your functions and ensure that only authorized entities can invoke them.

USE CASES OF AWS LAMBDA

1. **Event-Driven Processing:** Lambda is often used to respond to events from various AWS services, like S3 (object creation or deletion), DynamoDB (data changes), SNS (notifications), and more. For example, you could trigger a Lambda function to process images when they are uploaded to an S3 bucket.

2. **Real-time File Processing:** Lambda can be used to process files in real-time as they are uploaded to S3, such as converting file formats, extracting metadata, or generating thumbnails.

3. **REST API Endpoints:** You can create REST API endpoints using Amazon API Gateway and use Lambda to execute the backend logic for your API. This is a cost-effective way to build APIs without managing servers.

4. **Data Transformation:** Lambda can be used to transform data between different formats or structures. For example, you could use Lambda to transform JSON data into CSV format.

5. **Data Validation and Cleaning:** Lambda functions can be used to validate and clean incoming data. This is useful when you want to ensure that the data adheres to specific business rules before storing or processing it.

6. **Scheduled Tasks:** Lambda functions can be scheduled to run at specific intervals using CloudWatch Events. This is useful for tasks like periodic data backups, log rotation, or sending regular reports.

7. **Machine Learning Inference:** You can deploy machine learning models using Lambda to perform real-time inference on incoming data. This is particularly useful when you need to make predictions based on new data without maintaining a dedicated server.

8. **Image and Video Processing:** Lambda can be used to process images or videos, such as resizing images, generating thumbnails, or extracting frames from videos.

9. **IoT Data Processing:** Lambda functions can process data from IoT devices, allowing you to perform actions based on sensor data or control connected devices.

10. **User Authentication and Authorization:** Lambda functions can be used to handle user authentication and authorization for applications, interacting with authentication services and databases.

11. **Data Analysis and Aggregation:** Lambda can aggregate and analyze data from different sources, helping you gain insights from disparate data sets.

12. **Chatbots and Voice Assistants:** Lambda can be used to build chatbots or voice assistants that process user input and provide appropriate responses.

13. **Serverless Backend Logic:** Instead of running servers for backend logic, Lambda can be used to handle various tasks like user registration, order processing, and more.

14. **Webhooks and Integration:** You can use Lambda to integrate different services using webhooks, processing data between services as needed.

15. **Custom Application Logic:** Lambda allows you to create custom logic for your applications without managing the underlying infrastructure.

PRICING IN AWS LAMBDA

AWS Lambda pricing depends on seven major factors.

1. AWS Lambda charges you for the serverless compute time you use. The total time it takes your code to execute and the number of requests for your functions. There are no charges when your code is not running.

2. There is no charge for the first 1 million requests and the first 400,000 GB-seconds of compute time in a month.

3. The free allowance applies to functions running on both Graviton2 (Arm-based) and X86 CPUs.

4. After that, you pay $0.20 per 1 million requests.

5. Your Lambda cost will vary depending on the amount of memory (RAM) you allocate to your function.

6. Ephemeral storage pricing starts at $0.0000000309 per GB-second.

7. The following formula will help you calculate your AWS Lambda costs in GB-seconds: Number of executions X (Lambda memory size/1024) X (Lambda duration in milliseconds/1000) = GB-seconds

ADVANTAGES OF AWS LAMBDA

1. **Zero Server Management:** Since AWS Lambda automatically runs the user's code, there's no need for the user to manage the server. Simply write the code and upload it to Lambda.

2. **Scalability:** AWS Lambda runs code in response to each trigger, so the user's application is automatically scaled. The code also runs in parallel processes, each triggered individually, so scaling is done precisely with the size of the workload.

3. **Event-Driven Architecture:** AWS Lambda function can be triggered based on the events mapping in the AWS other service like when the file or video is added to the S3 bucket you can trigger the AWS Lambda function.

4. **Automatic High Availability:** When there is high demand or high incoming traffic AWS lambda function will automatically scale the servers.

5. **Affordable:** With AWS Lambda, one doesn't pay anything when the code isn't running. The user must only be charged for every 100ms of code execution and the number of times his code is triggered.

DISADVANTAGES OF AWS LAMBDA

1. **Latency while starting:** While AWS lambda is going to be activated after a long gap it will take some time to initialize the service which is required to deploy the application at that time end users will face latency issues.

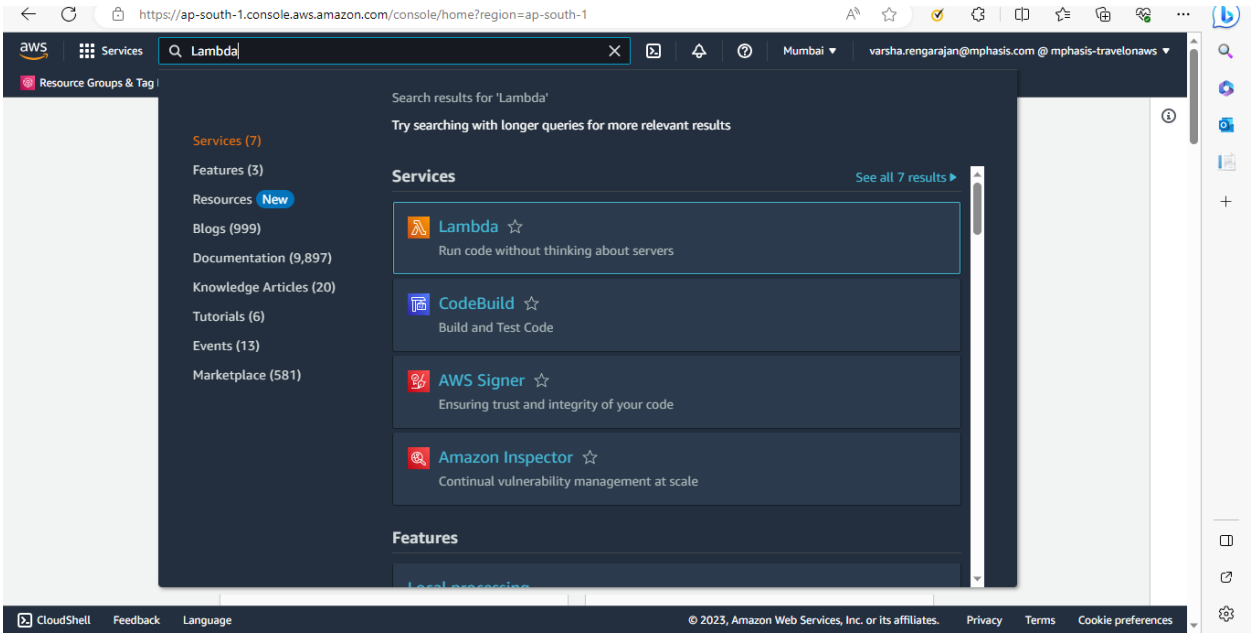
2. **Limited control of infrastructure:** Behalf of your lambda function is going to take of underlying infrastructure so you will have very limited control over undelaying infrastructure.

3. **Time Limit:** AWS Lambda enforces a maximum execution time limit for functions, which is currently set to 900 seconds (15 minutes). If your function exceeds this time limit, it will be forcibly terminated.

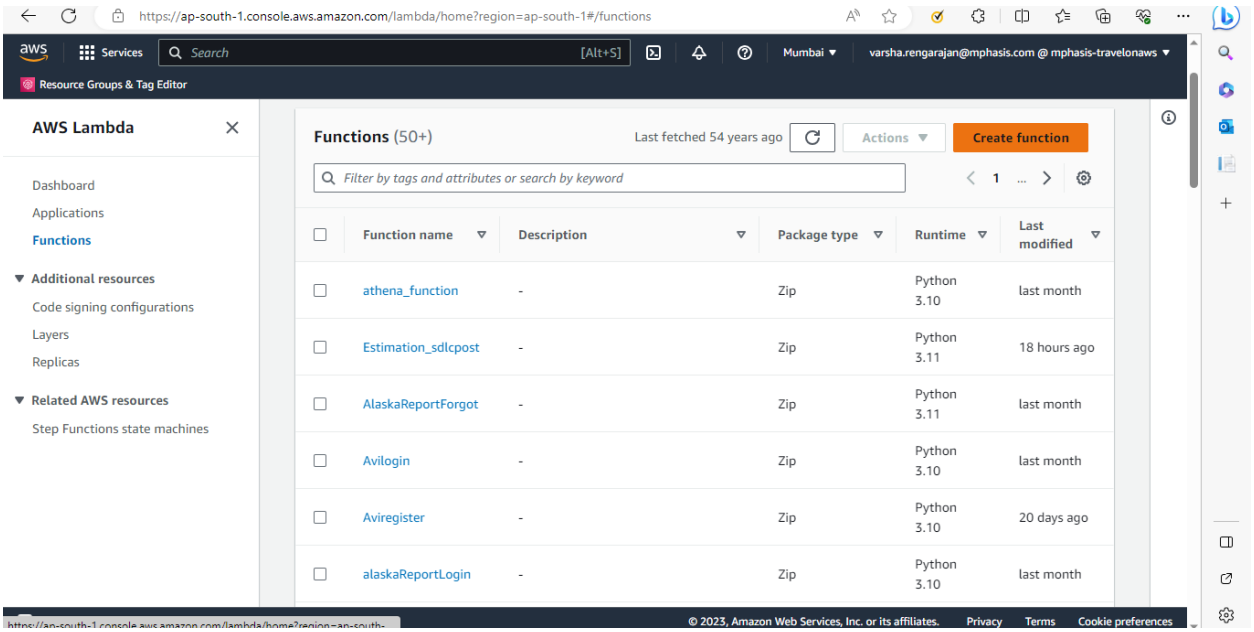
4. **Vendor Lock-In:** If you want to execute the lambda function then you need the support of any cloud provider as here, we are using AWS because it is widely used in the market.

5. **Limited Computing and Memory Options:** Limited configuration is there on the memory and CPU configuration. The predefined memory configuration was 120 MB to 120 GB and memory configuration determines the corresponding CPU power.

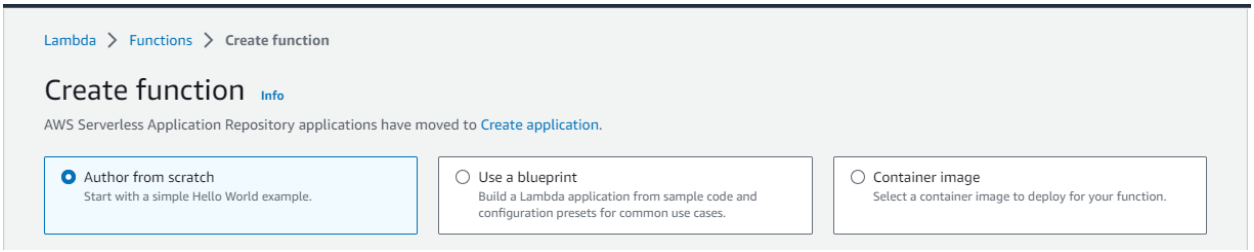
STEPS TO CREATE AWS LAMBDA



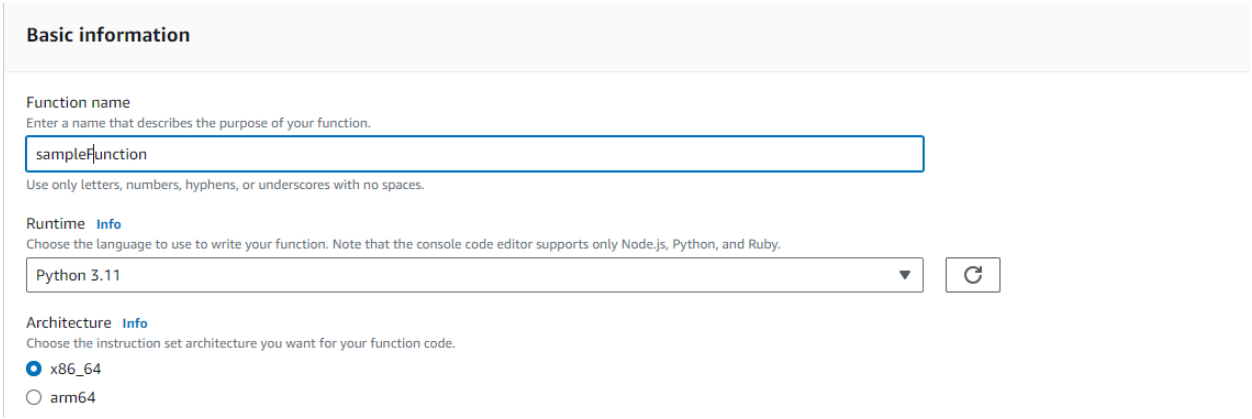
Log in to AWS account and choose AWS Lambda



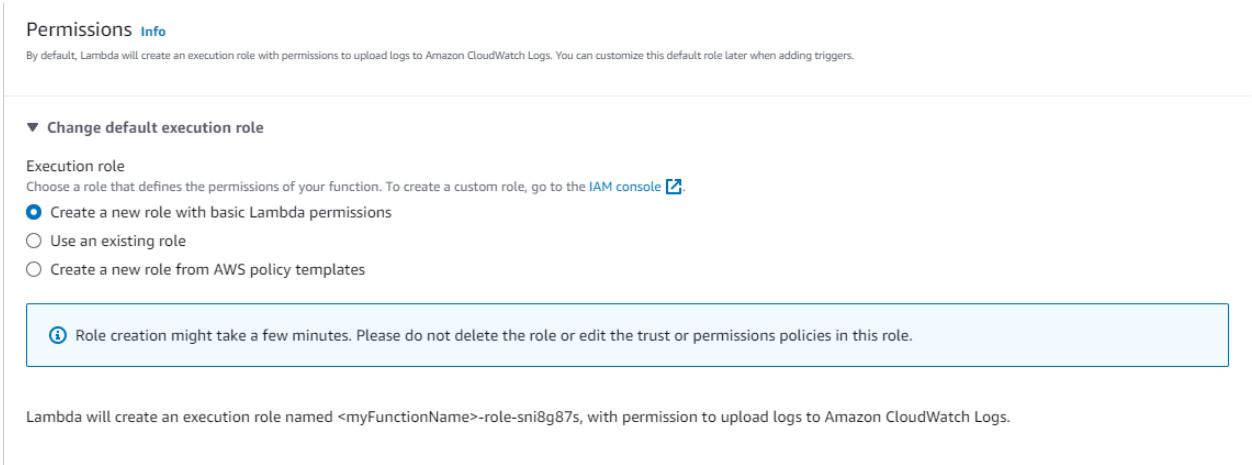
Click “Create function” button.



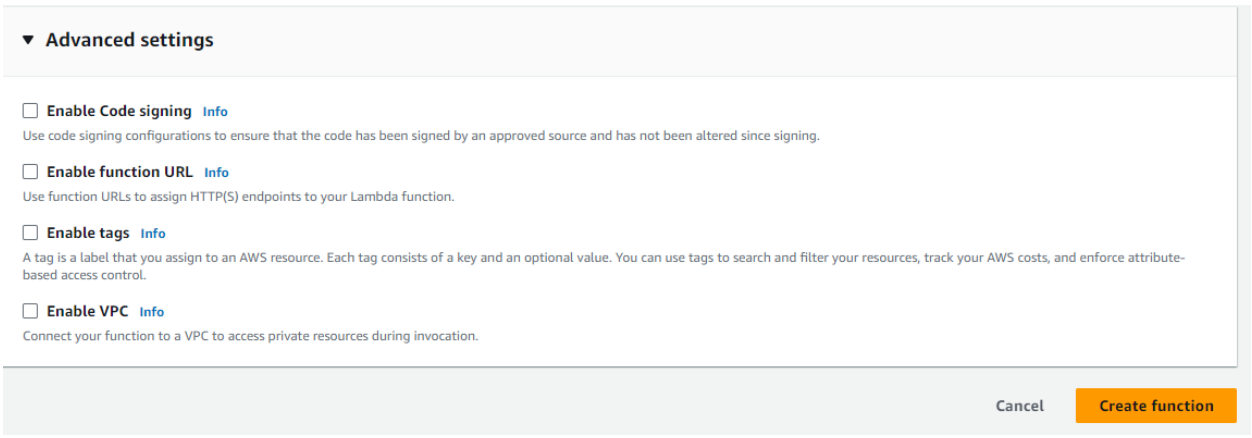
Choose the way of creating a code template.



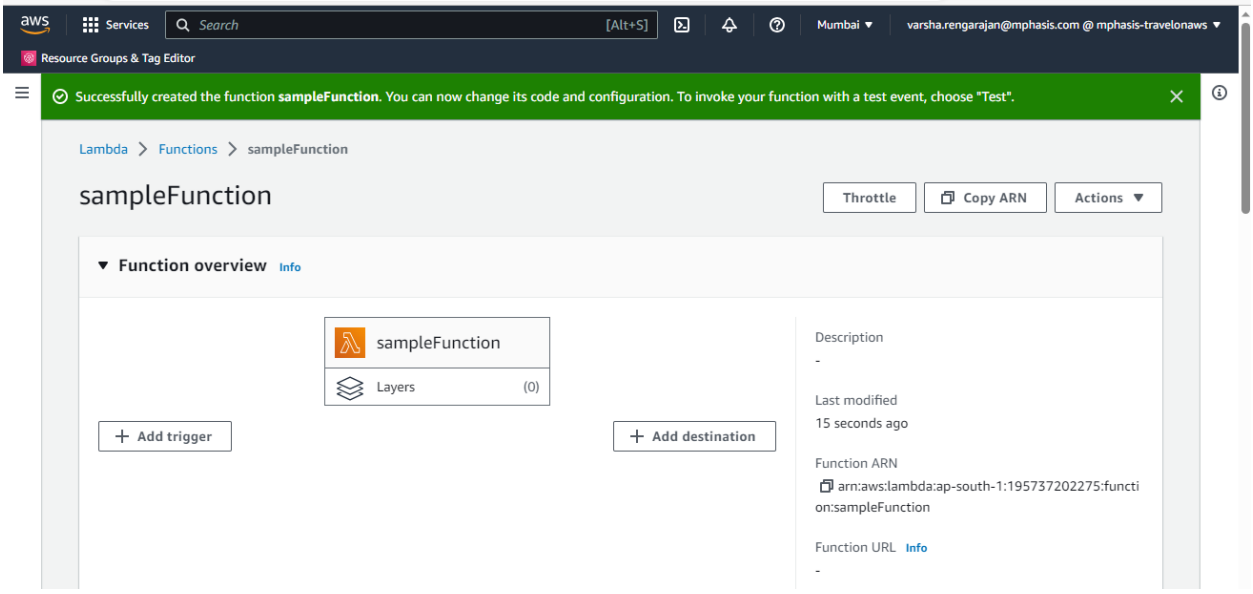
Fill the basic information like function name, runtime, and architecture.



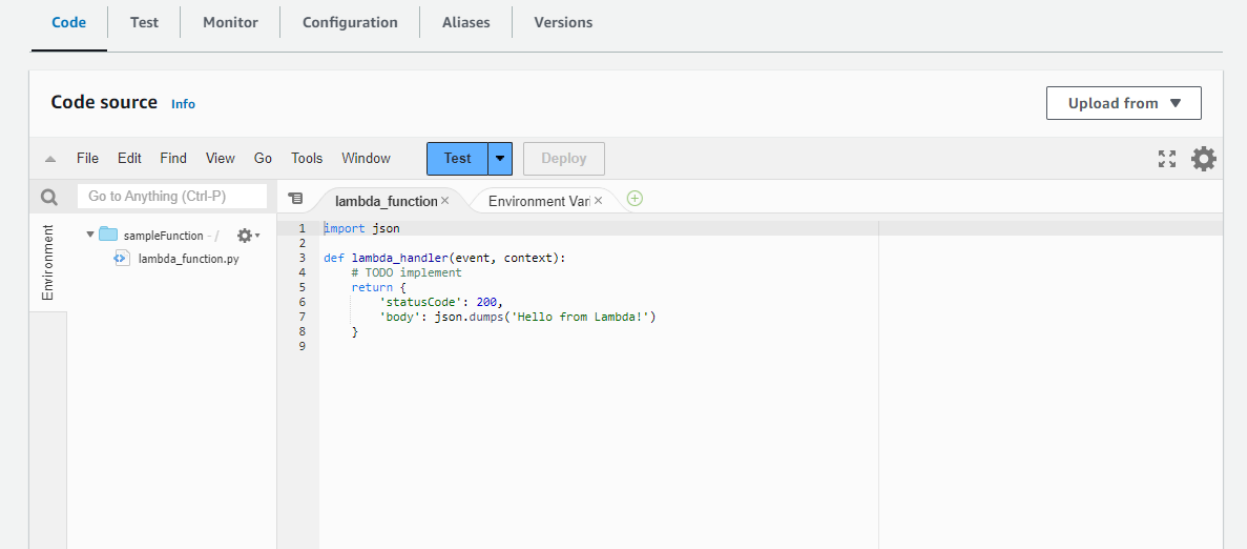
Choose the permissions based on your needs.



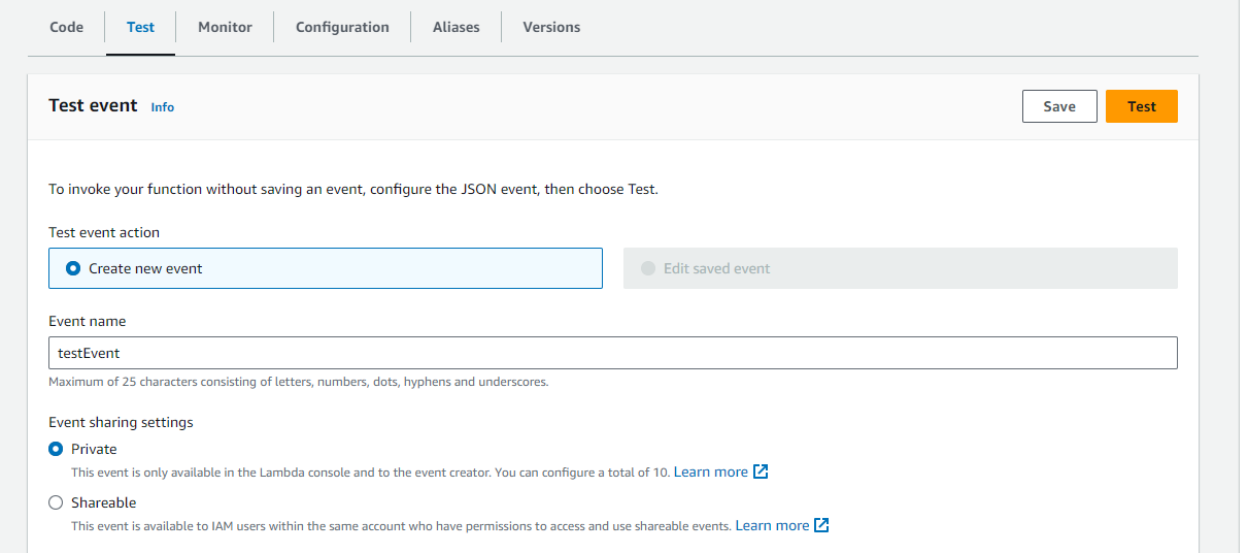
In advanced settings, enable the features based on your needs and click "Create function".



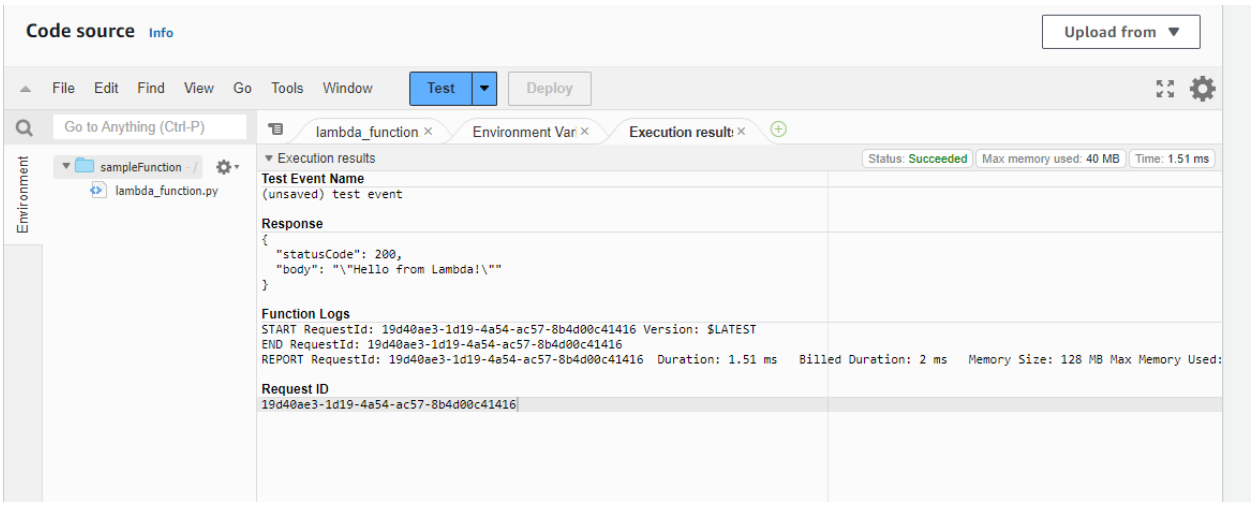
Function created successfully.



In coding portion, write the code according to your runtime.



In test portion, create new event and mention the event name. If required, insert the parameters in JSON format.

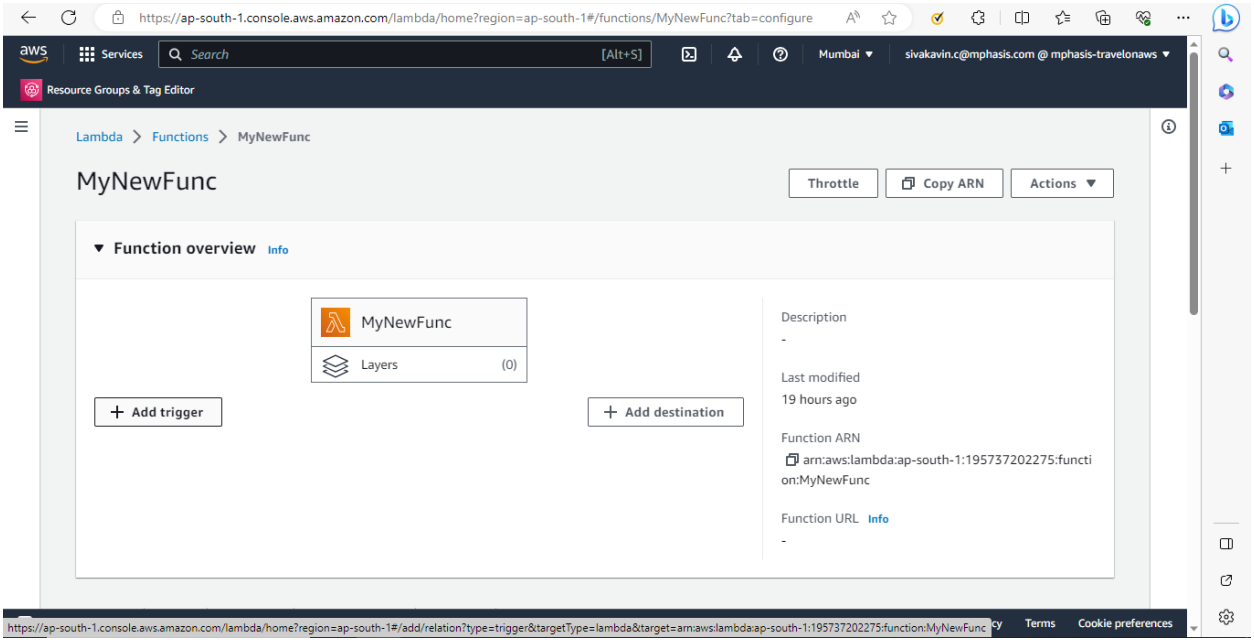


Click “Test”, it will display the output in execution results.

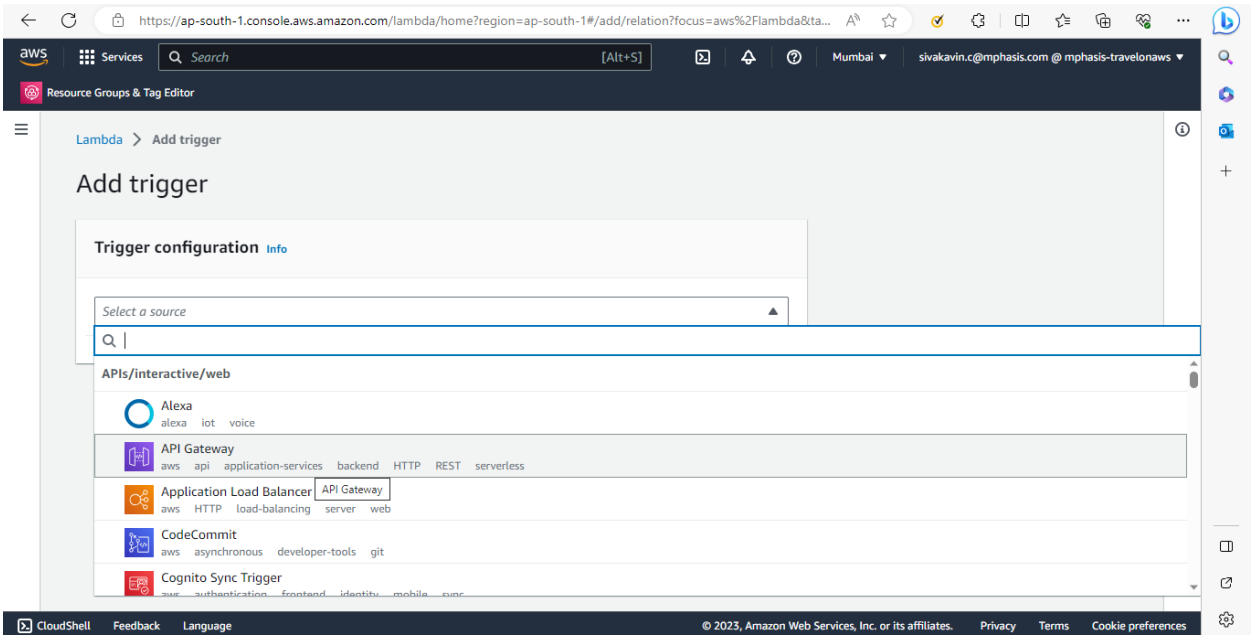
INTEGRATION

INTEGRATING LAMBDA WITH API GATEWAY

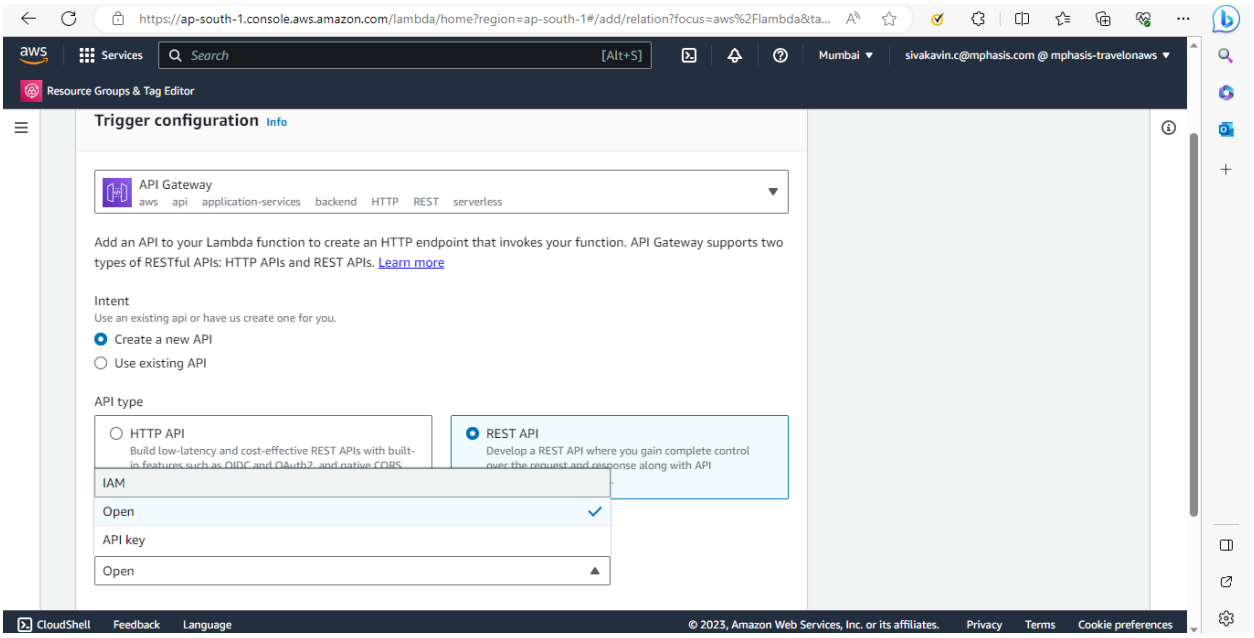
STEP 1: Click “Add trigger”



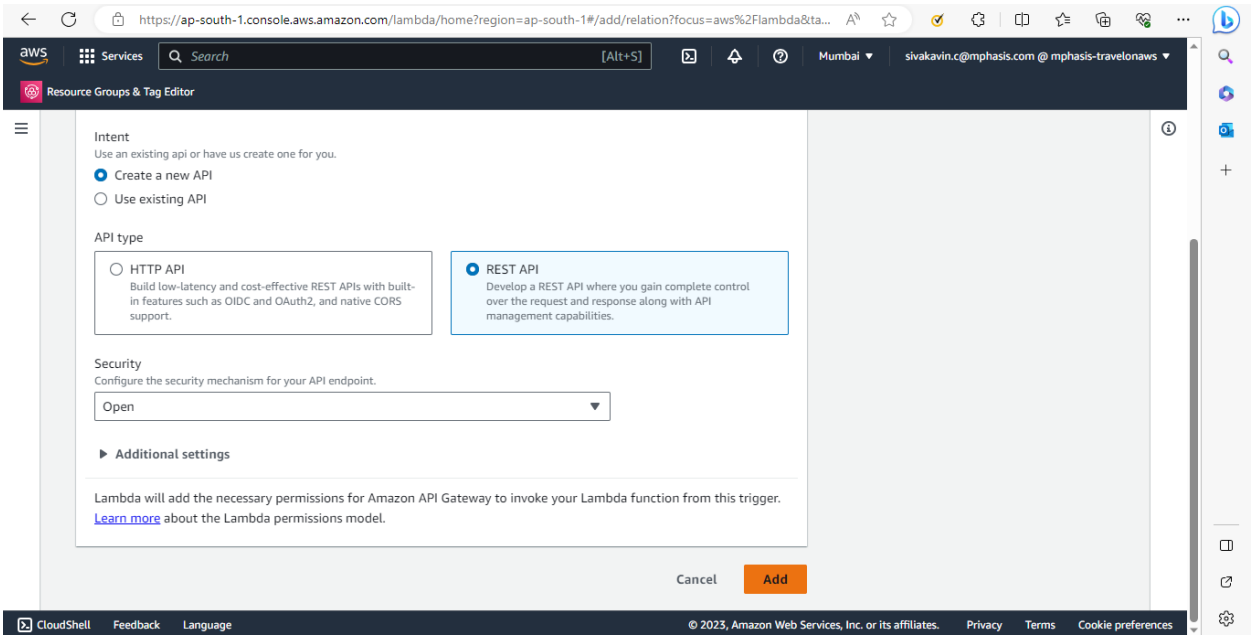
STEP 2: Select the source as “API Gateway.”



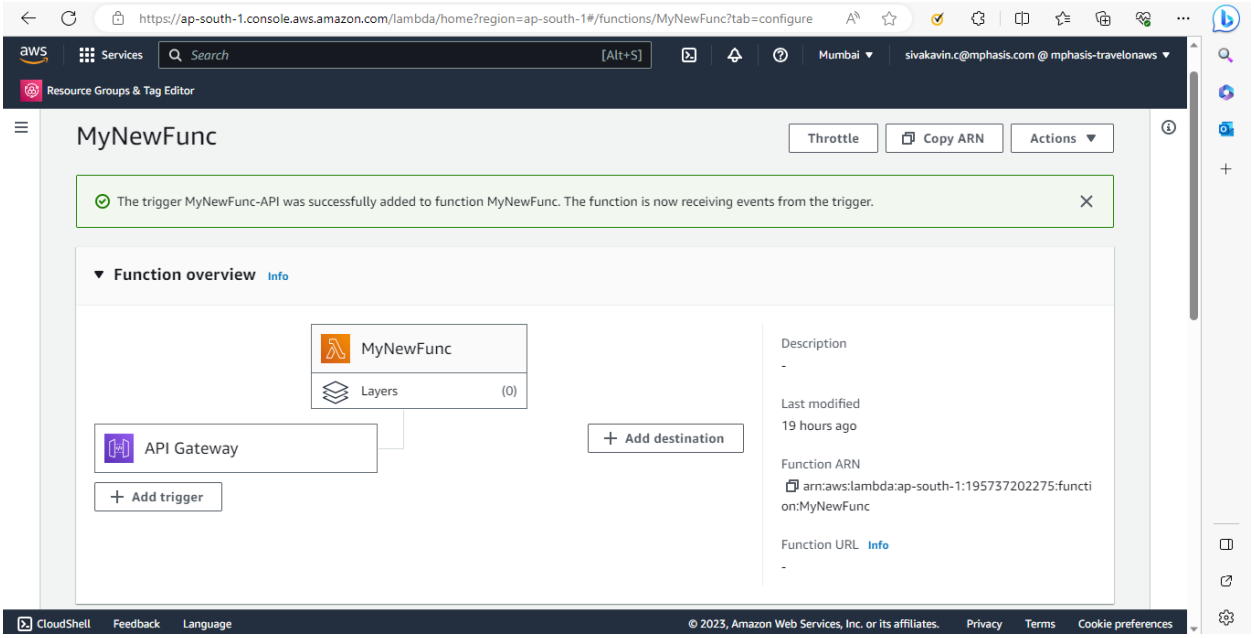
STEP 3: Choose “Create a new API” and choose API type as “REST API” and security type as “Open.”



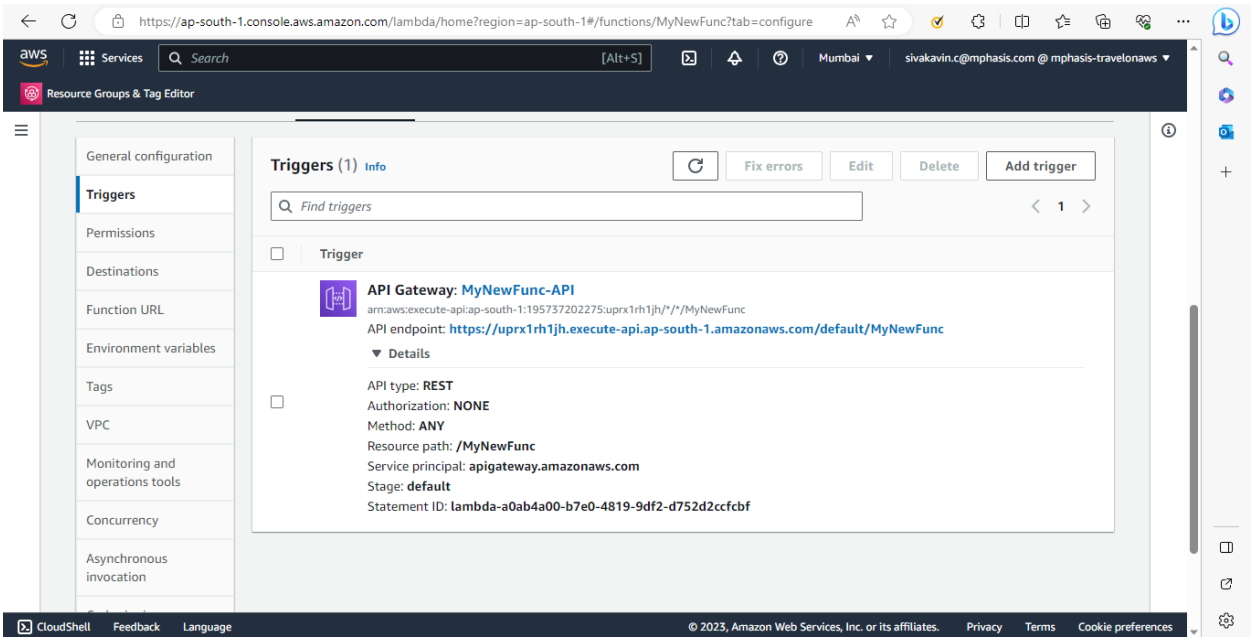
STEP 4: Click “Add” to create a trigger.



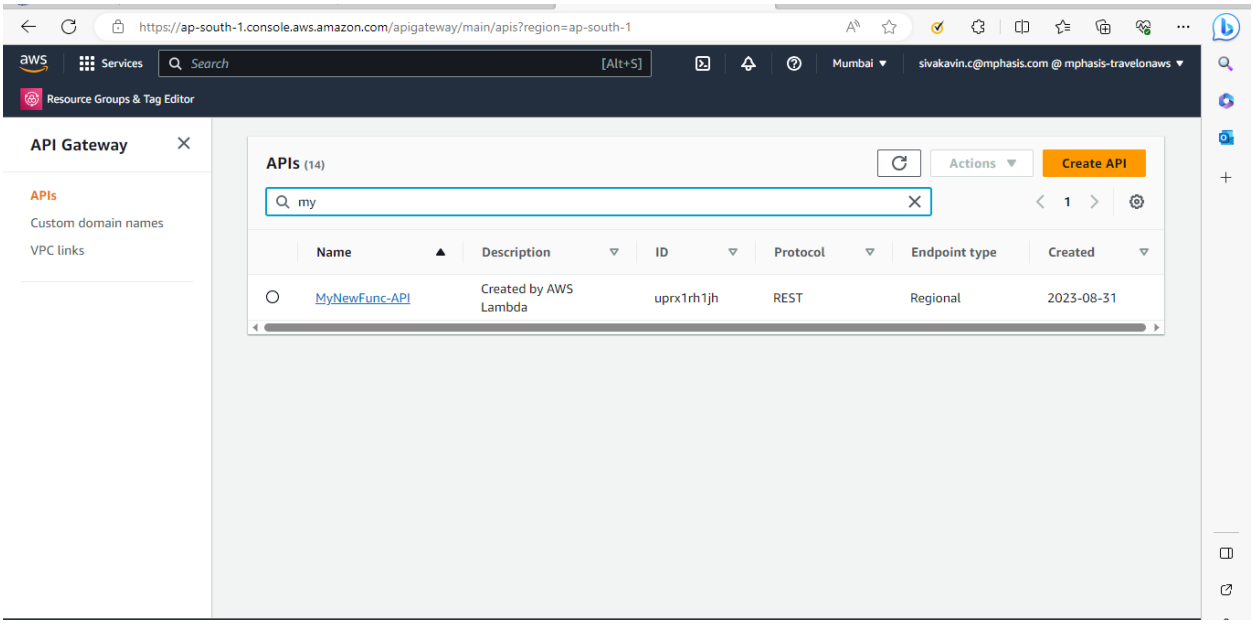
STEP 5: Trigger added successfully.



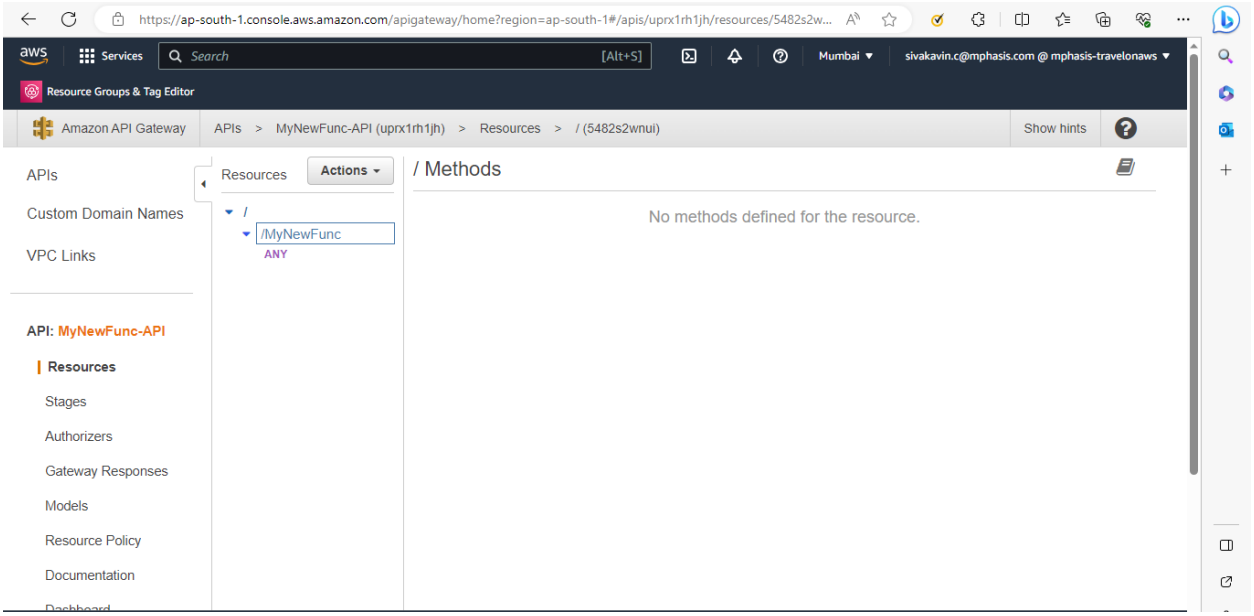
STEP 6: Trigger details are shown below



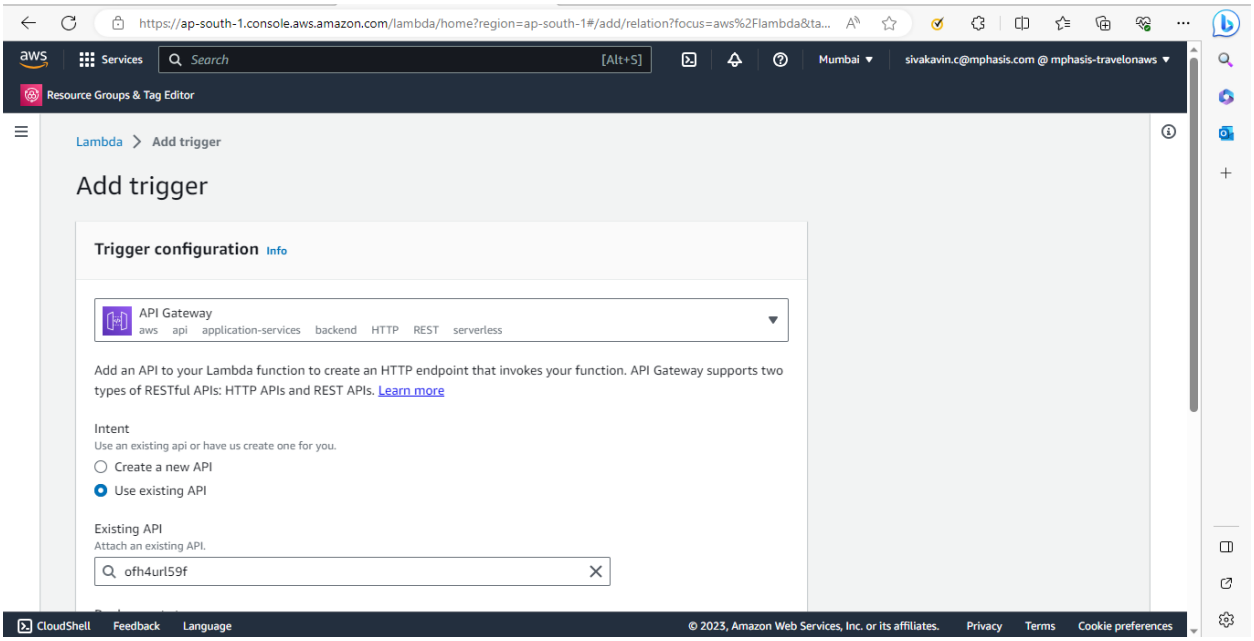
STEP 7: Open “API Gateway” service and newly created API is displaying in this API’s console.



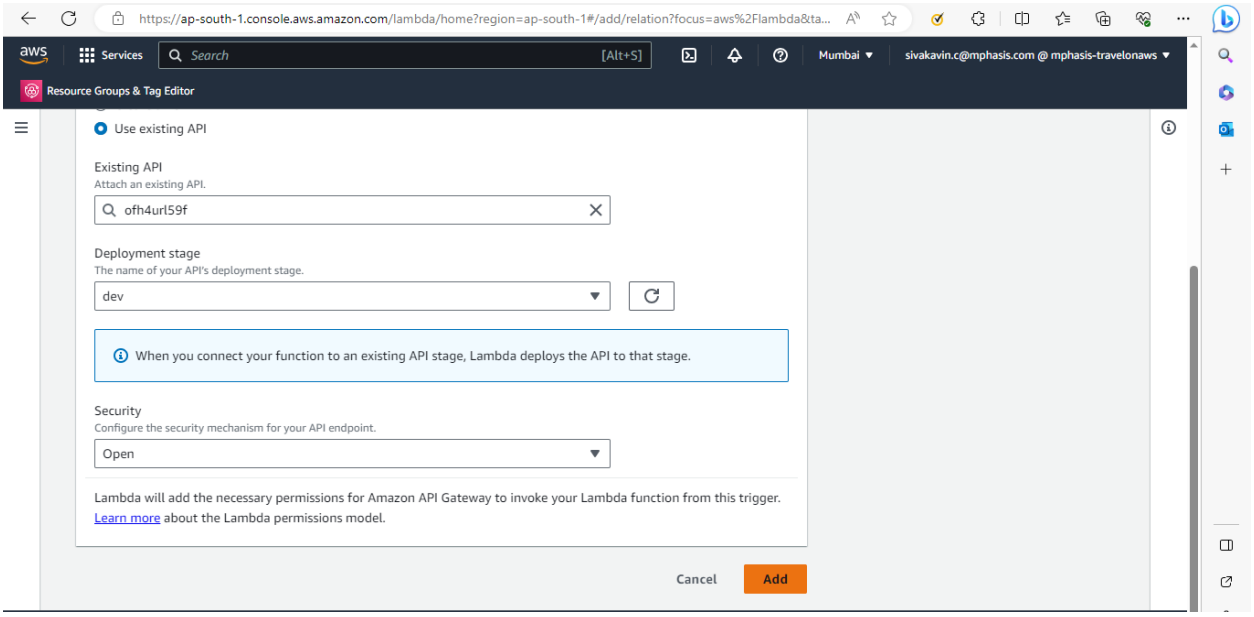
STEP 8: Resource also created by default.



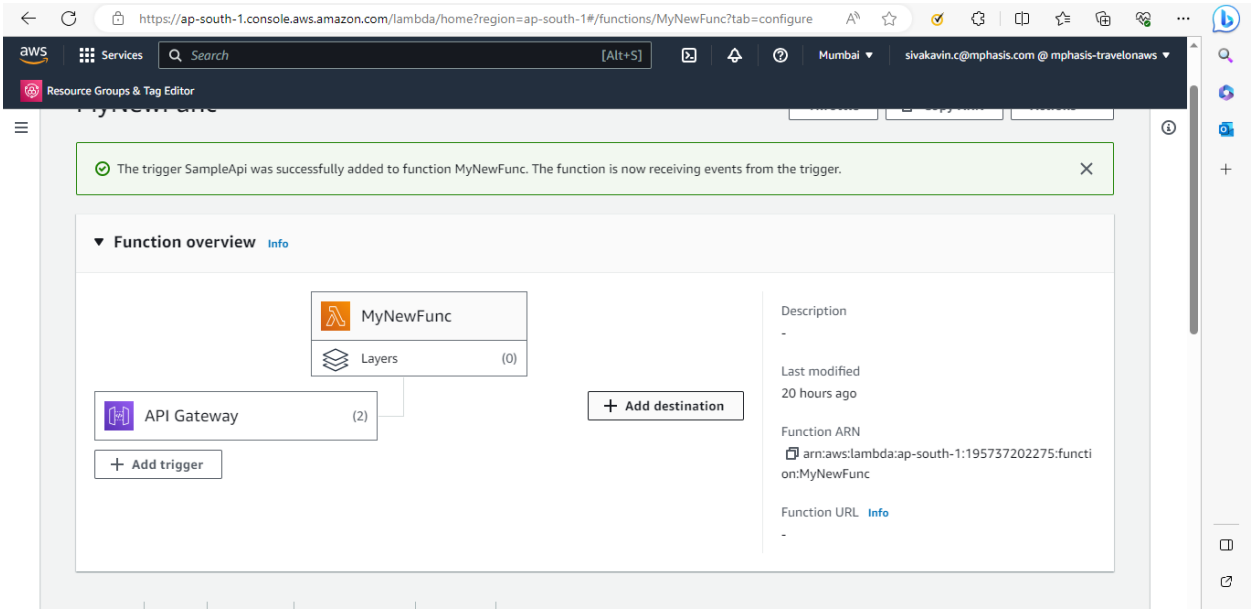
STEP 9: Instead of choosing “Create a new API”, choose “Use existing API” (If you have an existing API)



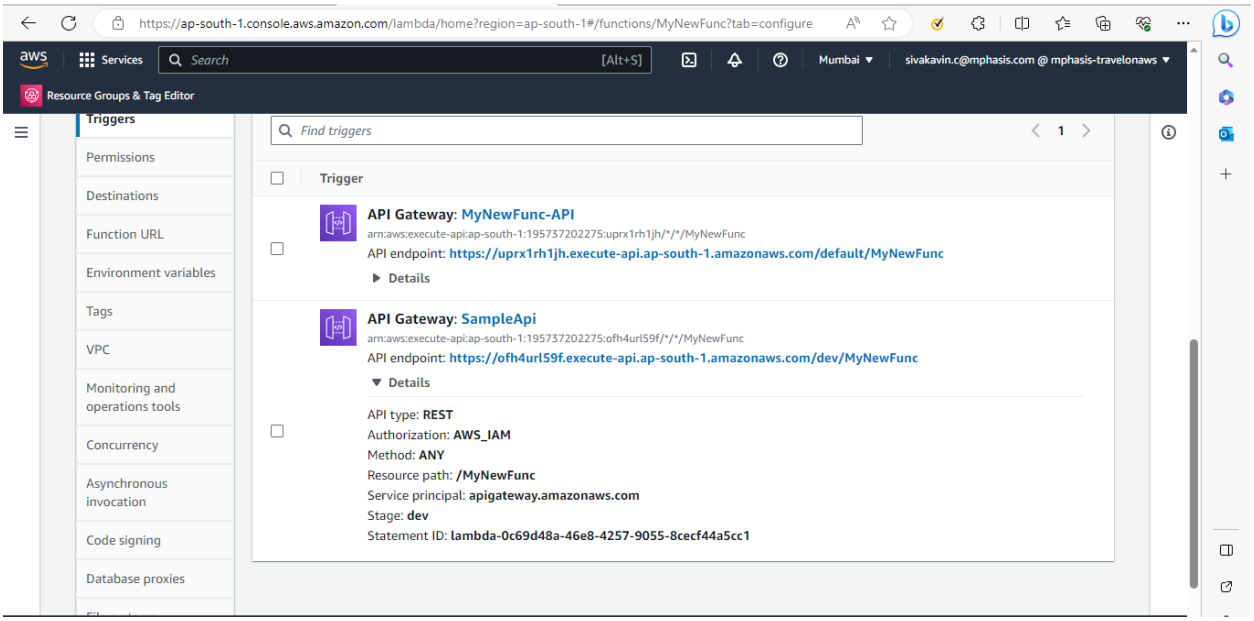
STEP 10: Choose “Existing API” and choose “Deployment stage” and security and click “Add”.



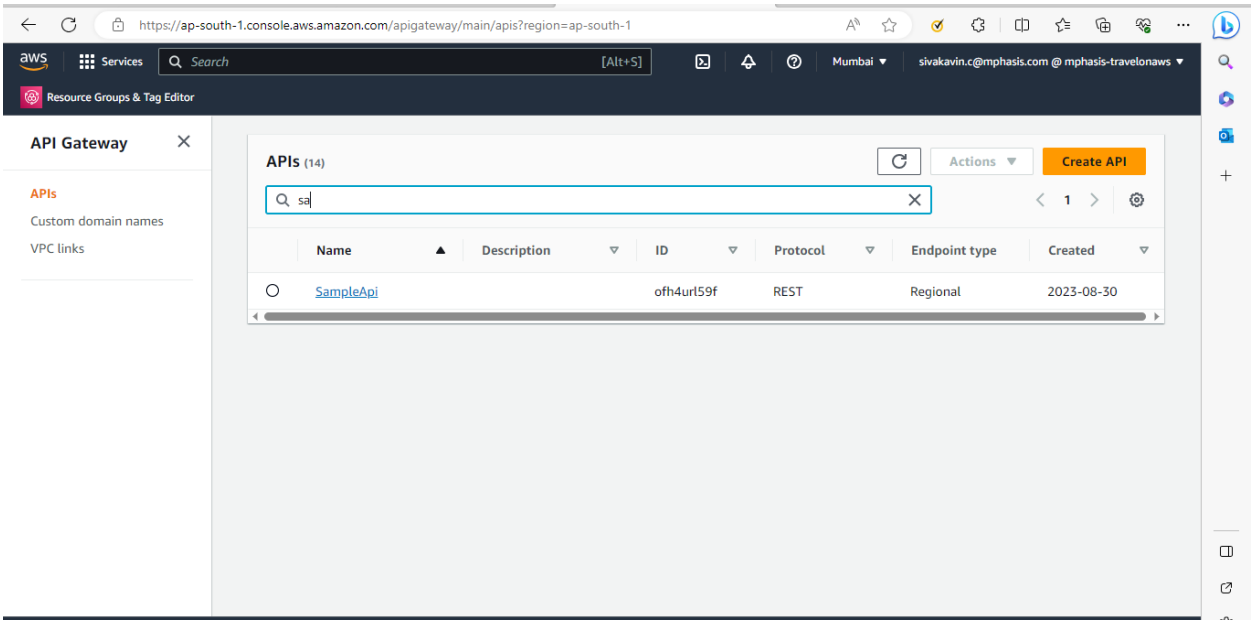
STEP 11: Trigger added successfully for an existing API.



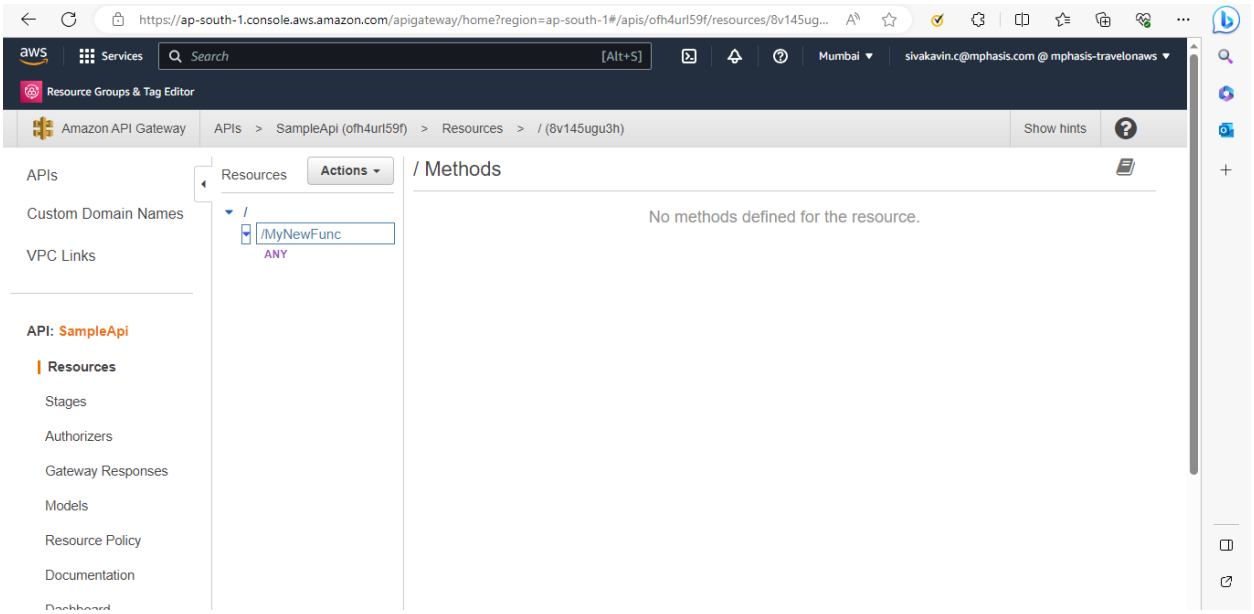
STEP 12: Here is the details of trigger.



STEP 13: Open “API Gateway” console and search for “SampleApi” API.

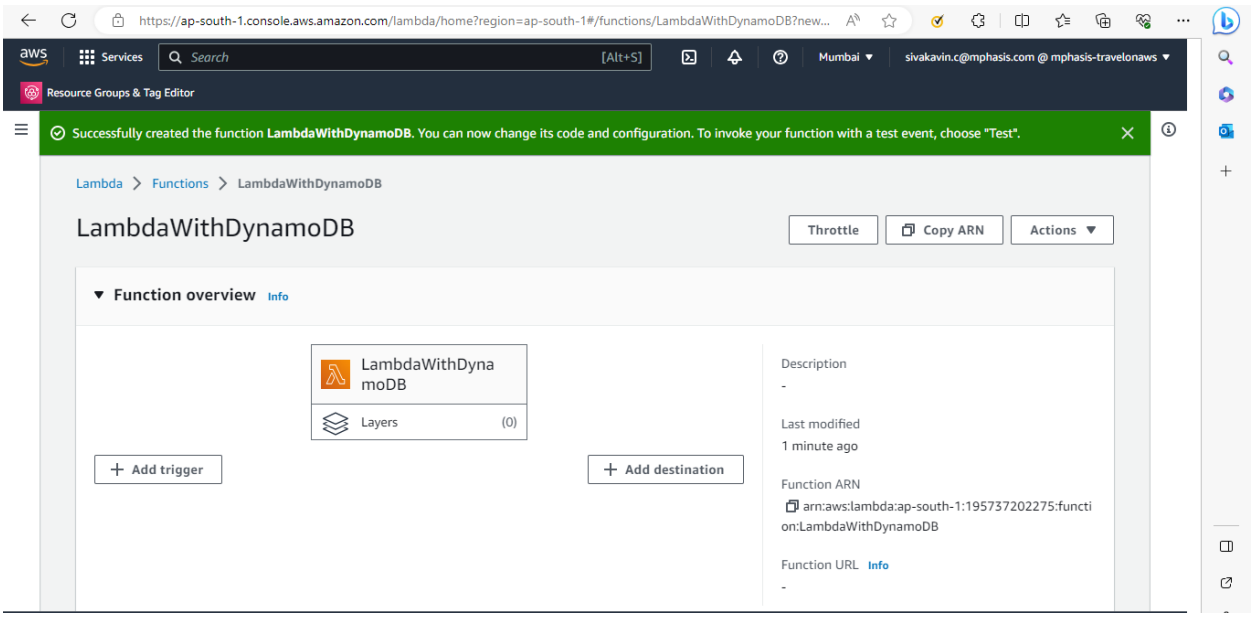


STEP 14: Resource created successfully.

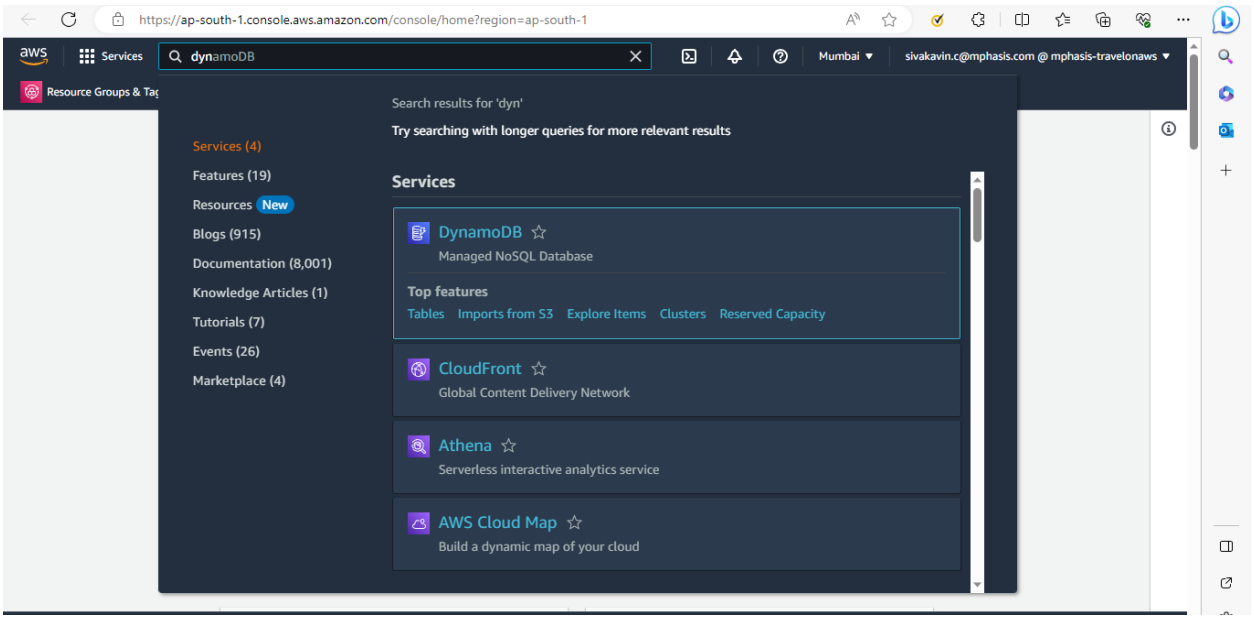


INTEGRATING LAMBDA WITH DYNAMODB

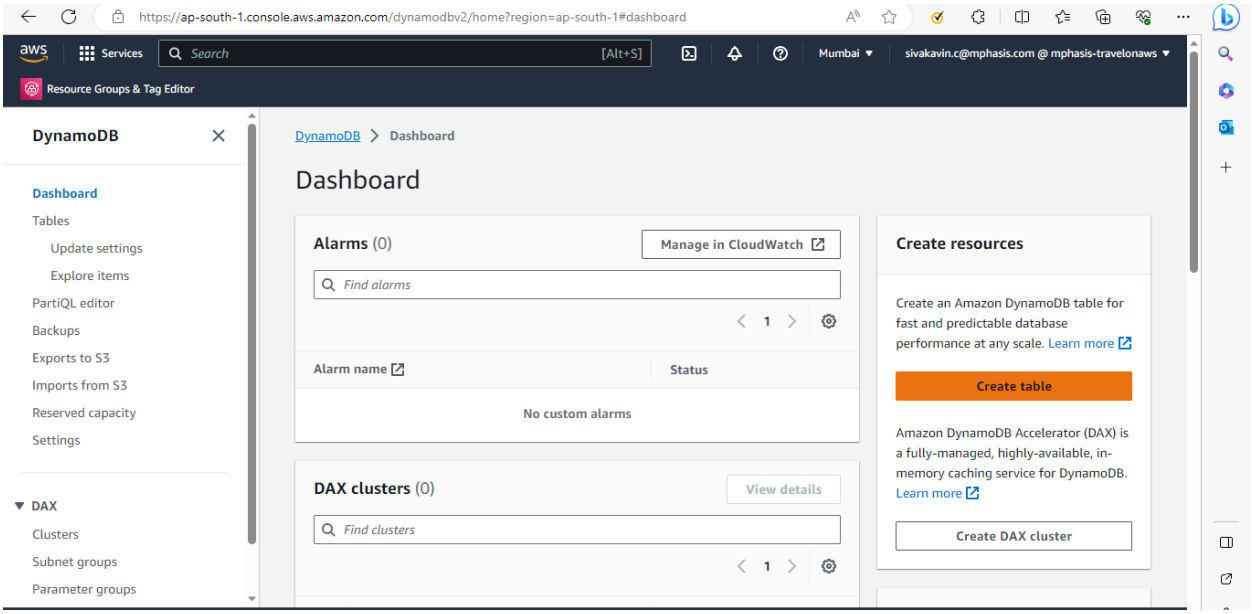
STEP 1: Create the “lambda function.”



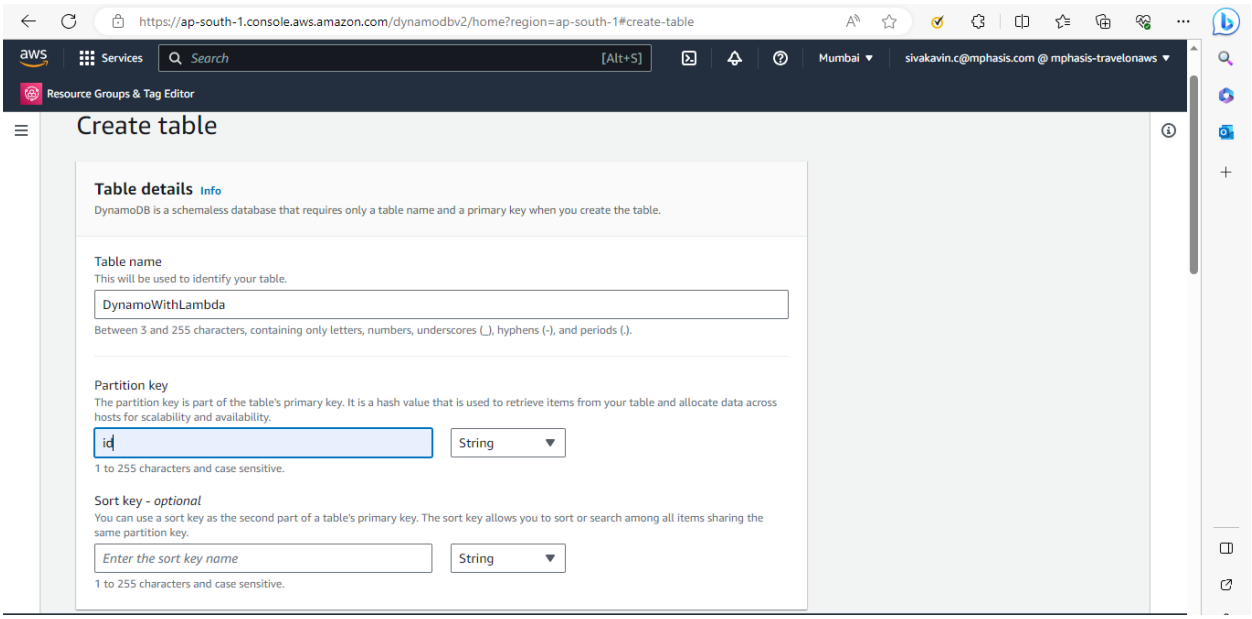
STEP 2: Click the “DynamoDB” service.



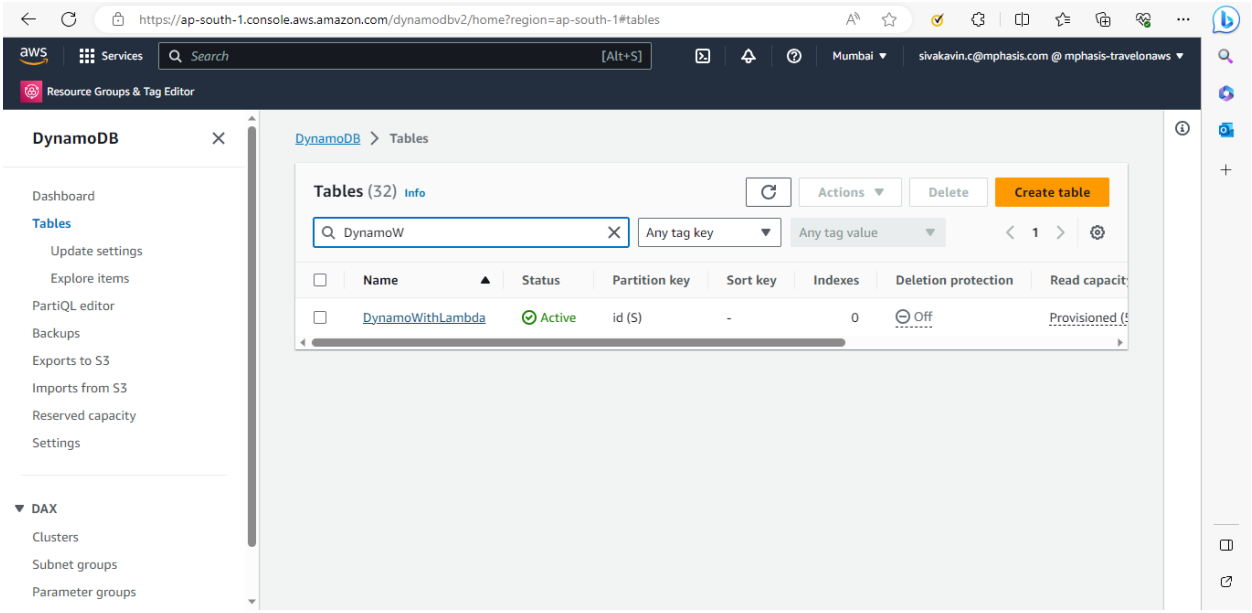
STEP 3: Click “Create table”.



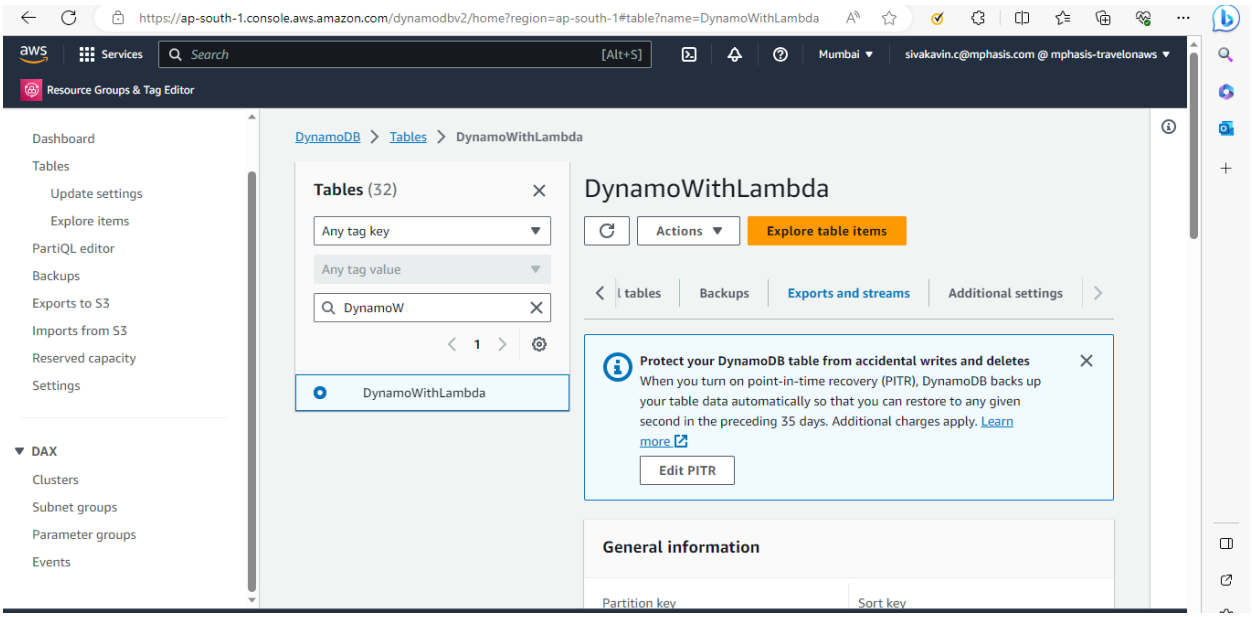
STEP 4: Enter table name, partition key, and click create table.



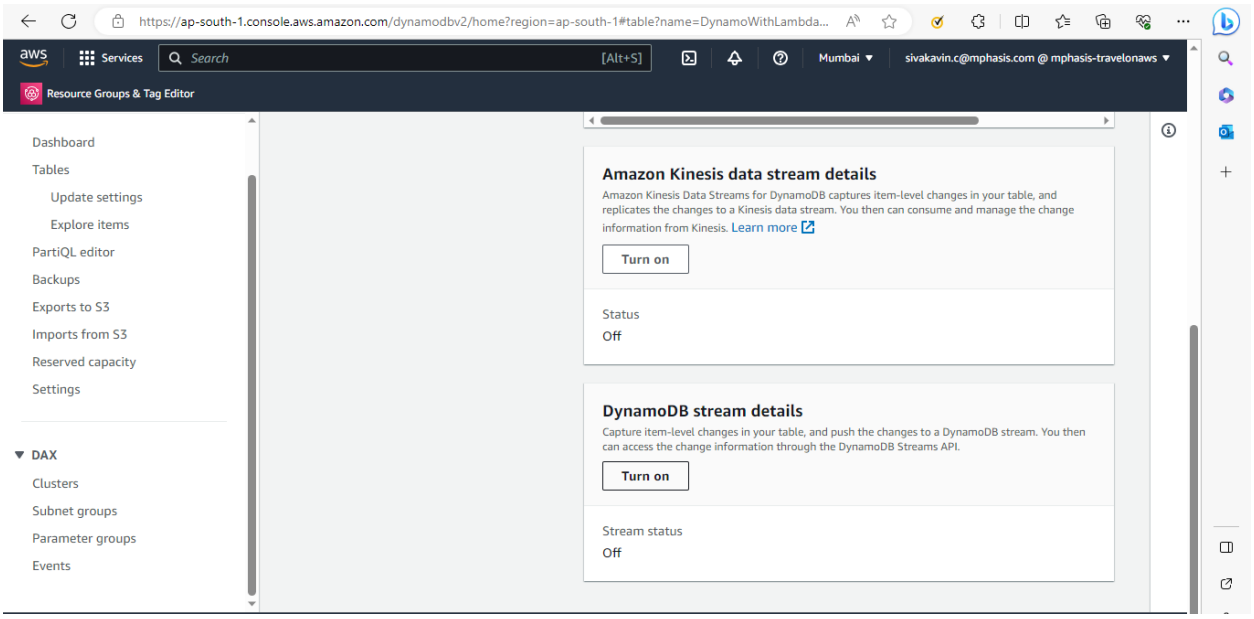
STEP 5: Table is created here.



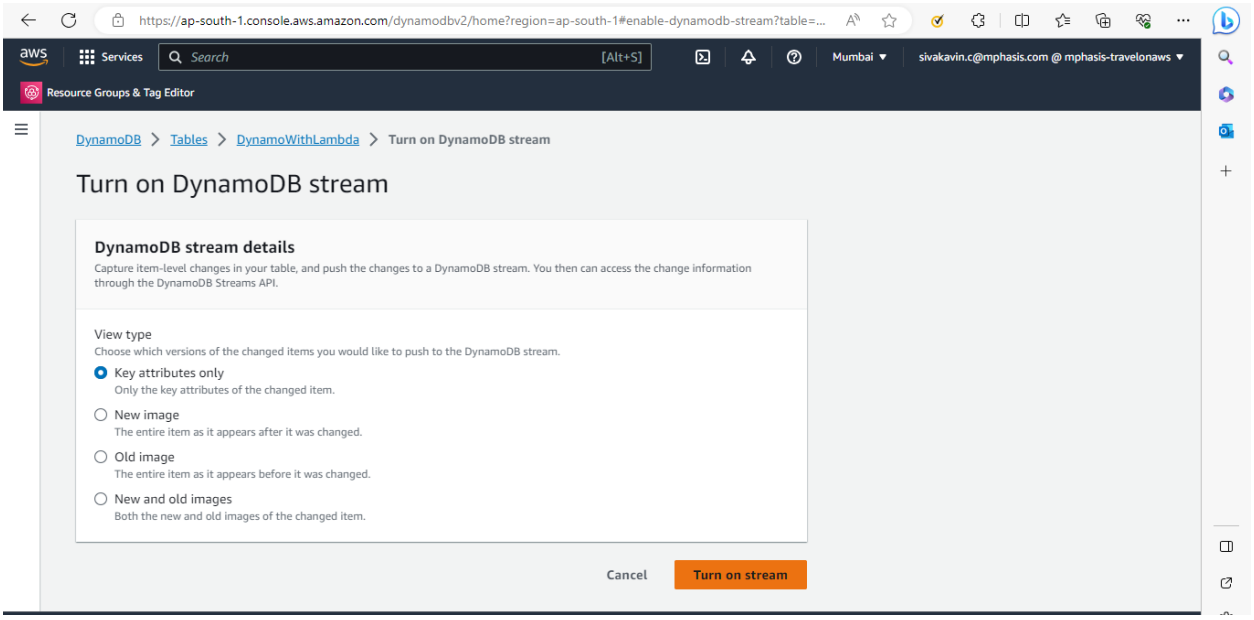
STEP 6: After table creation, In Dynamo DB console, click “Exports and streams”.



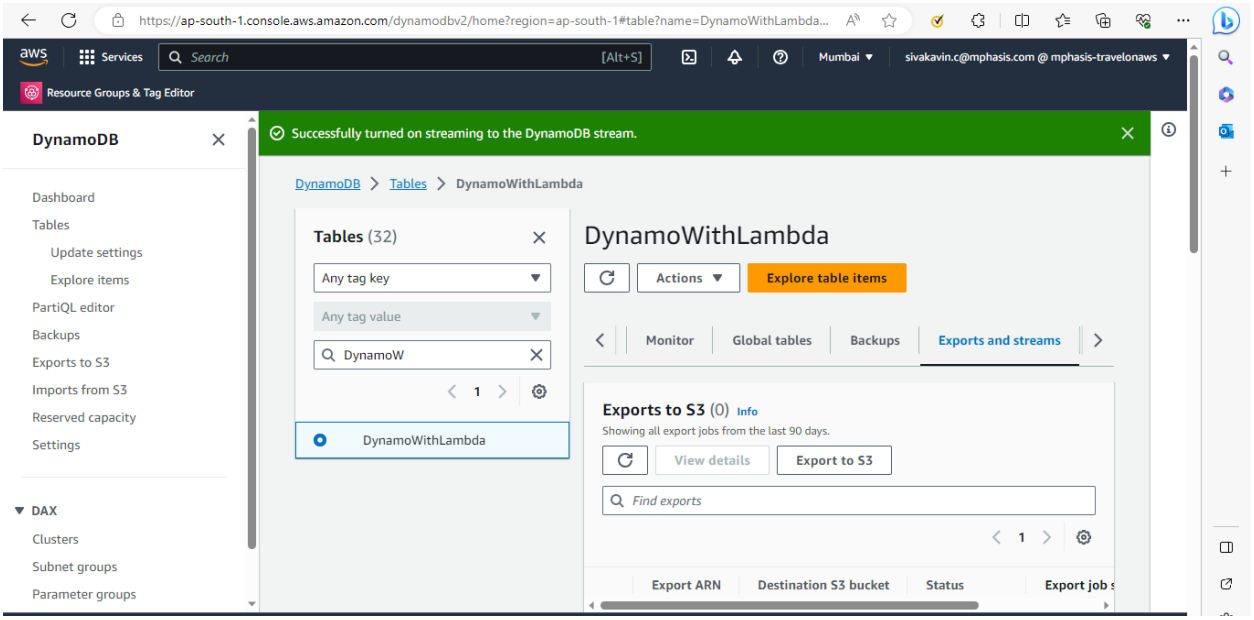
STEP 7: In “DynamoDB stream details”, click Turn on.



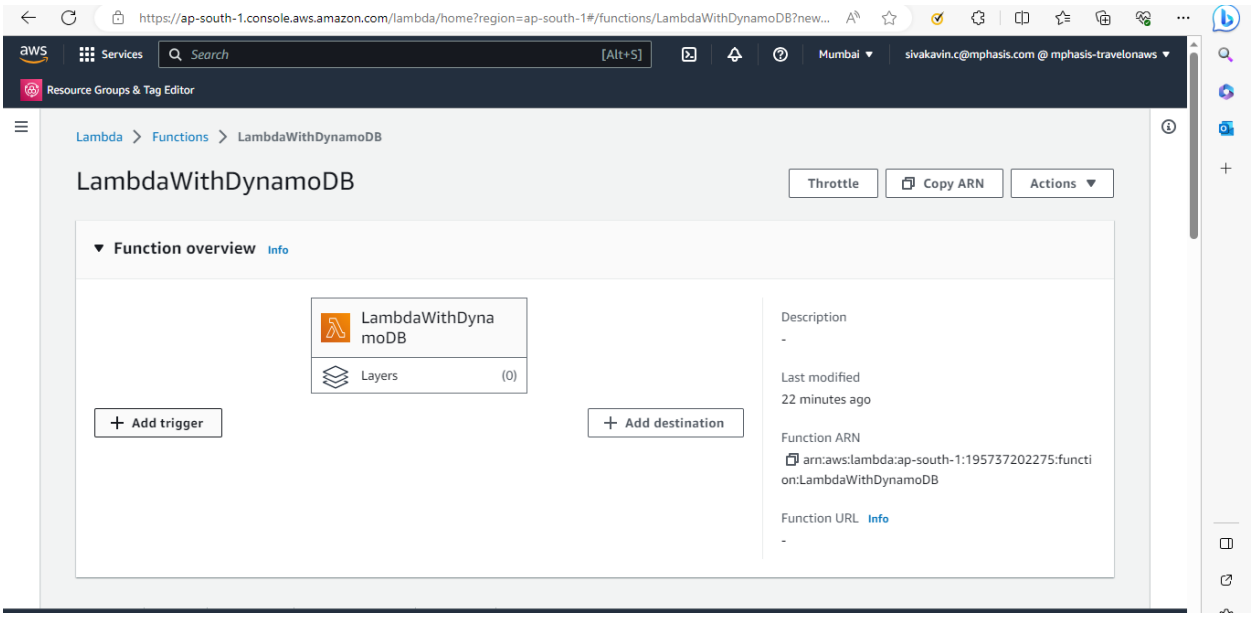
STEP 8: Choose view type and click “Turn on stream”.



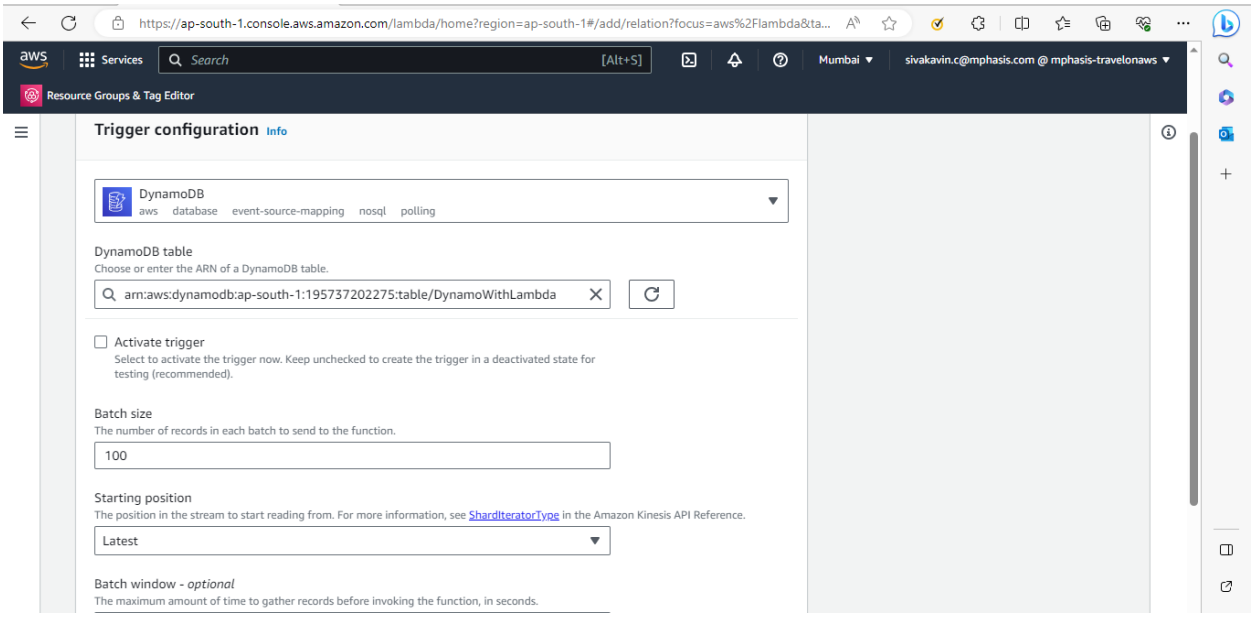
STEP 9: DynamoDB stream is turned on successfully. DynamoDB stream is used for logging the changes in DynamoDB table.



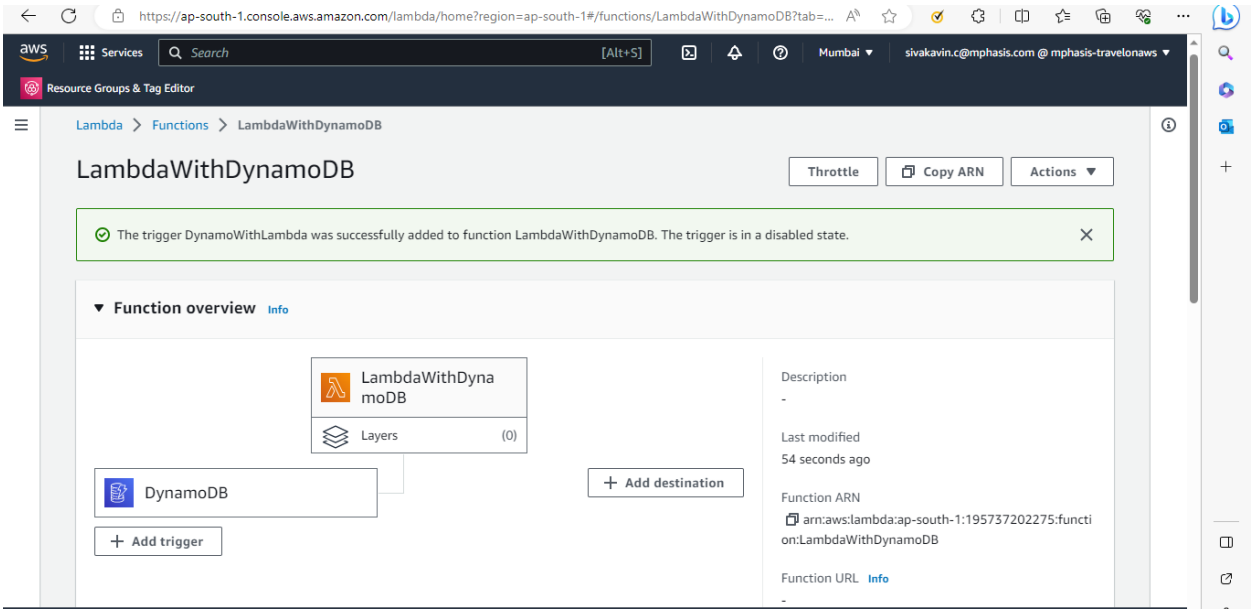
STEP 10: Click "Add trigger".



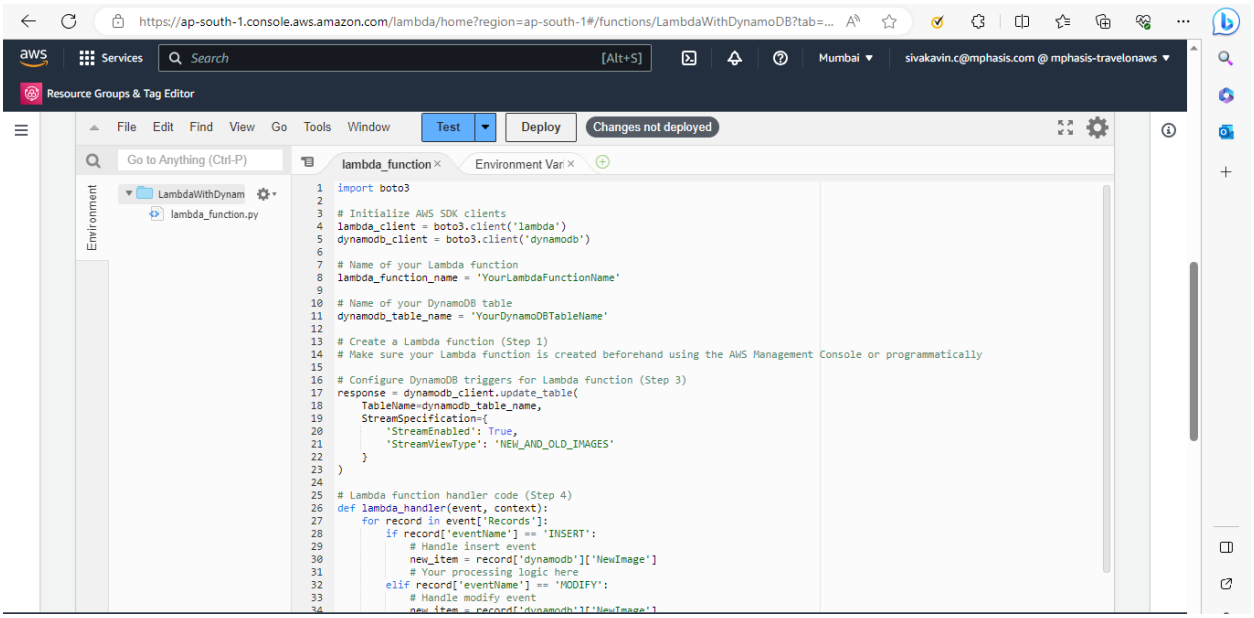
STEP 11: Choose “DynamoDB” as source, “DynamoDB Table name”, Batch size and click “Add”.



STEP 12: Trigger is added successfully.

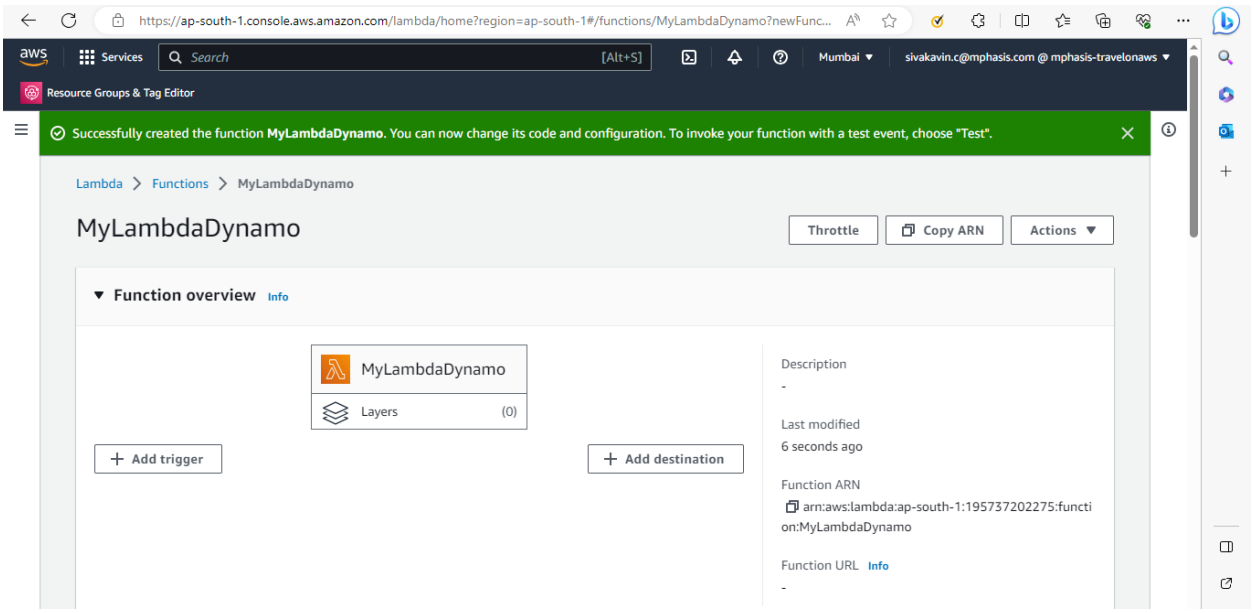


STEP 13: Add a code to access the data from “Dynamo DB”.

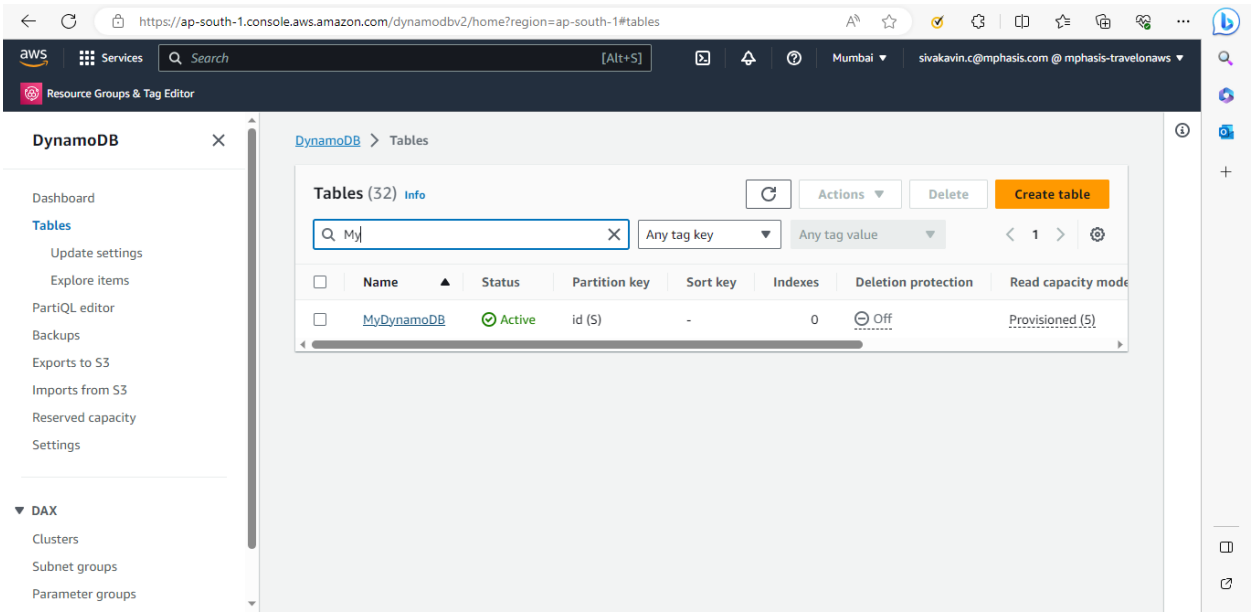


OTHER WAY OF INTEGRATION WITH DYNAMODB

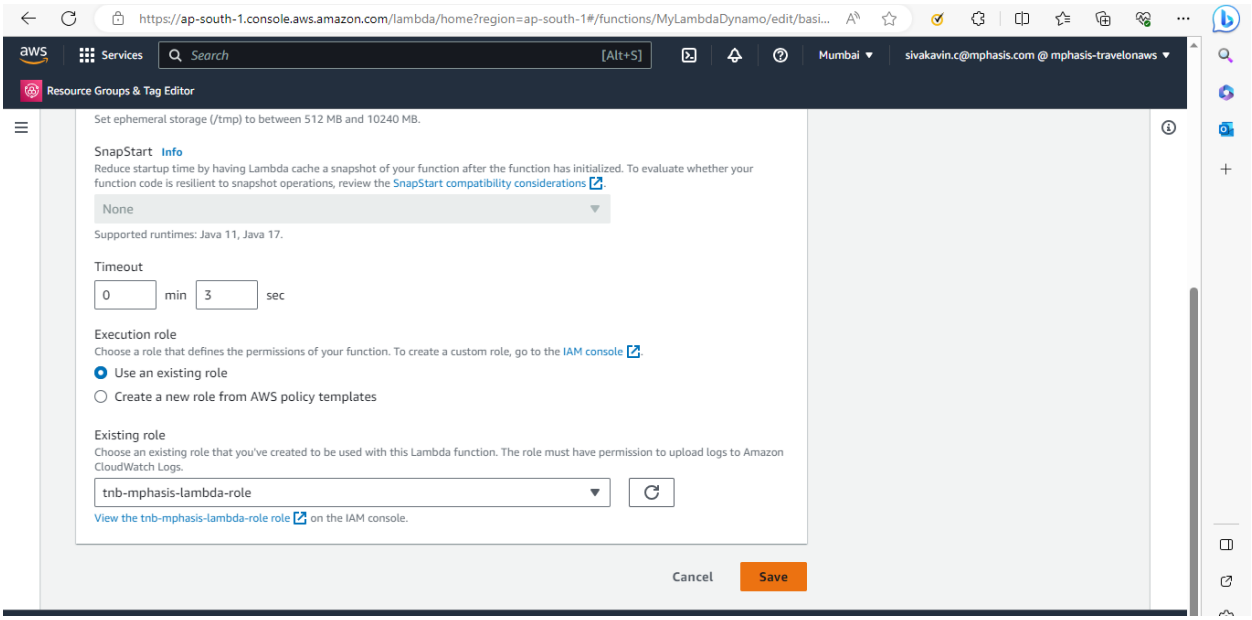
STEP 1: Create Lambda function.



STEP 2: Create Dynamo DB Table.



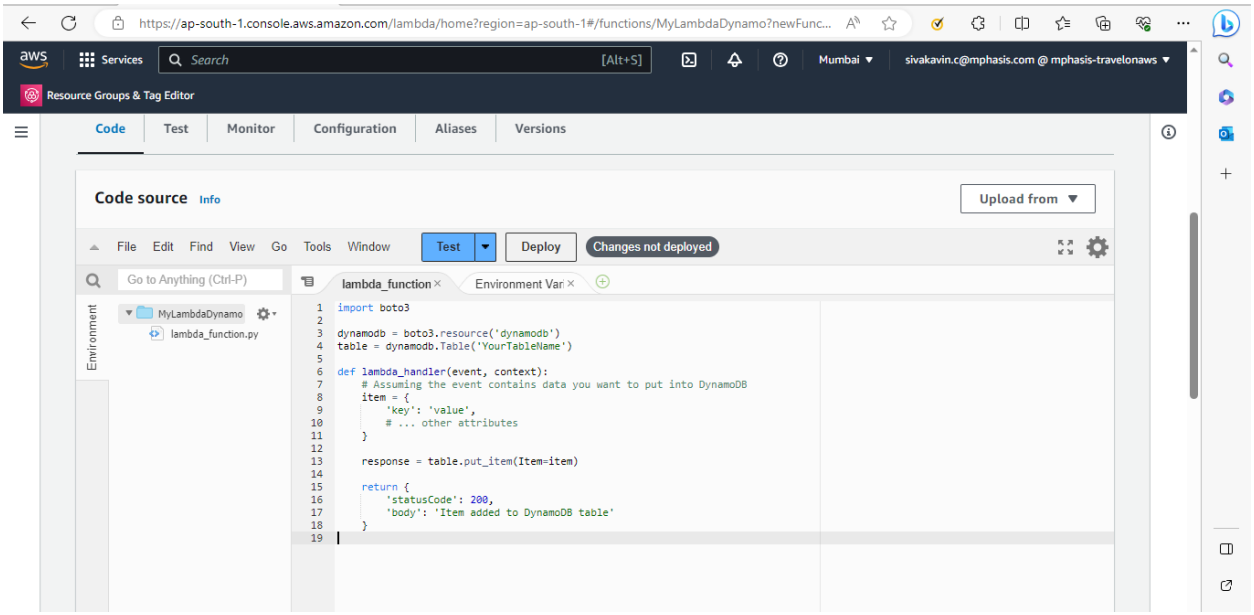
STEP 3: Create an “IAM role” or use existing role with access as “DynamoDBFullAccess”.



CODE FOR DYNAMODB FULL ACCESS

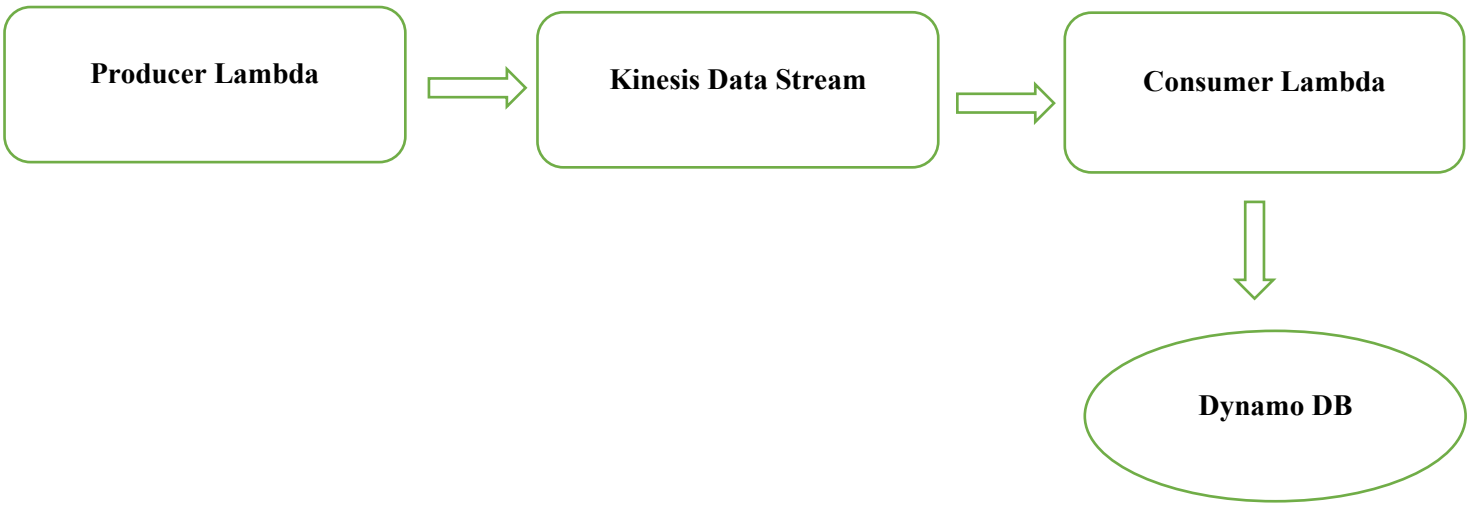
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:*",
      "Resource": "*"
    }
  ]
}
```

STEP 4: Write a code to access the dynamo DB table.

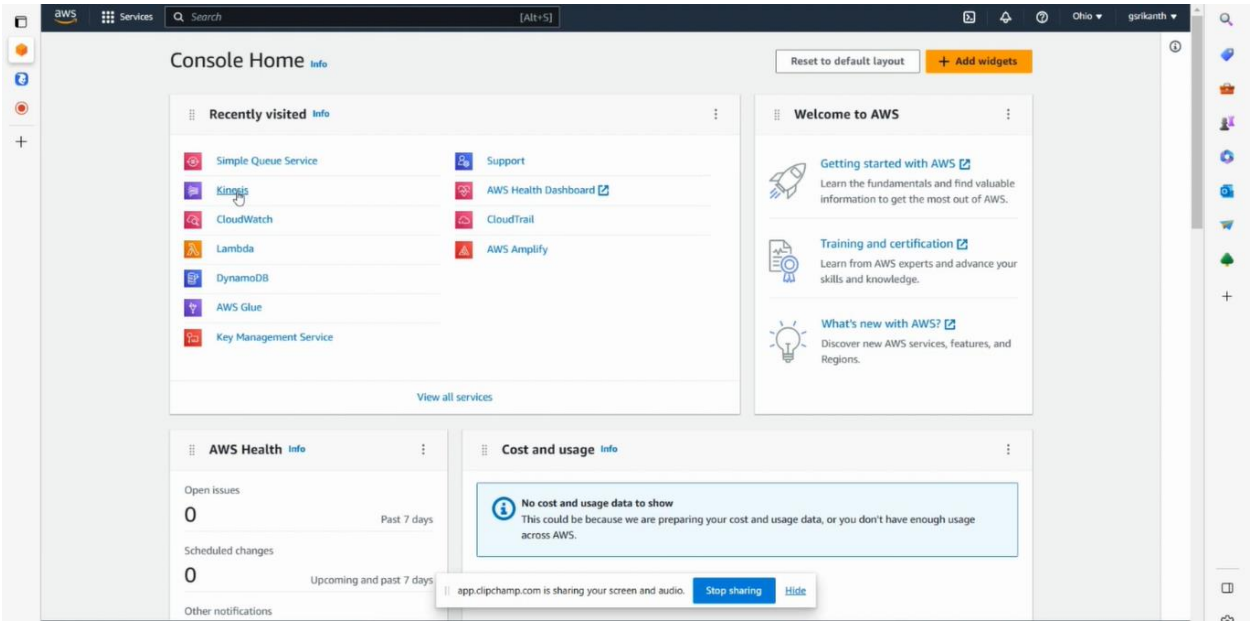


REAL TIME DATA PROCESSING WITH AWS KINESIS, LAMBDA AND DYNAMO DB

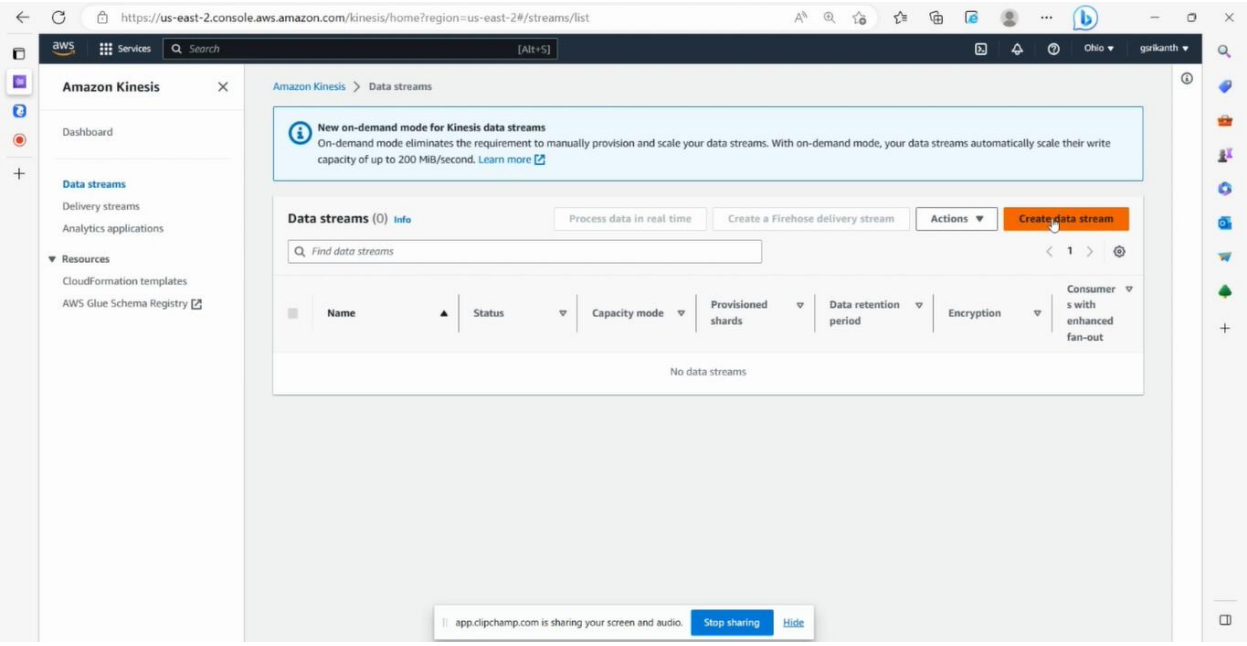
WORKFLOW DIAGRAM



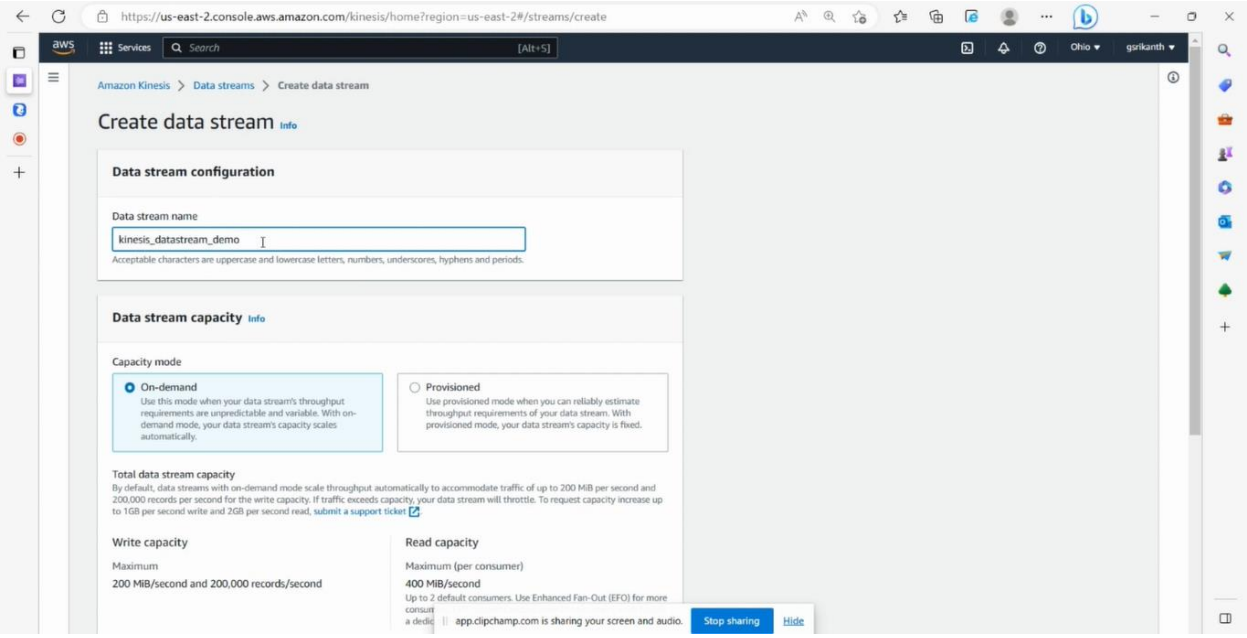
STEP 1: Choose the service “Kinesis”.



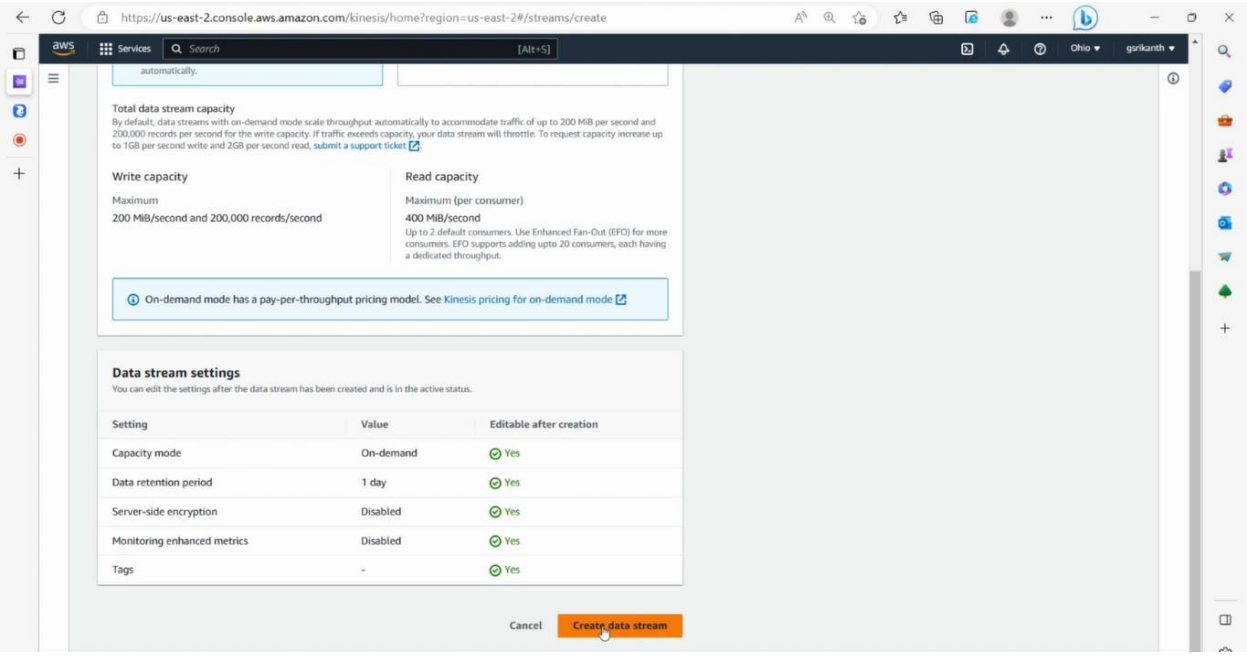
STEP 2: Click “Create data streams”.



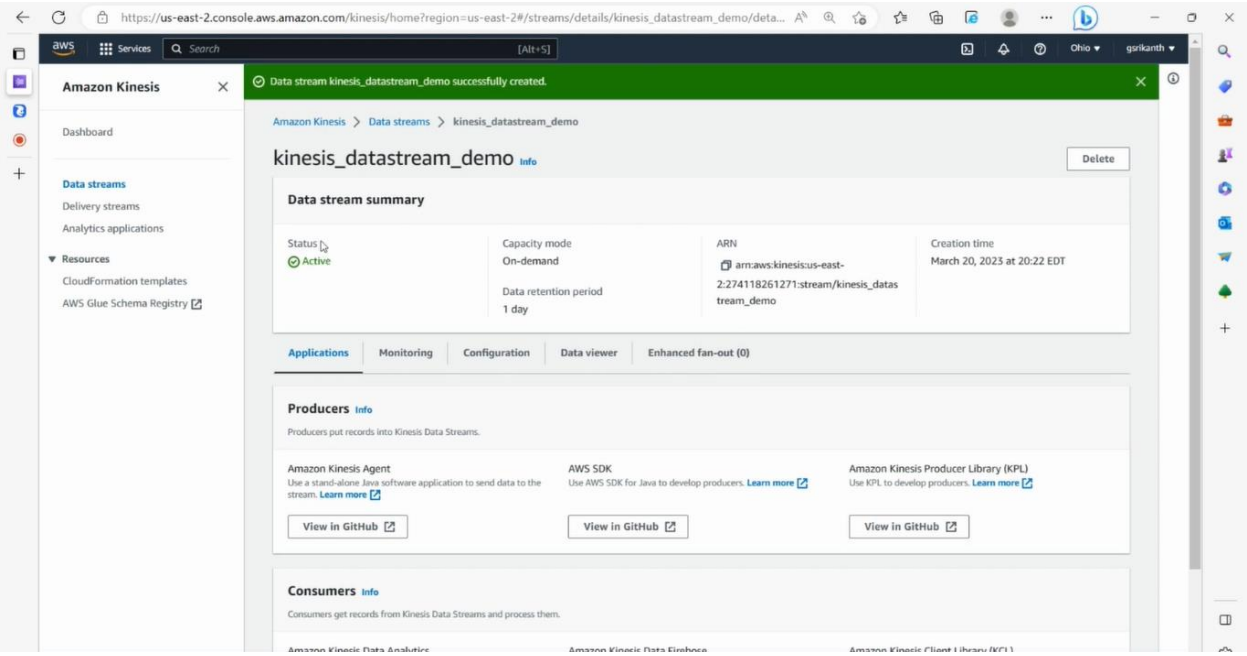
STEP 3: Mention name for data stream and choose data stream capacity.



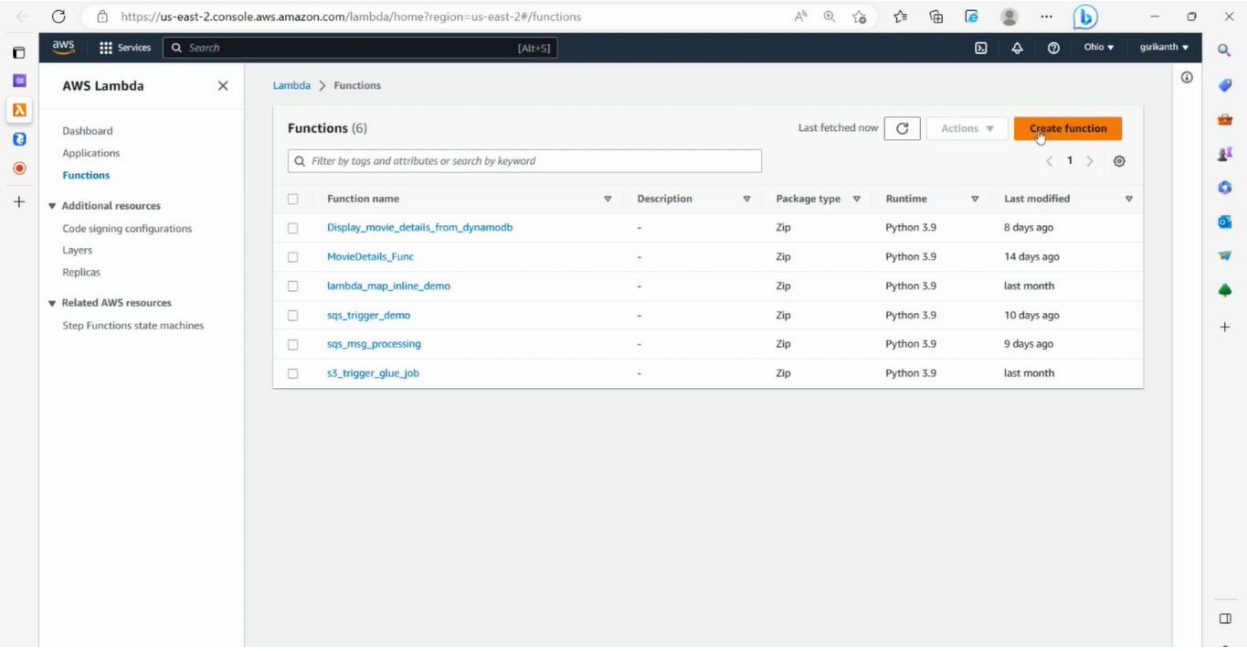
STEP 4: Click “Create data stream”.



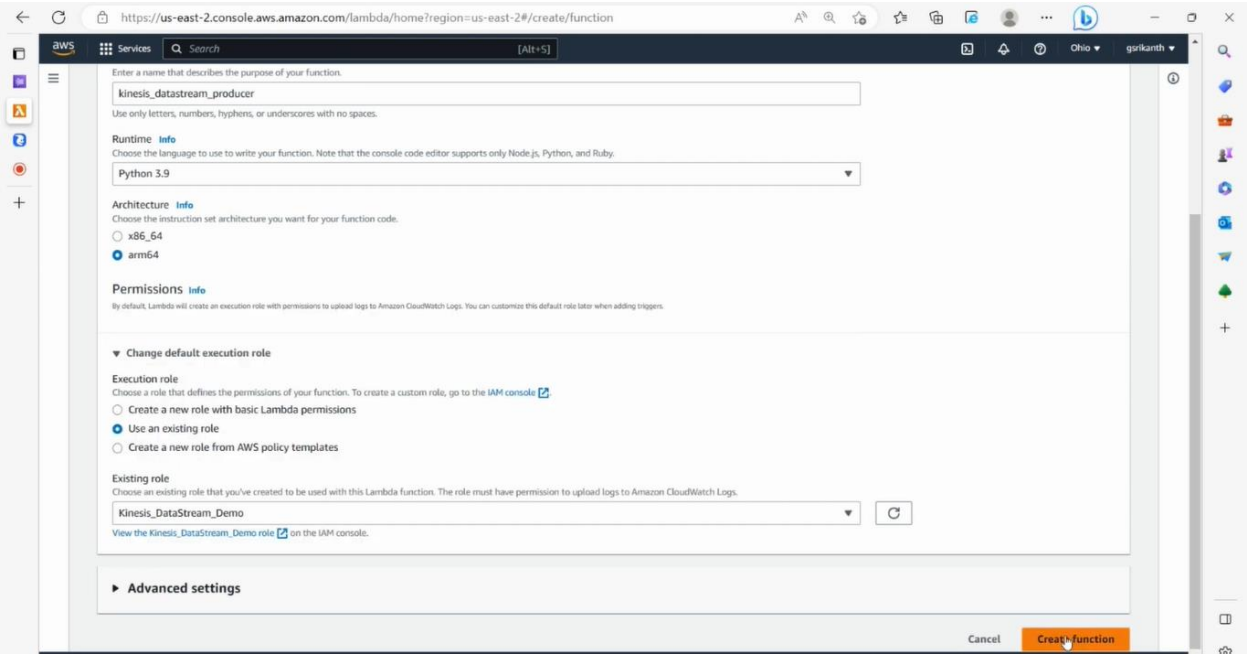
STEP 5: Kinesis data stream created successfully.



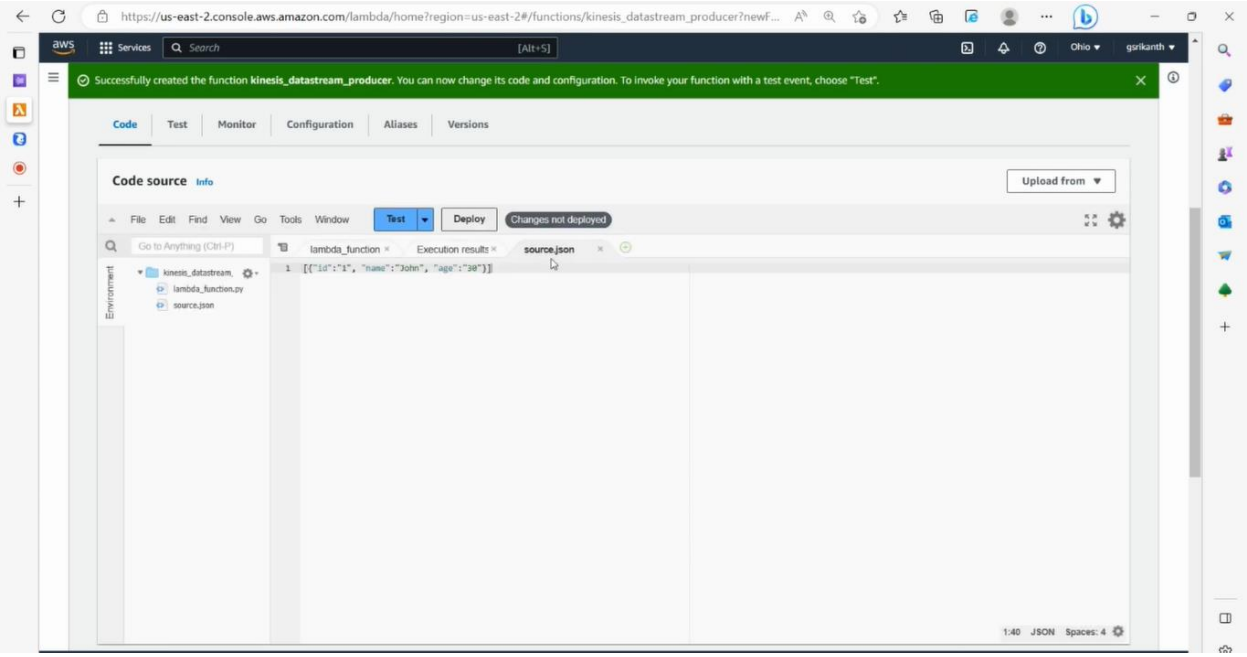
STEP 6: Create lambda function.



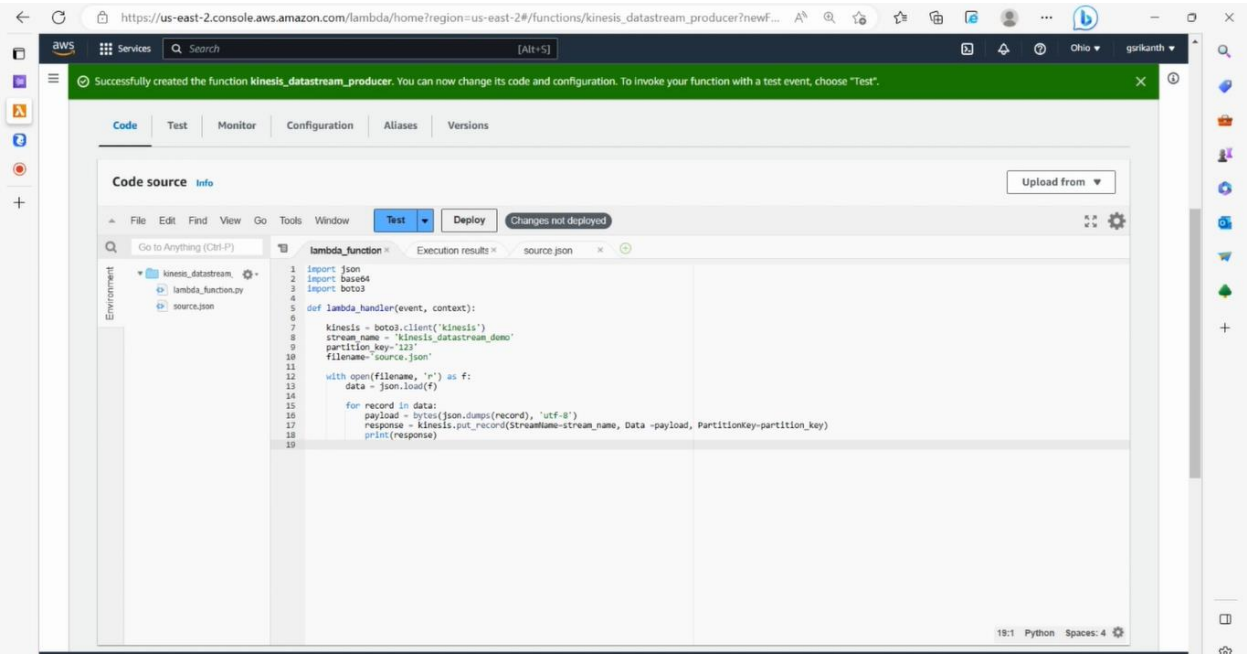
STEP 7: Enter all the necessary details and click “Create Function”.



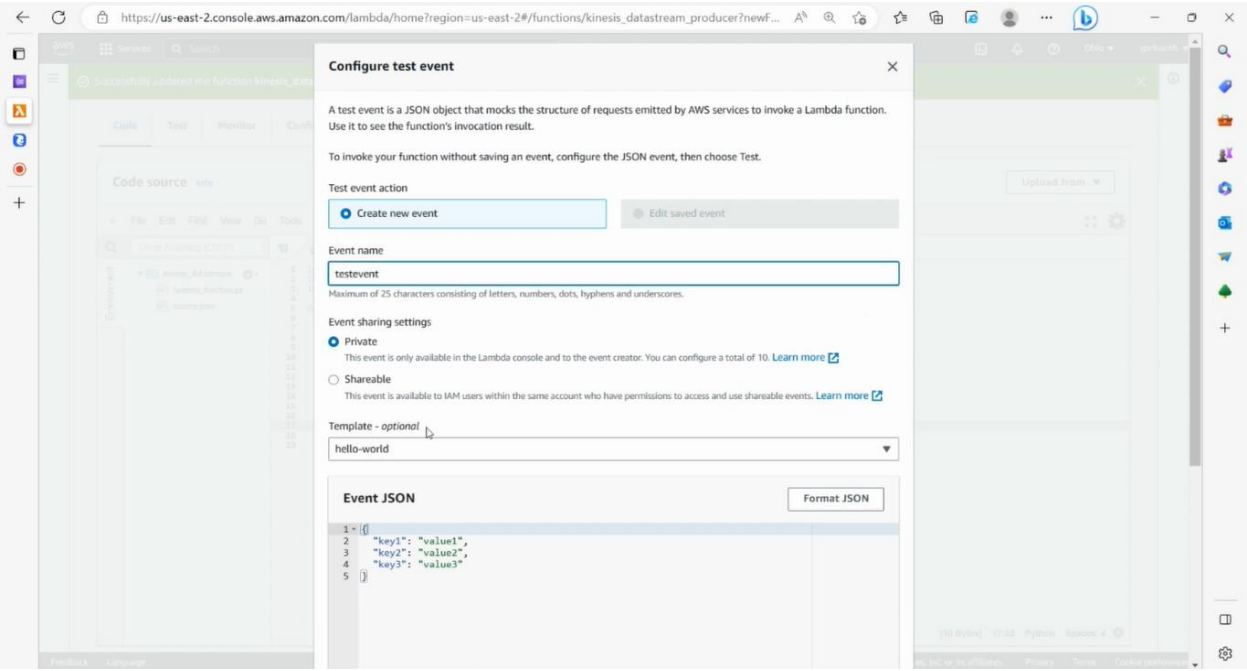
STEP 8: Just created source.json and add particular details.



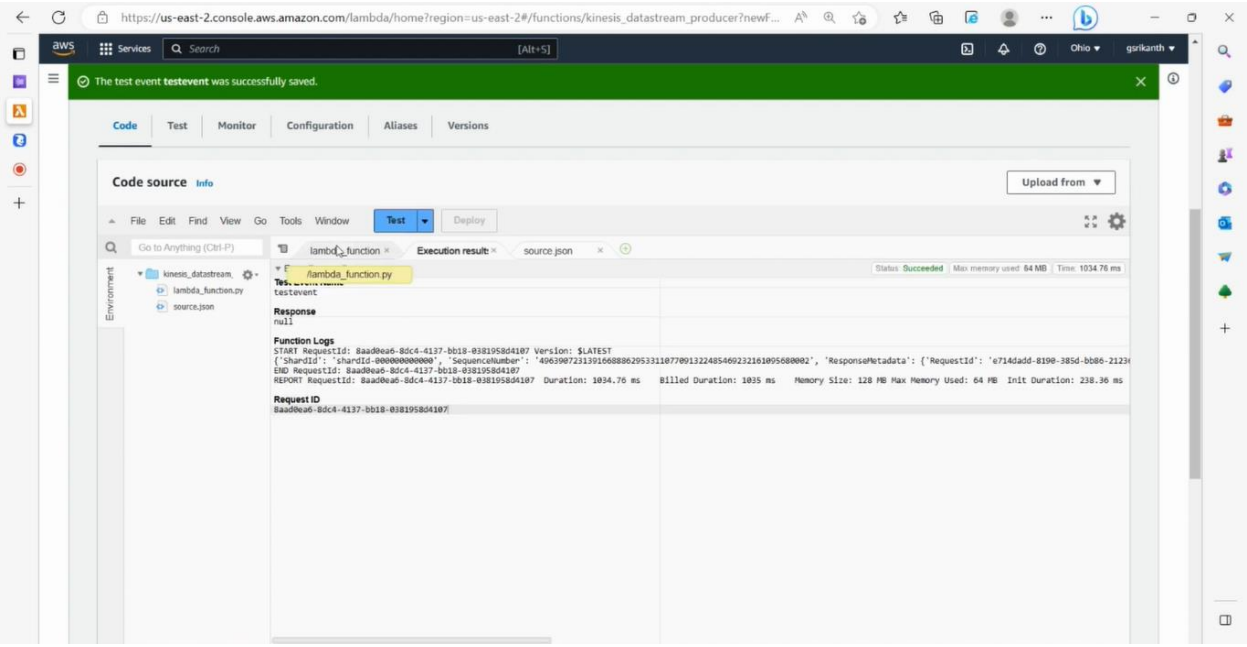
STEP 9: Write a code for sending a data to kinesis data stream.



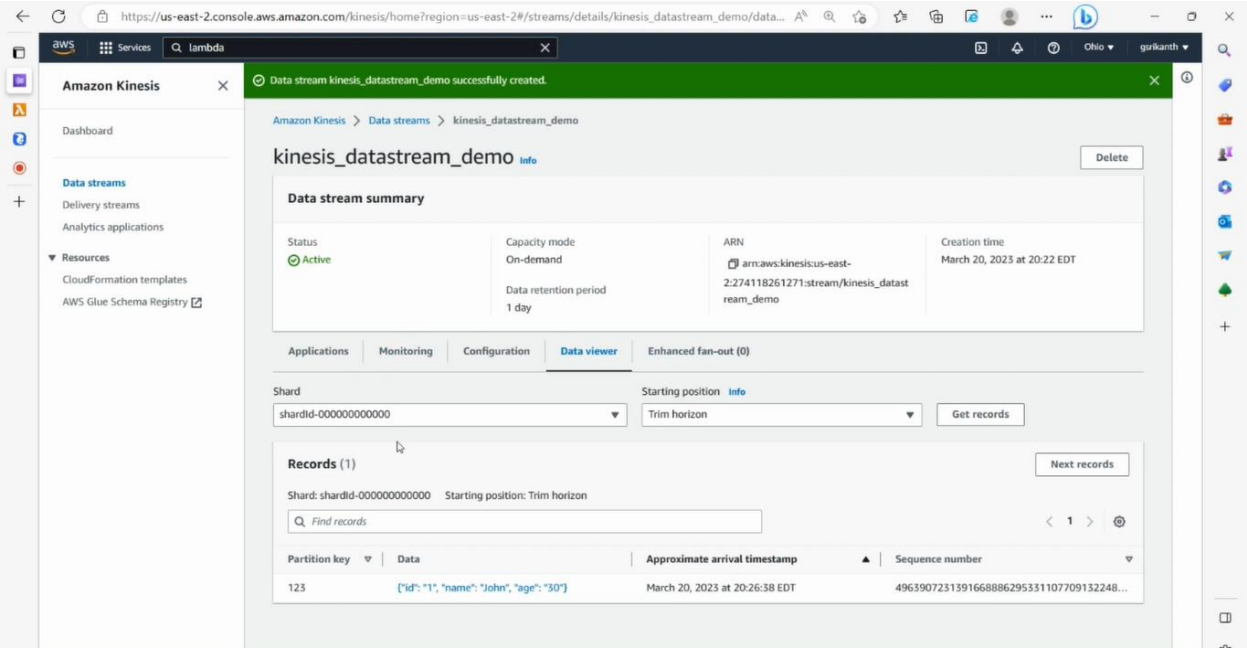
STEP 10: Create test event and save it.



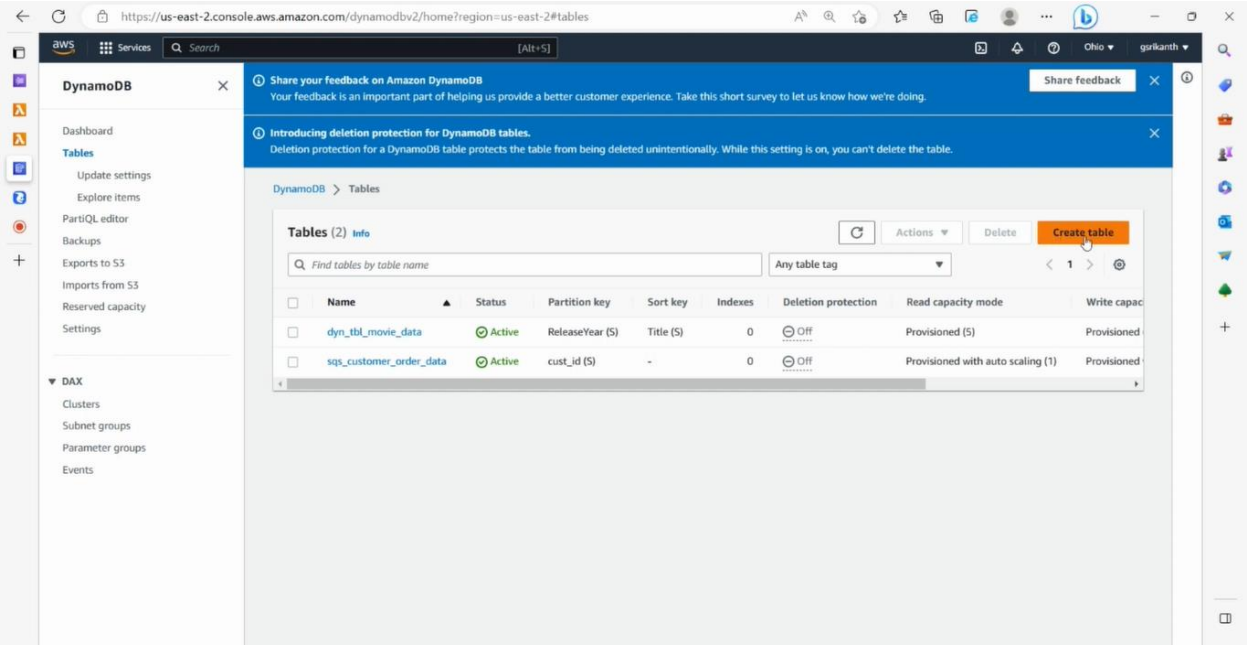
STEP 11: Click “Test” and result will show in execution results.



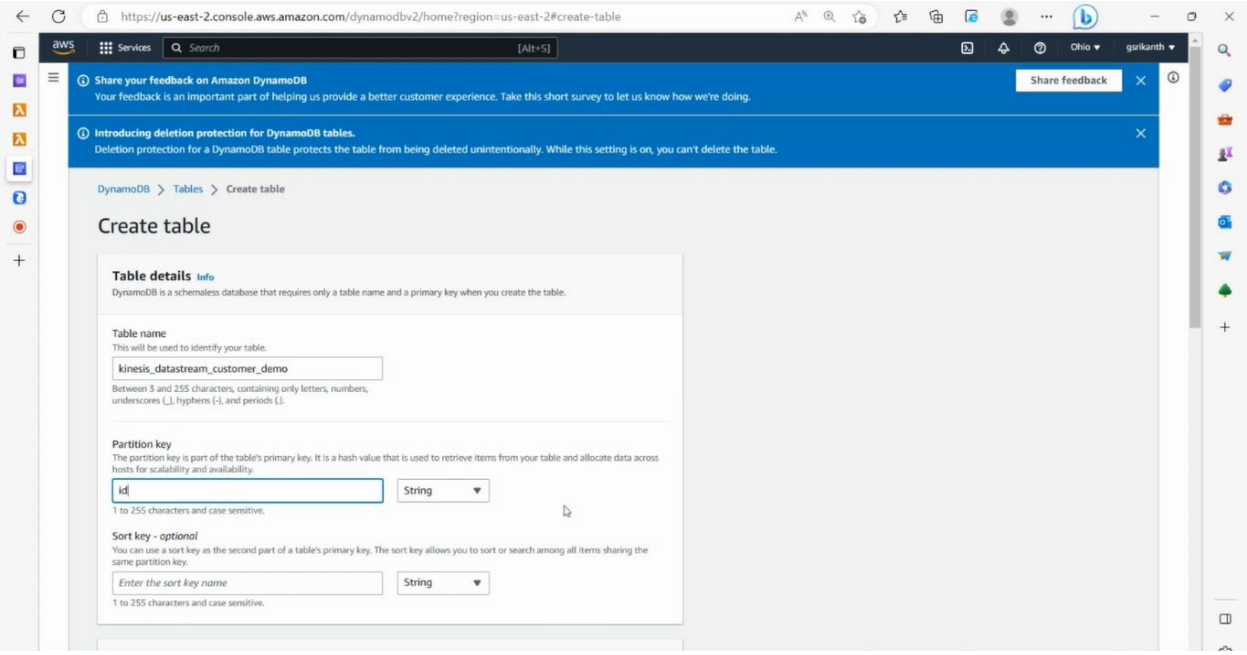
STEP 12: After testing the producer function, open kinesis console. In that, click “Data viewer” and it showing the data is in the kinesis data stream.



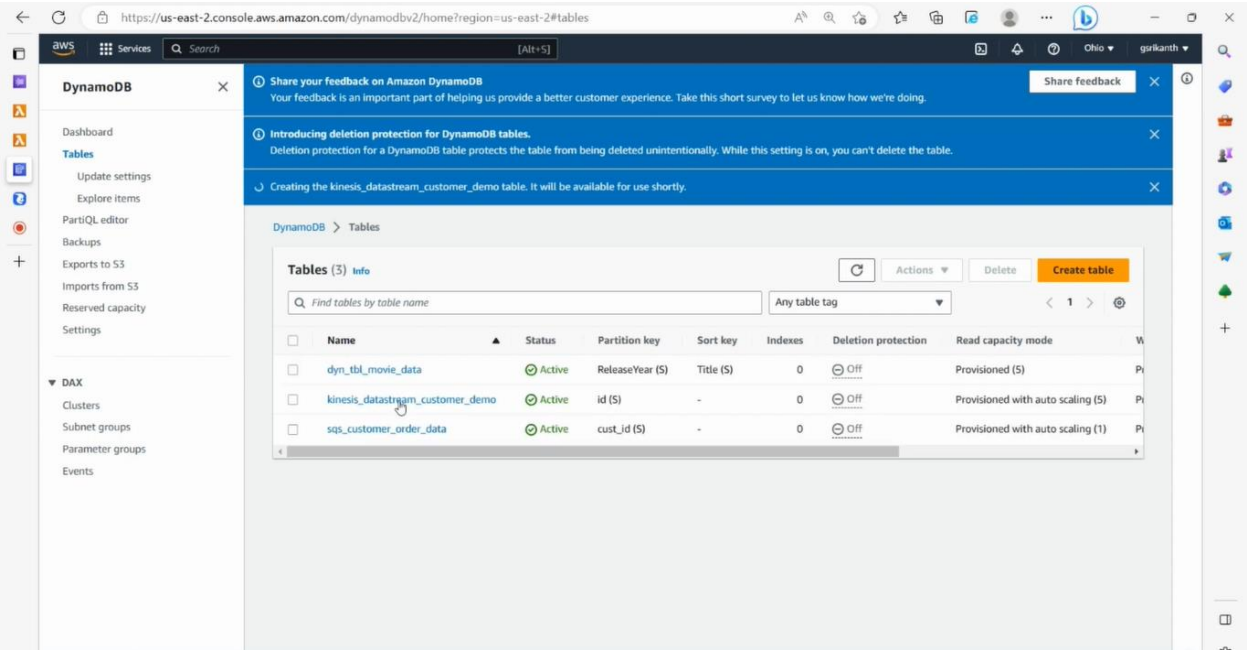
STEP 13: Navigate to Dynamo DB console and click “Create table”.



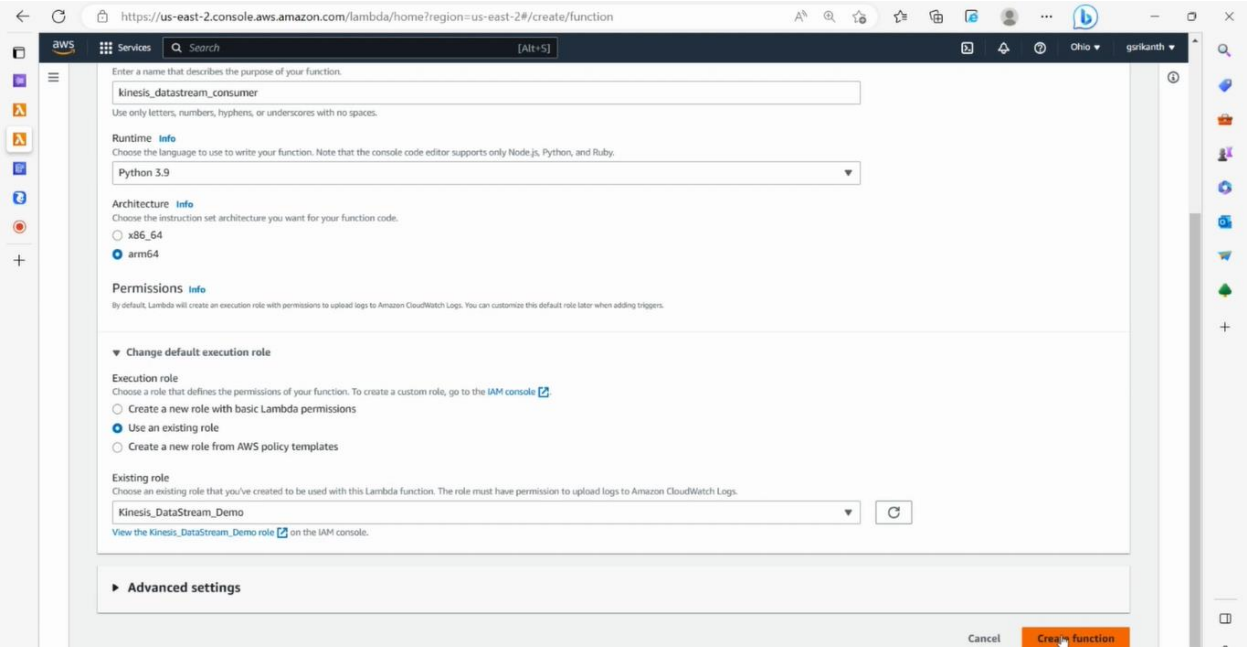
STEP 14: Enter table name and partition key and click create table.



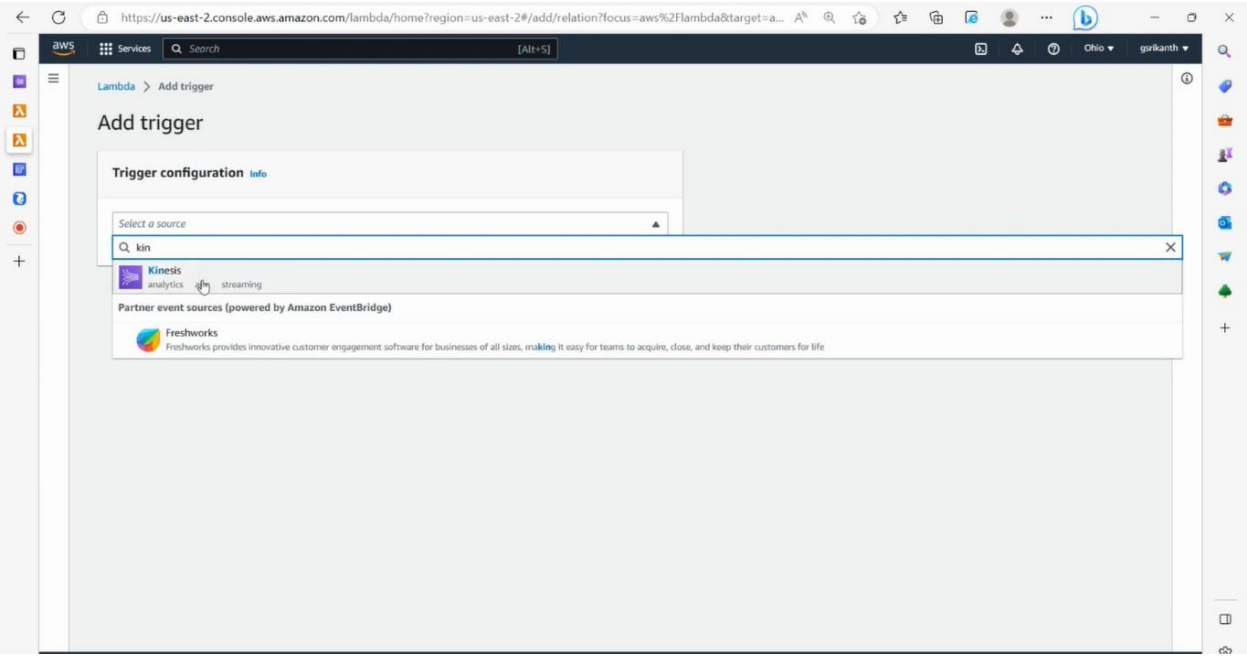
STEP 15: Table created successfully.



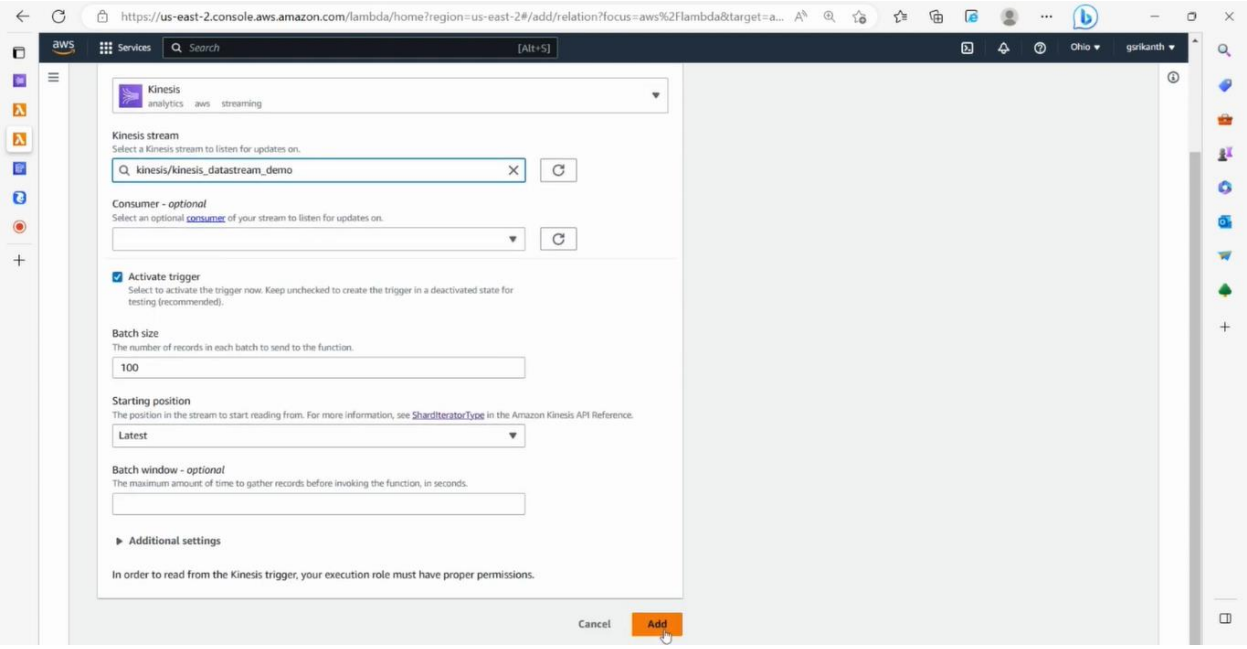
STEP 16: Now create another lambda function as consumer and fill the necessary details here.



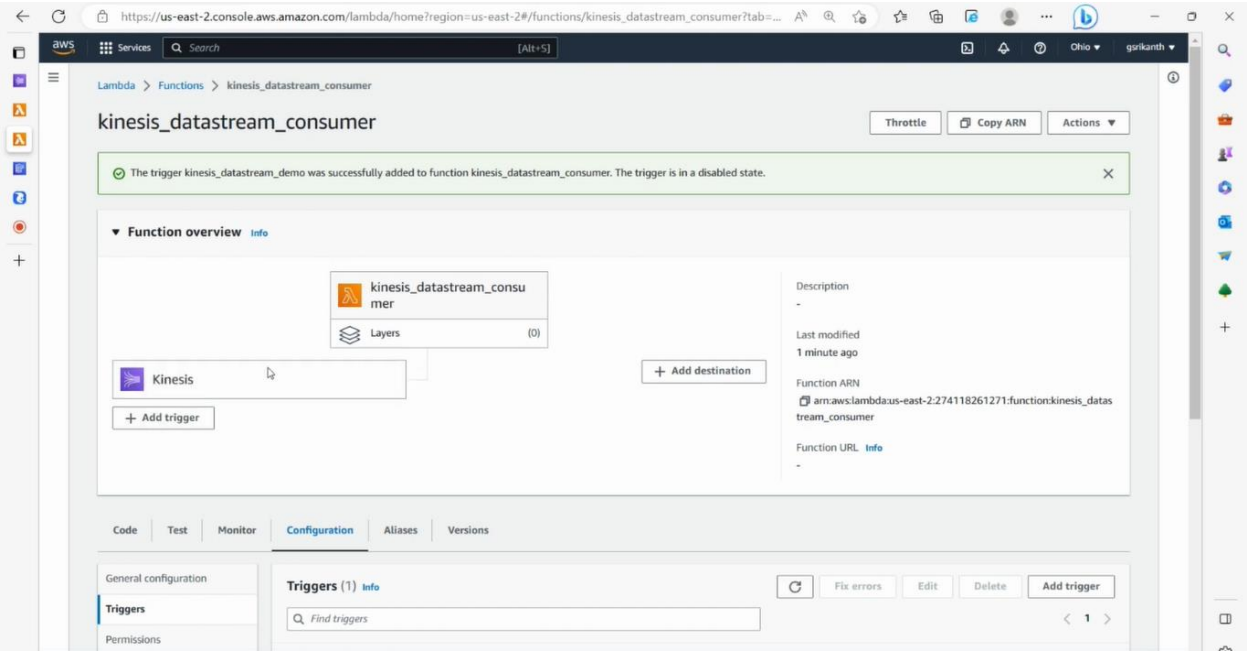
STEP 17: In this consumer lambda, add a trigger (Kinesis).



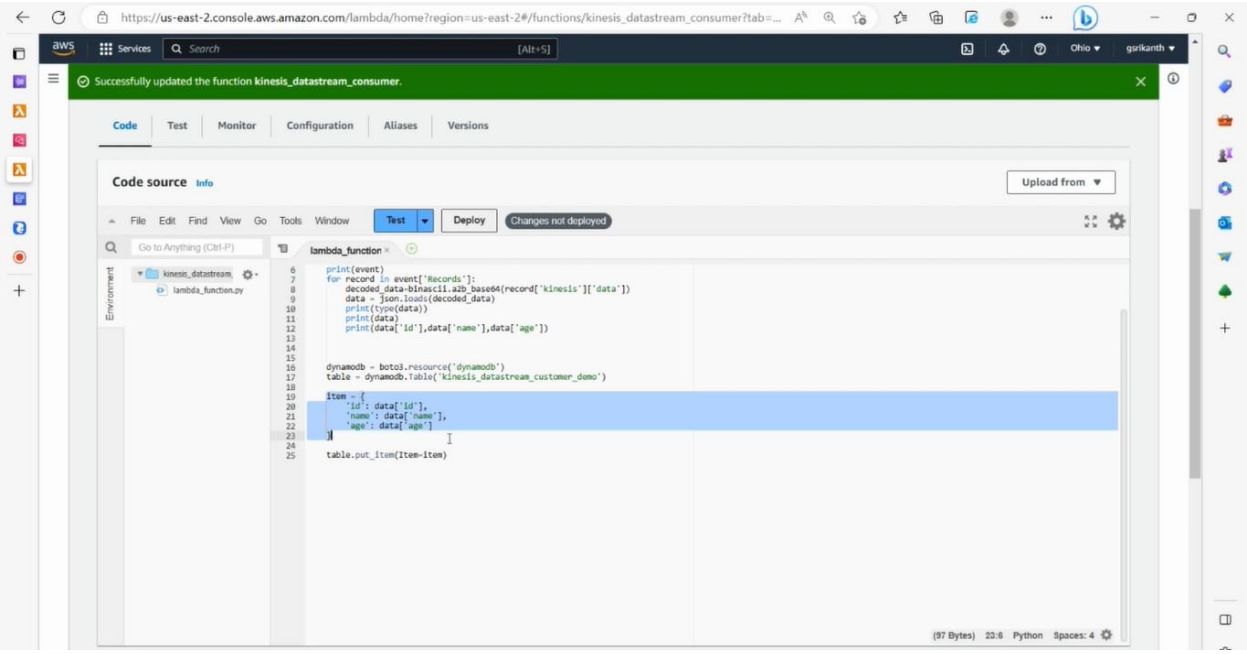
STEP 18: Choose Kinesis stream (already created) and click add.



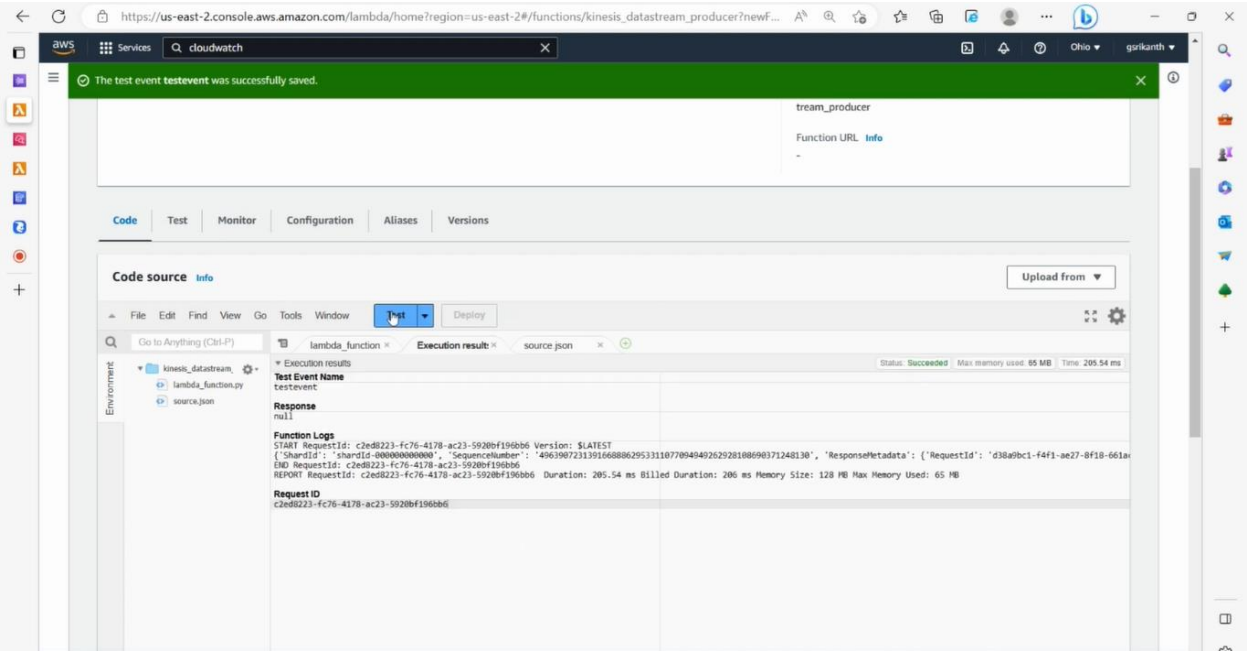
STEP 19: Trigger added successfully.



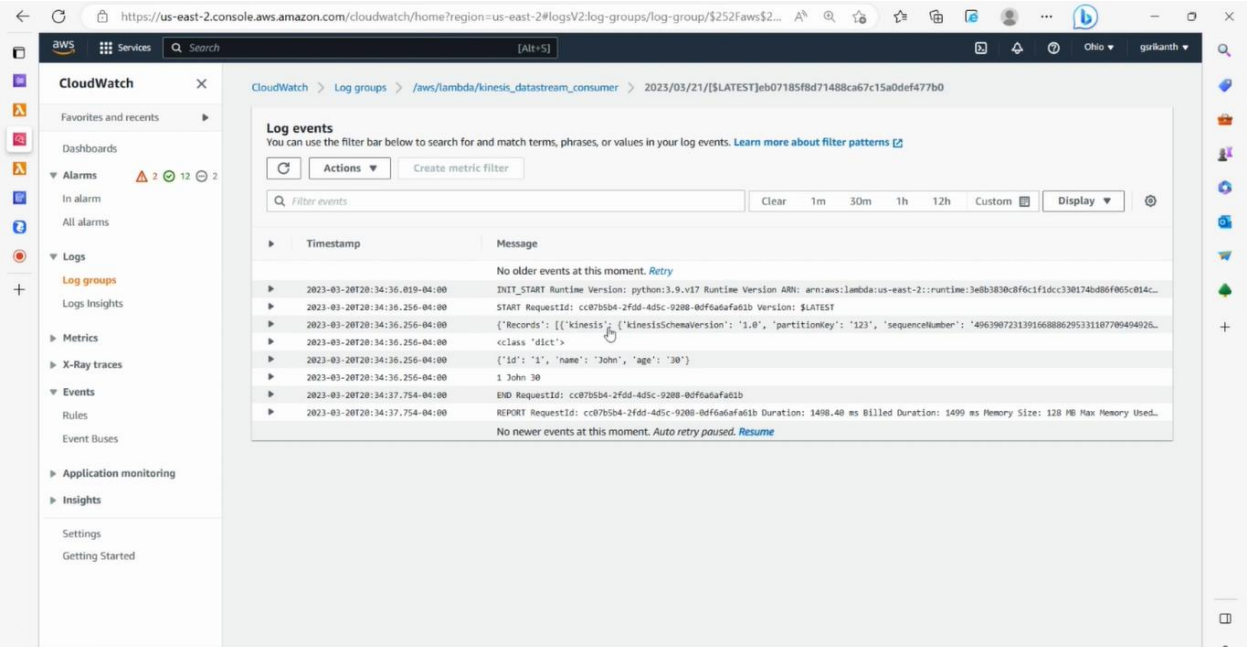
STEP 20: Write a code to push data in dynamo DB.



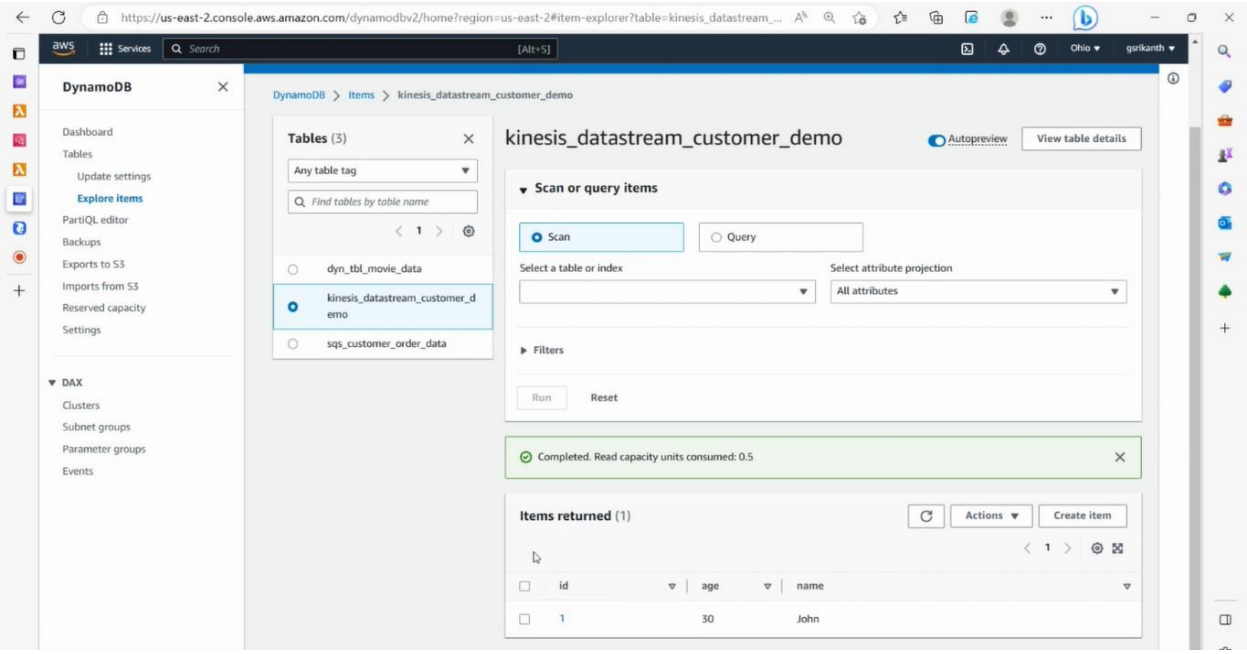
STEP 21: Now go to producer lambda function and click test.



STEP 22: After testing, open cloud watch console, it shows the data here.



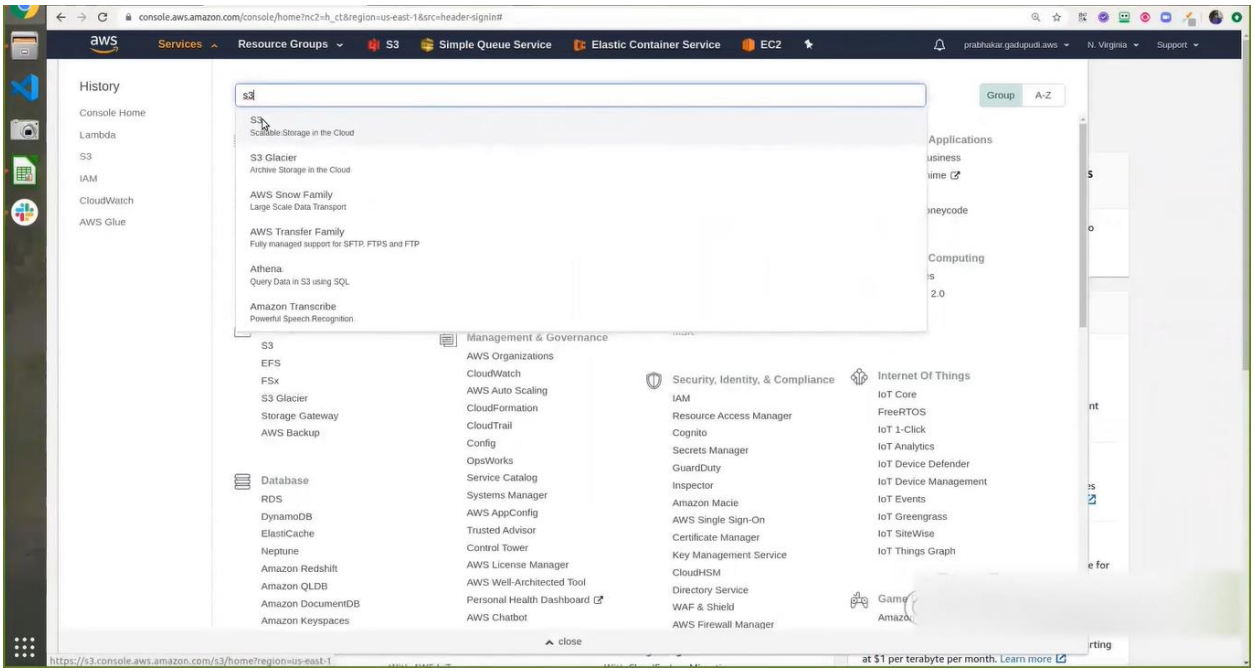
STEP 23: Open Dynamo DB table, the data stored successfully in the table.



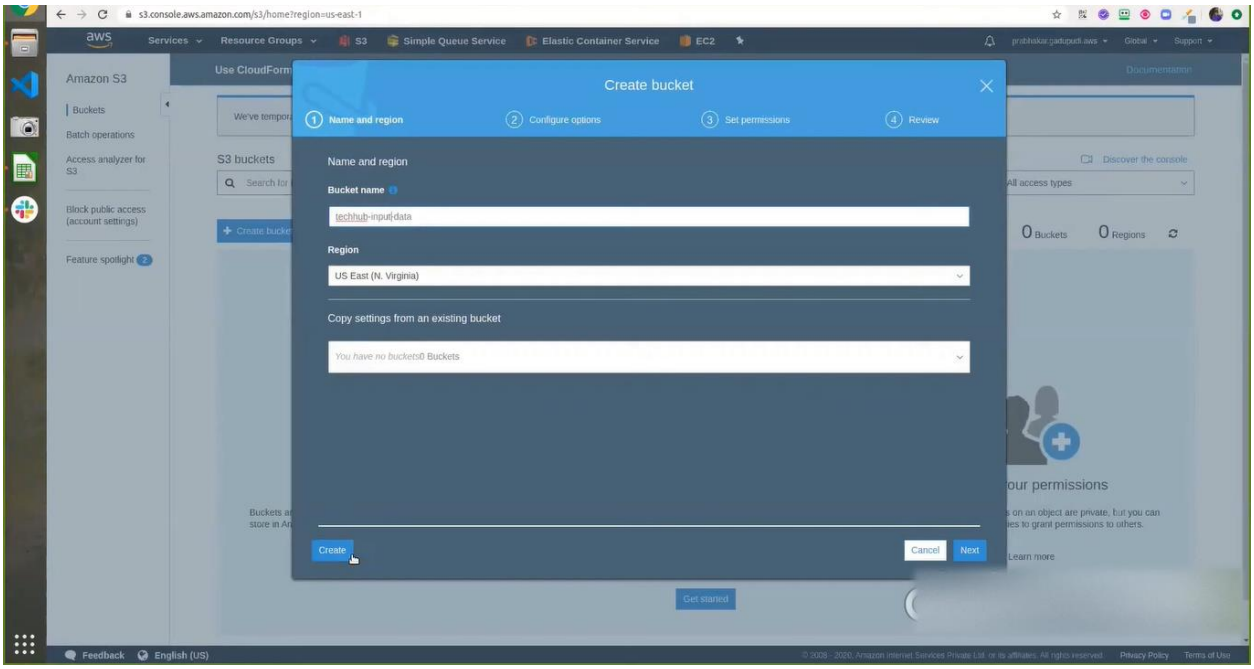
COPY FILES FROM ONE S3 BUCKET TO ANOTHER S3 BUCKET

INTEGRATING LAMBA WITH S3

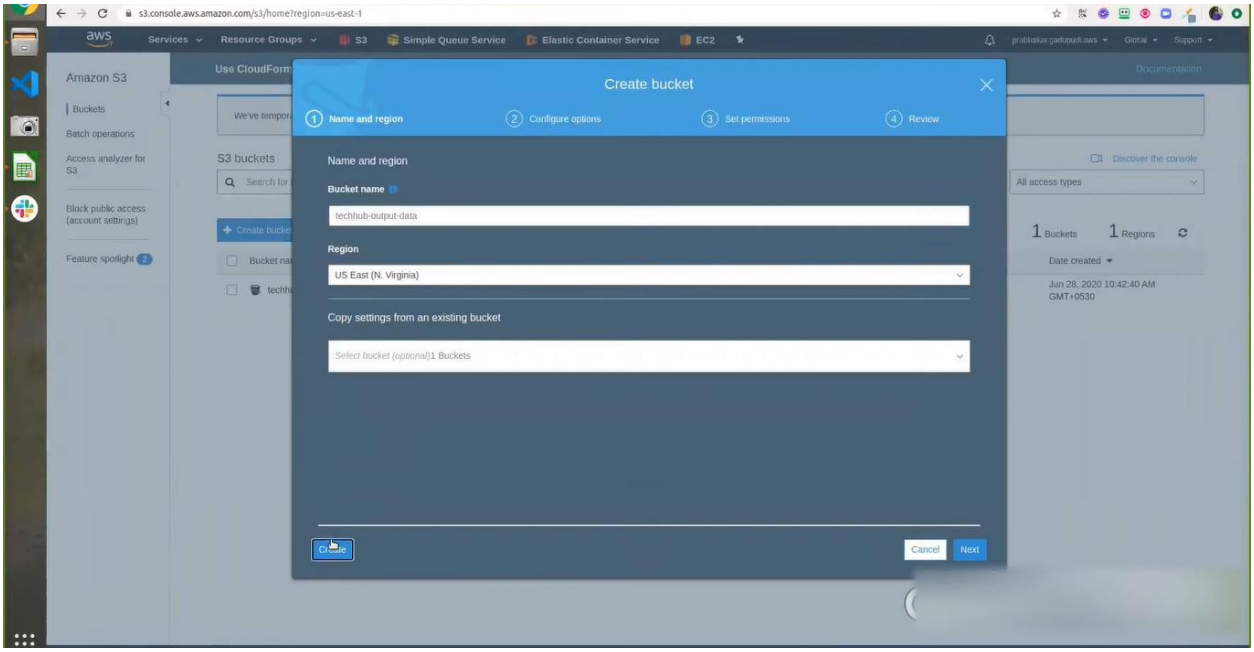
STEP 1: Choose S3 service.



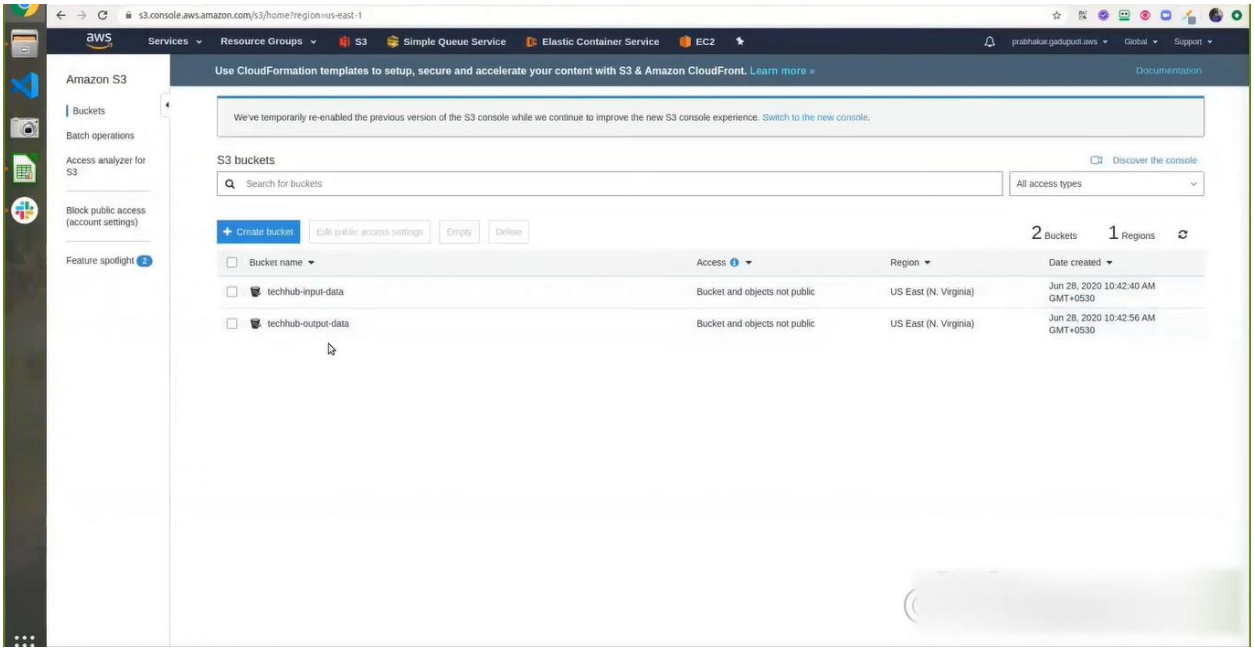
STEP 2: Create first bucket as “input-data”.



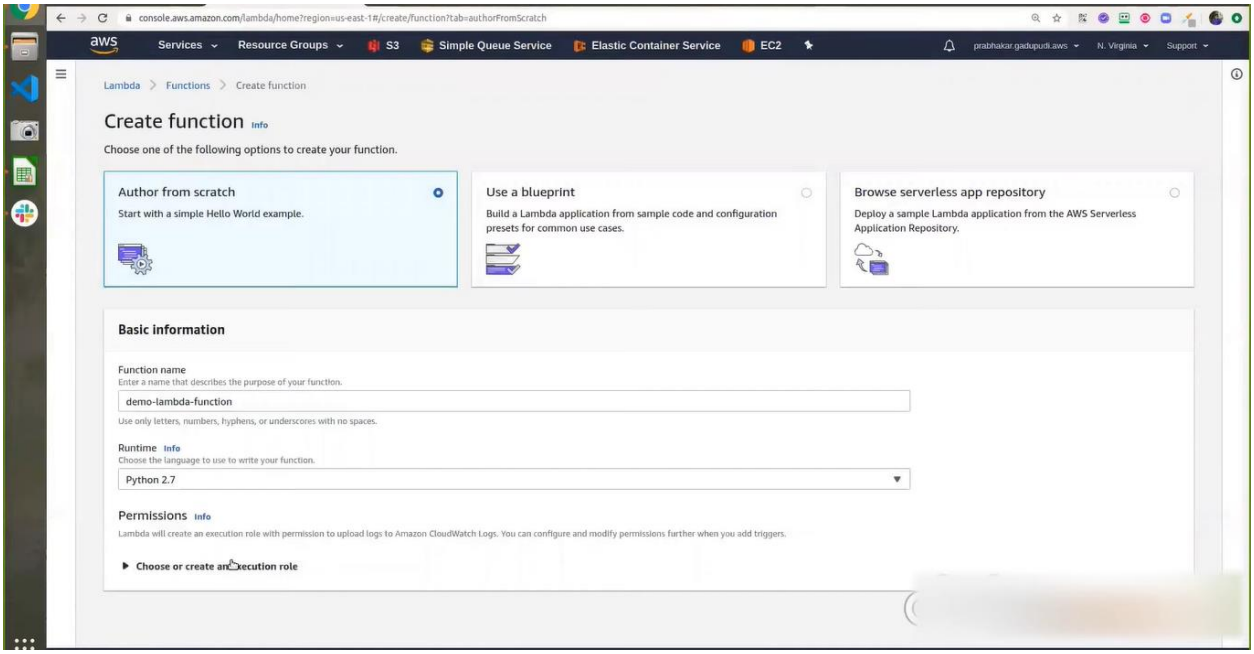
STEP 3: Create another bucket as “output-data”.



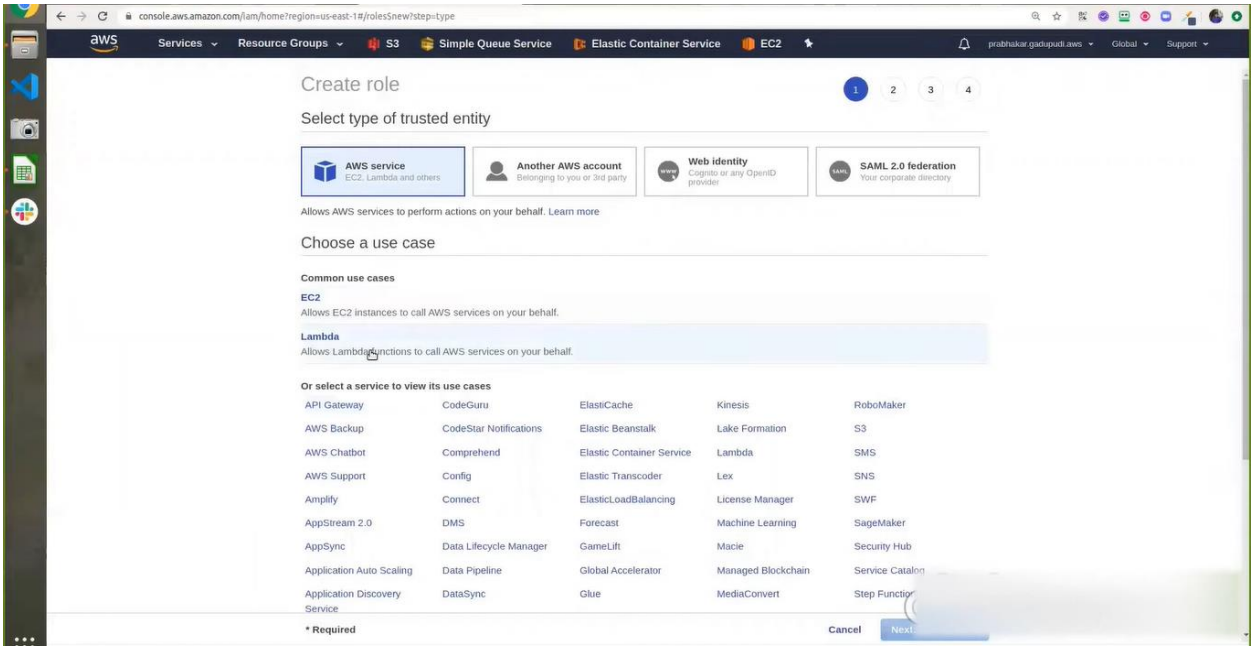
STEP 4: Two buckets created successfully.



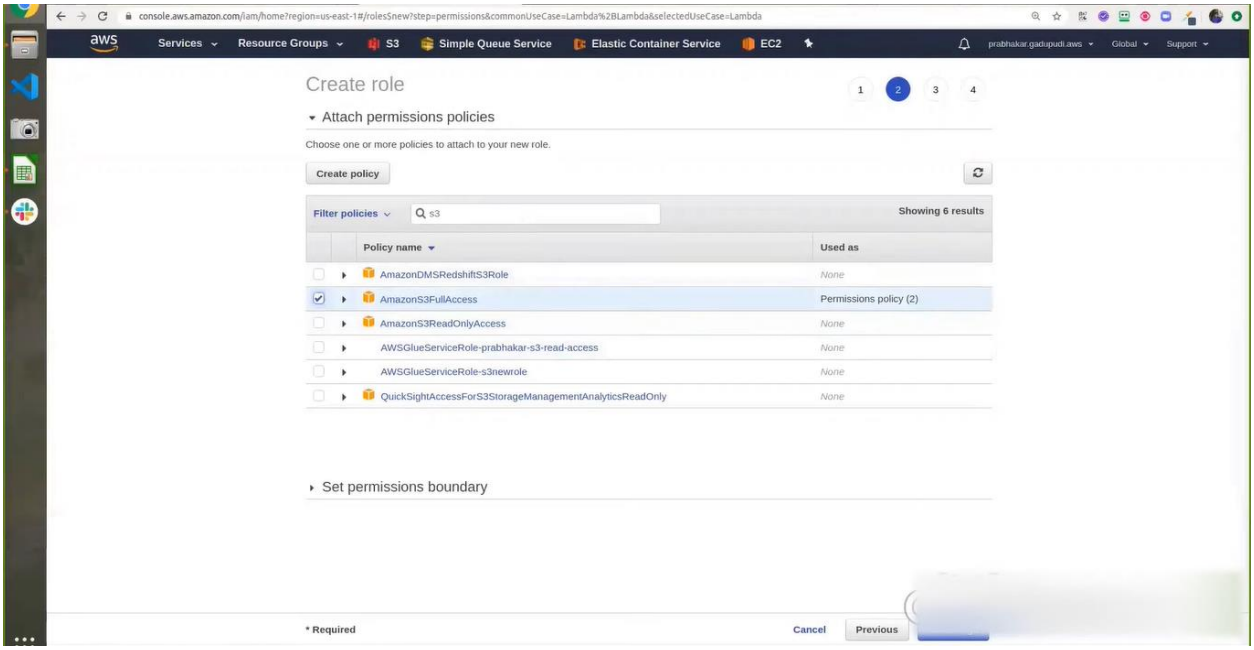
STEP 5: Create lambda function as “demo-lambda-function”.



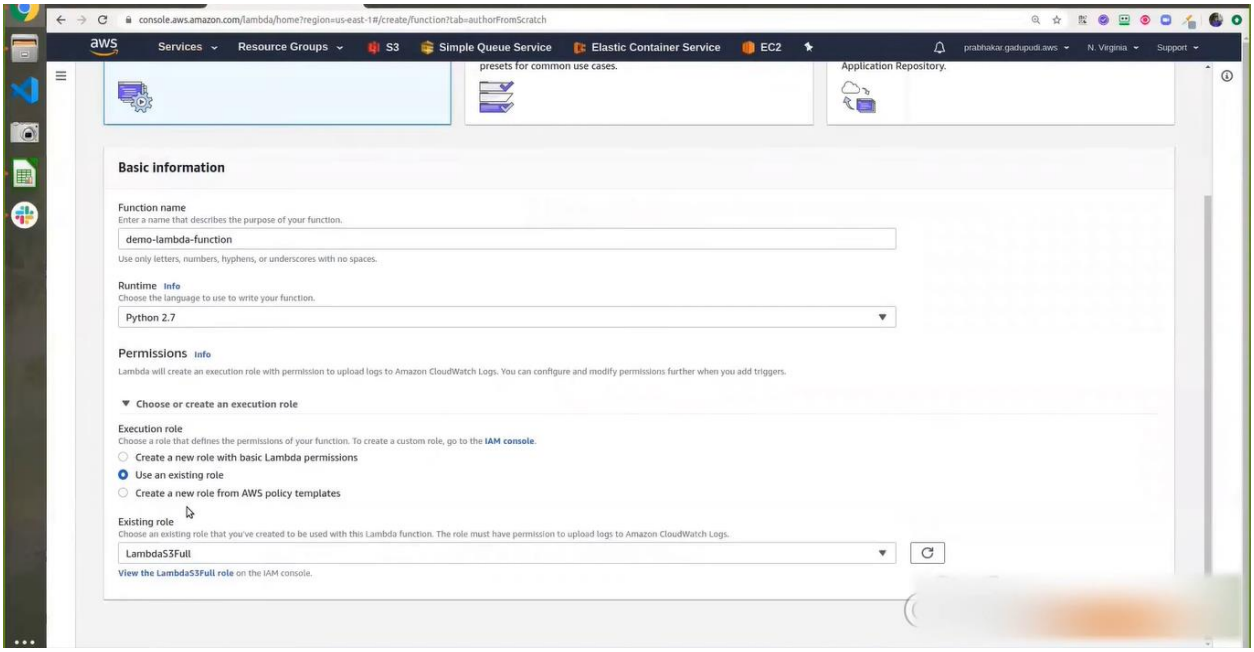
STEP 6: Choose role for lambda function.



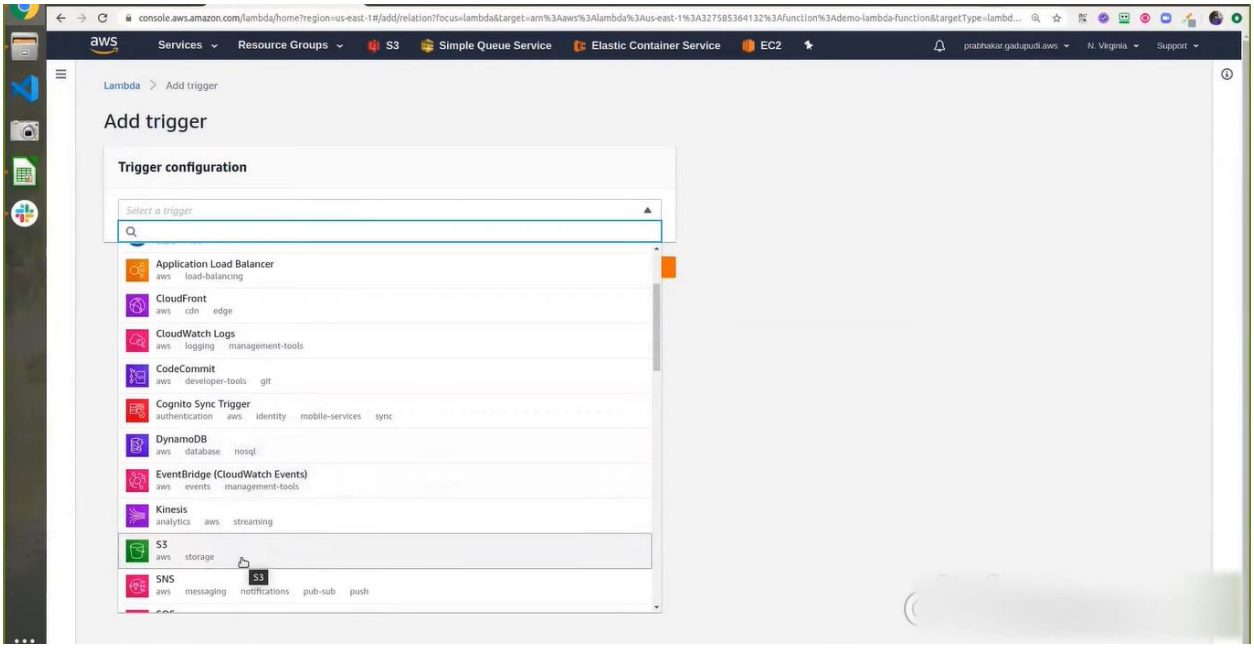
STEP 7: Choose “AmazonS3FullAccess” and create the role.



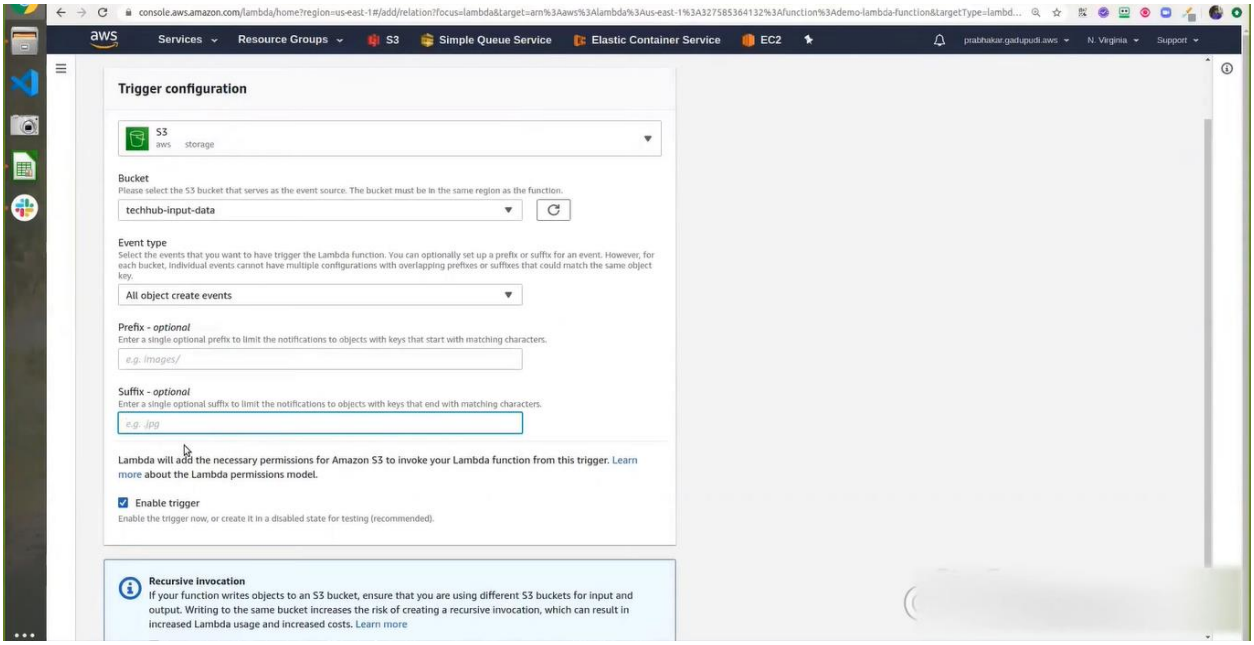
STEP 8: Enter the required details for creating lambda function and click create function.



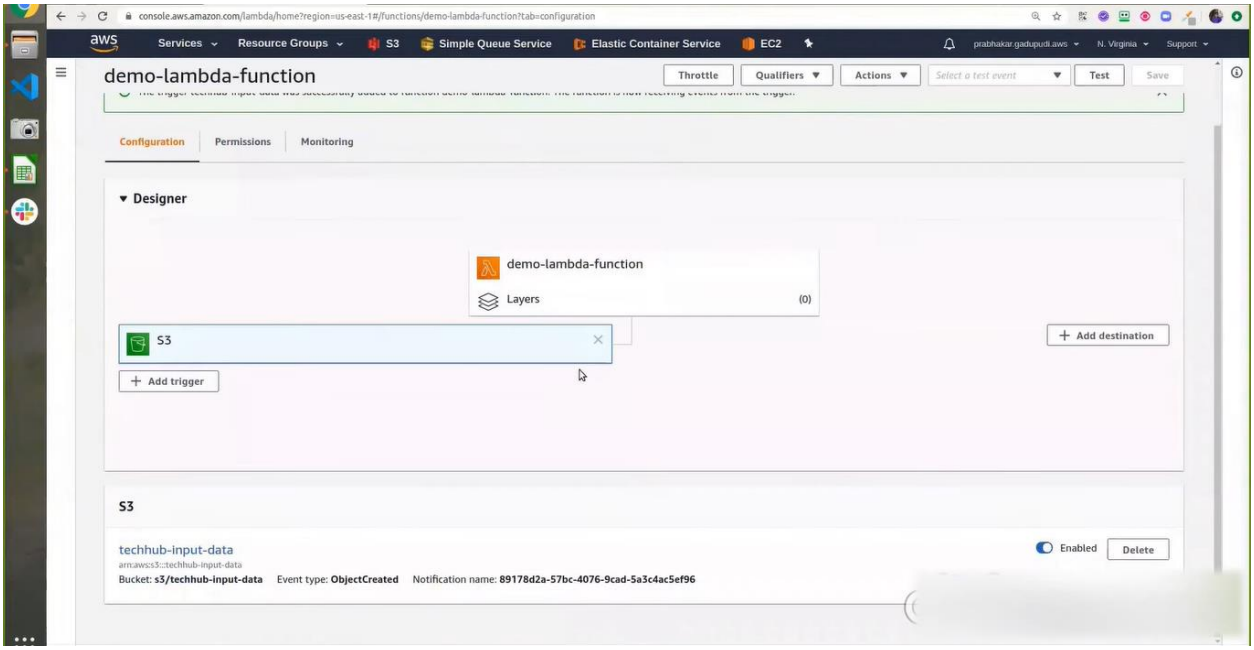
STEP 9: After creating lambda function, click “Add trigger” and choose S3.



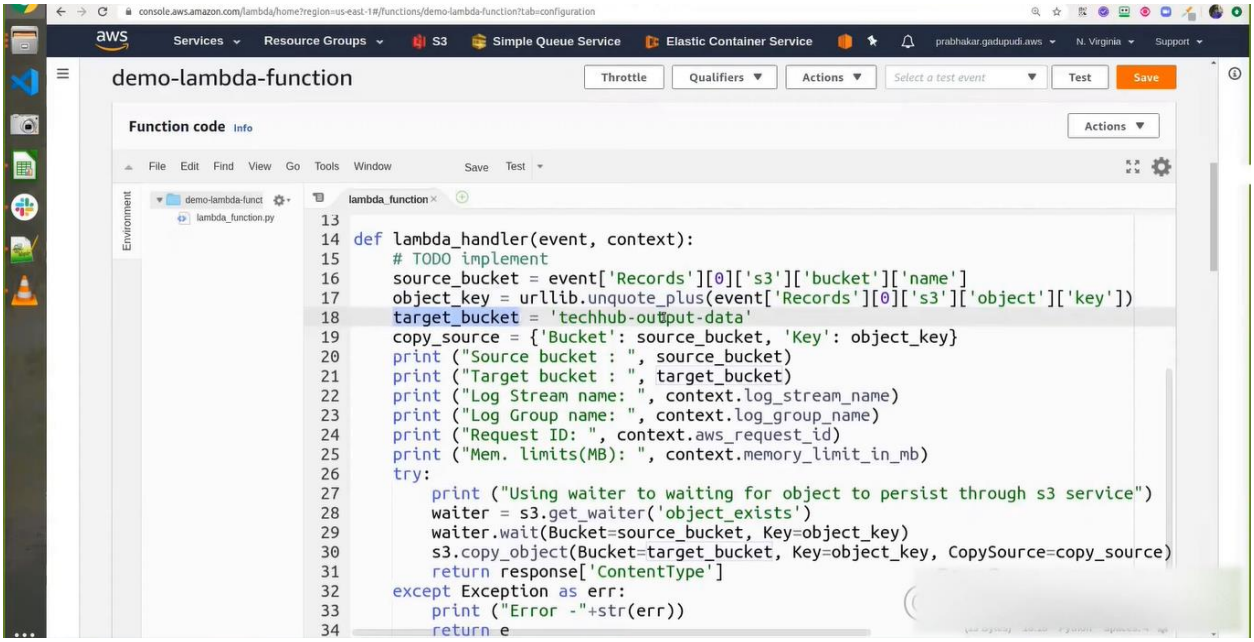
STEP 10: Choose bucket as “input-data” and fill the required details.



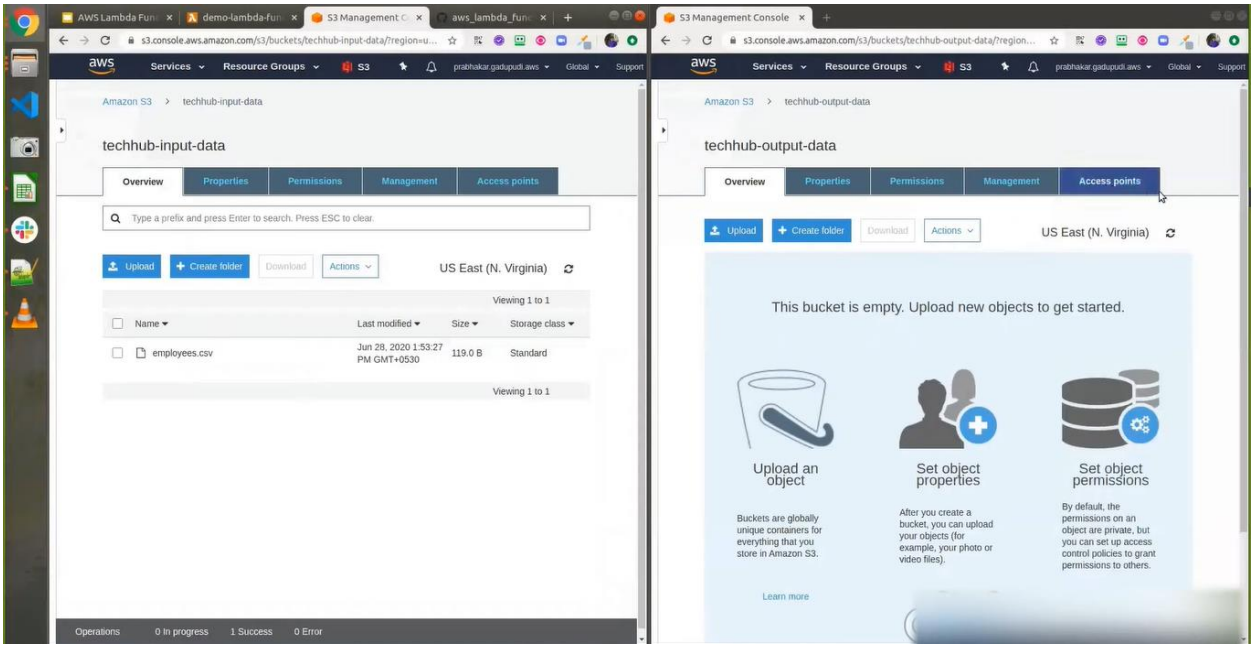
STEP 11: Trigger added successfully.



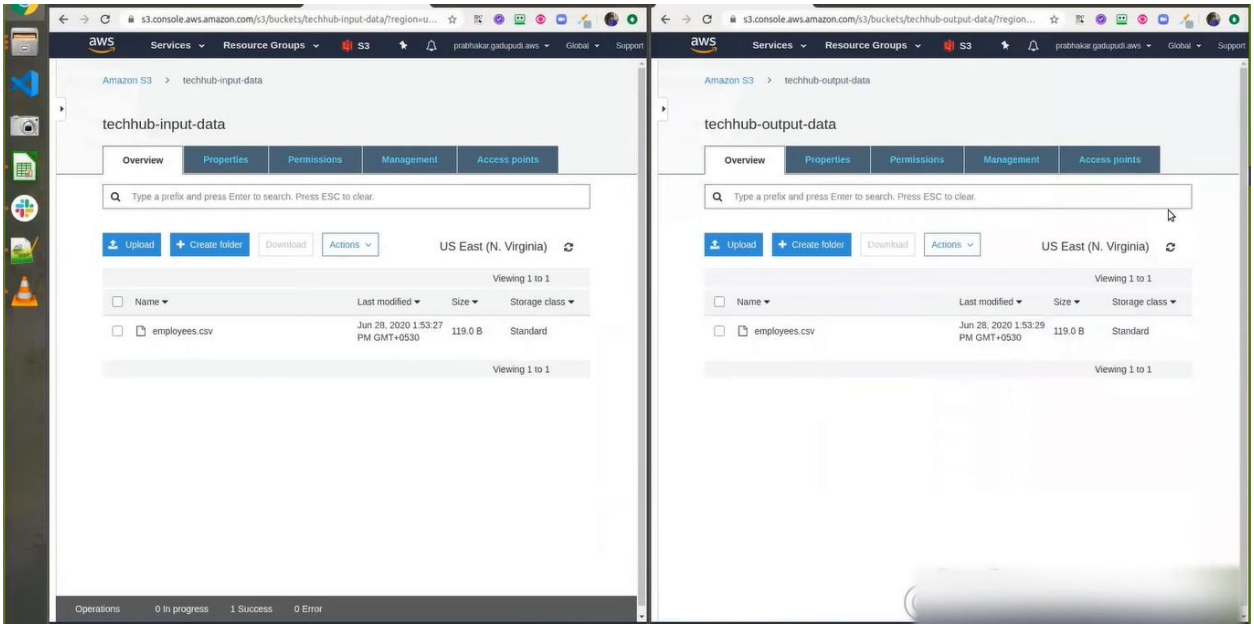
STEP 12: Write a code for copying a file from input-data bucket to output-data bucket.



STEP 13: Upload a file in input-data bucket as employees.csv.



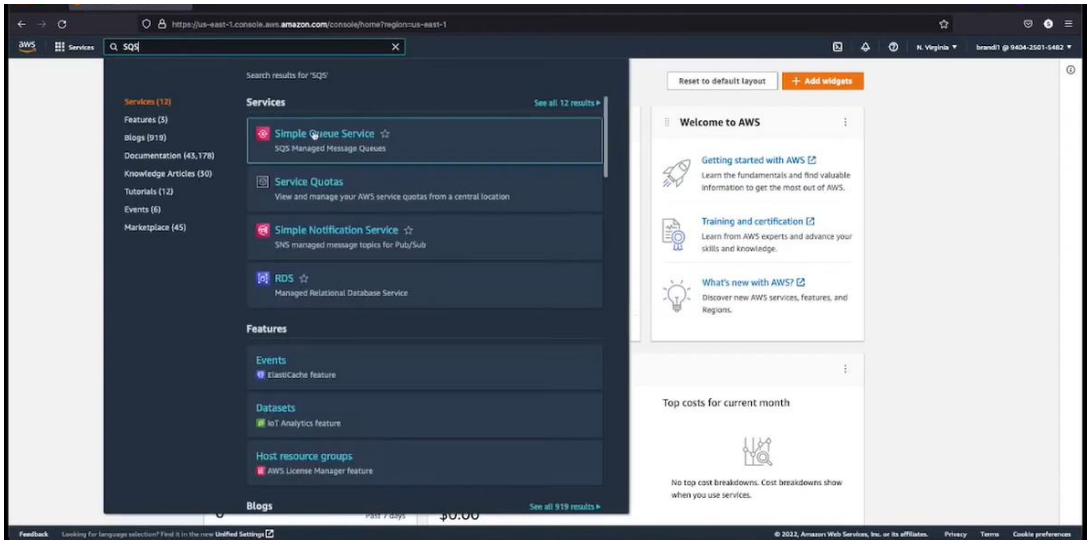
STEP 14: Open the output-data bucket and refresh. A copy of the same file is available in this bucket.



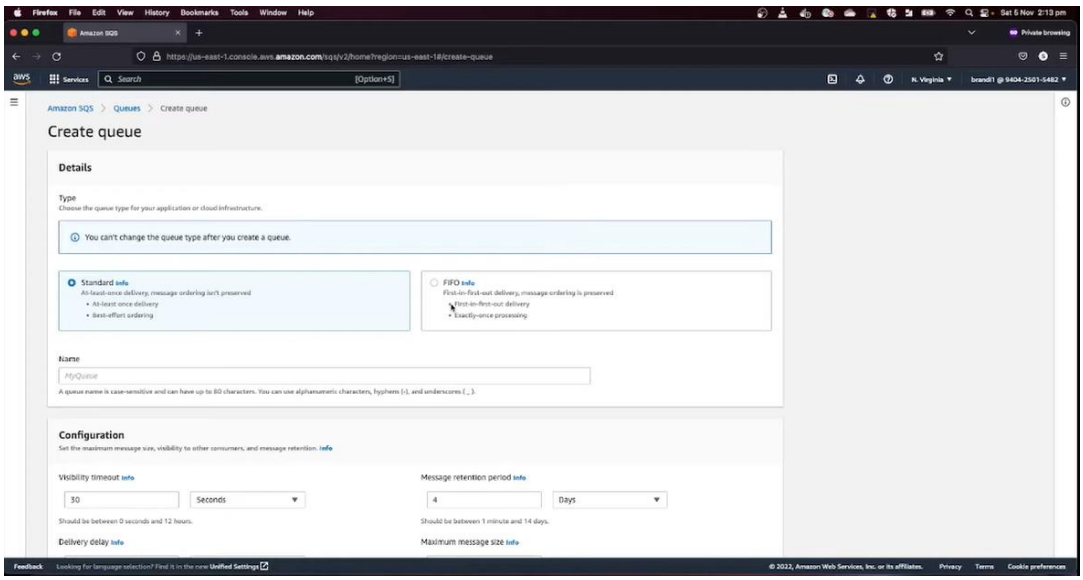
REAL TIME DATA PROCESSING WITH SQS, LAMBDA AND DYNAMODB

INTEGRATING LAMBDA WITH SQS

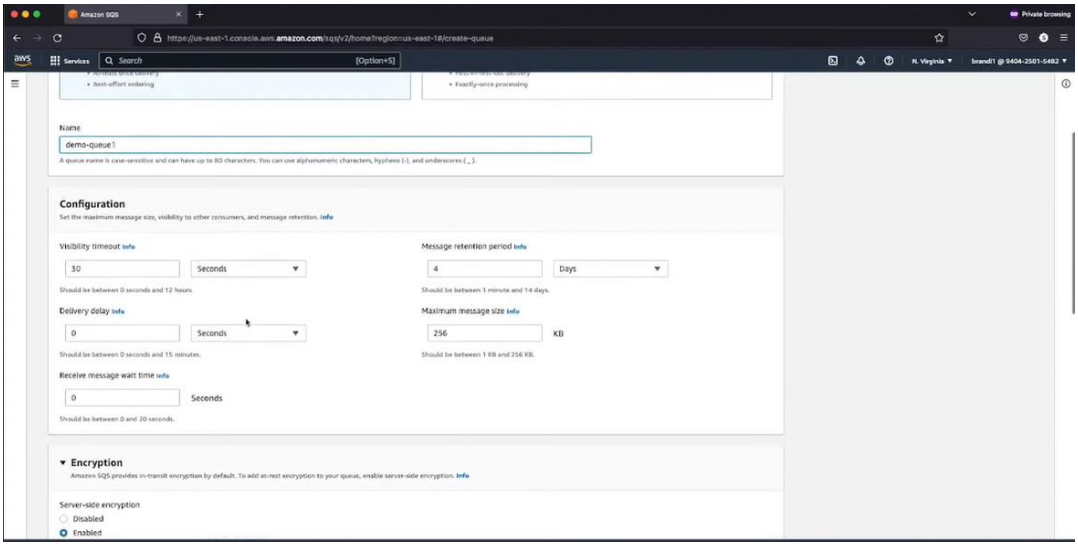
STEP 1: Choose SQS service and click.



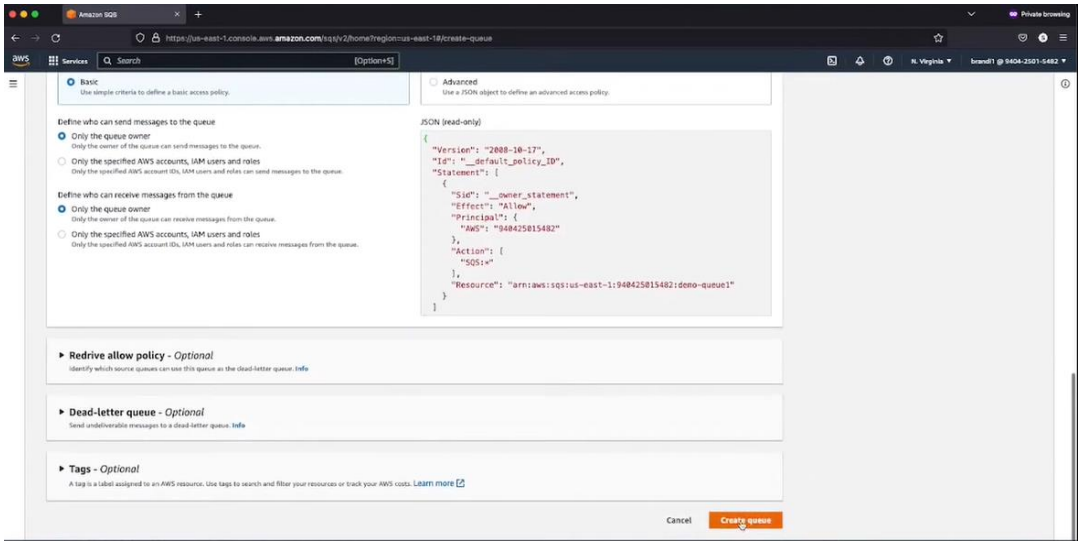
STEP 2: Choose a type and enter the necessary details like name.



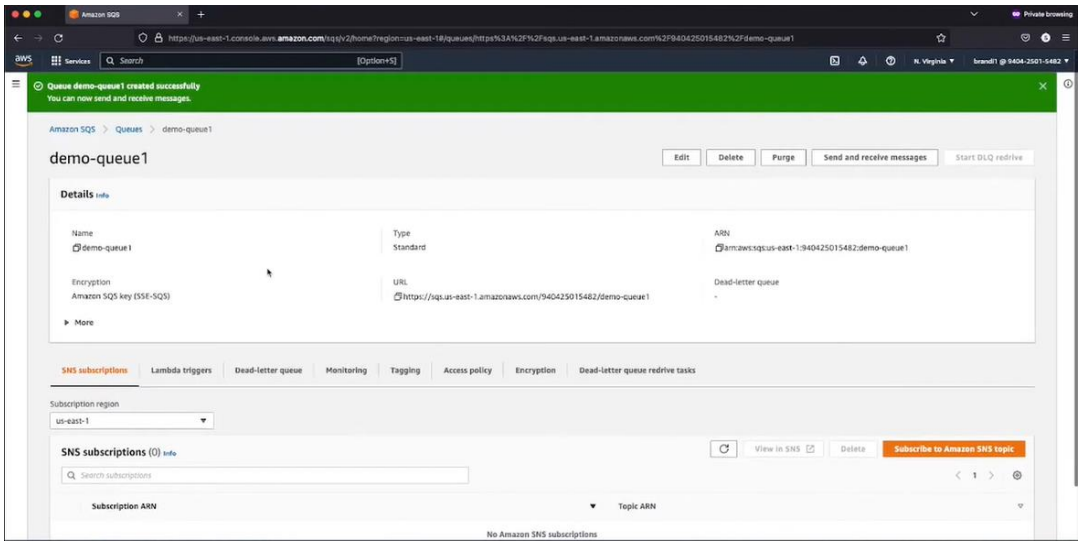
STEP 3: Choose the configuration details.



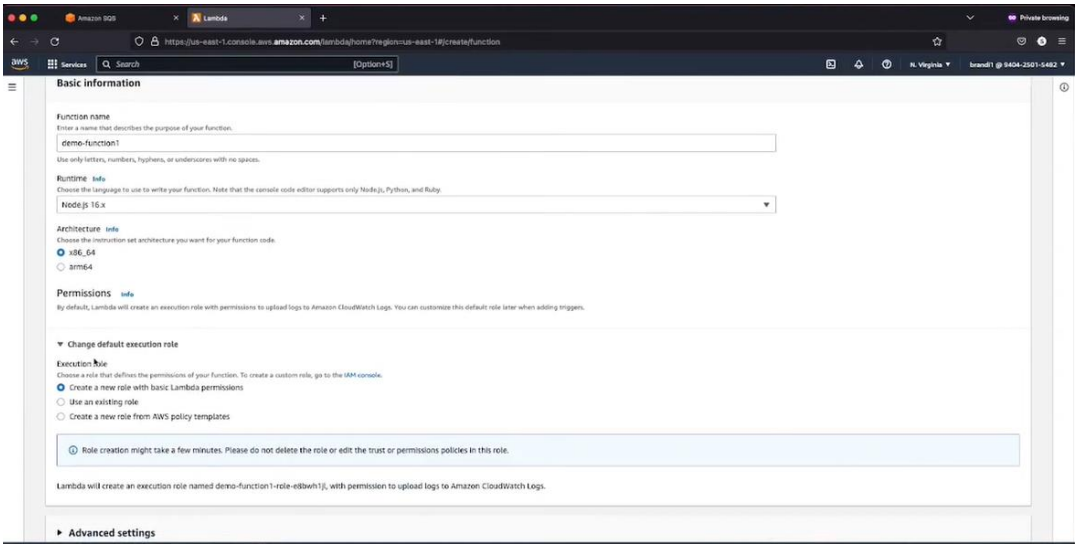
STEP 4: Click create queue.



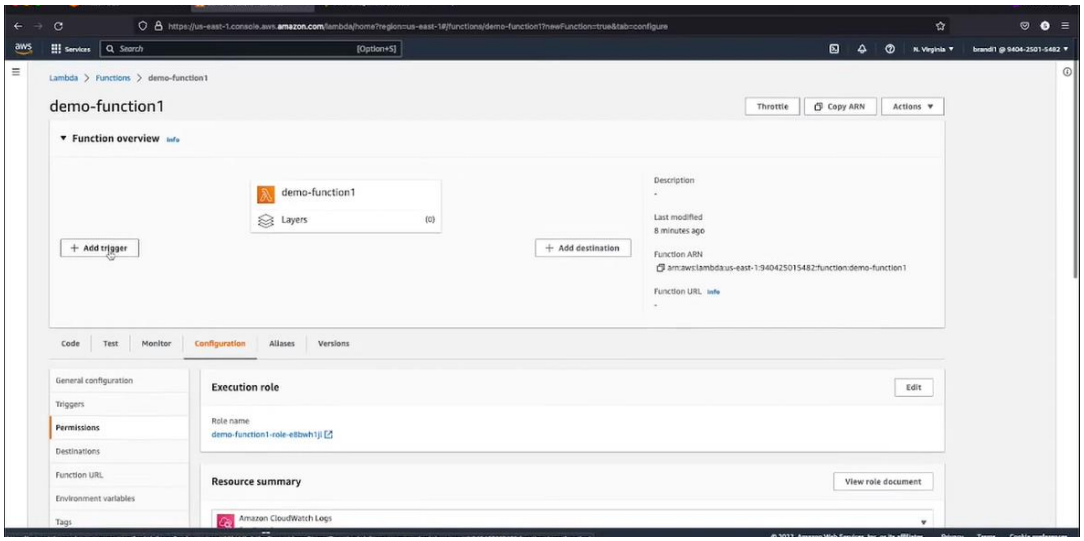
STEP 5: SQS queue created successfully.



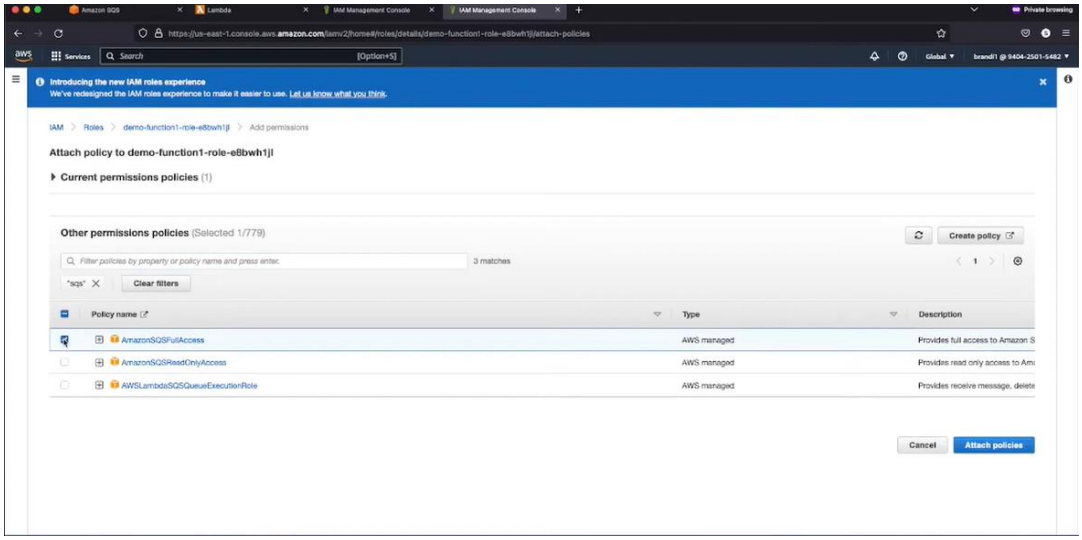
STEP 6: Create lambda function with necessary details.



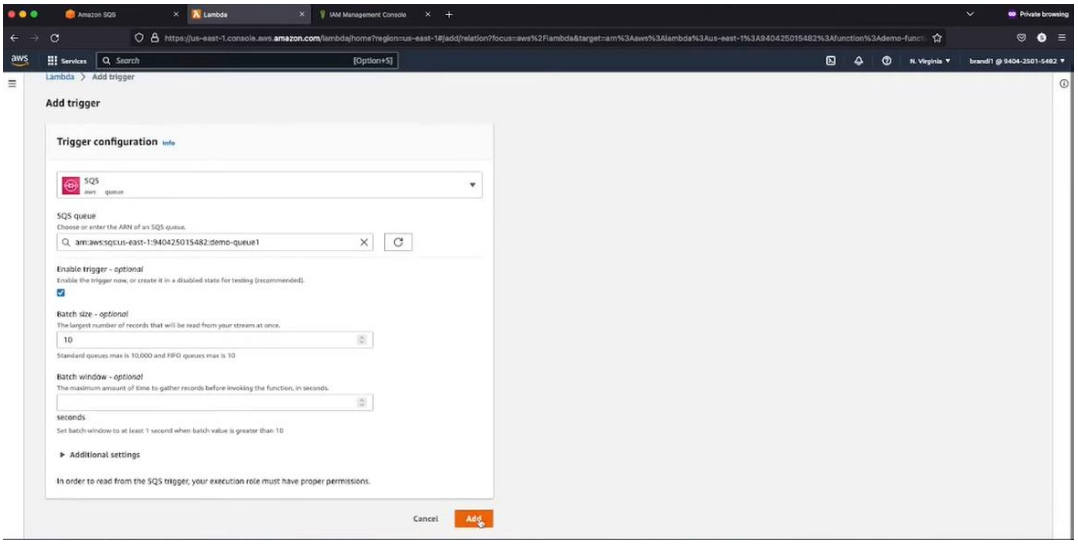
STEP 7: Click “Add trigger”.



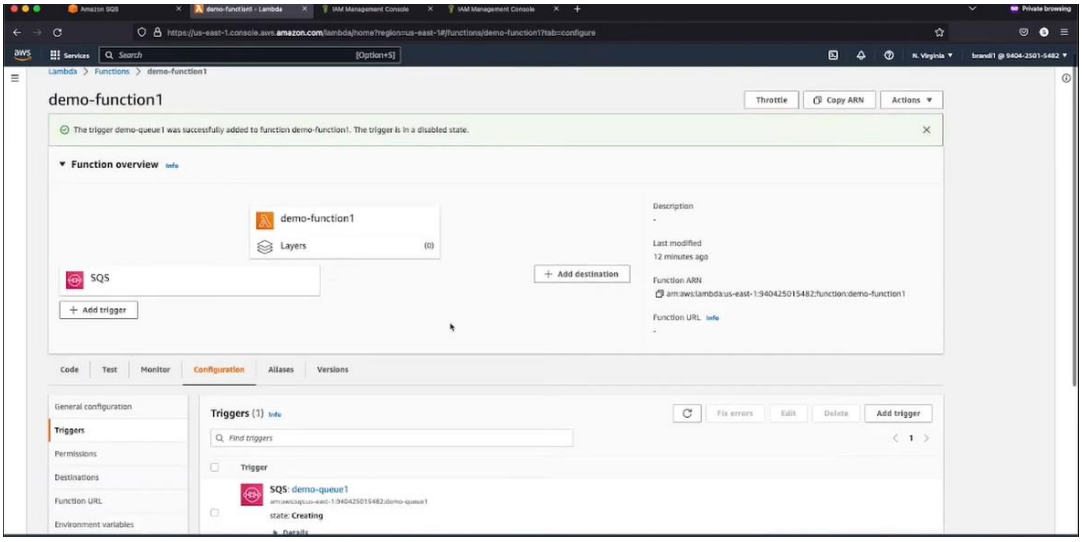
STEP 8: Add a permission in IAM role for SQS – AmazonSQSFullAccess



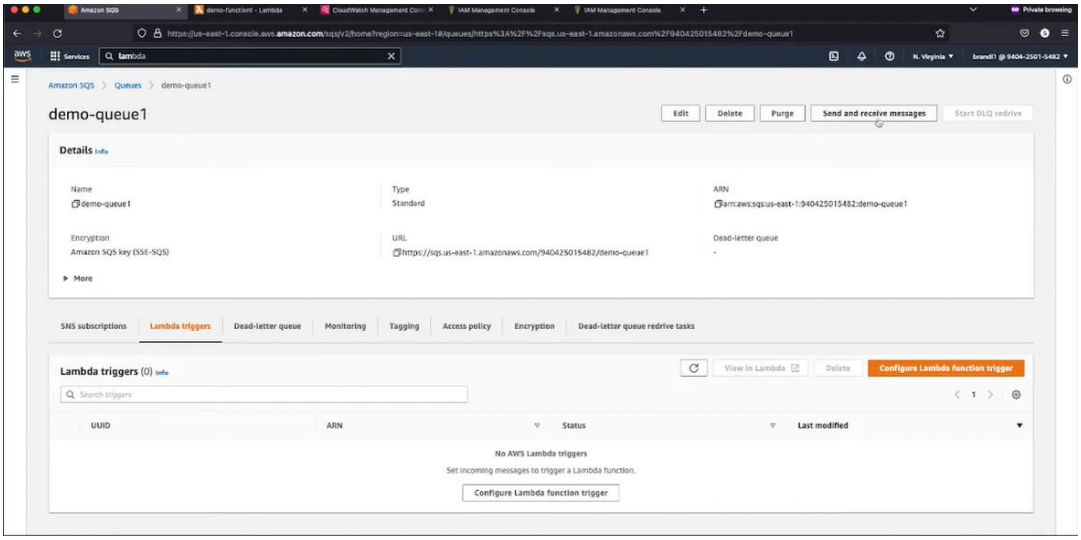
STEP 9: Choose SQS and enter necessary details and click “Add”.



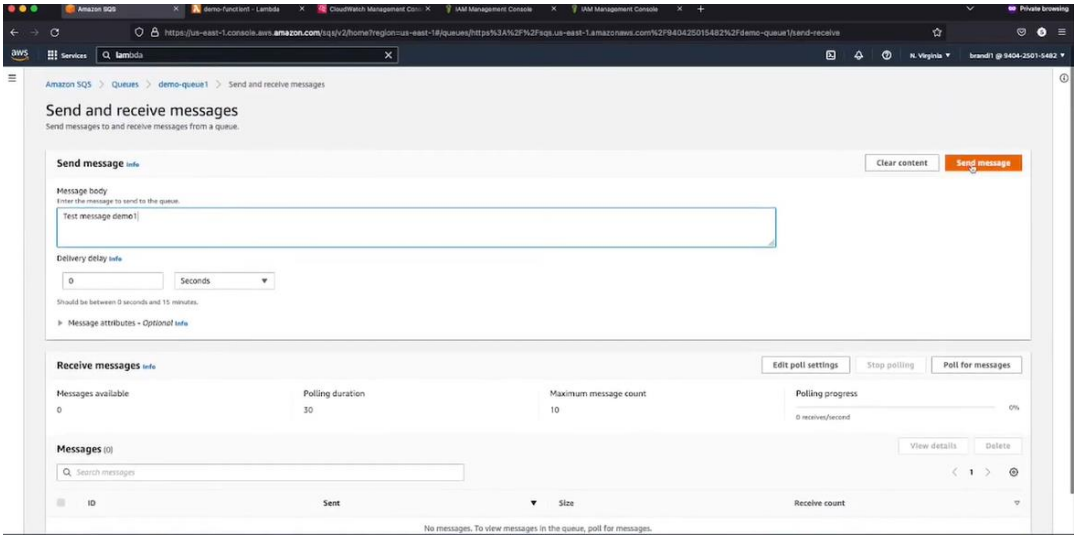
STEP 10: Trigger added successfully.



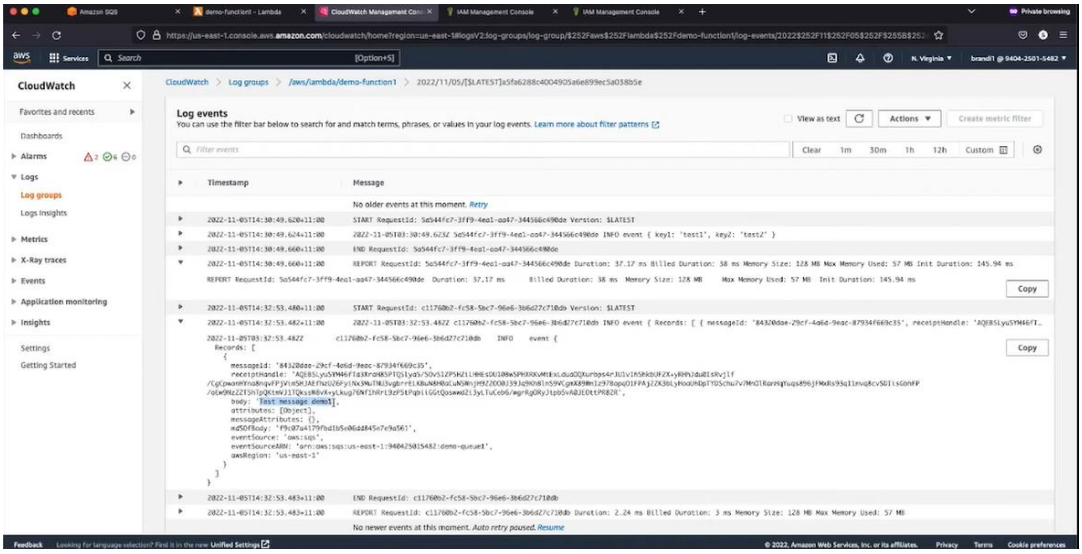
STEP 11: Open SQS service and click “send and receive messages”.



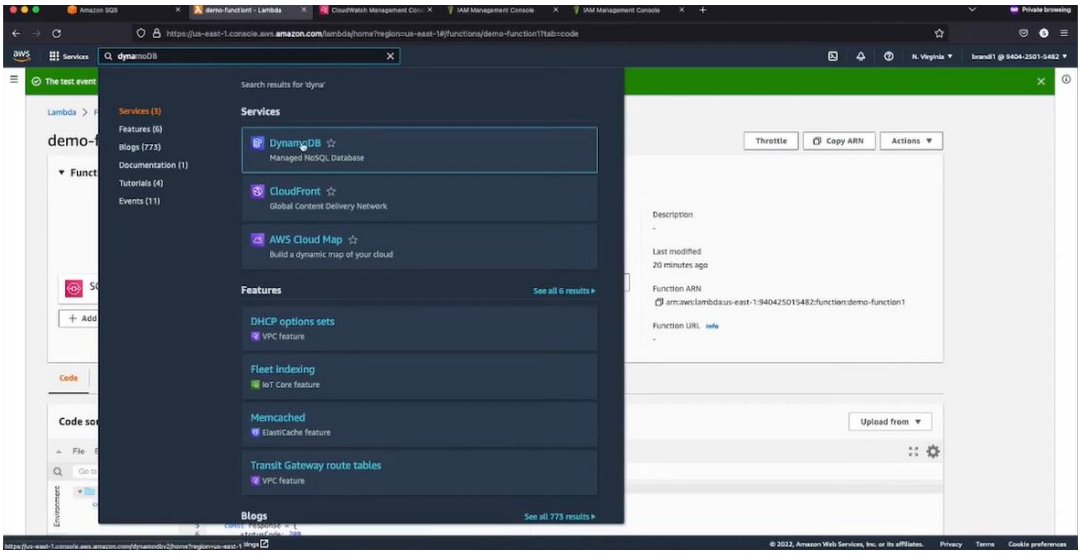
STEP 12: Enter some message in “Message body” and click “Send message”.



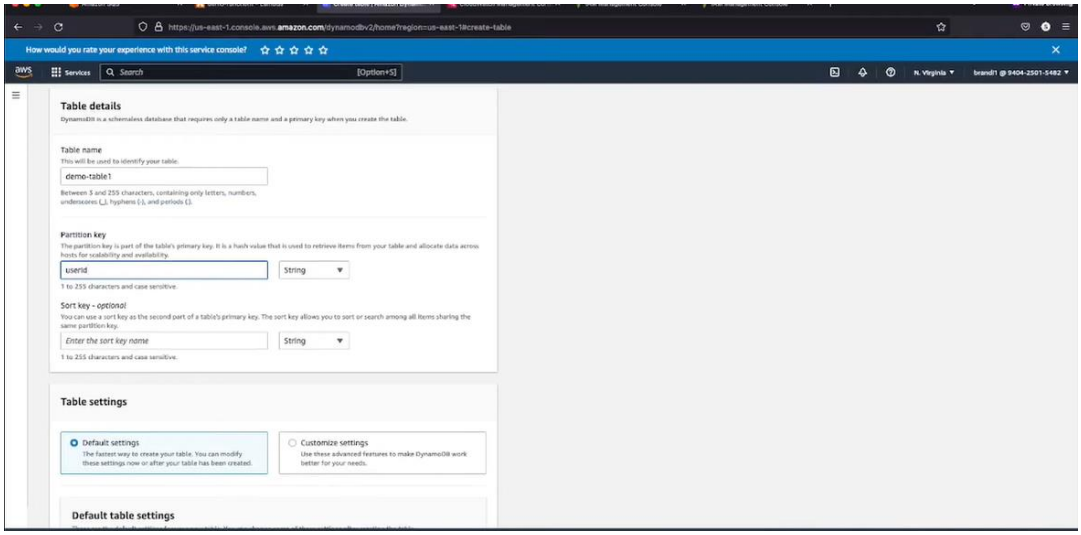
STEP 13: After sending a message, log in cloud watch for verification.



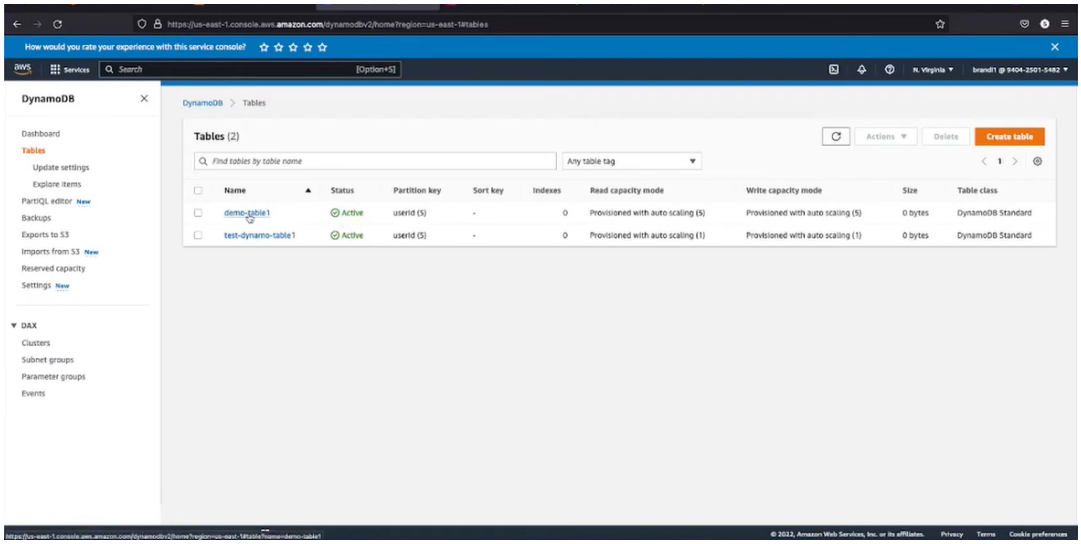
STEP 14: Next, choose “DynamoDB” service.



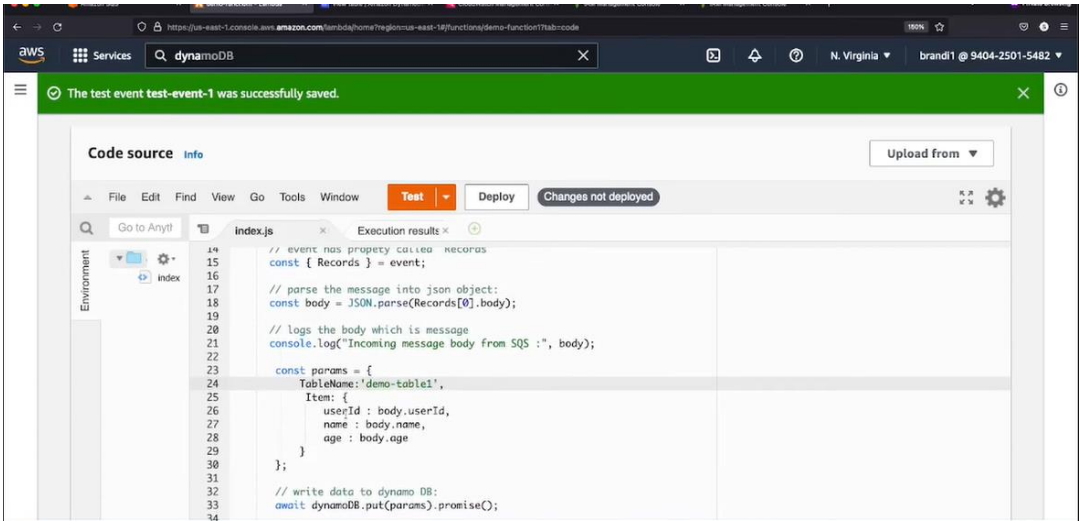
STEP 15: Create table with table name, partition key and click “Create table”.



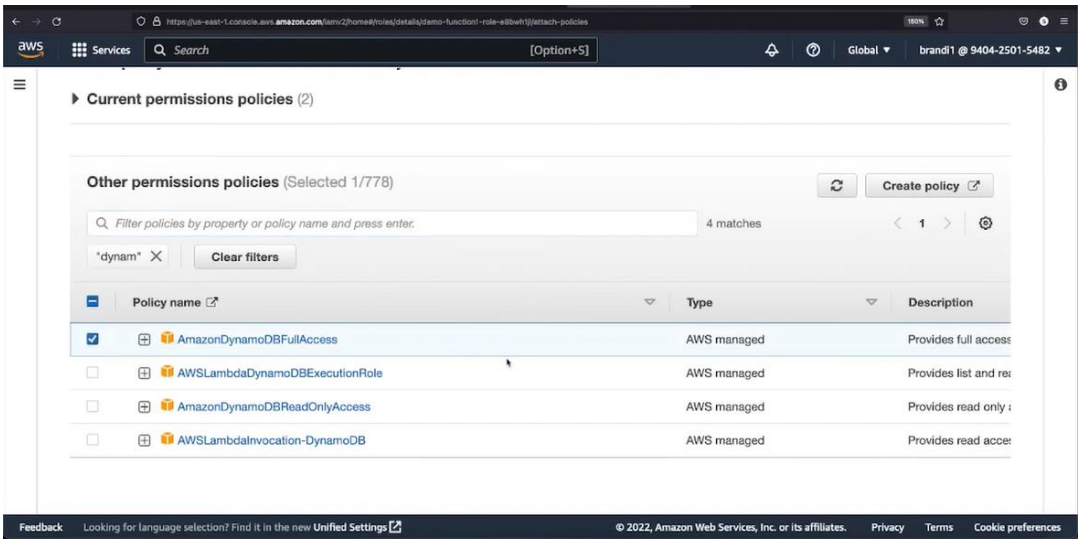
STEP 16: Table created successfully.



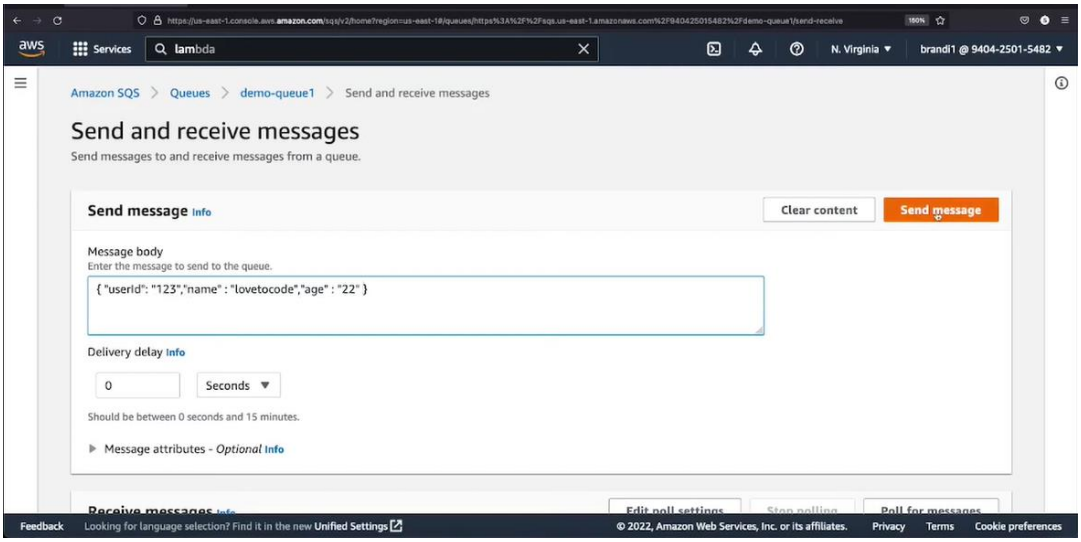
STEP 17: Write a code to store that message in Dynamo DB.



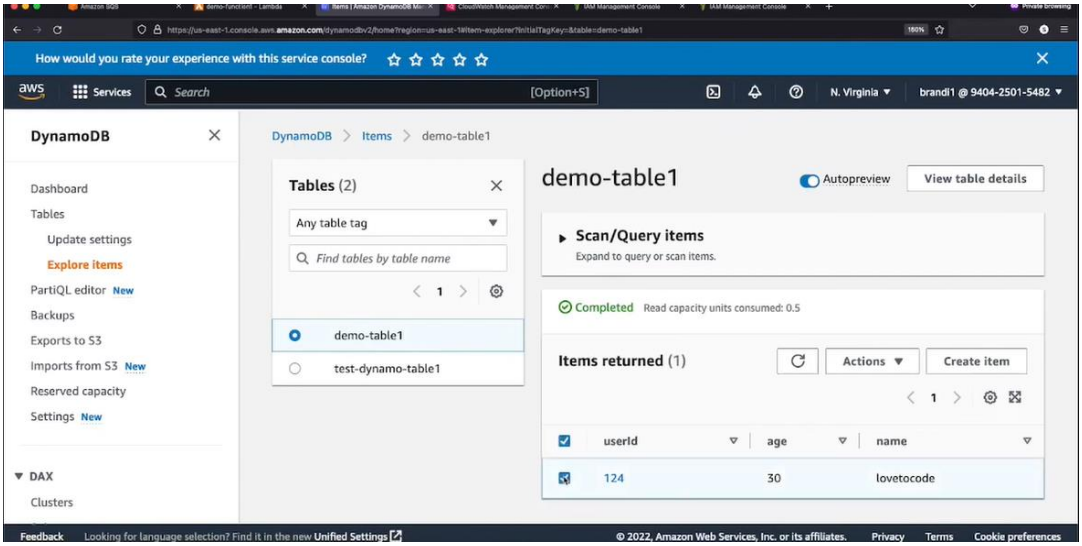
STEP 18: Enable permission for DynamoDB – AmazonDynamoDBFullAccess.



STEP 19: Open SQS service and send a message.

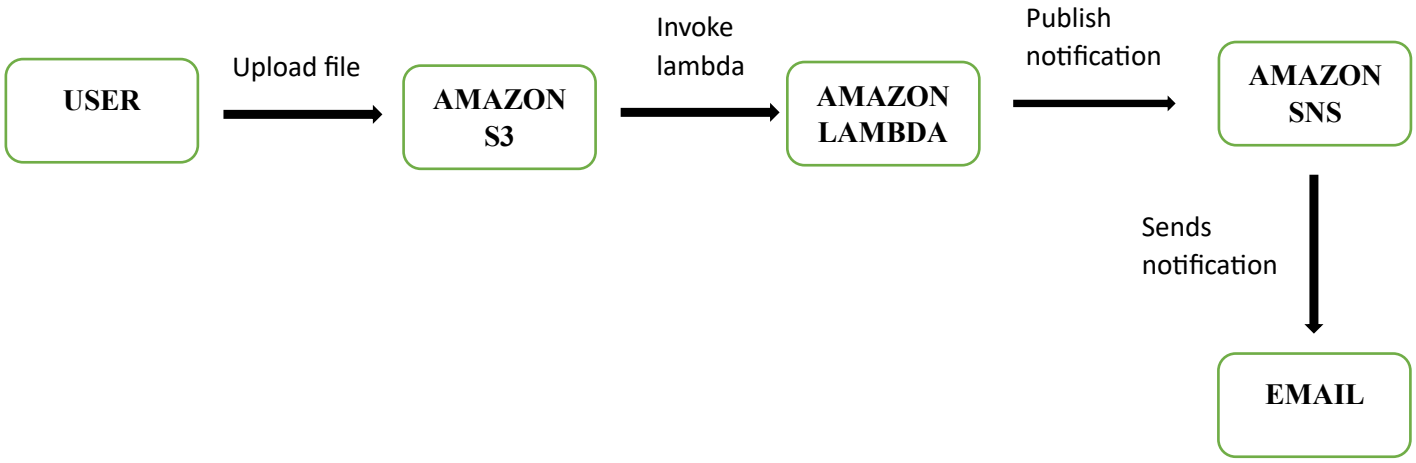


STEP 20: Open DynamoDB table, the data stored here successfully.

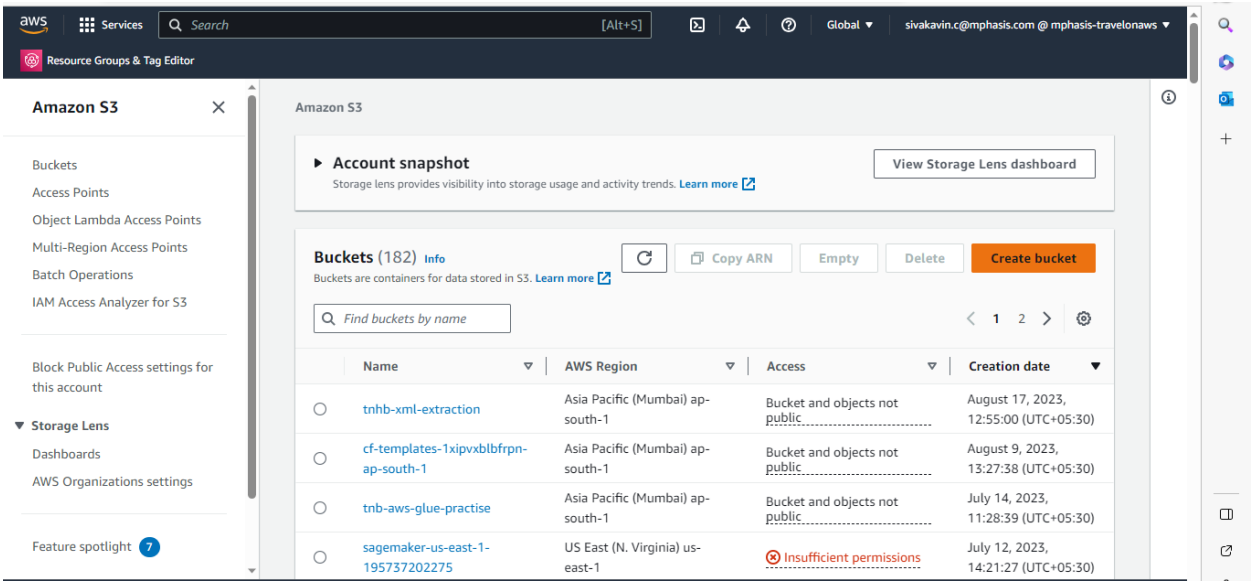


AUTOMATE SENDING NOTIFICATIONS WITH USING AMAZON SNS AND AWS LAMBDA

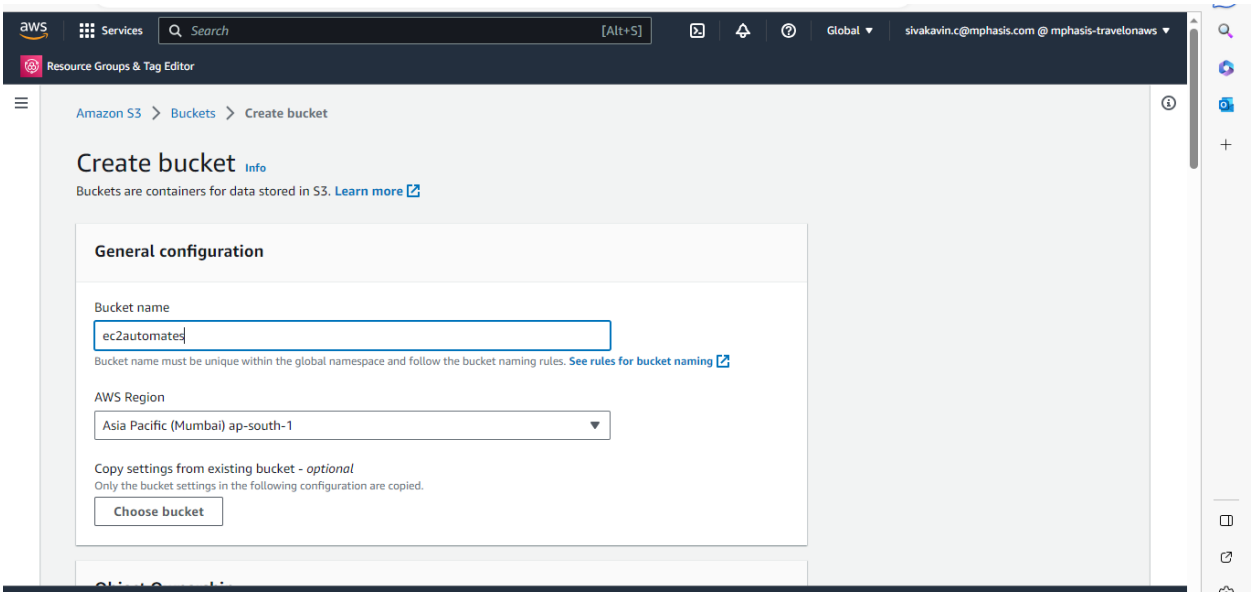
WORKFLOW DIAGRAM



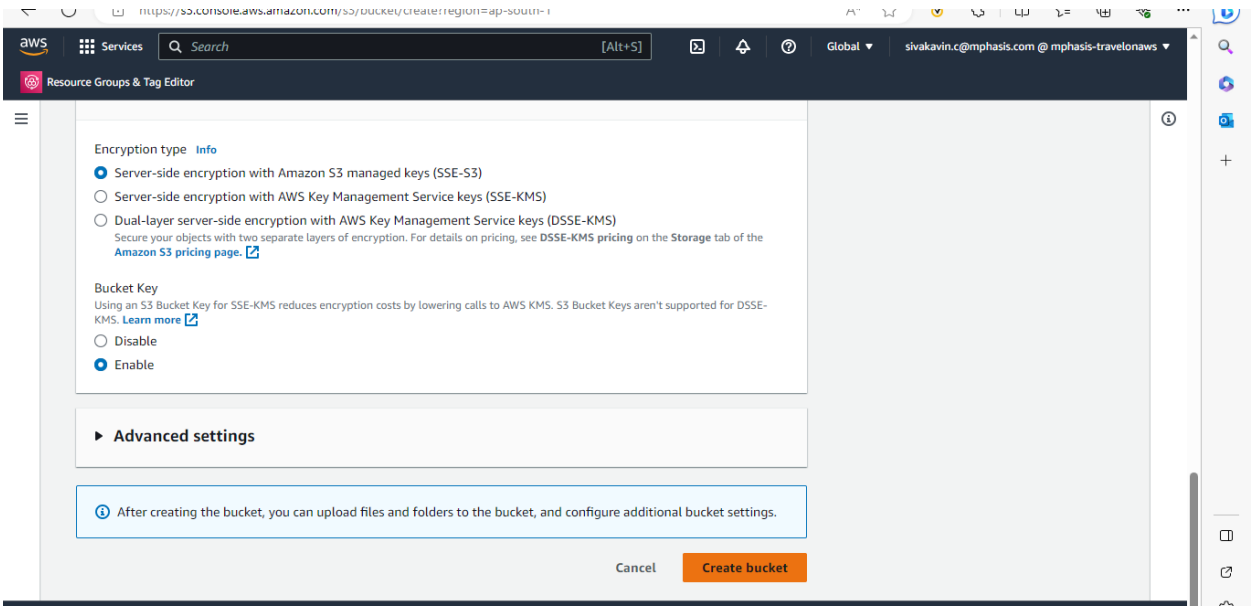
STEP 1: Choose Amazon S3 service and click on “Create bucket”.



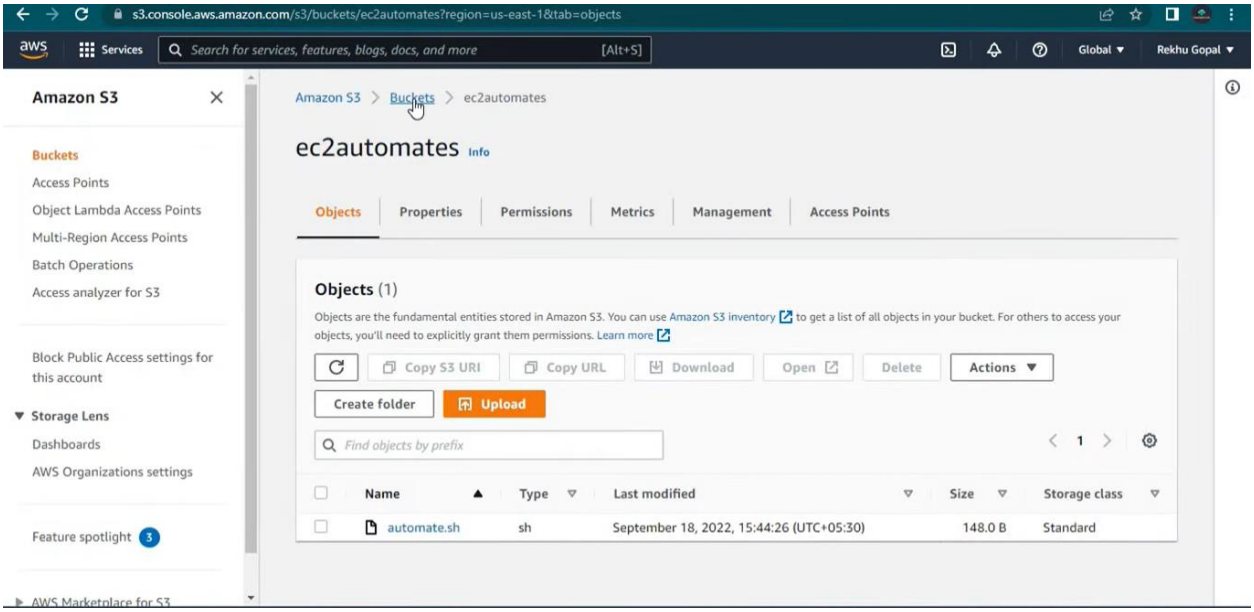
STEP 2: Fill bucket name and other information.



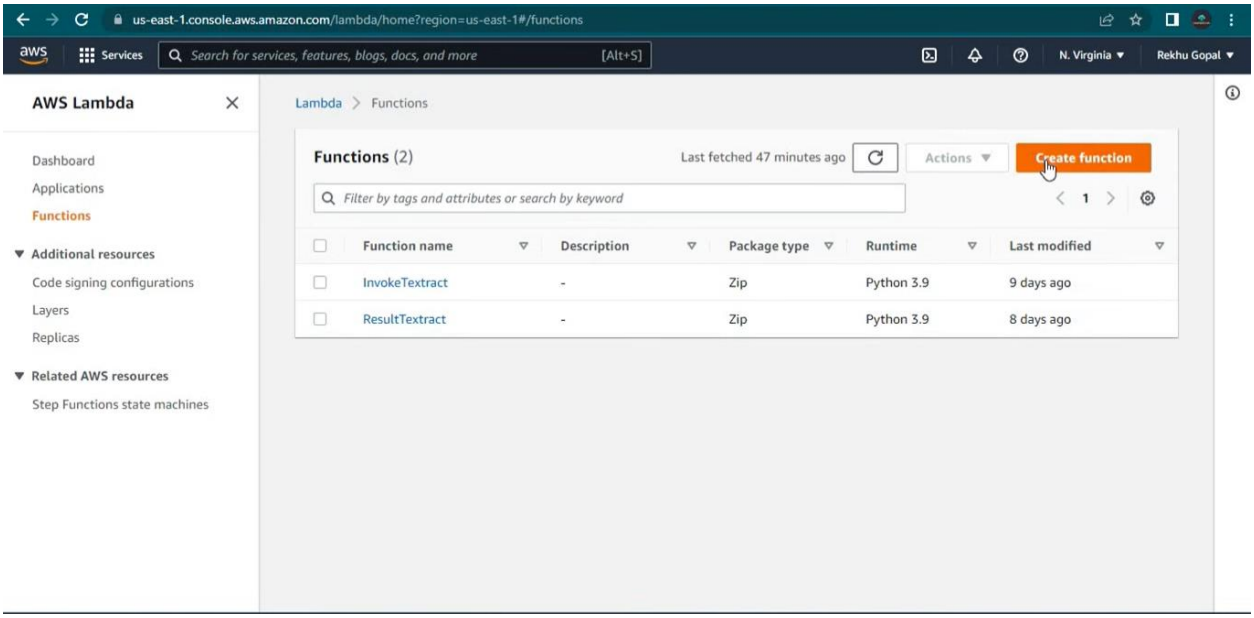
STEP 3: Choose encryption type and bucket key and click “create bucket”.



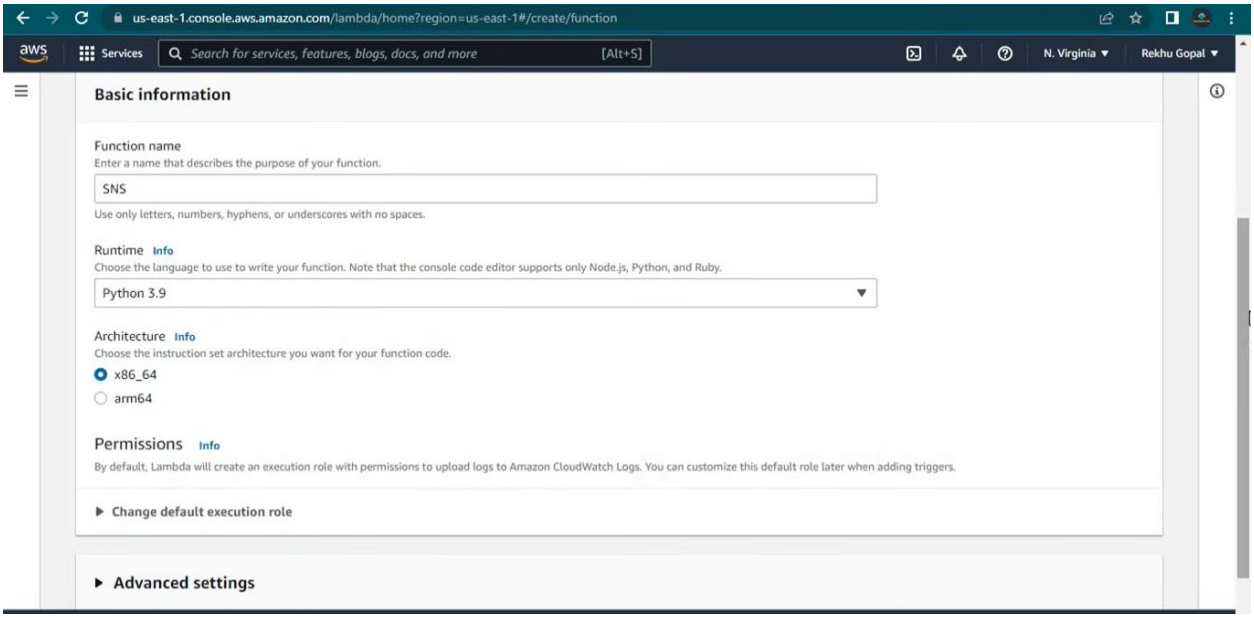
STEP 4: Bucket created successfully.



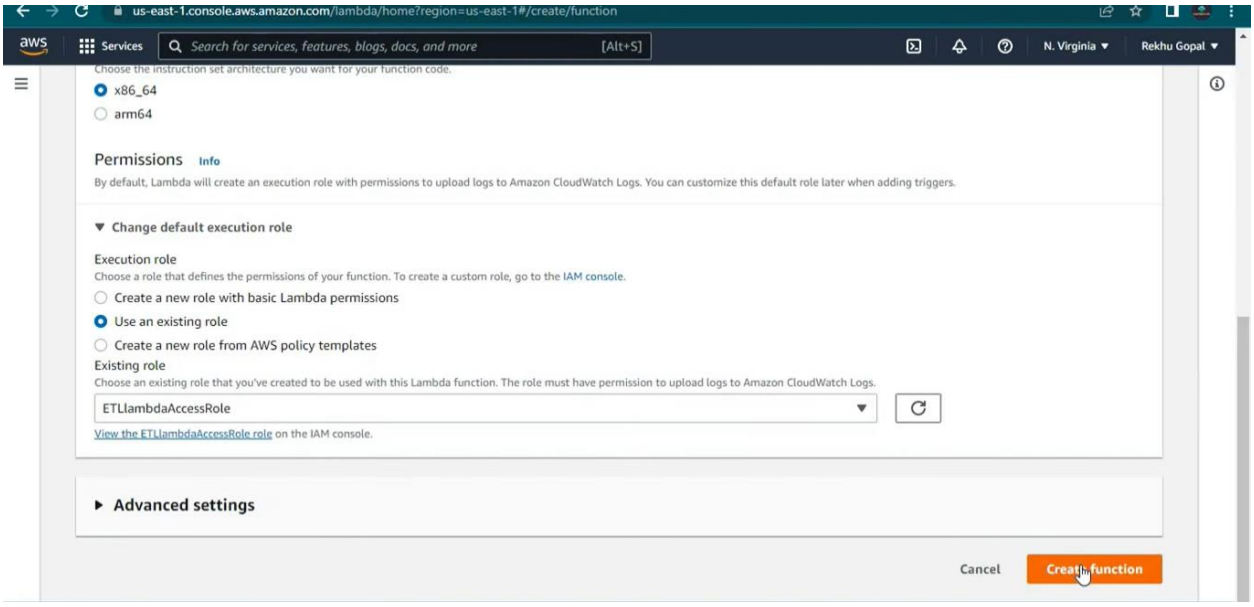
STEP 5: Create lambda function.



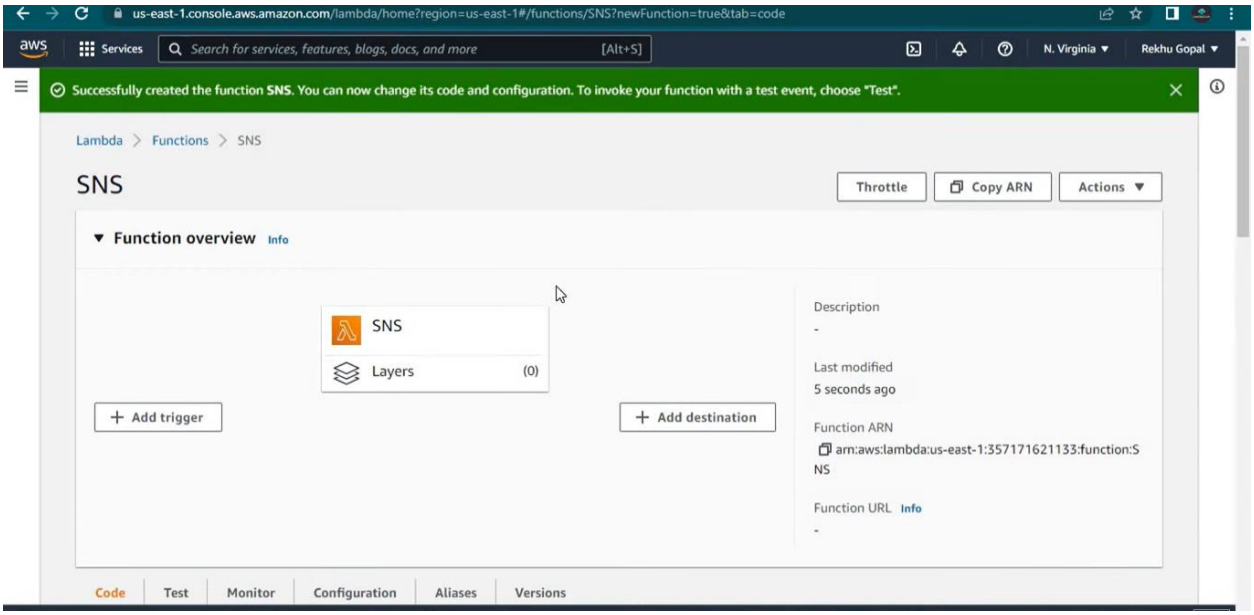
STEP 6: Fill function name, runtime, architecture, and permissions.



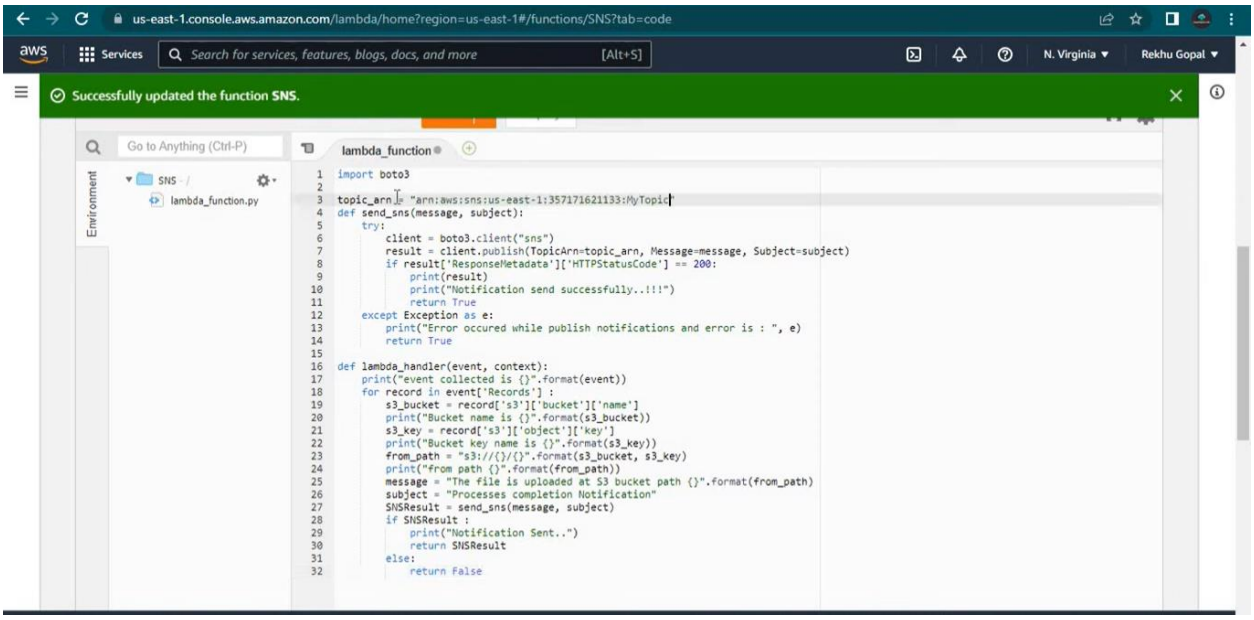
STEP 7: Click “Create function”.



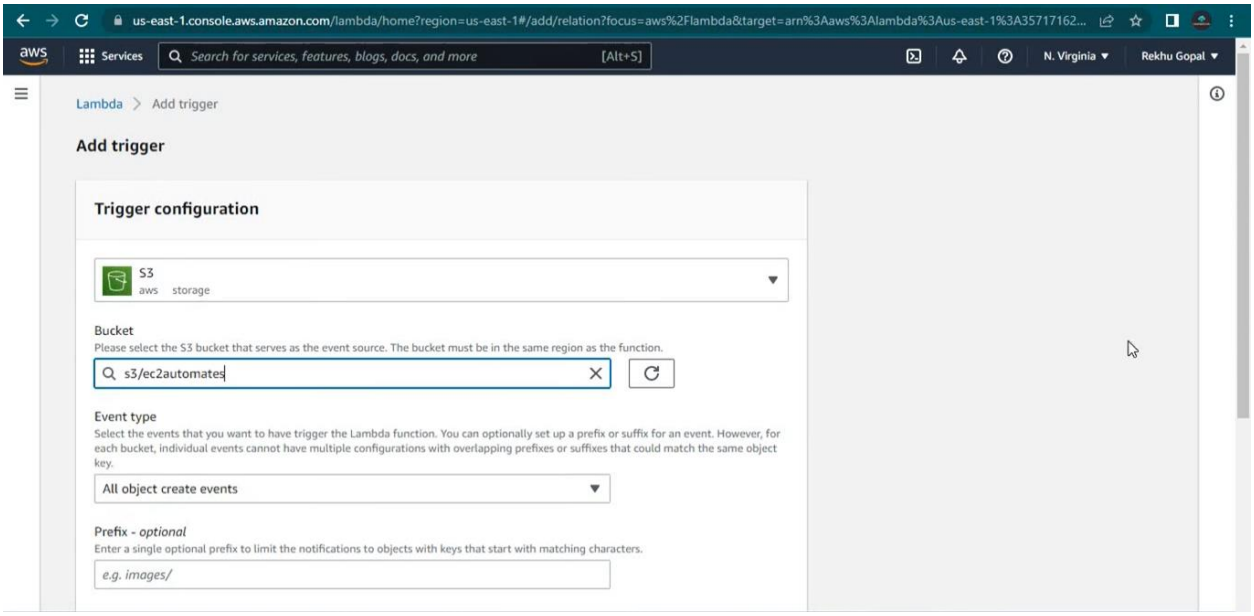
STEP 8: Lambda function created successfully.



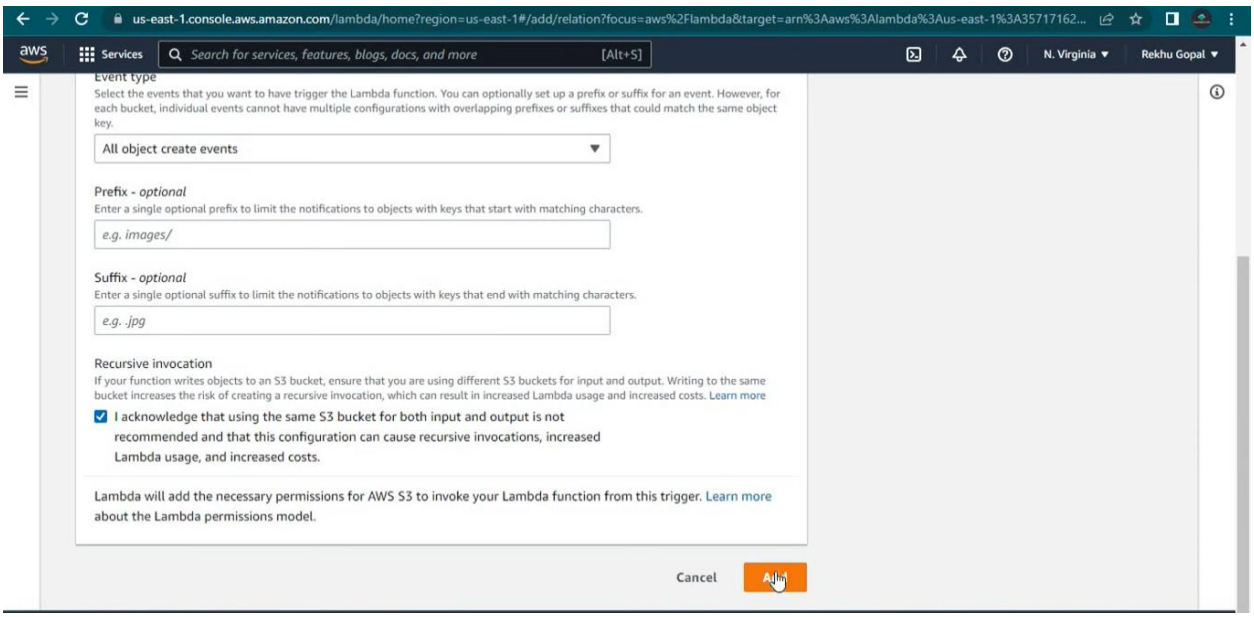
STEP 9: Write a code to get a detail from S3 and push to SNS.



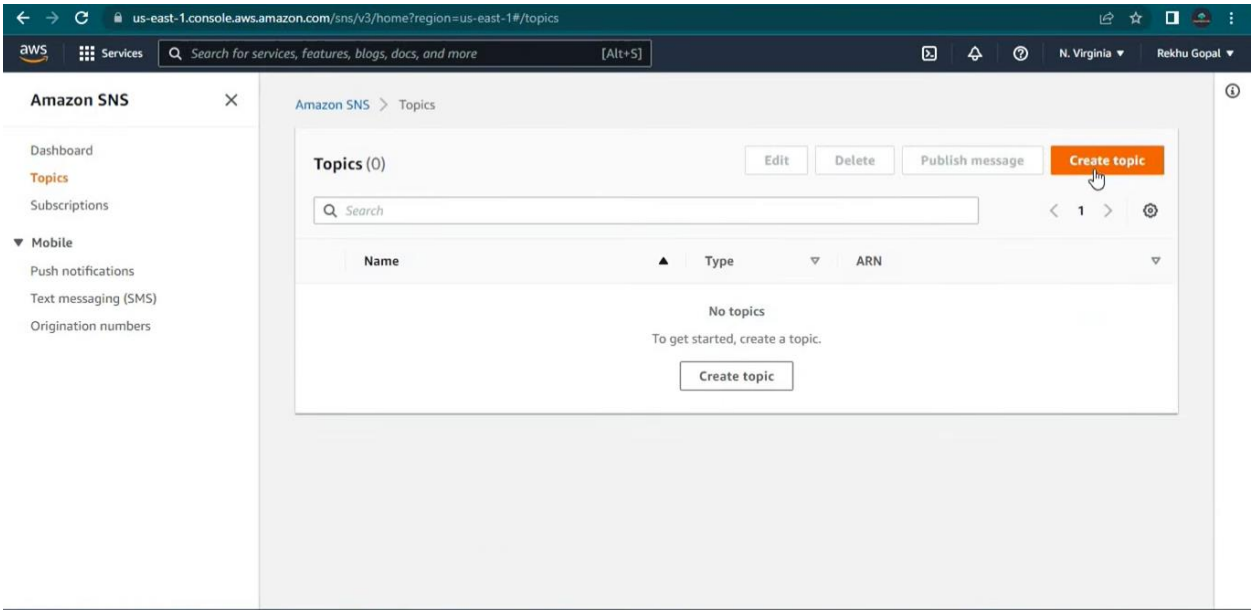
STEP 10: Add trigger with S3 – mention bucket name and event type.



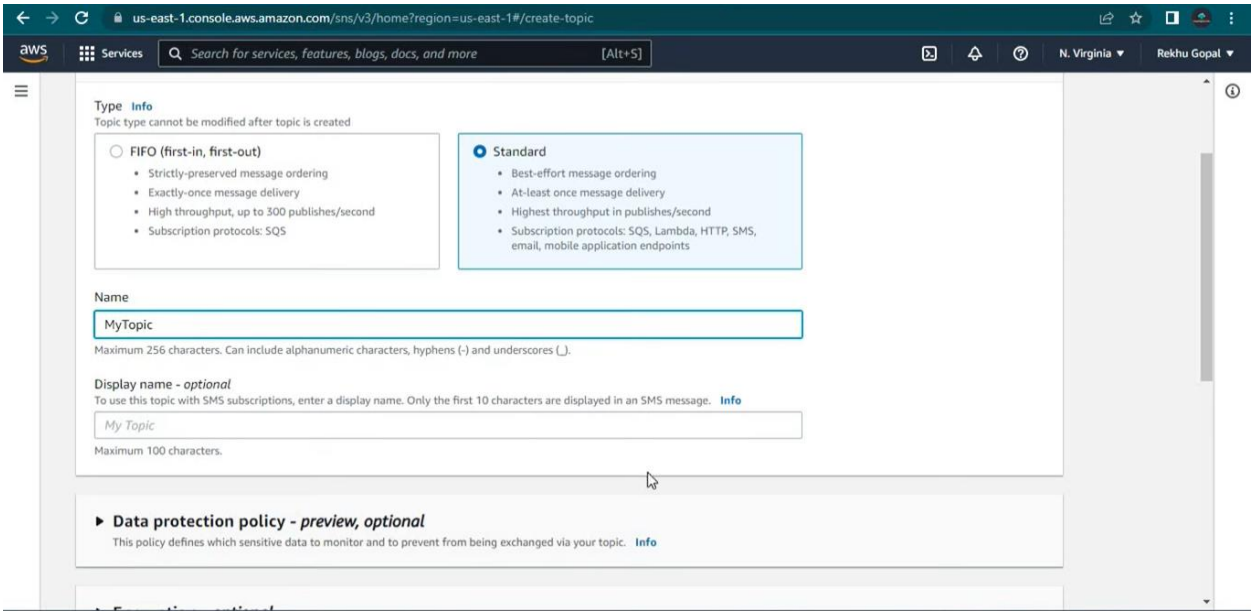
STEP 11: Click “Add”.



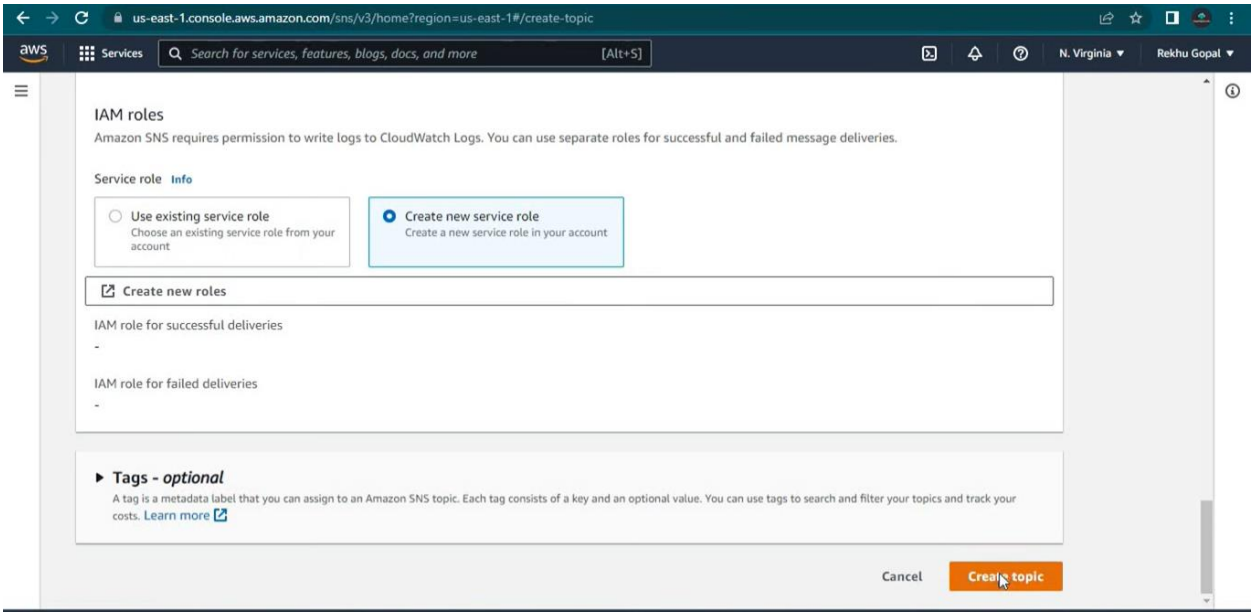
STEP 12: In Amazon SNS, click “Create topic”.



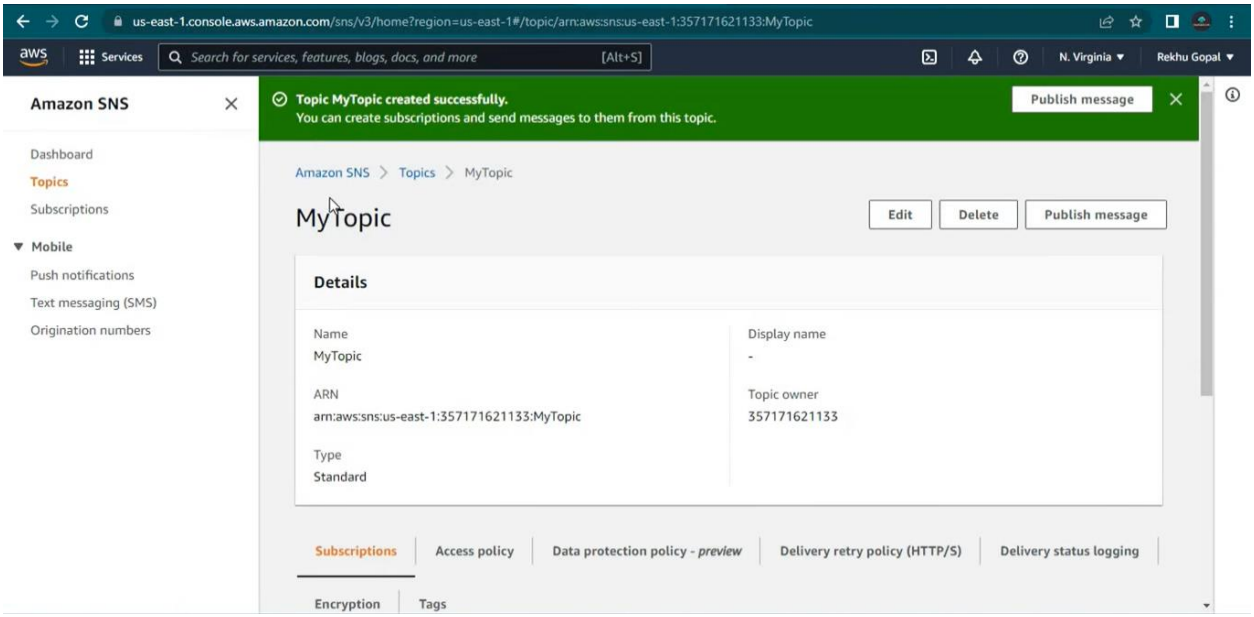
STEP 13: Mention topic name.



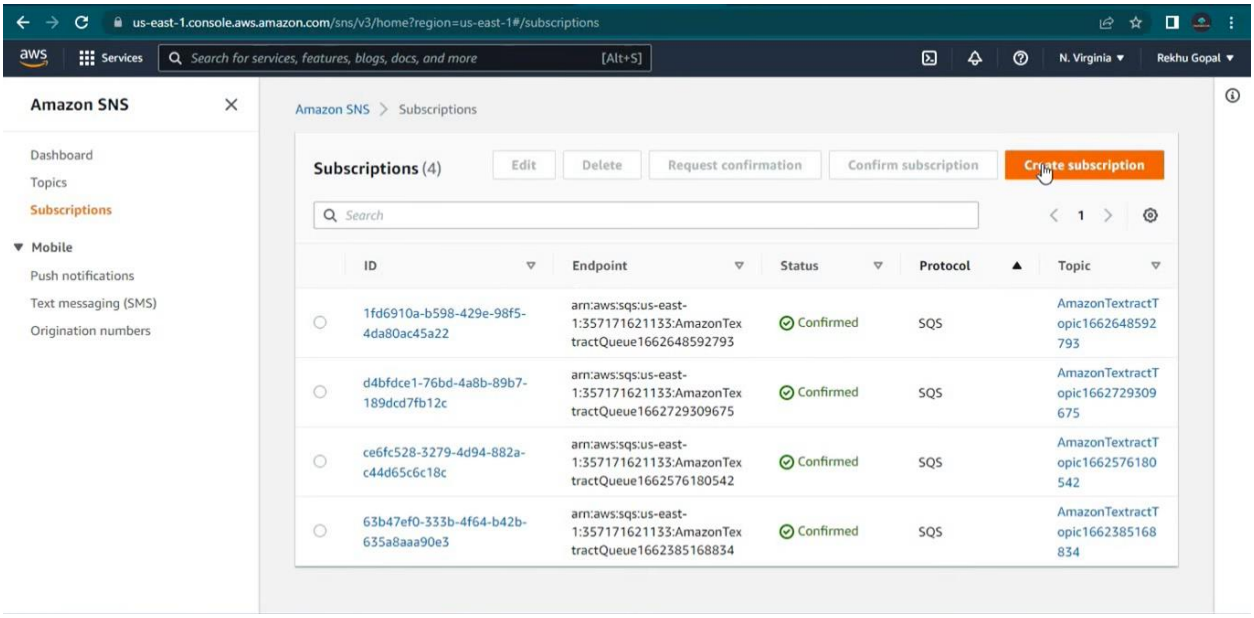
STEP 14: Choose IAM role and click “create topic”.



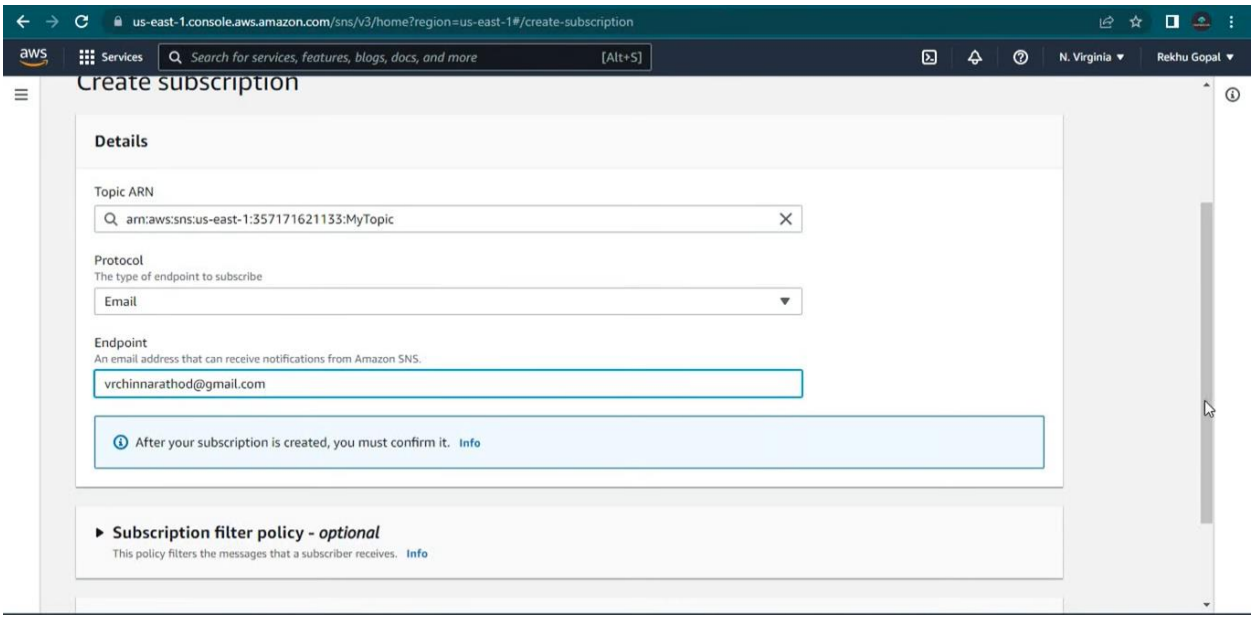
STEP 15: Topic created successfully.



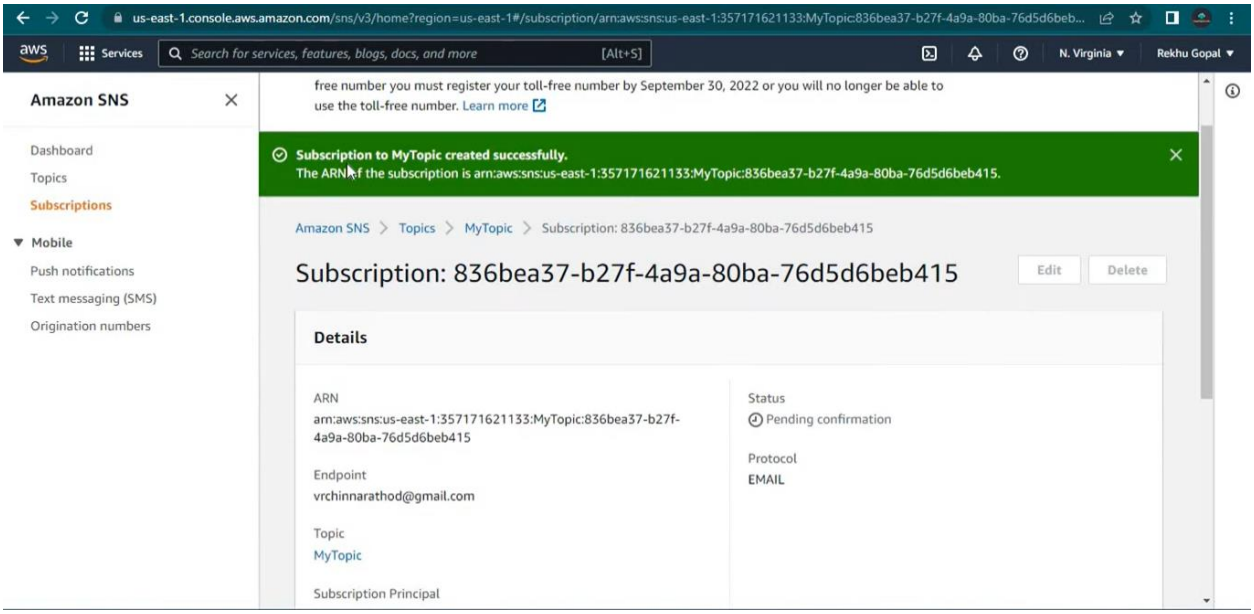
STEP 16: Next, click “Create subscription” to subscribe the topic.



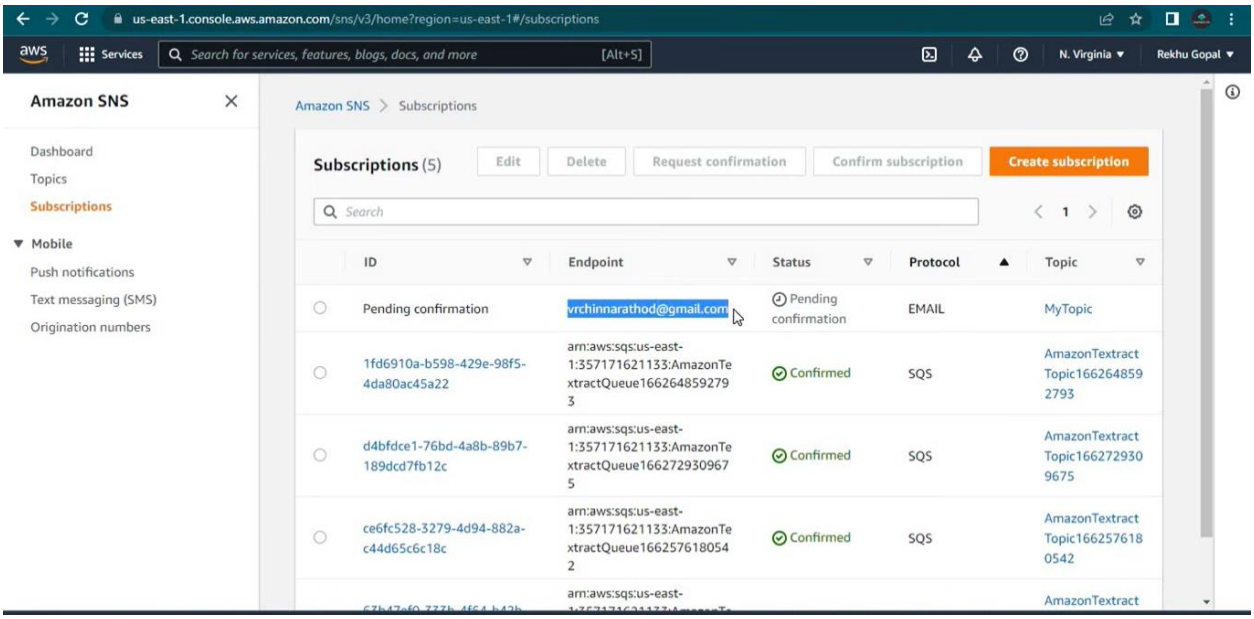
STEP 17: Choose topic ARN, protocol and endpoint.



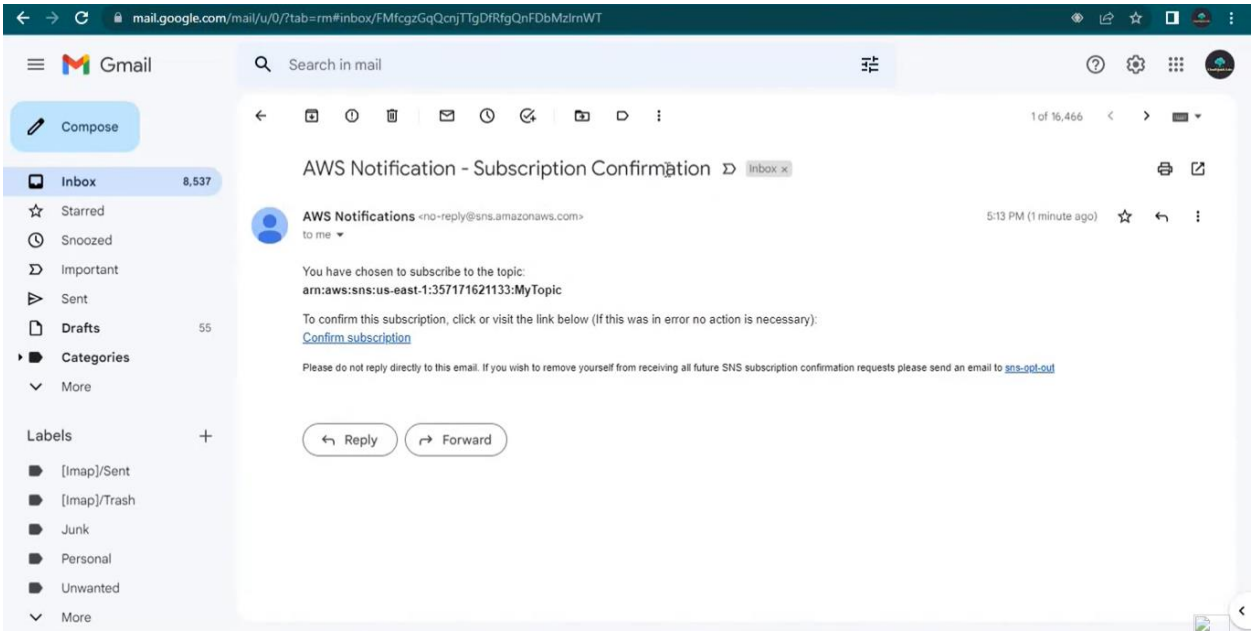
STEP 18: Subscription to topic created successfully.



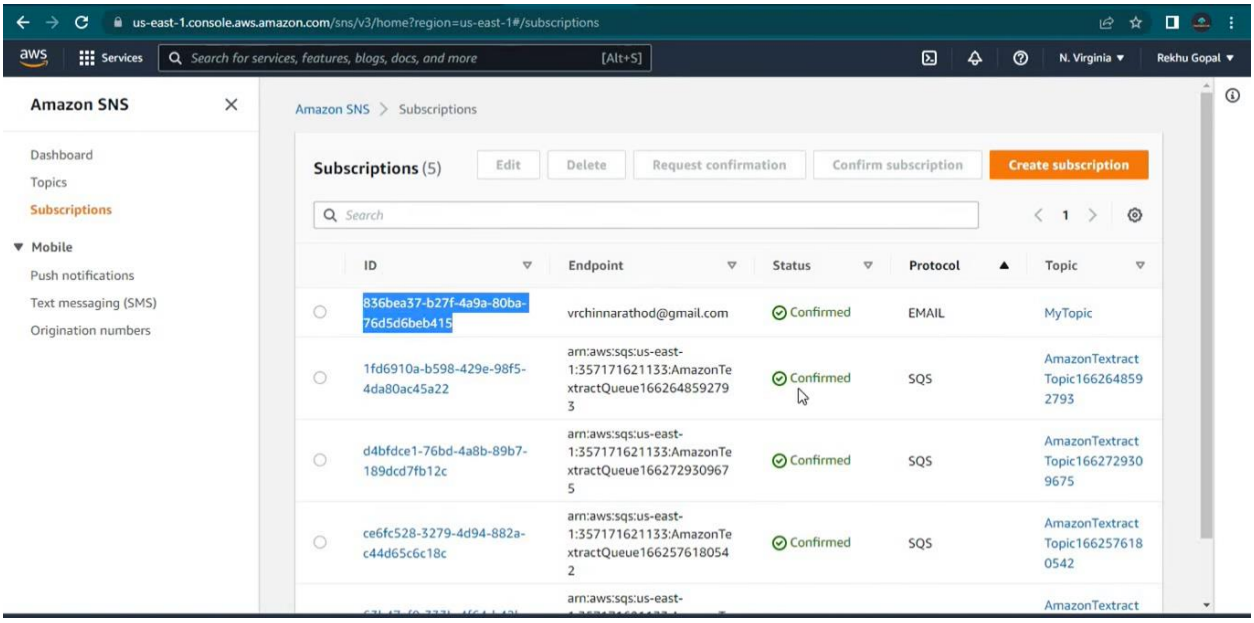
STEP 19: Subscription is in pending state. It will send an email to mentioned endpoint (Email).



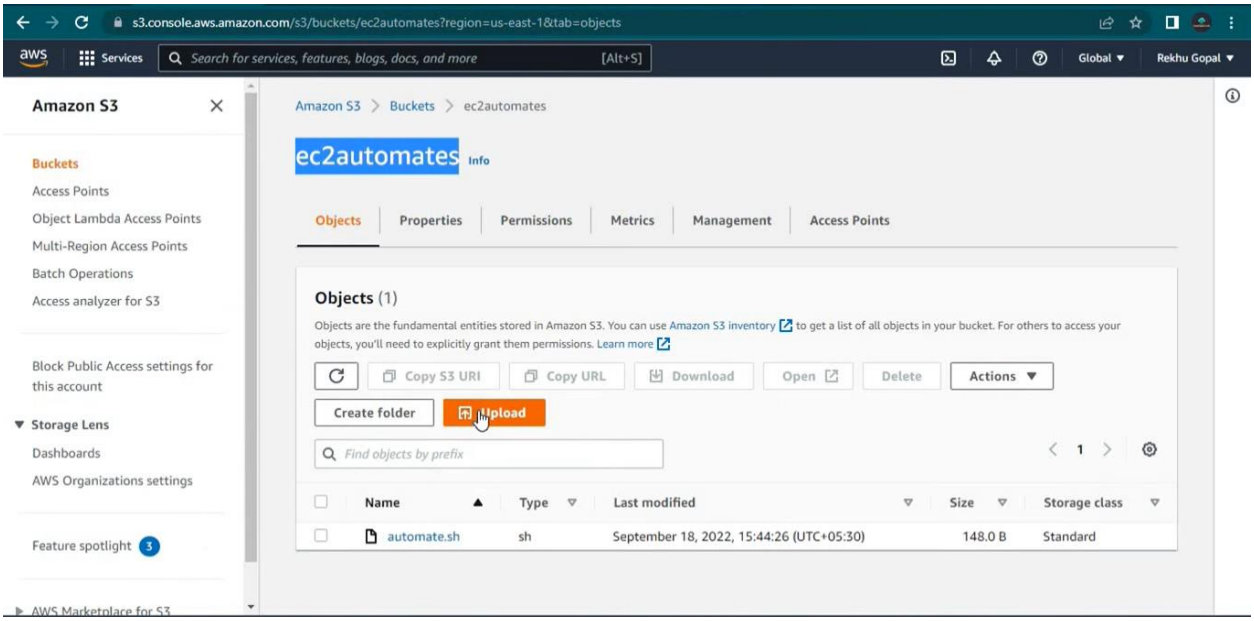
STEP 20: Click “confirm subscription” in email to confirm.



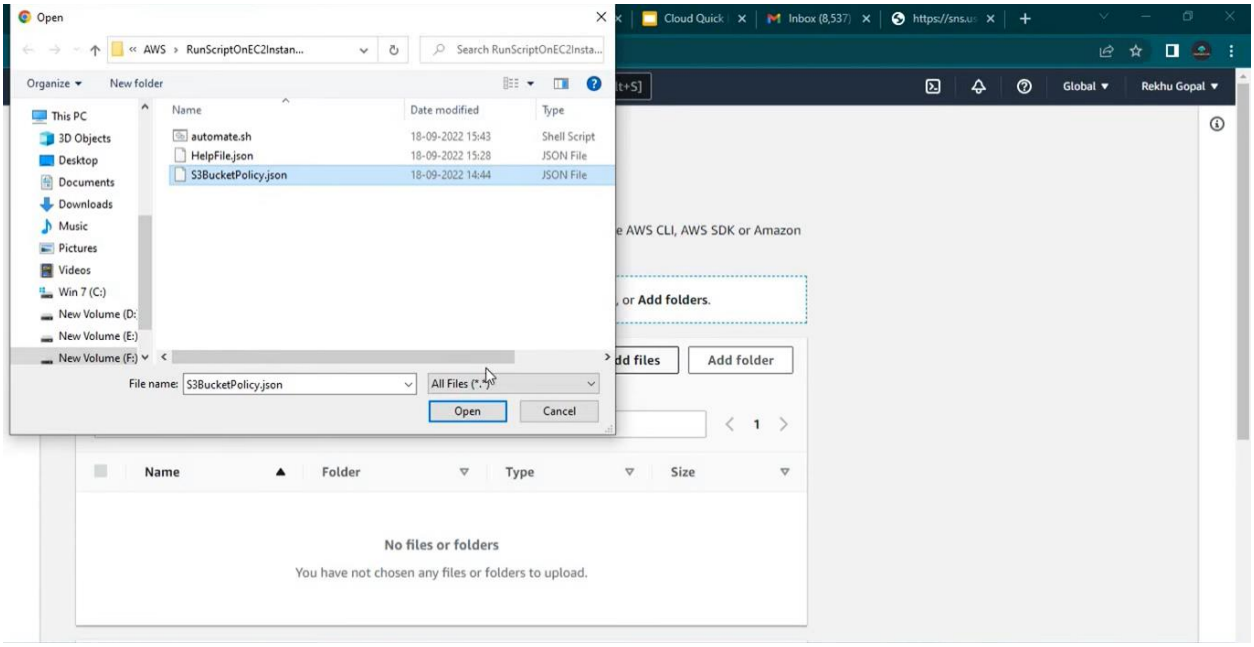
STEP 21: Then status changed to confirmed state.



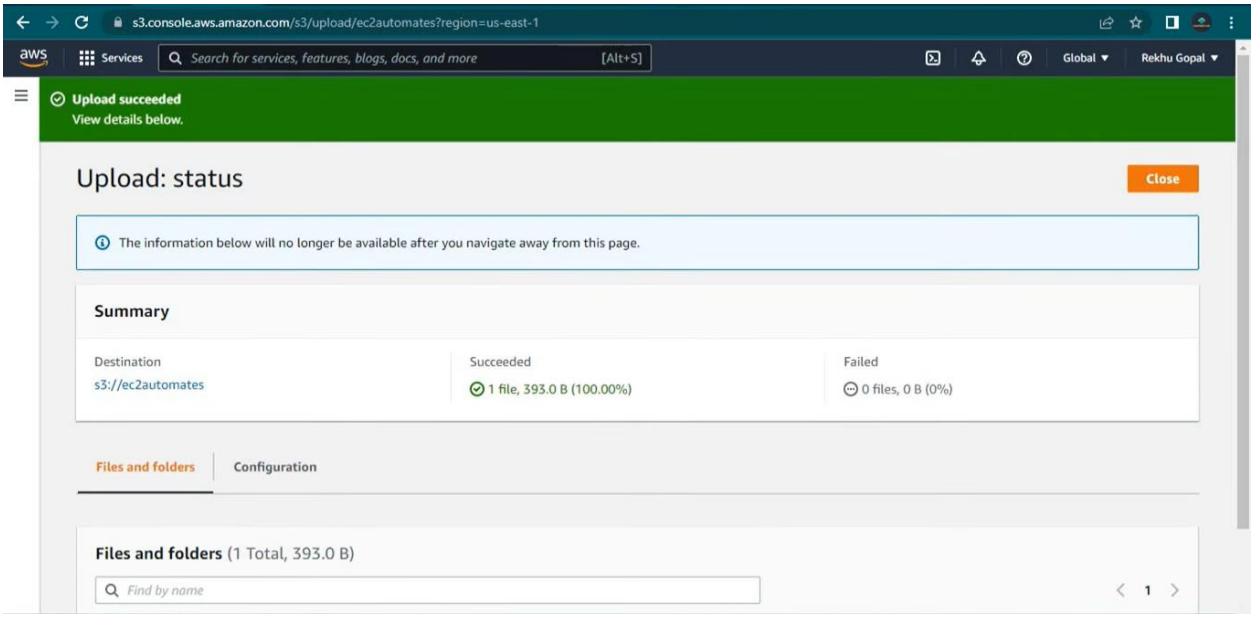
STEP 22: Now, upload a file in S3 bucket.



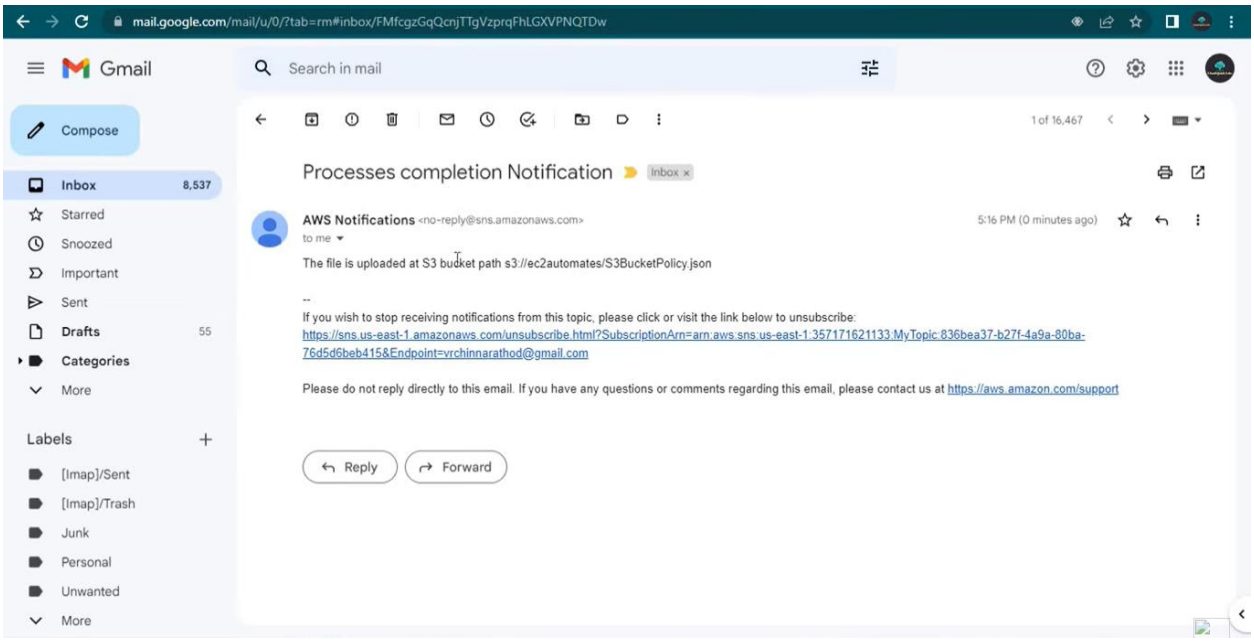
STEP 23: Choose one file from PC.



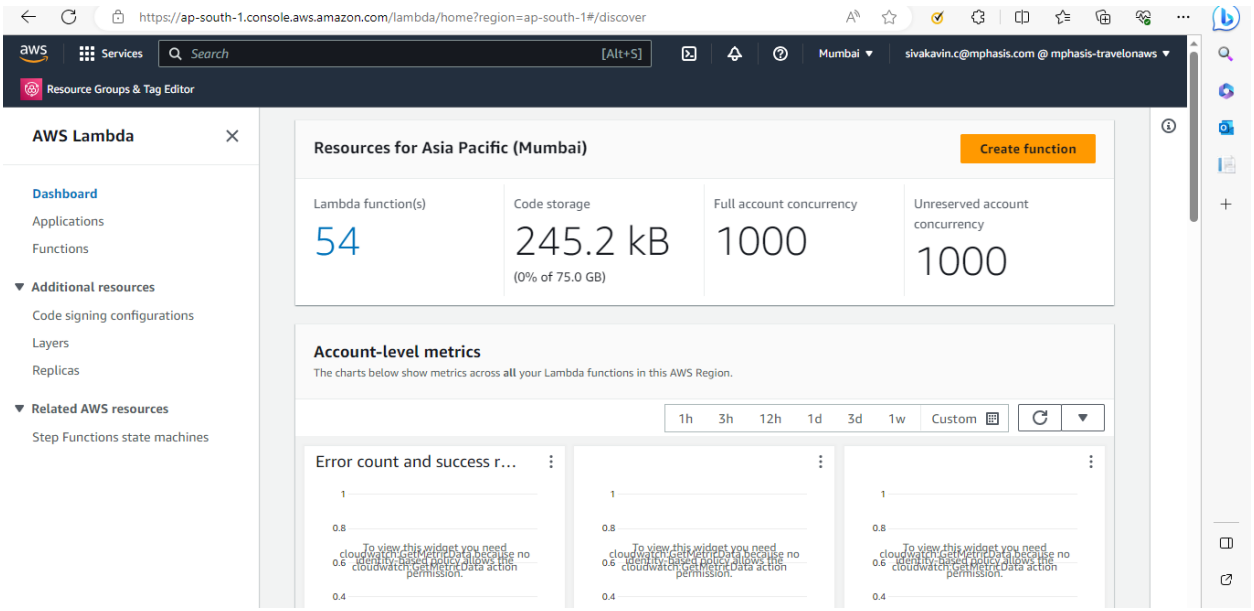
STEP 24: File uploaded successfully in S3 bucket.

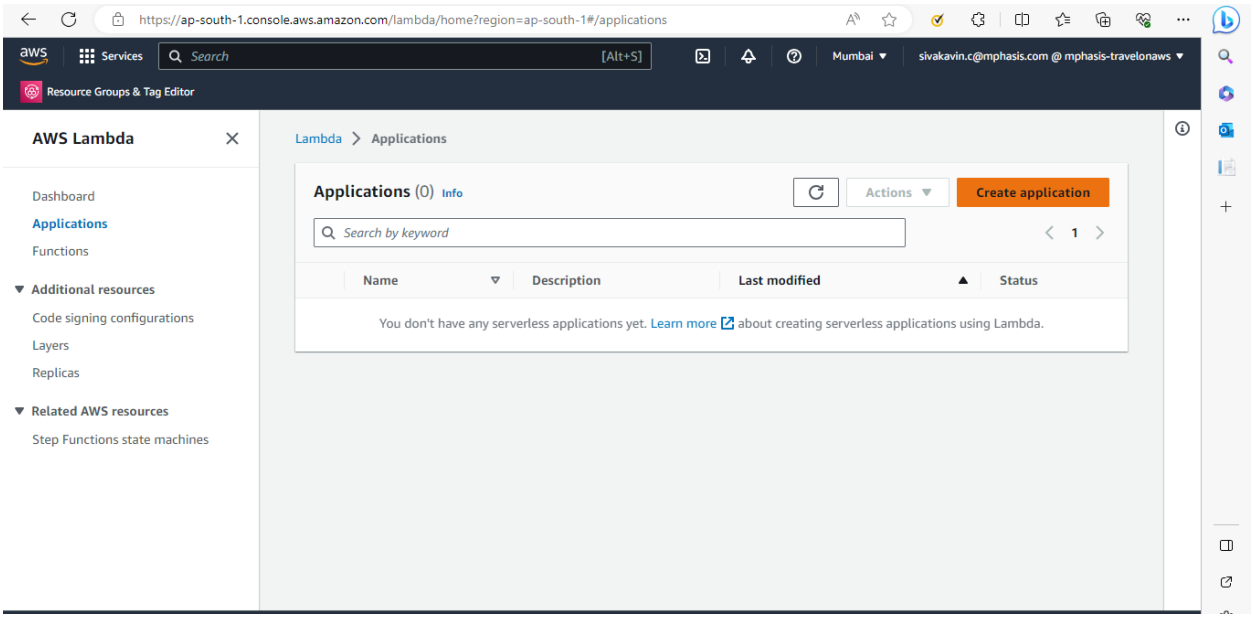


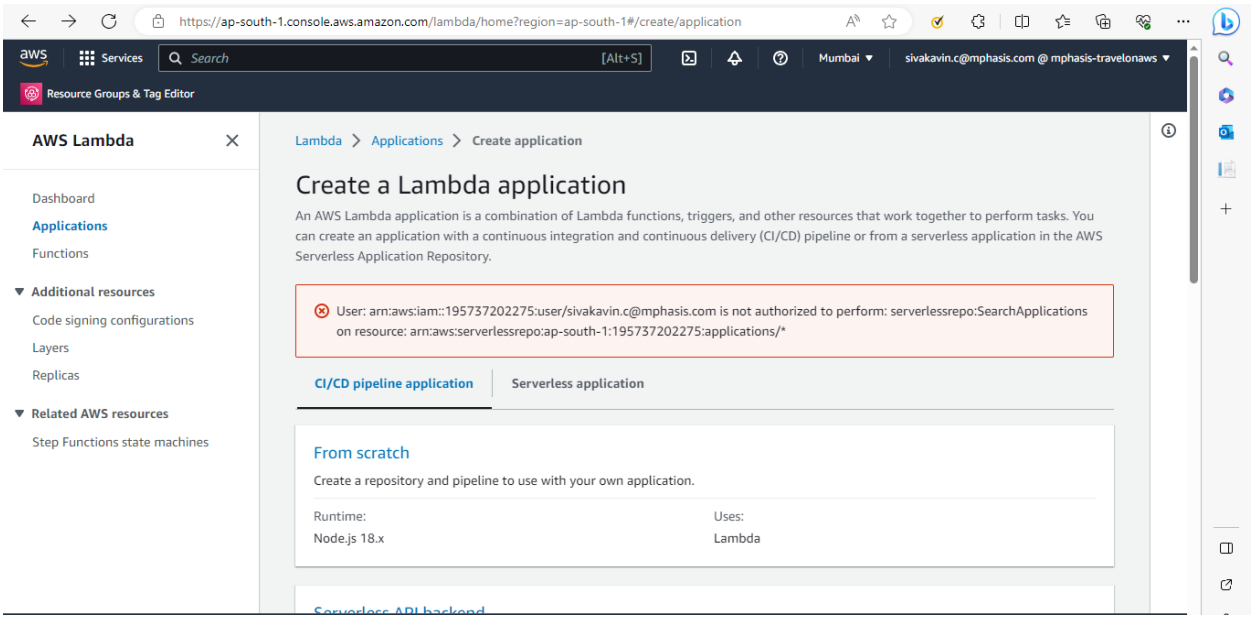
STEP 25: Notification received to email successfully.

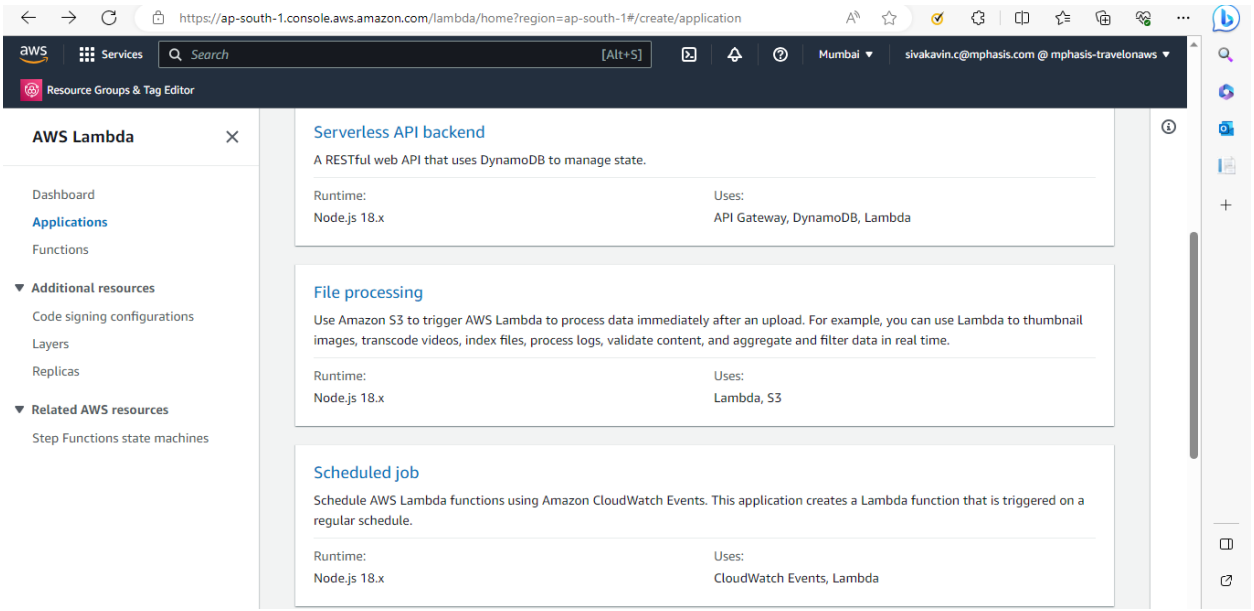


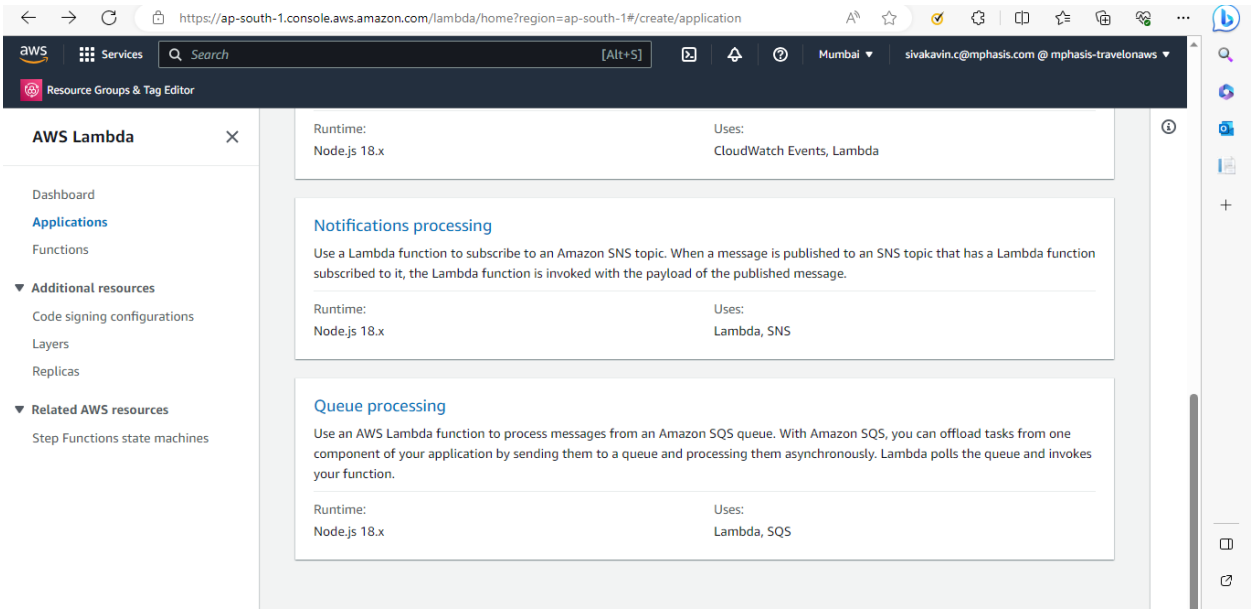
DASHBOARD AND APPLICATIONS

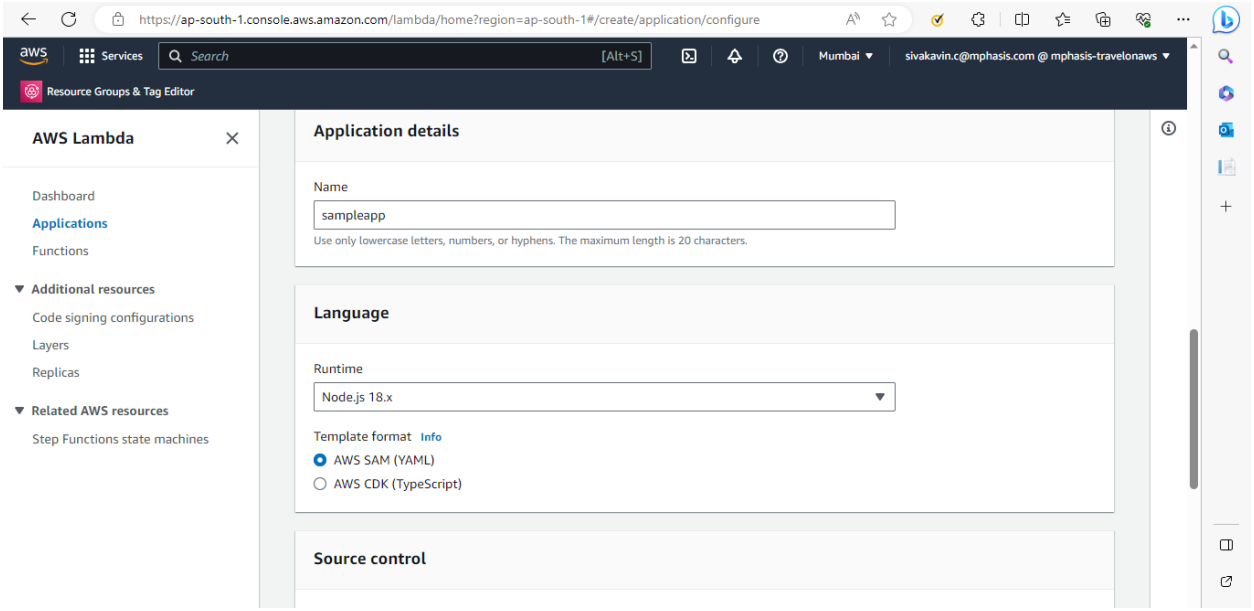


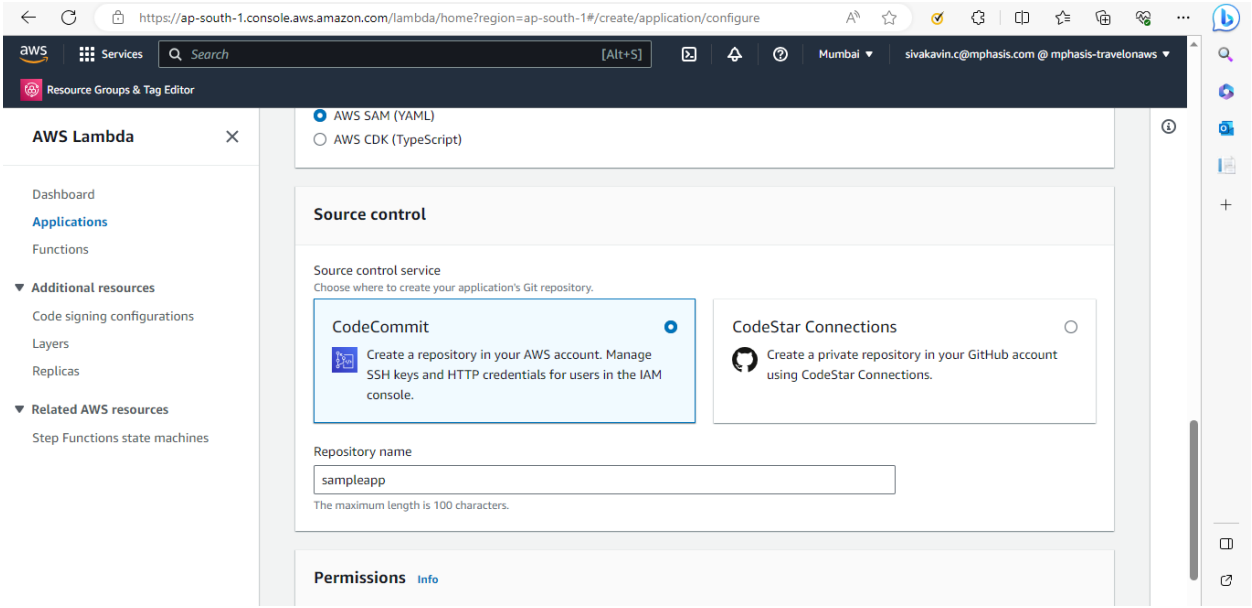


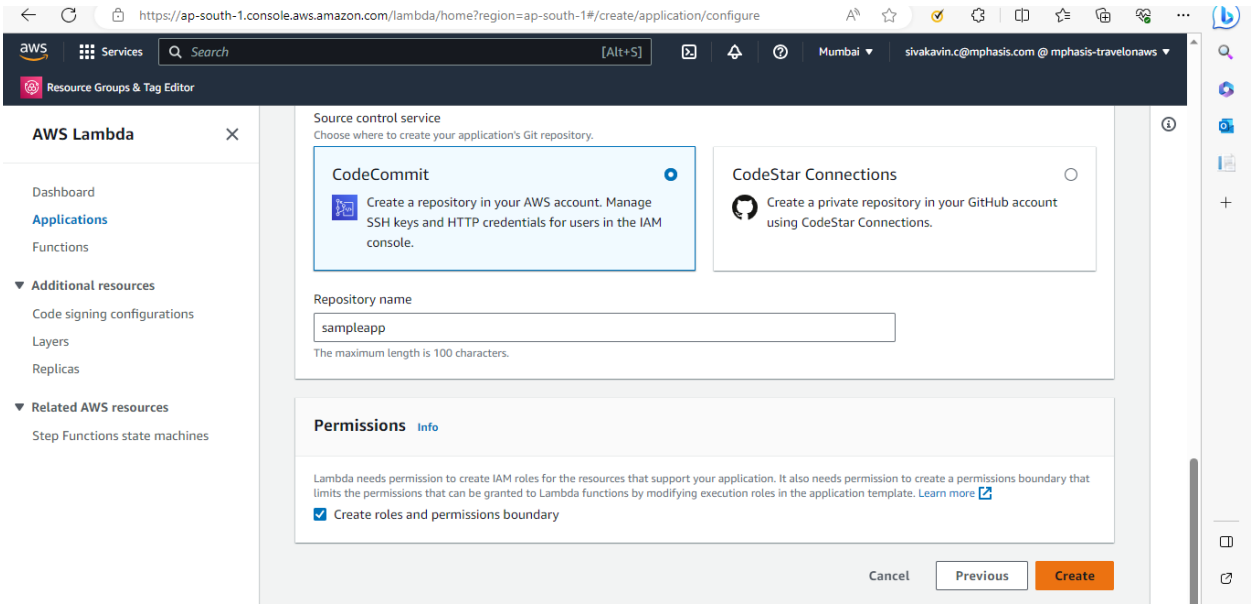

https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/create/application
aws
Services
Search
[Alt+S]
Mumbai
sivakavin.c@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Replicas
Related AWS resources
Step Functions state machines
Lambda > Applications > Create application
Create a Lambda application
An AWS Lambda application is a combination of Lambda functions, triggers, and other resources that work together to perform tasks. You can create an application with a continuous integration and continuous delivery (CI/CD) pipeline or from a serverless application in the AWS Serverless Application Repository.
User: arn:aws:iam::195737202275:user/sivakavin.c@mphasis.com is not authorized to perform: serverlessrepo:SearchApplications on resource: arn:aws:serverlessrepo:ap-south-1:195737202275:applications/*
CI/CD pipeline application
Serverless application
From scratch
Create a repository and pipeline to use with your own application.
Runtime:
Node.js 18.x
Uses:
Lambda


https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/create/application
aws
Services
Search
[Alt+S]
Mumbai
sivakavin.c@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Replicas
Related AWS resources
Step Functions state machines
Serverless API backend
A RESTful web API that uses DynamoDB to manage state.
Runtime:
Node.js 18.x
Uses:
API Gateway, DynamoDB, Lambda
File processing
Use Amazon S3 to trigger AWS Lambda to process data immediately after an upload. For example, you can use Lambda to thumbnail images, transcode videos, index files, process logs, validate content, and aggregate and filter data in real time.
Runtime:
Node.js 18.x
Uses:
Lambda, S3
Scheduled job
Schedule AWS Lambda functions using Amazon CloudWatch Events. This application creates a Lambda function that is triggered on a regular schedule.
Runtime:
Node.js 18.x
Uses:
CloudWatch Events, Lambda


https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/create/application
aws
Services
Search
[Alt+S]
Mumbai
sivakavin.c@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Replicas
Related AWS resources
Step Functions state machines
Runtime:
Node.js 18.x
Uses:
CloudWatch Events, Lambda
Notifications processing
Use a Lambda function to subscribe to an Amazon SNS topic. When a message is published to an SNS topic that has a Lambda function subscribed to it, the Lambda function is invoked with the payload of the published message.
Runtime:
Node.js 18.x
Uses:
Lambda, SNS
Queue processing
Use an AWS Lambda function to process messages from an Amazon SQS queue. With Amazon SQS, you can offload tasks from one component of your application by sending them to a queue and processing them asynchronously. Lambda polls the queue and invokes your function.
Runtime:
Node.js 18.x
Uses:
Lambda, SQS

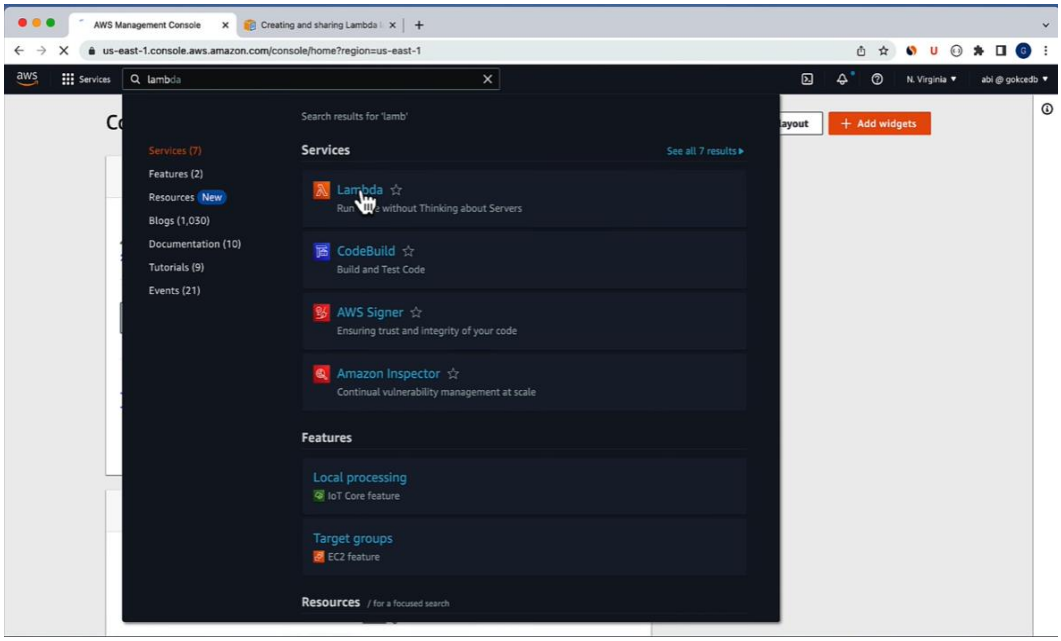

https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/create/application/configure
aws
Services
Search
[Alt+S]
Mumbai
sivakavin.c@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Replicas
Related AWS resources
Step Functions state machines
Application details
Name
sampleapp
Use only lowercase letters, numbers, or hyphens. The maximum length is 20 characters.
Language
Runtime
Node.js 18.x
Template format Info
AWS SAM (YAML)
AWS CDK (TypeScript)
Source control


https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/create/application/configure
Services
Search
[Alt+S]
Mumbai
sivakavin.c@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Replicas
Related AWS resources
Step Functions state machines
AWS SAM (YAML)
AWS CDK (TypeScript)
Source control
Source control service
Choose where to create your application's Git repository.
CodeCommit
Create a repository in your AWS account. Manage SSH keys and HTTP credentials for users in the IAM console.
CodeStar Connections
Create a private repository in your GitHub account using CodeStar Connections.
Repository name
sampleapp
The maximum length is 100 characters.
Permissions
Info

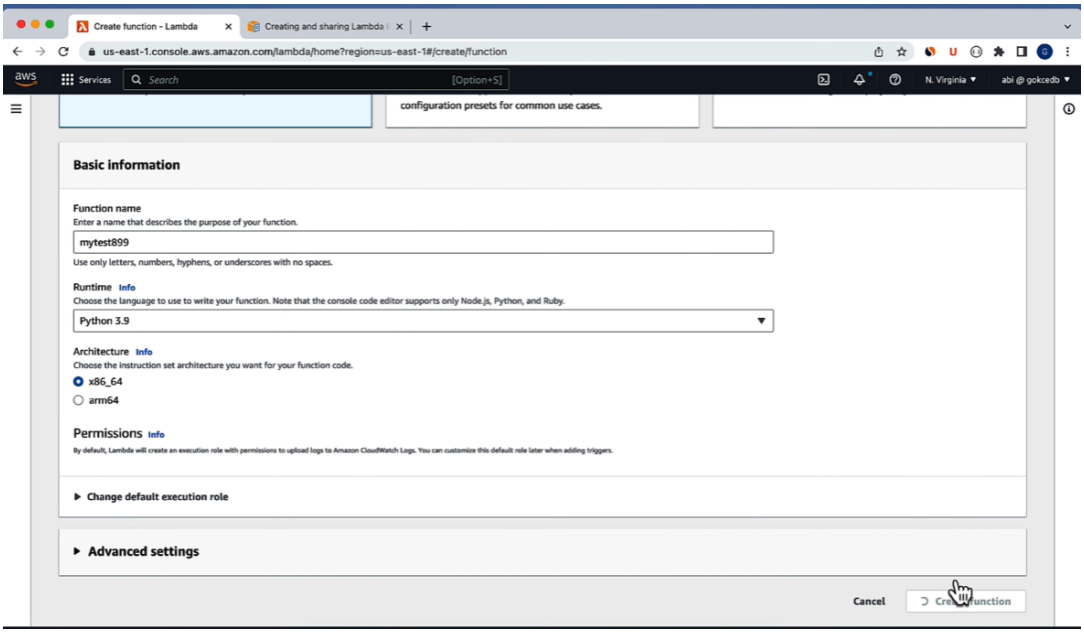

https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/create/application/configure
Services
Search
[Alt+S]
Mumbai
sivakavin.c@mphasis.com @ mphasis-travelonaws
Resource Groups & Tag Editor
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Replicas
Related AWS resources
Step Functions state machines
Source control service
Choose where to create your application's Git repository.
CodeCommit
Create a repository in your AWS account. Manage SSH keys and HTTP credentials for users in the IAM console.
CodeStar Connections
Create a private repository in your GitHub account using CodeStar Connections.
Repository name
sampleapp
The maximum length is 100 characters.
Permissions
Info
Lambda needs permission to create IAM roles for the resources that support your application. It also needs permission to create a permissions boundary that limits the permissions that can be granted to Lambda functions by modifying execution roles in the application template. Learn more
Create roles and permissions boundary
Cancel
Previous
Create

LAYERS

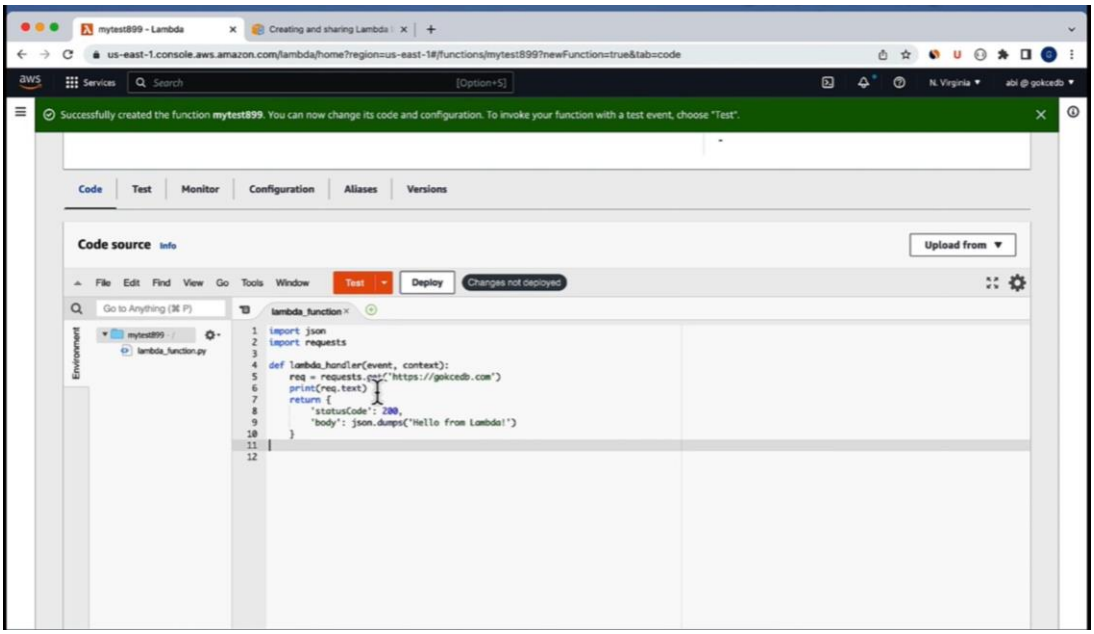
STEP 1: Search Lambda

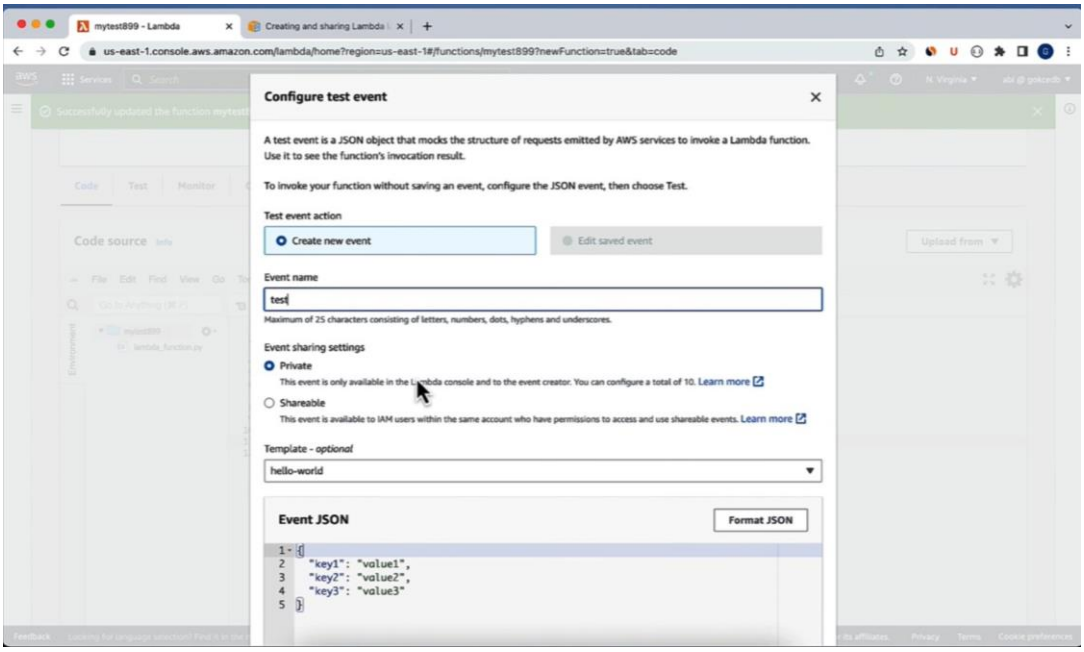


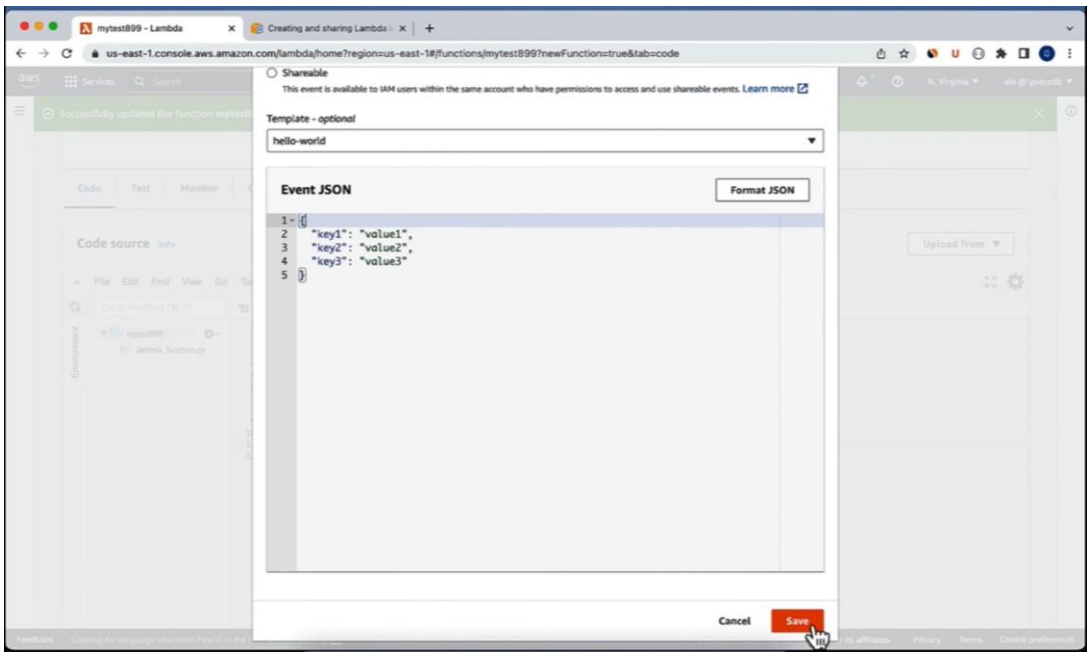
STEP 2: Create A Function

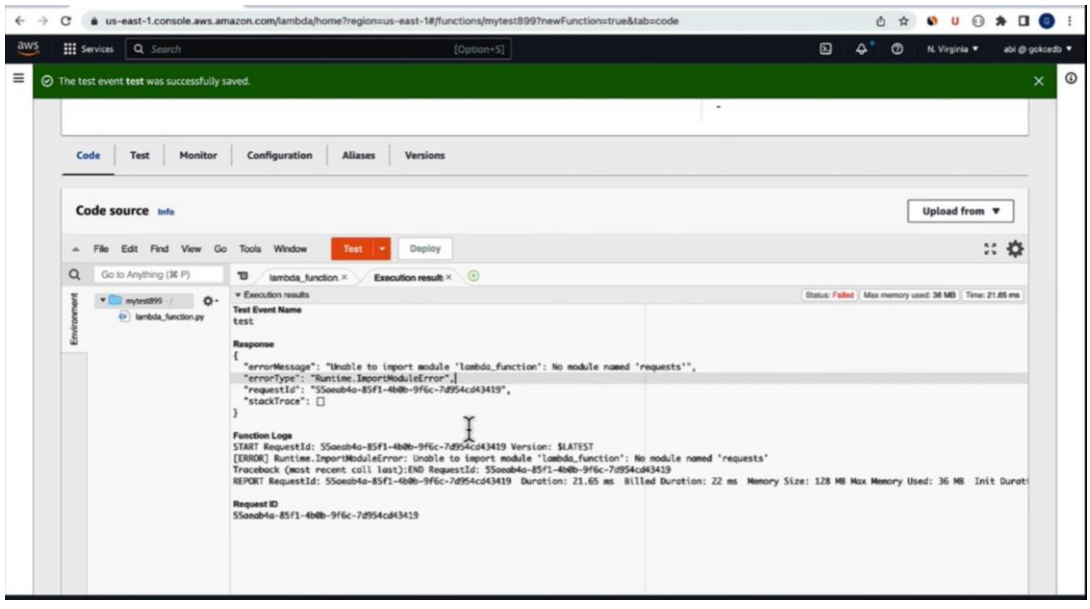


STEP 3: Deploy and Test The Code

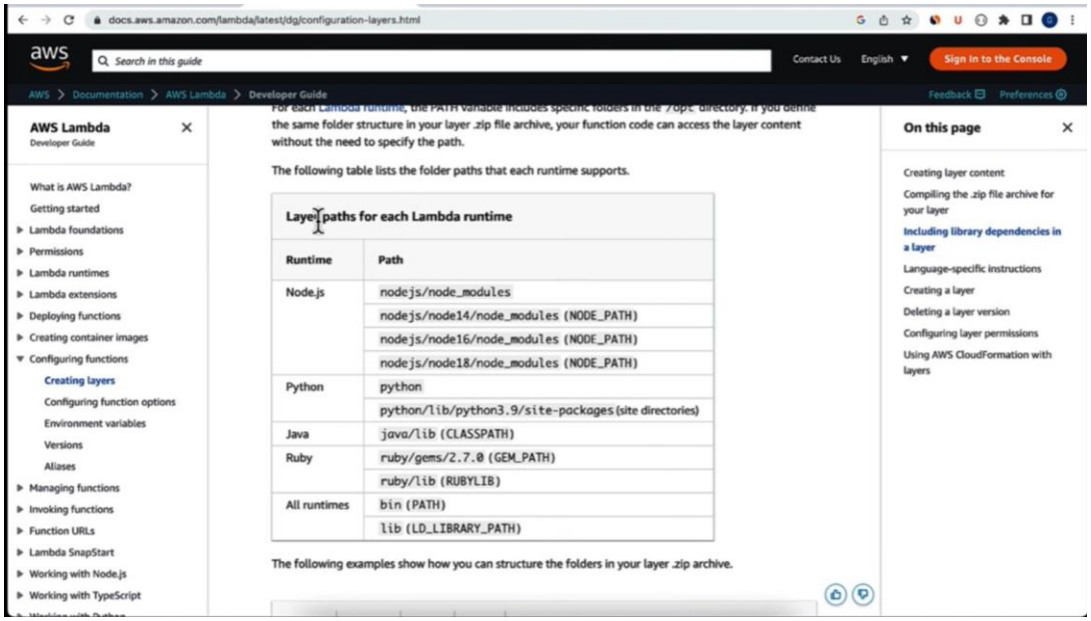


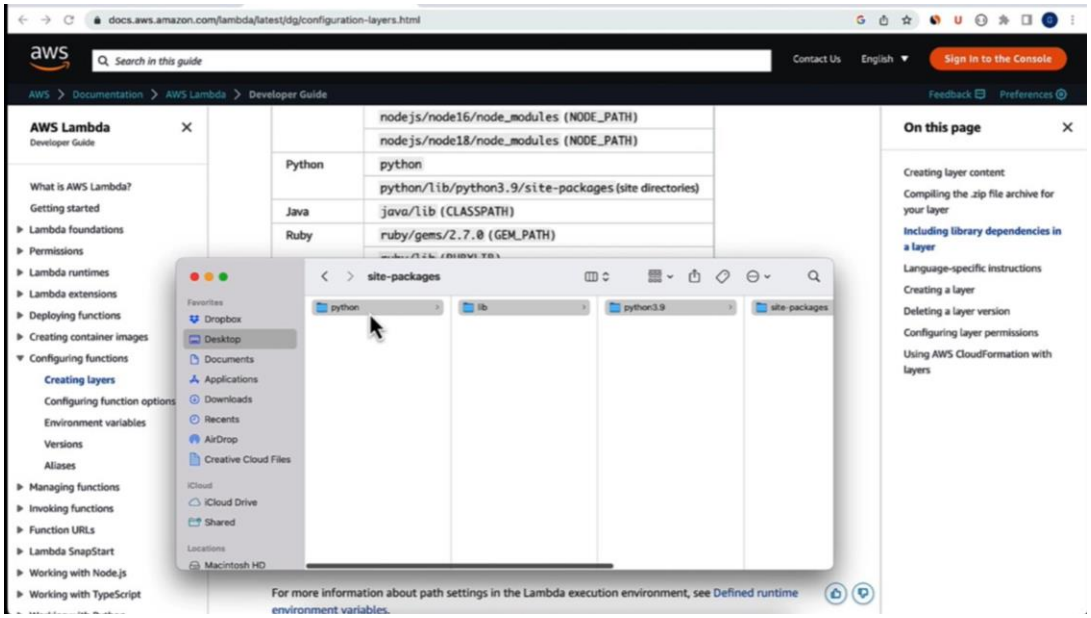


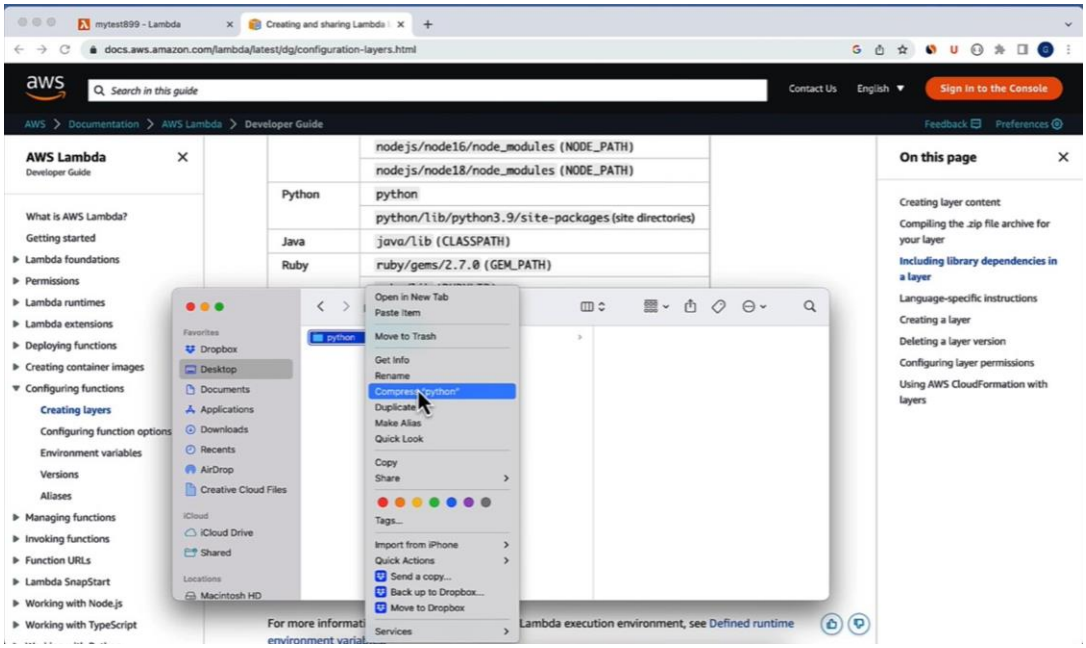

mytest899 - Lambda
Creating and sharing Lambda
us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/mytest899?newFunction=true&tab=code
Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more
Template - optional
hello-world
Event JSON
Format JSON
{
"key1": "value1",
"key2": "value2",
"key3": "value3"
}
Cancel
Save

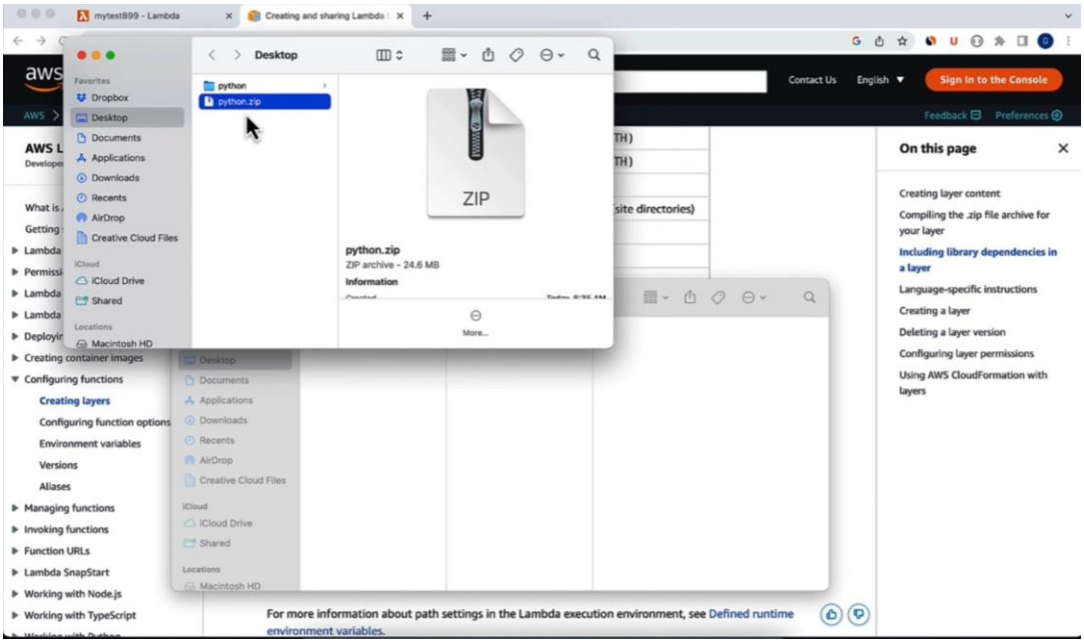

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/mytest899?newFunction=true&tab=code
The test event test was successfully saved.
Code
Test
Monitor
Configuration
Aliases
Versions
Code source
Info
Upload from
File Edit Find View Go Tools Window
Test
Deploy
Go to Anything (⌘ P)
lambda_function
Execution result
Environment
mytest899
lambda_function.py
Execution results
Test Event Name
test
Response
{
"errorMessage": "Unable to import module 'lambda_function': No module named 'requests'",
"errorType": "Runtime.ImportModuleError",
"requestId": "55aeab4a-85f1-4b0b-9f6c-7d954cd43419",
"stackTrace": []
}
Function Logs
START RequestId: 55aeab4a-85f1-4b0b-9f6c-7d954cd43419 Version: $LATEST
[ERROR] Runtime.ImportModuleError: Unable to import module 'lambda_function': No module named 'requests'
Traceback (most recent call last):END RequestId: 55aeab4a-85f1-4b0b-9f6c-7d954cd43419
REPORT RequestId: 55aeab4a-85f1-4b0b-9f6c-7d954cd43419 Duration: 21.65 ms Billed Duration: 22 ms Memory Size: 128 MB Max Memory Used: 36 MB
Request ID
55aeab4a-85f1-4b0b-9f6c-7d954cd43419

STEP 4: Compressing Package Dependencies into A Zip File

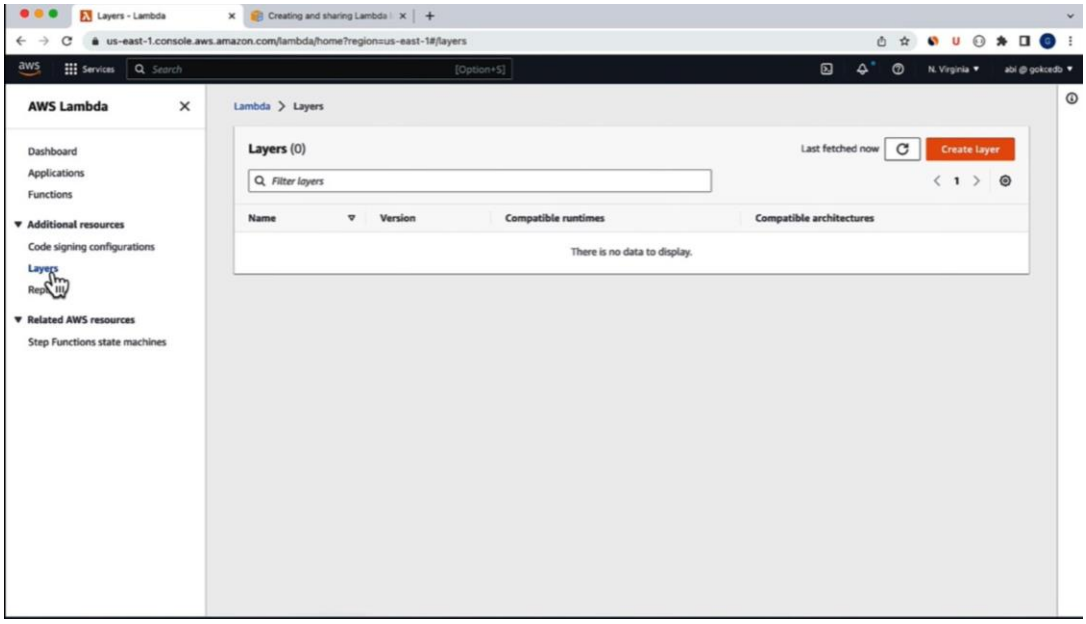




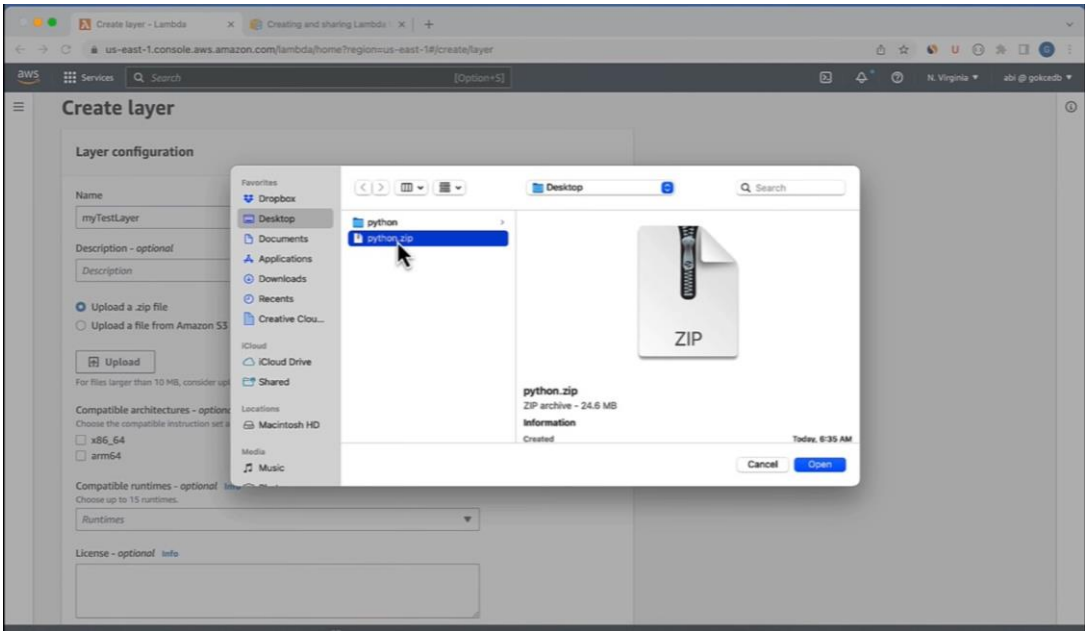

mytest899 - Lambda
Creating and sharing Lambda
docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html
Search in this guide
Contact Us
English
Sign in to the Console
AWS > Documentation > AWS Lambda > Developer Guide
Feedback
Preferences
AWS Lambda
Developer Guide
What is AWS Lambda?
Getting started
Lambda foundations
Permissions
Lambda runtimes
Lambda extensions
Deploying functions
Creating container images
Configuring functions
Creating layers
Configuring function options
Environment variables
Versions
Aliases
Managing functions
Invoking functions
Function URLs
Lambda SnapStart
Working with Node.js
Working with TypeScript
nodejs/node16/node_modules (NODE_PATH)
nodejs/node18/node_modules (NODE_PATH)
Python
python
python/lib/python3.9/site-packages (site directories)
Java
java/lib (CLASSPATH)
Ruby
ruby/gems/2.7.0 (GEM_PATH)
Open in New Tab
Paste Item
Move to Trash
Get Info
Rename
Compress "python"
Duplicate
Make Alias
Quick Look
Copy
Share
Tags...
Import from iPhone
Quick Actions
Send a copy...
Back up to Dropbox...
Move to Dropbox
Services
Favorites
Dropbox
Desktop
Documents
Applications
Downloads
Recents
AirDrop
Creative Cloud Files
iCloud
iCloud Drive
Shared
Locations
Macintosh HD
On this page
Creating layer content
Compiling the .zip file archive for your layer
Including library dependencies in a layer
Language-specific instructions
Creating a layer
Deleting a layer version
Configuring layer permissions
Using AWS CloudFormation with layers
For more information
Lambda execution environment, see Defined runtime
environment variables.

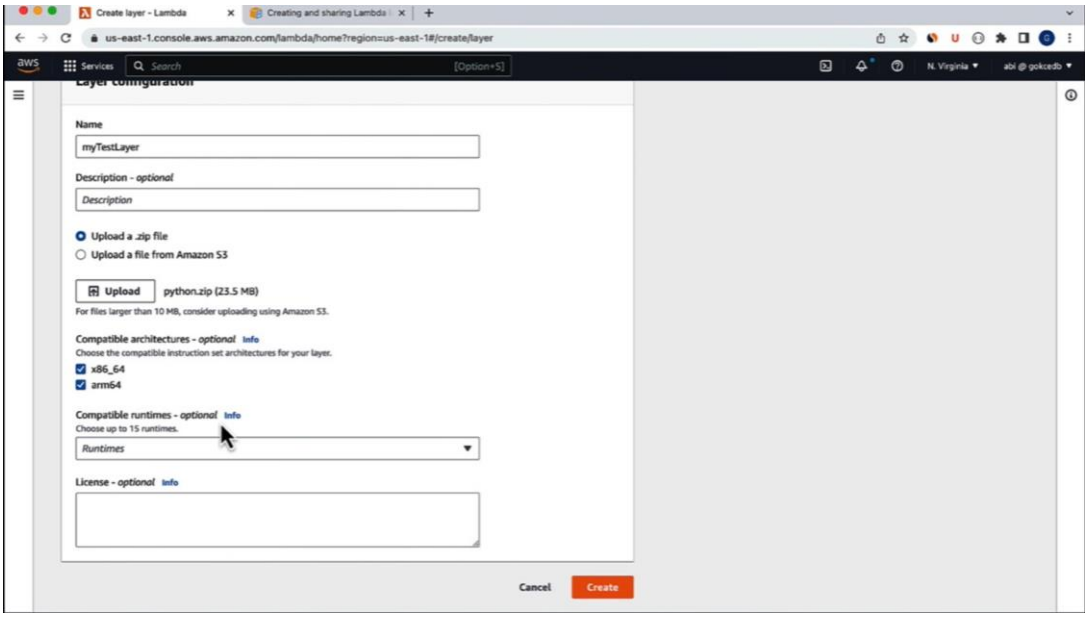

mytest899 - Lambda
Creating and sharing Lambda
Desktop
python
python.zip
ZIP
python.zip
ZIP archive - 24.6 MB
Information
More...
Contact Us
English
Sign in to the Console
Feedback
Preferences
On this page
Creating layer content
Compiling the .zip file archive for your layer
Including library dependencies in a layer
Language-specific instructions
Creating a layer
Deleting a layer version
Configuring layer permissions
Using AWS CloudFormation with layers
For more information about path settings in the Lambda execution environment, see Defined runtime
environment variables.

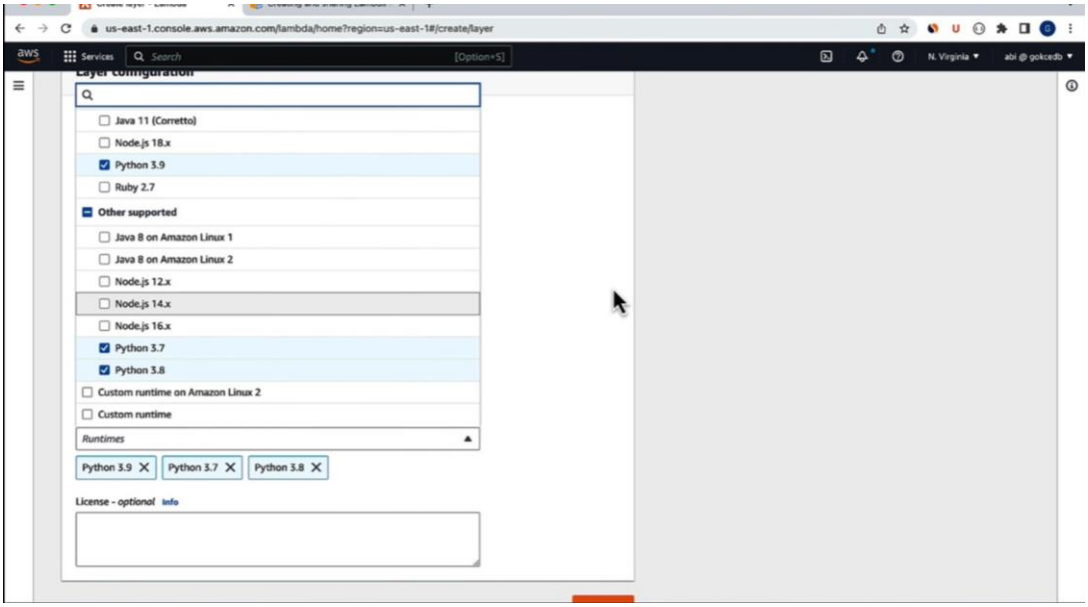
STEP 5: In Layers, Create A New Layer

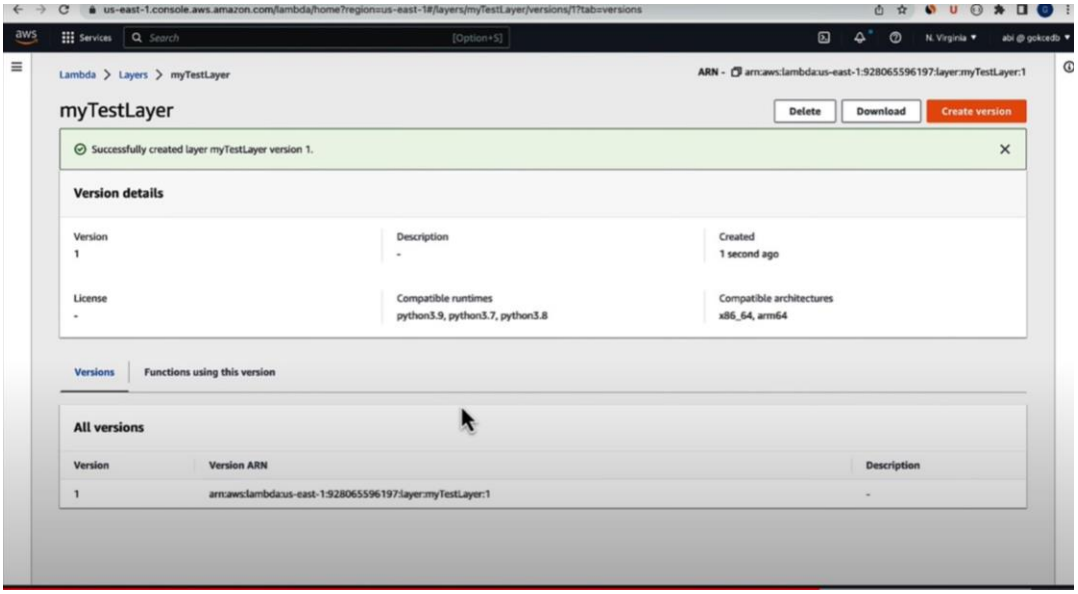


STEP 6: Upload the Zip File And Create A Layer

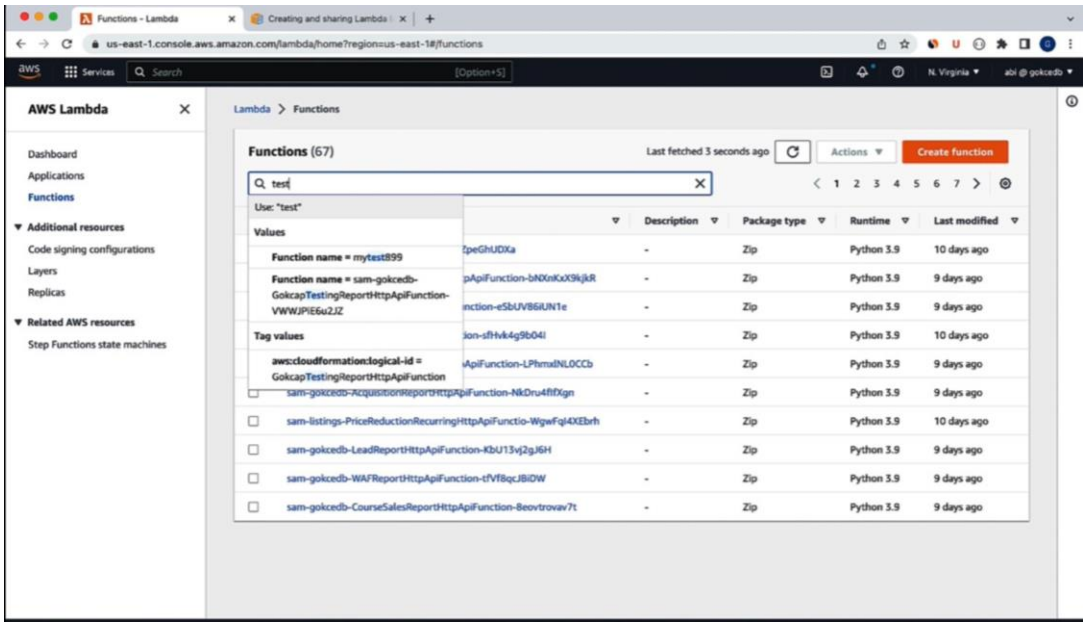


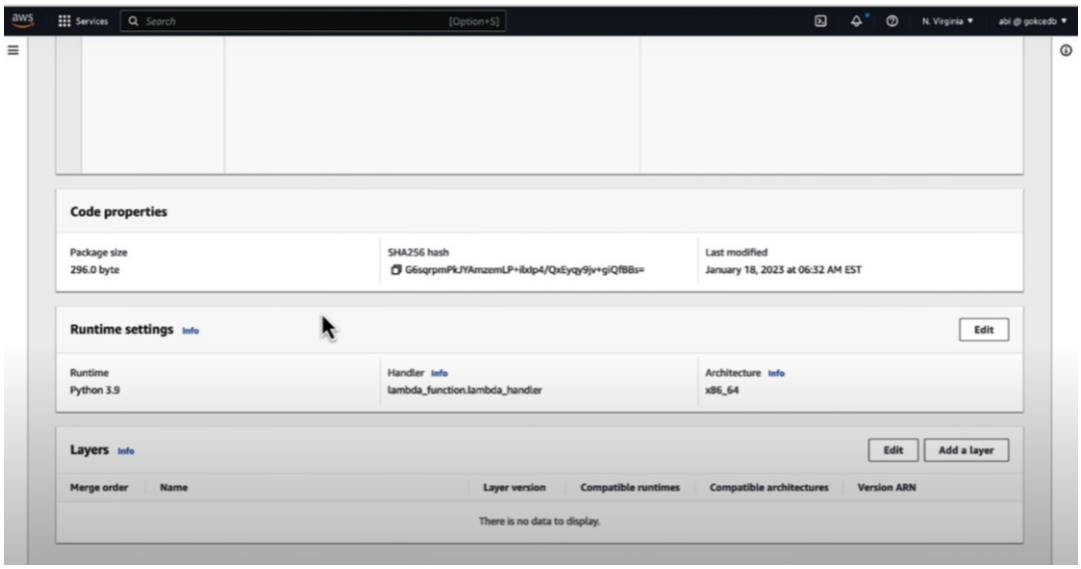

Create layer - Lambda
Creating and sharing Lambda
us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/layer
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Layer configuration
Name
myTestLayer
Description - optional
Description
Upload a .zip file
Upload a file from Amazon S3
Upload
python.zip (23.5 MB)
For files larger than 10 MB, consider uploading using Amazon S3.
Compatible architectures - optional Info
Choose the compatible instruction set architectures for your layer.
x86_64
arm64
Compatible runtimes - optional Info
Choose up to 15 runtimes.
Runtimes
License - optional Info
Cancel
Create

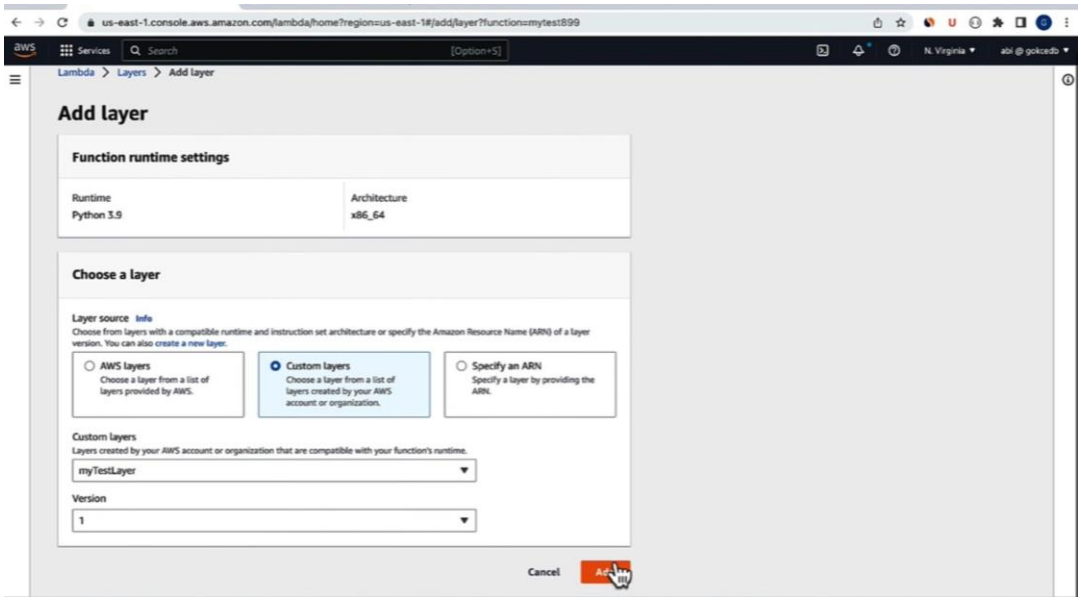

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/layer
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Layer configuration
Java 11 (Corretto)
Node.js 18.x
Python 3.9
Ruby 2.7
Other supported
Java 8 on Amazon Linux 1
Java 8 on Amazon Linux 2
Node.js 12.x
Node.js 14.x
Node.js 16.x
Python 3.7
Python 3.8
Custom runtime on Amazon Linux 2
Custom runtime
Runtimes
Python 3.9
Python 3.7
Python 3.8
License - optional Info



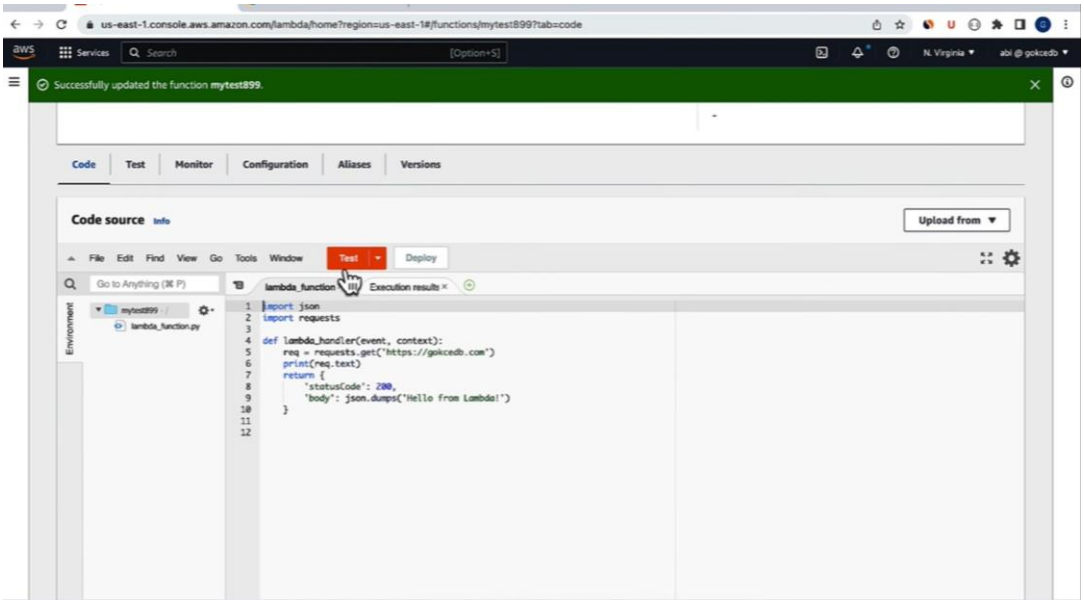
STEP 7: In the Created Function, Add A Layer

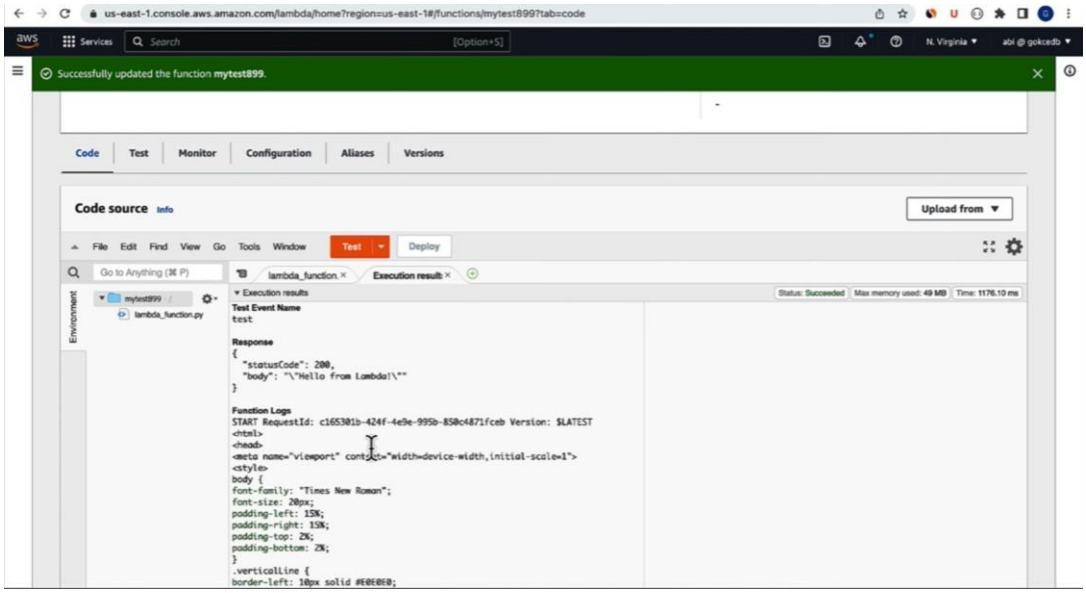



Services
Search
[Option+S]
N. Virginia
Code properties
Package size
296.0 byte
SHA256 hash
G6sqrpmPkJYAmzemLP+ilxlp4/QxEyqy9jv+gjQfBBs=
Last modified
January 18, 2023 at 06:32 AM EST
Runtime settings Info
Edit
Runtime
Python 3.9
Handler Info
lambda_function.lambda_handler
Architecture Info
x86_64
Layers Info
Edit
Add a layer
Merge order
Name
Layer version
Compatible runtimes
Compatible architectures
Version ARN
There is no data to display.


us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/add/layer?function=mytest899
Services
Search
[Option+S]
N. Virginia
Lambda > Layers > Add layer
Add layer
Function runtime settings
Runtime
Python 3.9
Architecture
x86_64
Choose a layer
Layer source Info
Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also create a new layer.
AWS layers
Choose a layer from a list of layers provided by AWS.
Custom layers
Choose a layer from a list of layers created by your AWS account or organization.
Specify an ARN
Specify a layer by providing the ARN.
Custom layers
Layers created by your AWS account or organization that are compatible with your function's runtime.
myTestLayer
Version
1
Cancel

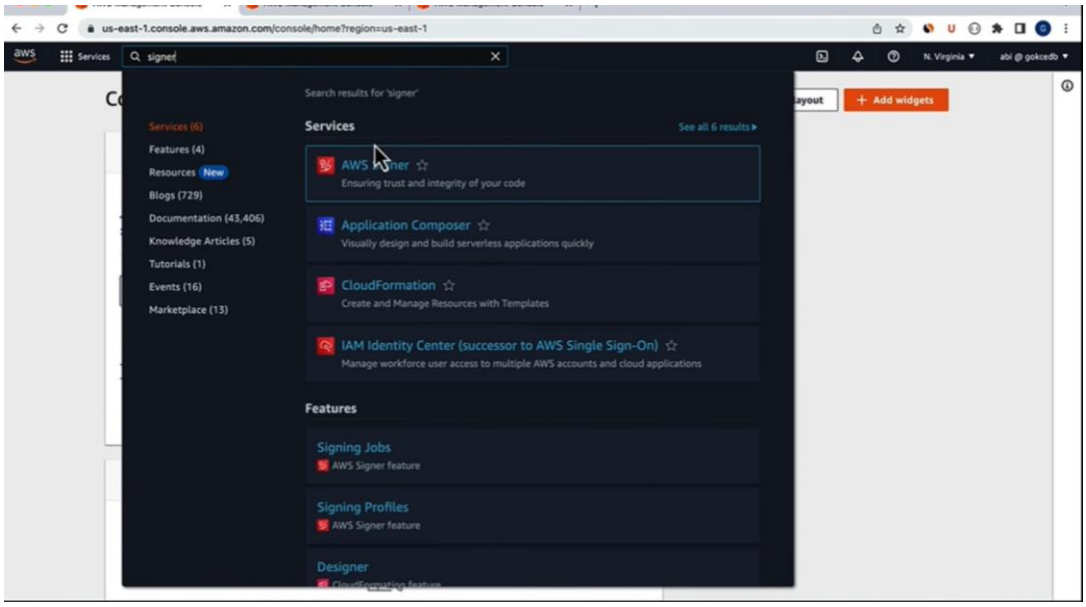
STEP 8: Test the Code



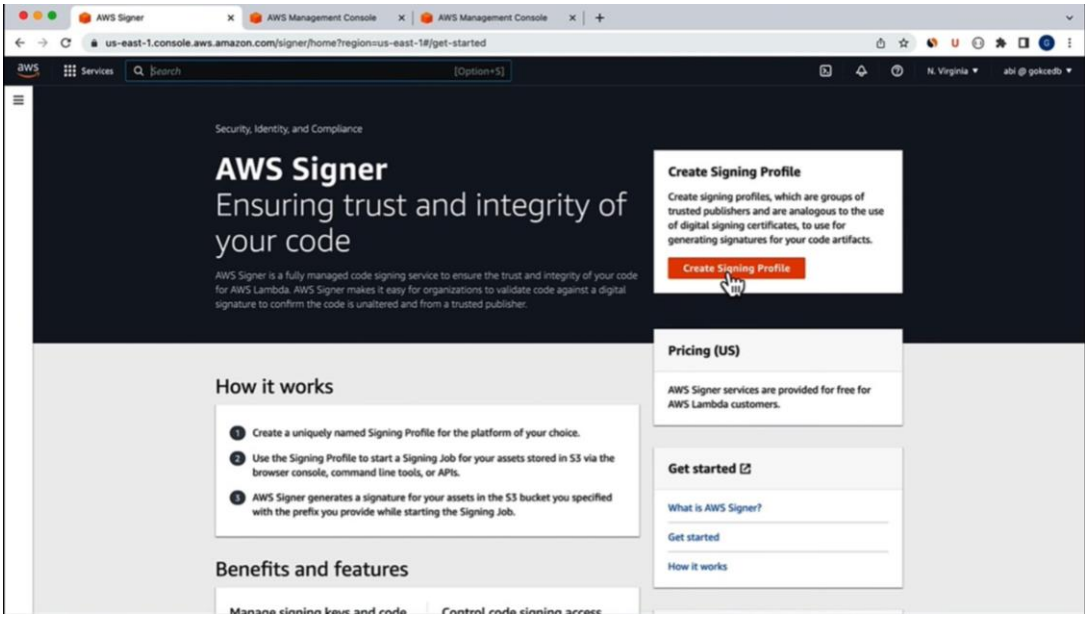


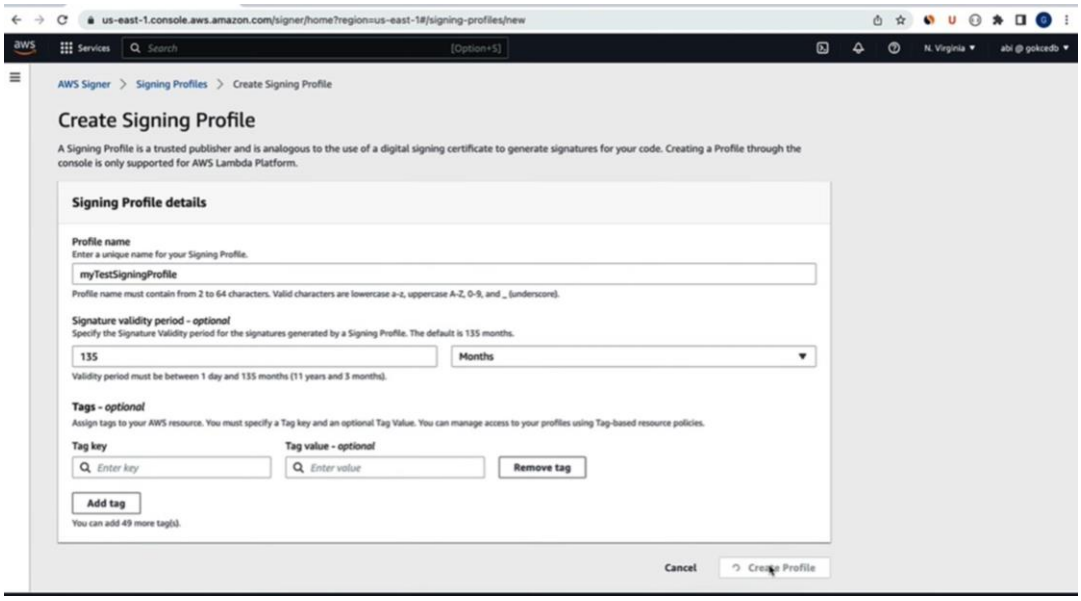
CODE SIGNING CONFIGURATIONS

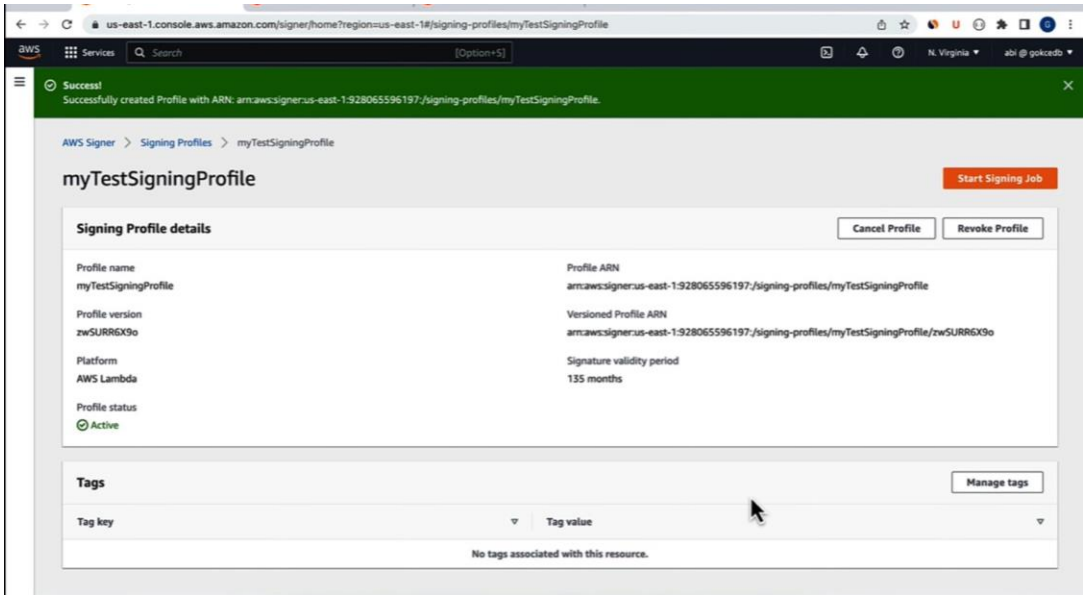
STEP 1: Search Aws Signer



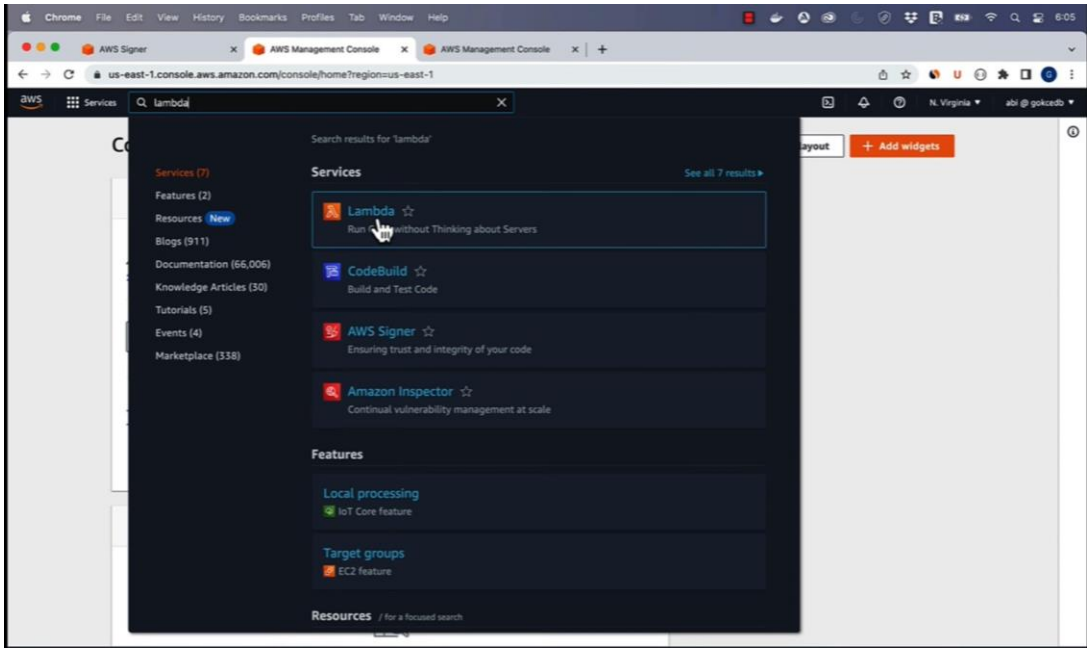
STEP 2: Create Signing Profile



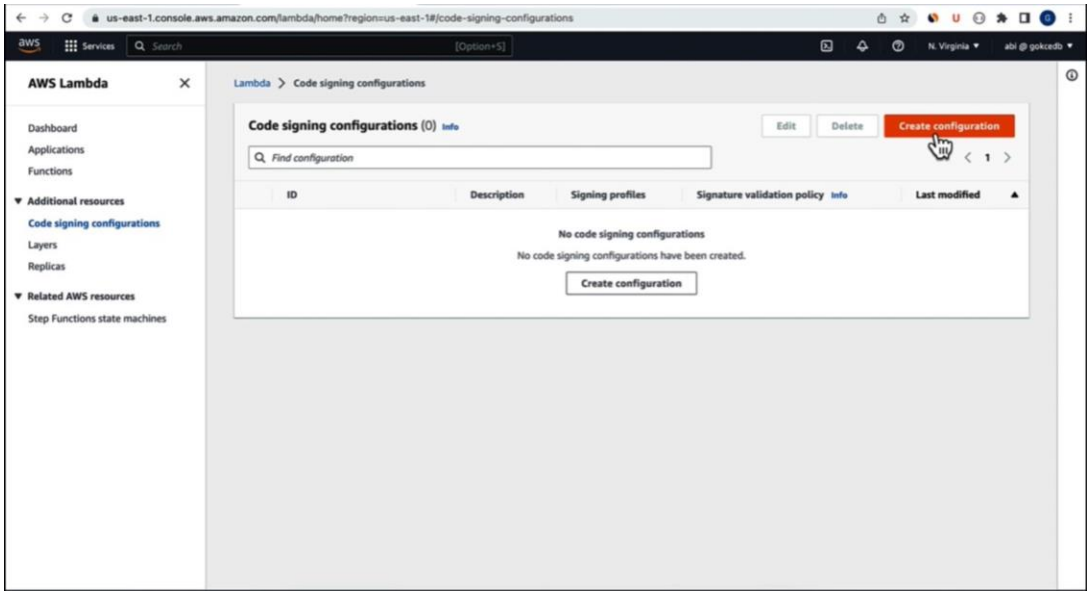

us-east-1.console.aws.amazon.com/signer/home?region=us-east-1#/signing-profiles/new
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
AWS Signer > Signing Profiles > Create Signing Profile
Create Signing Profile
A Signing Profile is a trusted publisher and is analogous to the use of a digital signing certificate to generate signatures for your code. Creating a Profile through the console is only supported for AWS Lambda Platform.
Signing Profile details
Profile name
Enter a unique name for your Signing Profile.
myTestSigningProfile
Profile name must contain from 2 to 64 characters. Valid characters are lowercase a-z, uppercase A-Z, 0-9, and _ (underscore).
Signature validity period - optional
Specify the Signature Validity period for the signatures generated by a Signing Profile. The default is 135 months.
135
Months
Validity period must be between 1 day and 135 months (11 years and 3 months).
Tags - optional
Assign tags to your AWS resource. You must specify a Tag key and an optional Tag Value. You can manage access to your profiles using Tag-based resource policies.
Tag key
Tag value - optional
Enter key
Enter value
Remove tag
Add tag
You can add 49 more tag(s).
Cancel
Create Profile

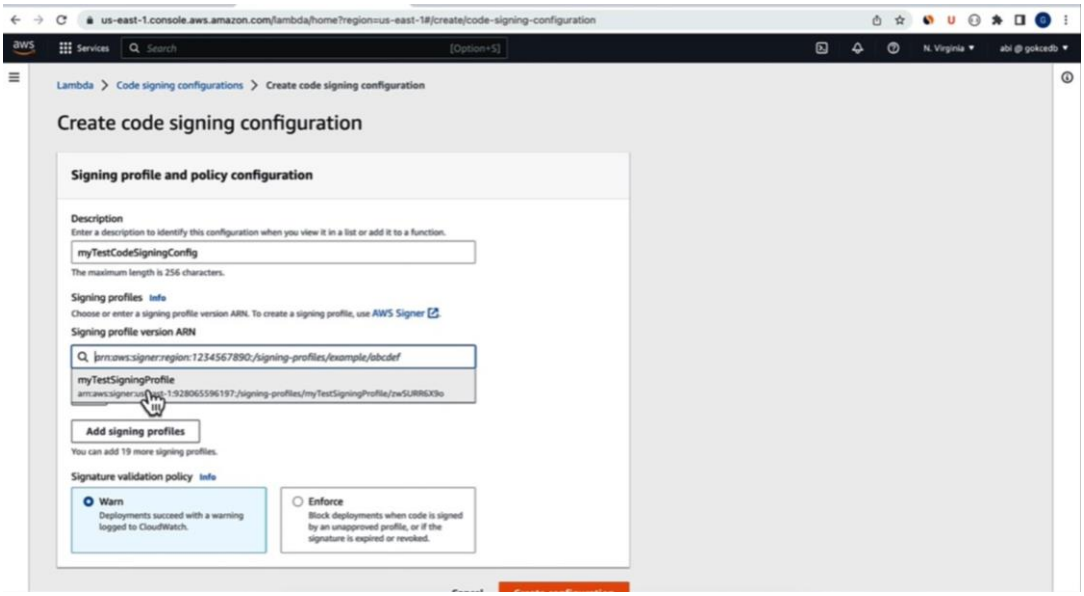

us-east-1.console.aws.amazon.com/signer/home?region=us-east-1#/signing-profiles/myTestSigningProfile
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Success!
Successfully created Profile with ARN: arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile.
AWS Signer > Signing Profiles > myTestSigningProfile
myTestSigningProfile
Start Signing Job
Signing Profile details
Cancel Profile
Revoke Profile
Profile name
myTestSigningProfile
Profile ARN
arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile
Profile version
zwSURR6X9o
Versioned Profile ARN
arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile/zwSURR6X9o
Platform
AWS Lambda
Signature validity period
135 months
Profile status
Active
Tags
Manage tags
Tag key
Tag value
No tags associated with this resource.

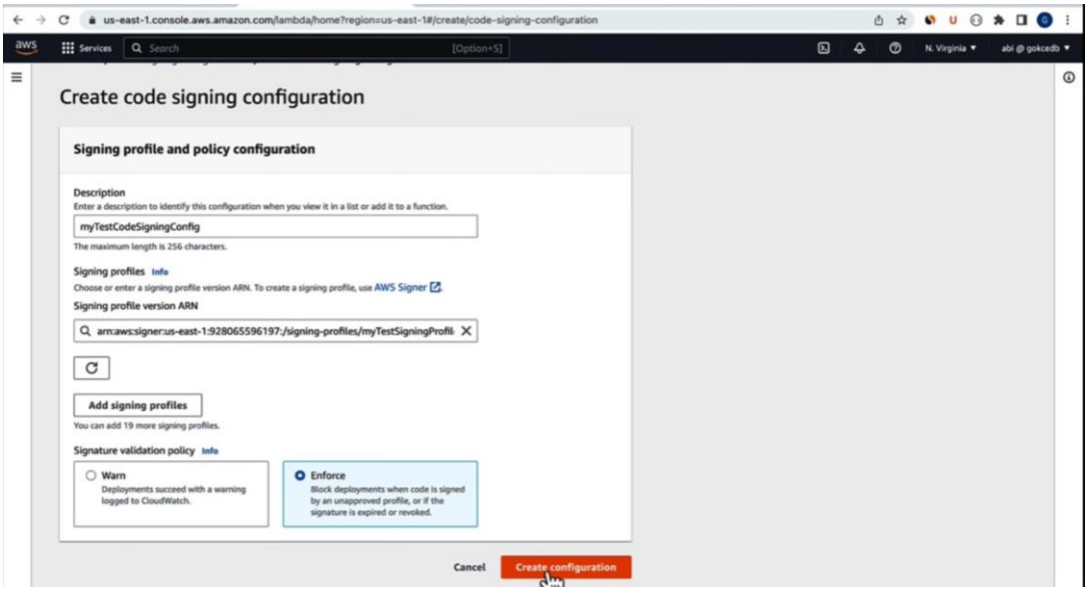
STEP 3: Navigate to Lambda

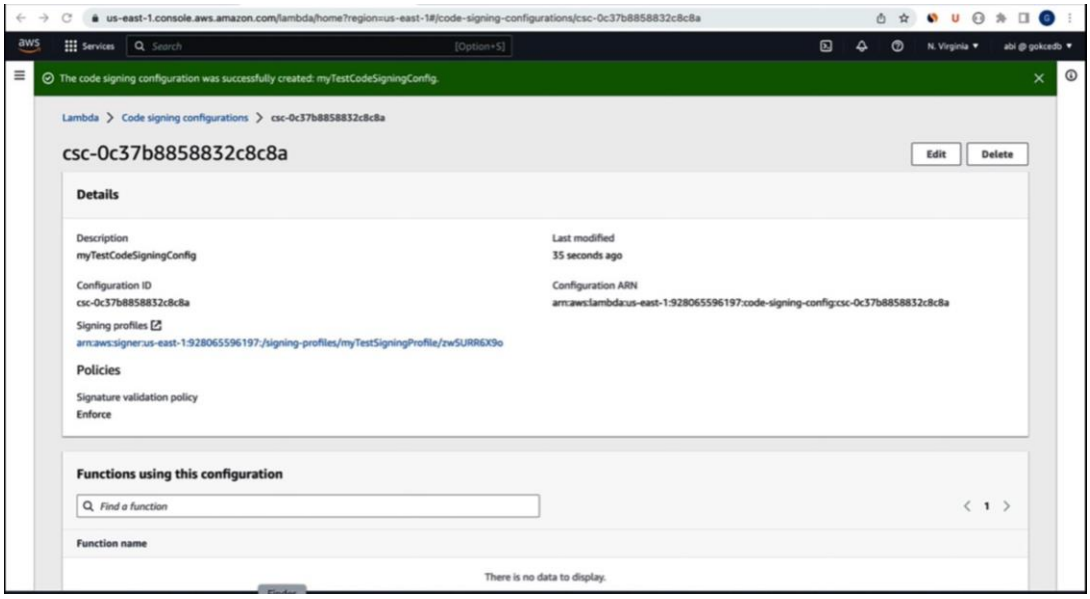


STEP 4: In Code Signing Configurations, Create A Configuration

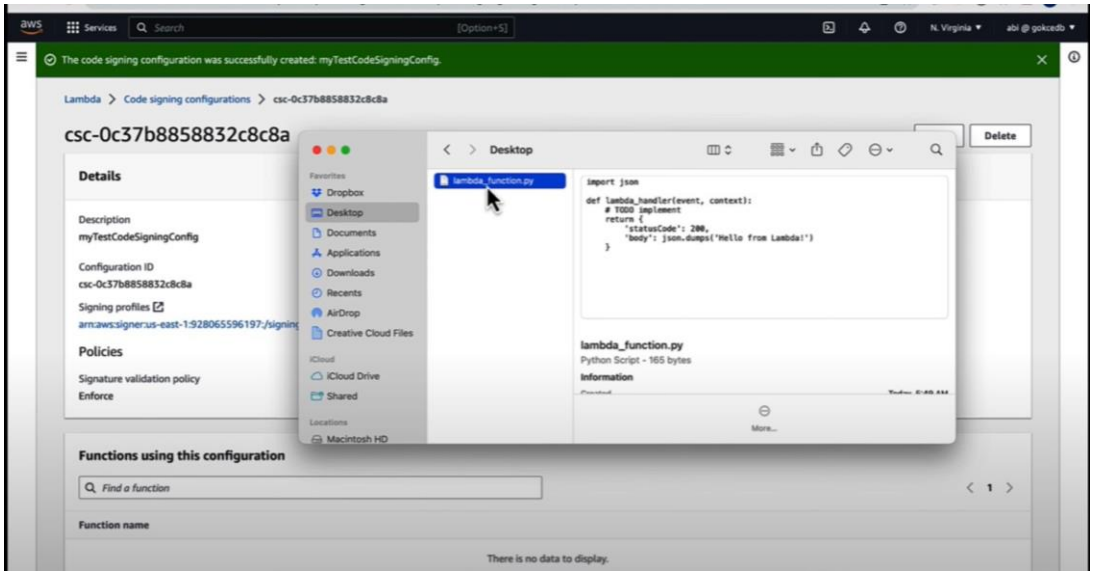


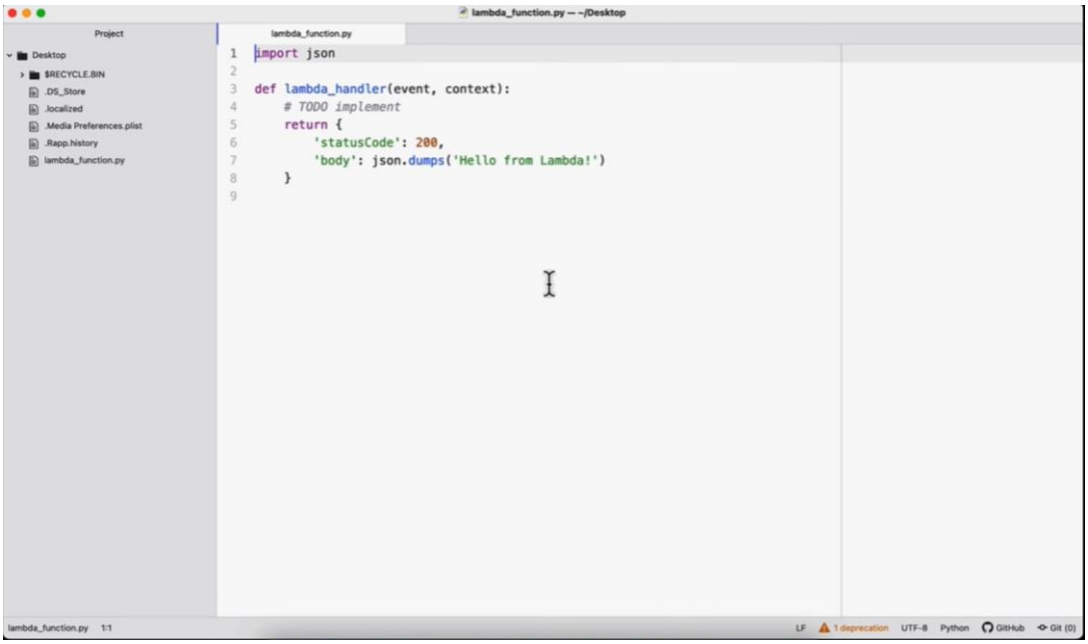

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/code-signing-configuration
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Lambda > Code signing configurations > Create code signing configuration
Create code signing configuration
Signing profile and policy configuration
Description
Enter a description to identify this configuration when you view it in a list or add it to a function.
myTestCodeSigningConfig
The maximum length is 256 characters.
Signing profiles Info
Choose or enter a signing profile version ARN. To create a signing profile, use AWS Signer.
Signing profile version ARN
arn:aws:signer:region:123456789O:/signing-profiles/example/abcdef
myTestSigningProfile
Add signing profiles
You can add 19 more signing profiles.
Signature validation policy Info
Warn
Deployments succeed with a warning logged to CloudWatch.
Enforce
Block deployments when code is signed by an unapproved profile, or if the signature is expired or revoked.

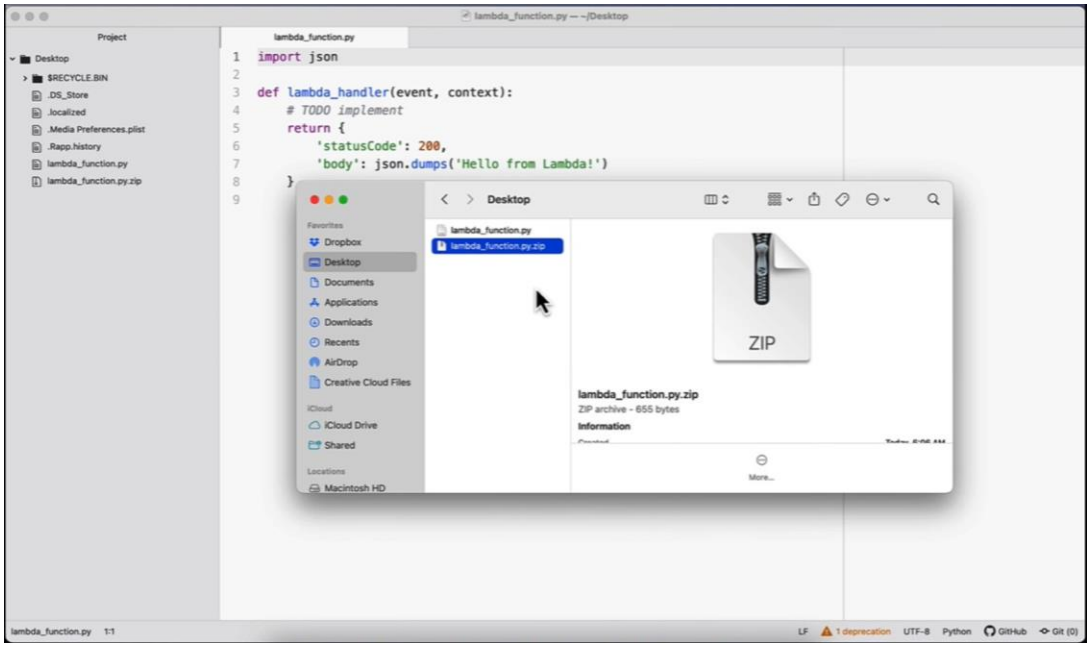

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/code-signing-configuration
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Create code signing configuration
Signing profile and policy configuration
Description
Enter a description to identify this configuration when you view it in a list or add it to a function.
myTestCodeSigningConfig
The maximum length is 256 characters.
Signing profiles Info
Choose or enter a signing profile version ARN. To create a signing profile, use AWS Signer.
Signing profile version ARN
arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfi
Add signing profiles
You can add 19 more signing profiles.
Signature validation policy Info
Warn
Deployments succeed with a warning logged to CloudWatch.
Enforce
Block deployments when code is signed by an unapproved profile, or if the signature is expired or revoked.
Cancel
Create configuration



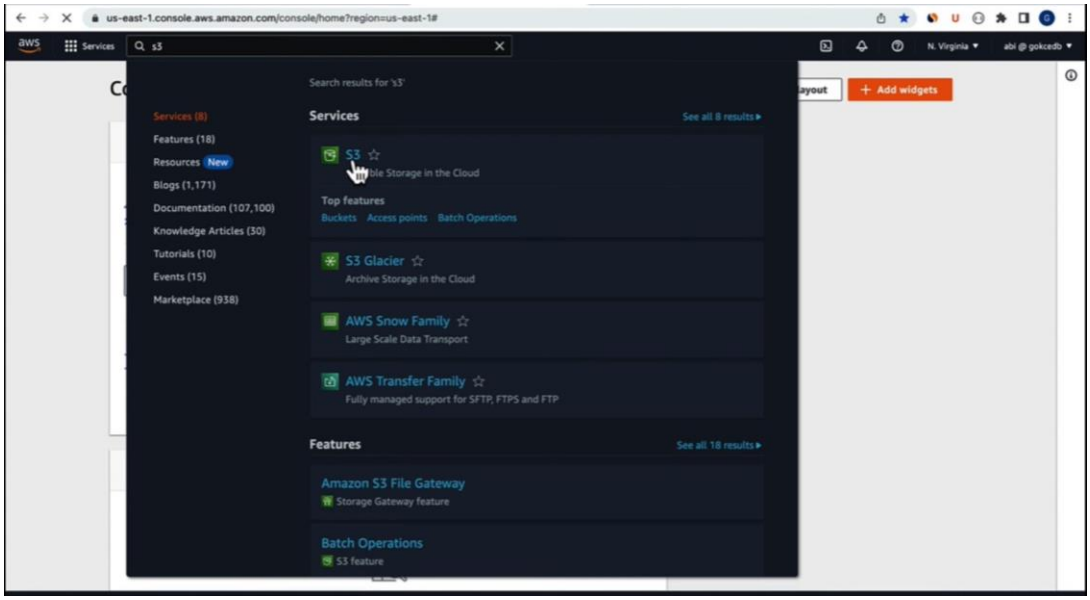
STEP 5: Compressing the Code File Into A Zip File



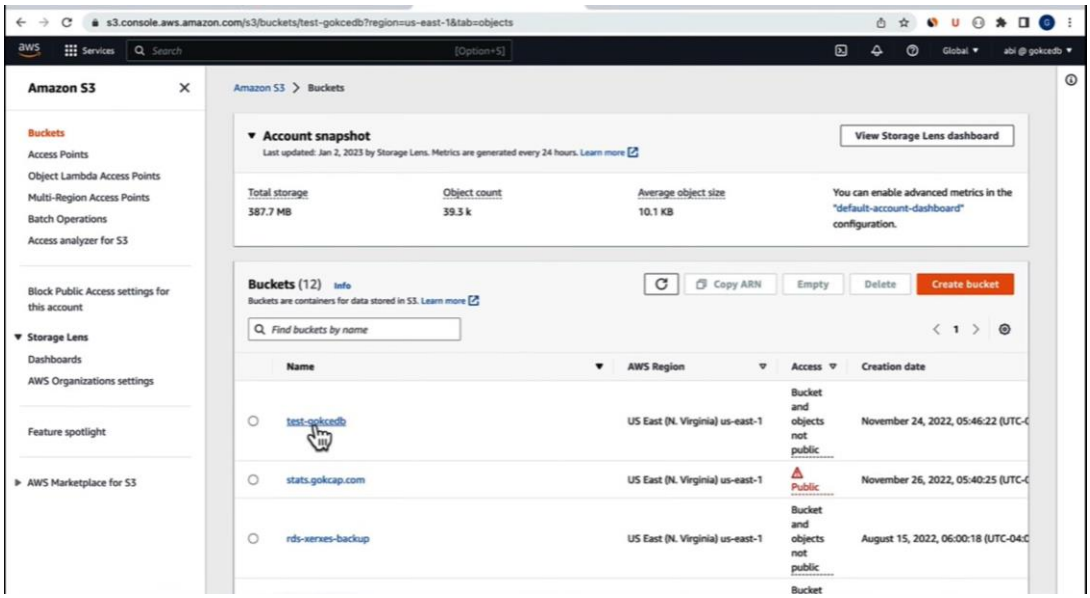

lambda_function.py — ~/Desktop
Project
Desktop
$RECYCLE.BIN
.DS_Store
.localized
.Media Preferences.plist
.Rapp.history
lambda_function.py
lambda_function.py
import json
def lambda_handler(event, context):
# TODO implement
return {
'statusCode': 200,
'body': json.dumps('Hello from Lambda!')
}
lambda_function.py 1:1
LF
1 deprecation
UTF-8
Python
GitHub
Git (0)


lambda_function.py — ~/Desktop
Project
Desktop
$RECYCLE.BIN
.DS_Store
.localized
.Media Preferences.plist
.Rapp.history
lambda_function.py
lambda_function.py.zip
lambda_function.py
import json
def lambda_handler(event, context):
# TODO implement
return {
'statusCode': 200,
'body': json.dumps('Hello from Lambda!')
}
Desktop
Favorites
Dropbox
Desktop
Documents
Applications
Downloads
Recents
AirDrop
Creative Cloud Files
iCloud
iCloud Drive
Shared
Locations
Macintosh HD
lambda_function.py
lambda_function.py.zip
ZIP
lambda_function.py.zip
ZIP archive - 655 bytes
Information
More...
lambda_function.py 1:1
LF
1 deprecation
UTF-8
Python
GitHub
Git (0)

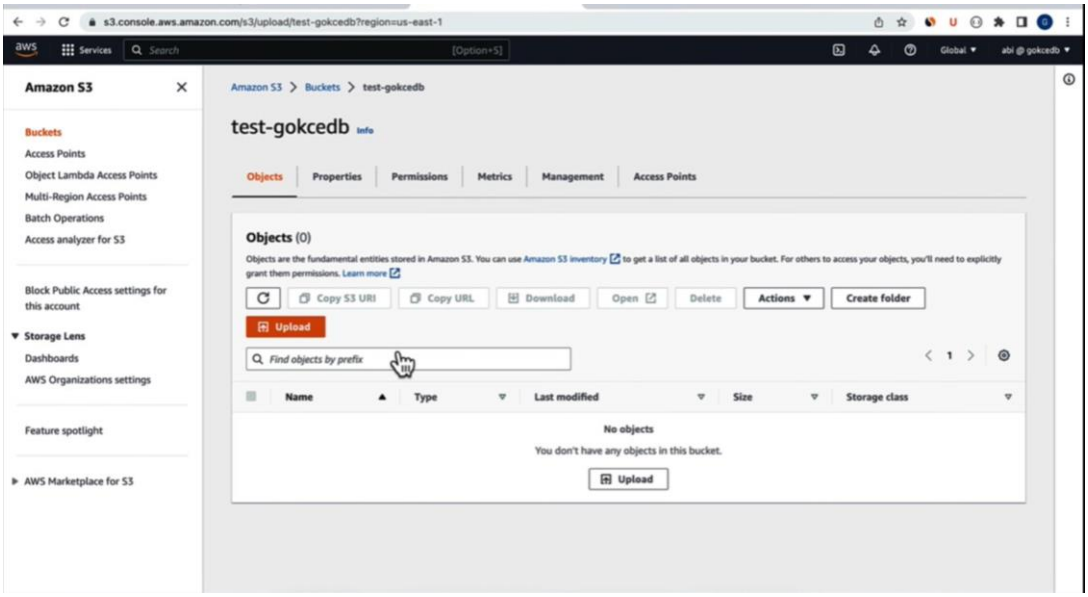
STEP 6: Navigate to S3

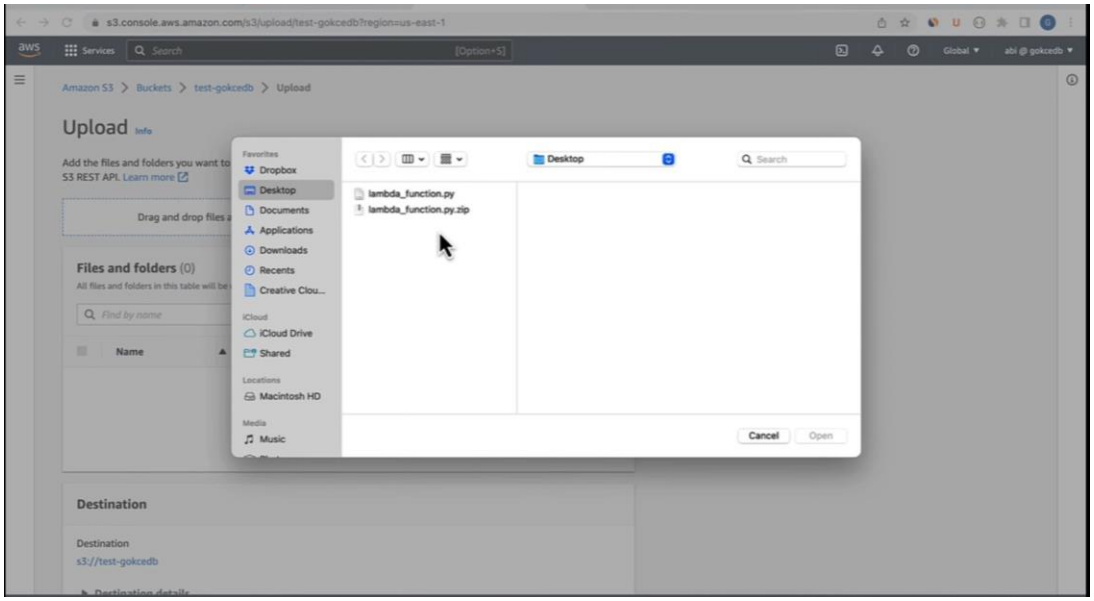


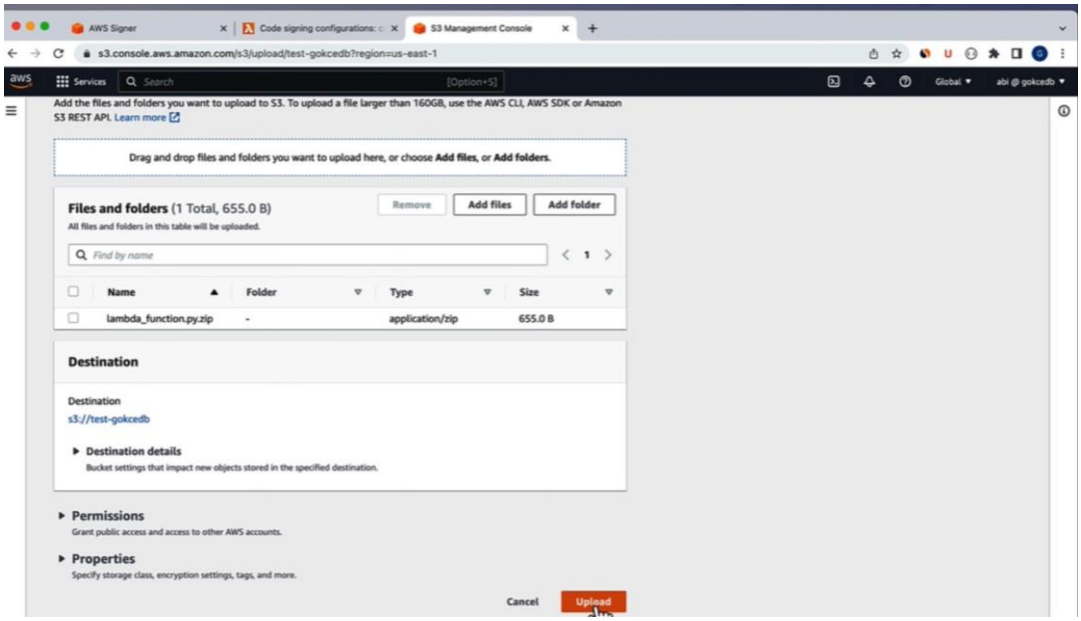
STEP 7: Select an Already Created Bucket

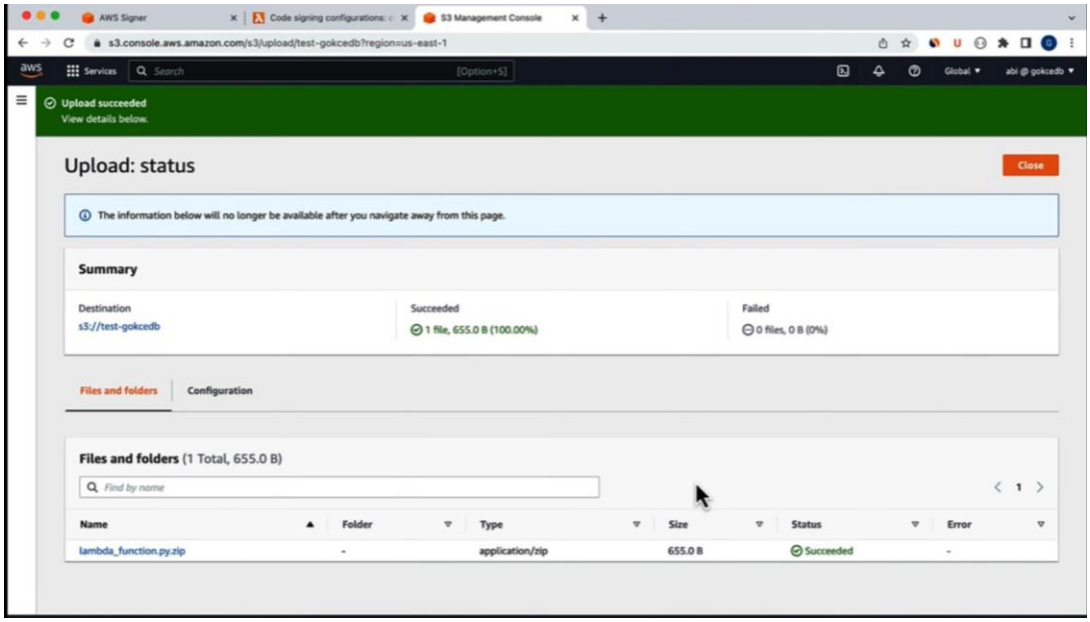


STEP 8: Upload the Zip File

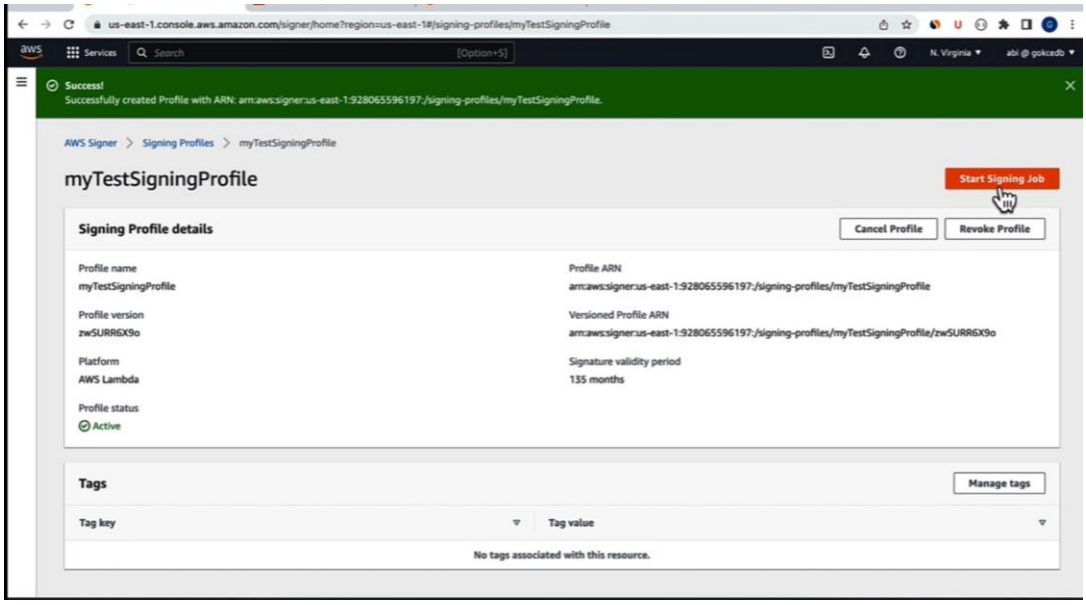


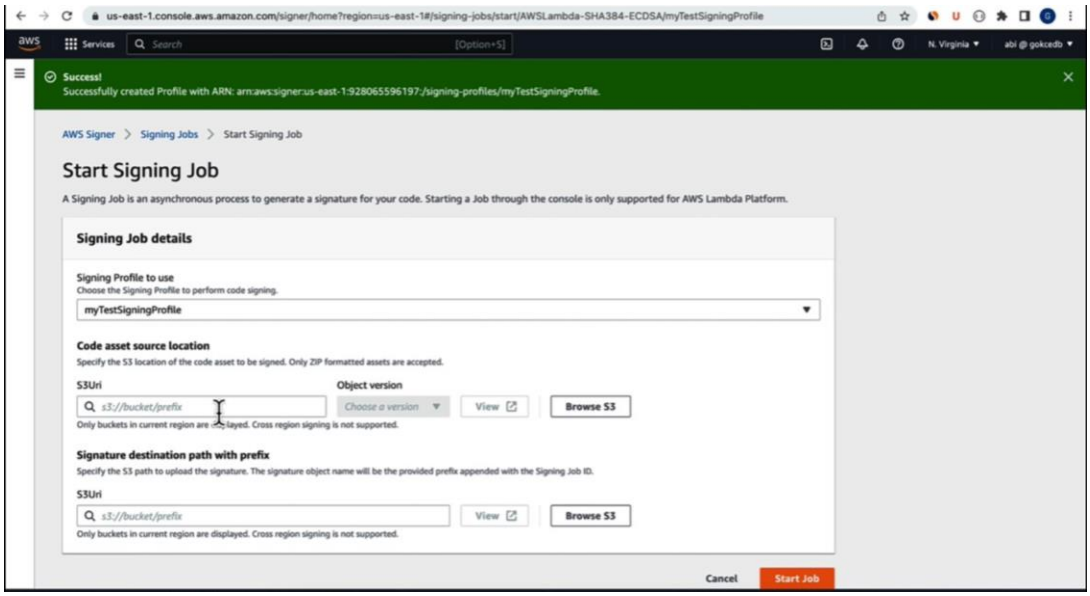



AWS Signer
Code signing configurations:
S3 Management Console
s3.console.aws.amazon.com/s3/upload/test-gokcedb?region=us-east-1
Services
Search
[Option+S]
Global
abi @ gokcedb
Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more
Drag and drop files and folders you want to upload here, or choose Add files, or Add folders.
Files and folders (1 Total, 655.0 B)
All files and folders in this table will be uploaded.
Remove
Add files
Add folder
Find by name
Name
Folder
Type
Size
lambda_function.py.zip
-
application/zip
655.0 B
Destination
Destination
s3://test-gokcedb
Destination details
Bucket settings that impact new objects stored in the specified destination.
Permissions
Grant public access and access to other AWS accounts.
Properties
Specify storage class, encryption settings, tags, and more.
Cancel
Upload

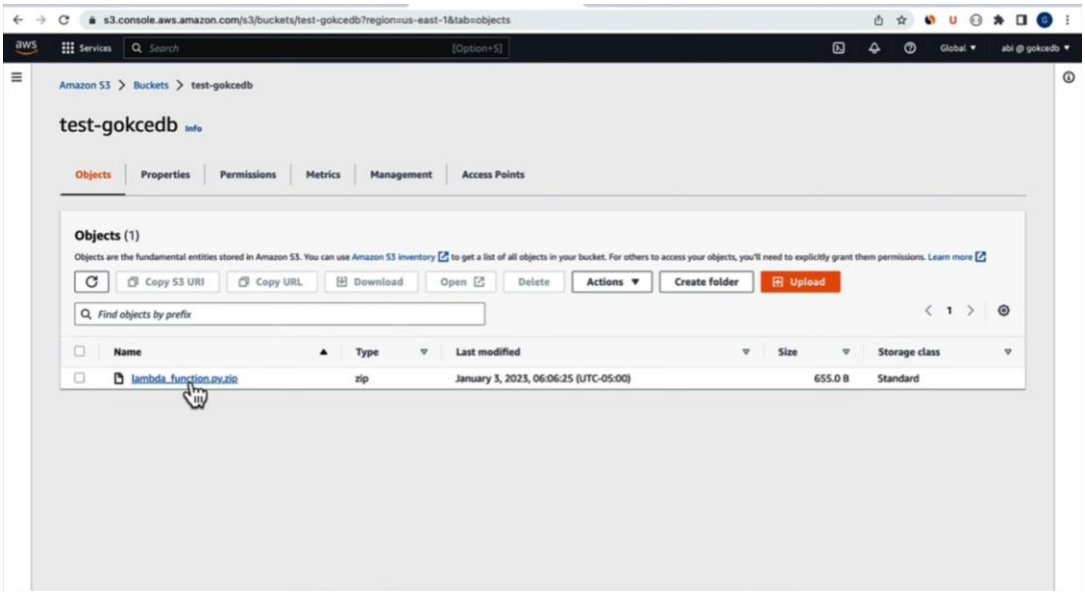

AWS Signer
Code signing configurations:
S3 Management Console
s3.console.aws.amazon.com/s3/upload/test-gokcedb?region=us-east-1
Services
Search
[Option+S]
Global
abi @ gokcedb
Upload succeeded
View details below.
Upload: status
Close
The information below will no longer be available after you navigate away from this page.
Summary
Destination
s3://test-gokcedb
Succeeded
1 file, 655.0 B (100.00%)
Failed
0 files, 0 B (0%)
Files and folders
Configuration
Files and folders (1 Total, 655.0 B)
Find by name
Name
Folder
Type
Size
Status
Error
lambda_function.py.zip
-
application/zip
655.0 B
Succeeded
-

STEP 9: In the Created Signing Profile, Now Start Signing Job

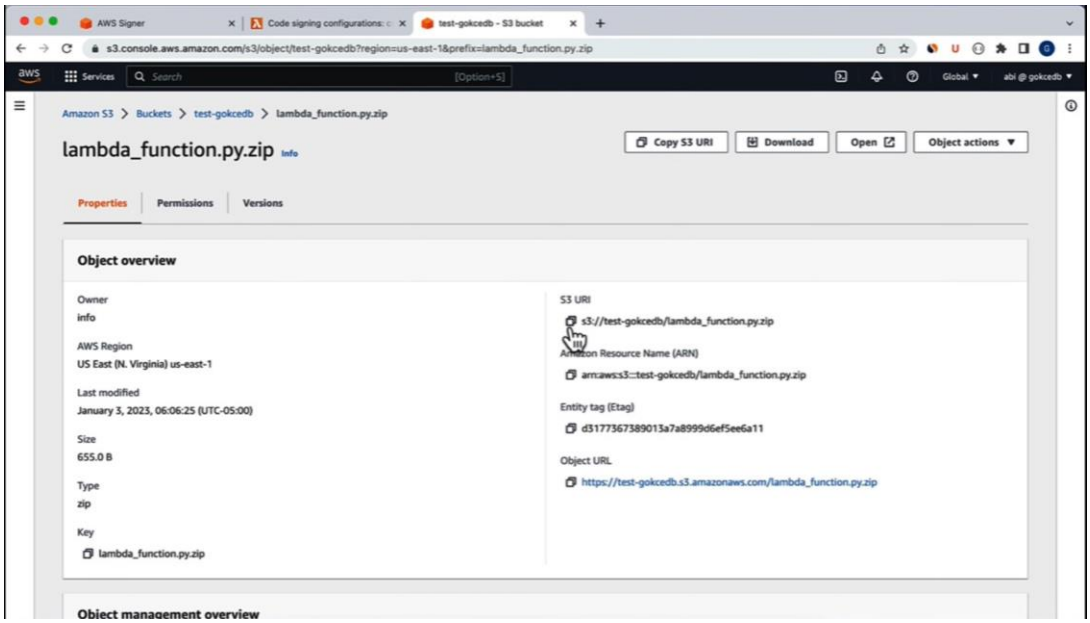


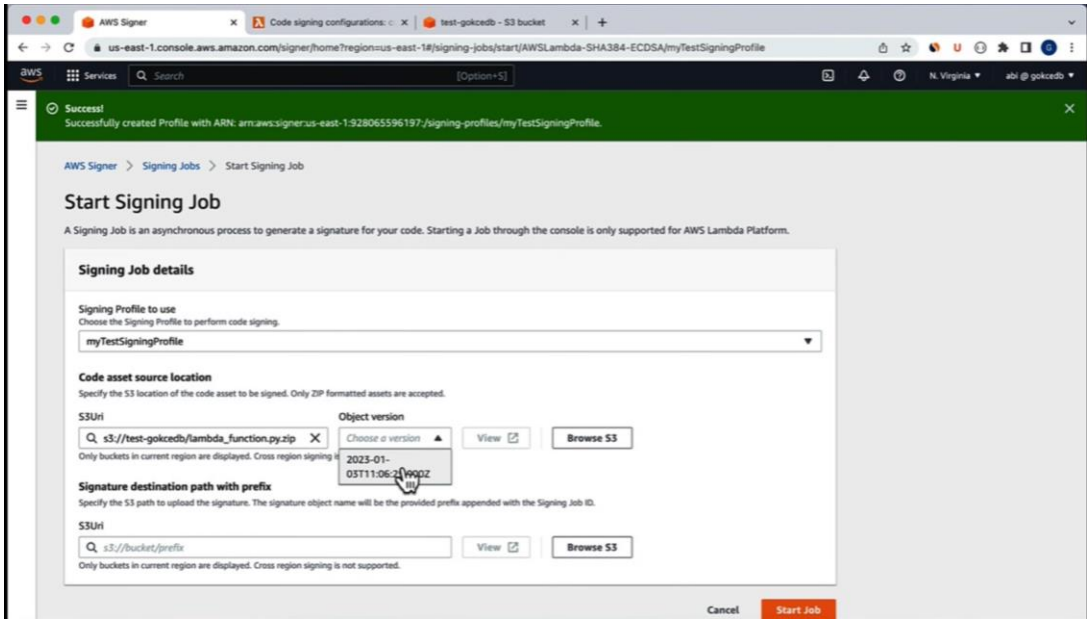


STEP 10: Now Click the Uploaded The Zip File Code In The S3 Bucket

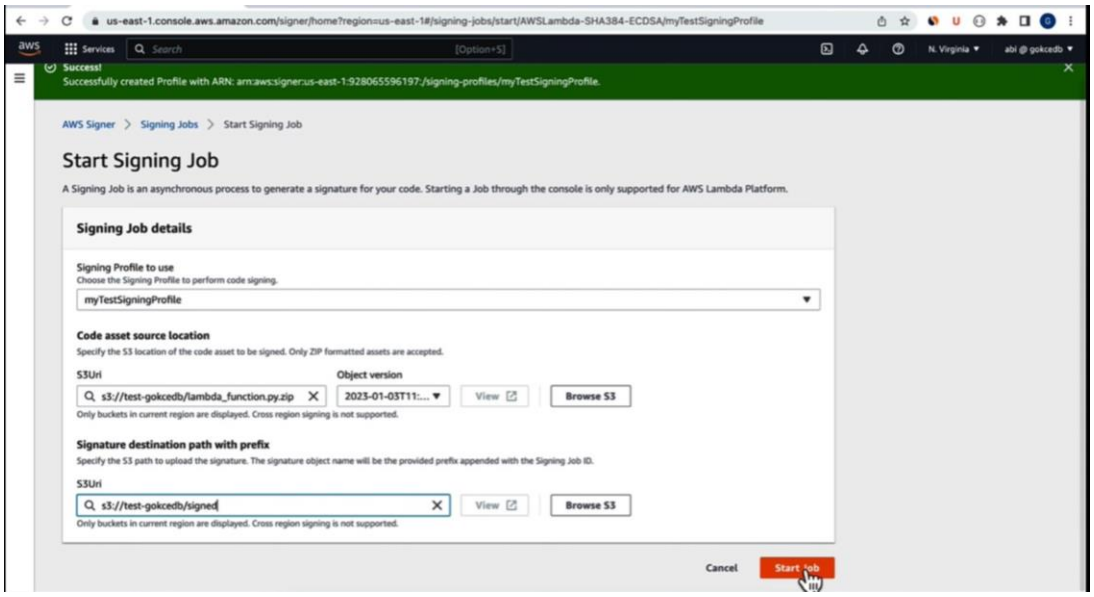


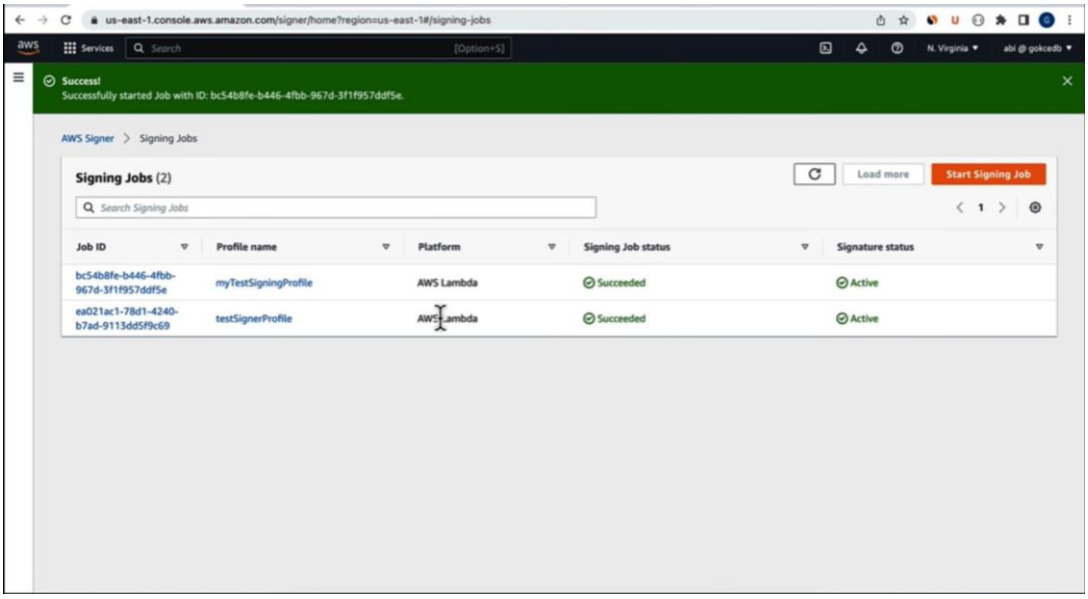
STEP 11: Copy the S3 Url And Paste It In The Signing Job Source Location

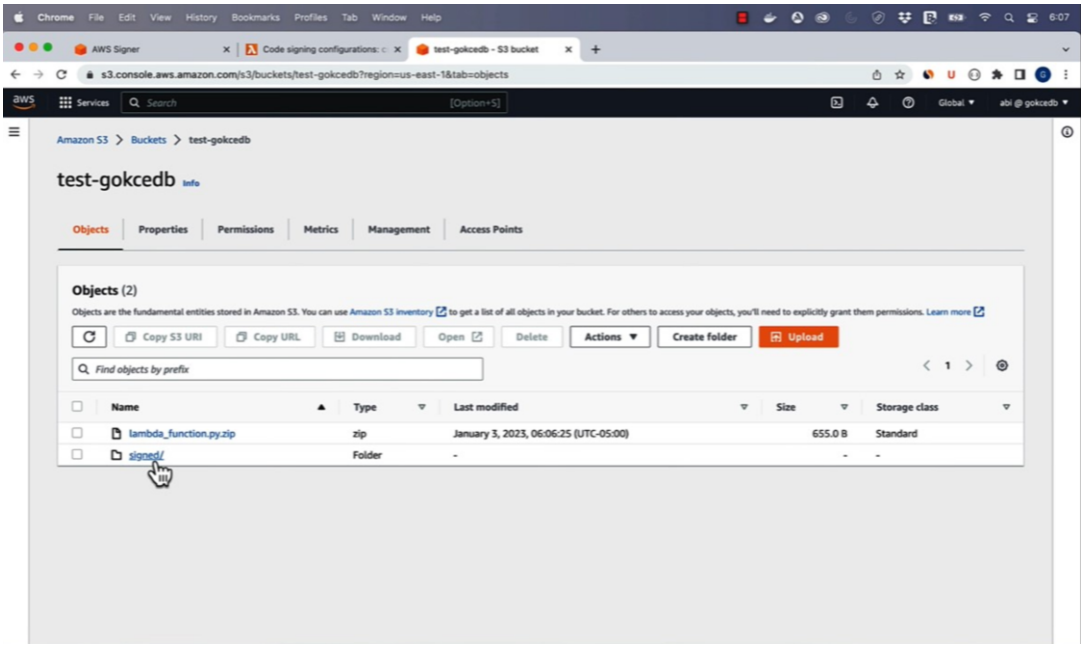


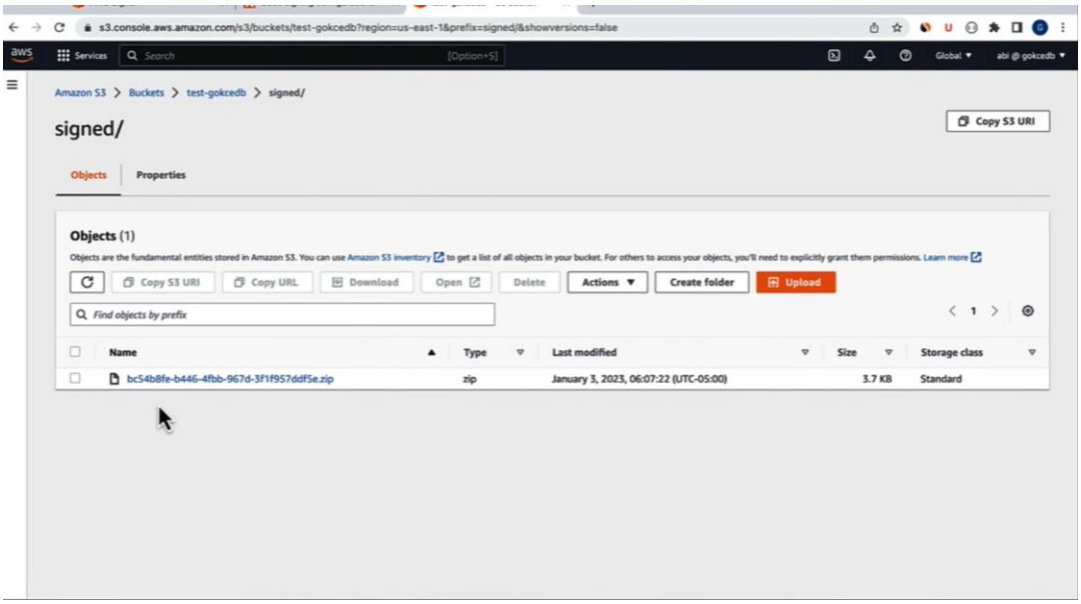


STEP 12: Enter the Destination Path

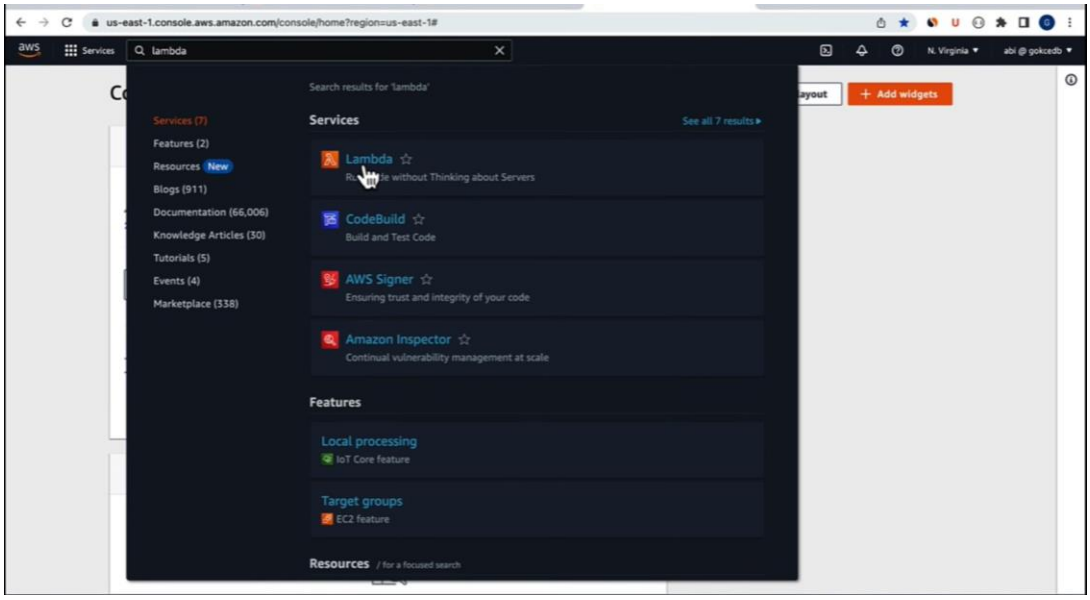


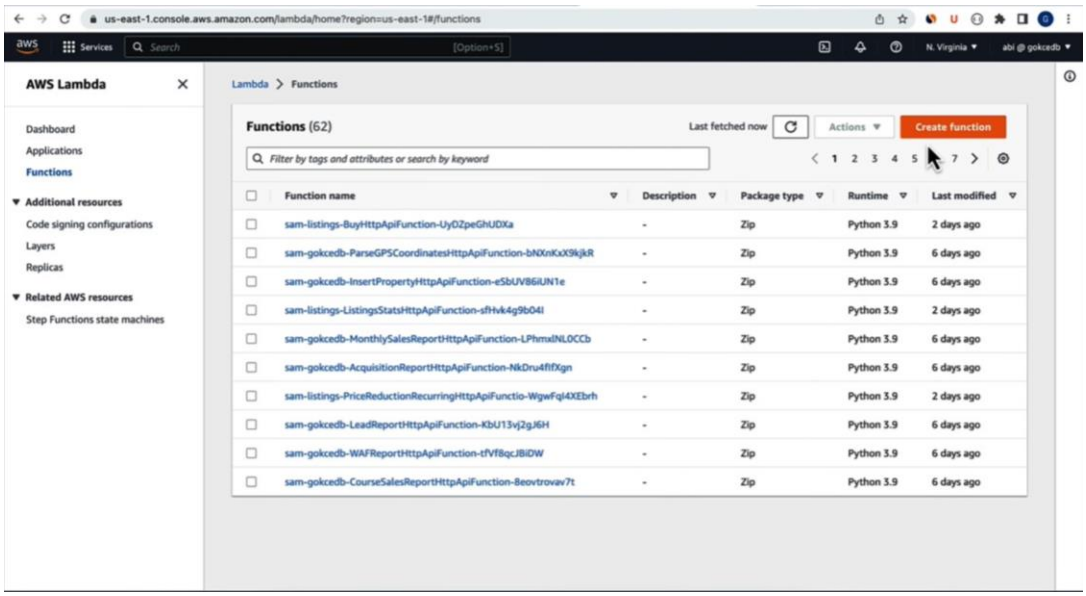

us-east-1.console.aws.amazon.com/signer/home?region=us-east-1#/signing-jobs
Success!
Successfully started Job with ID: bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.
AWS Signer > Signing Jobs
Signing Jobs (2)
Load more
Start Signing Job
Search Signing Jobs
Job ID
Profile name
Platform
Signing Job status
Signature status
bc54b8fe-b446-4fbb-967d-3f1f957ddf5e
myTestSigningProfile
AWS Lambda
Succeeded
Active
ea021ac1-78d1-4240-b7ad-9113dd5f9c69
testSignerProfile
AWS Lambda
Succeeded
Active

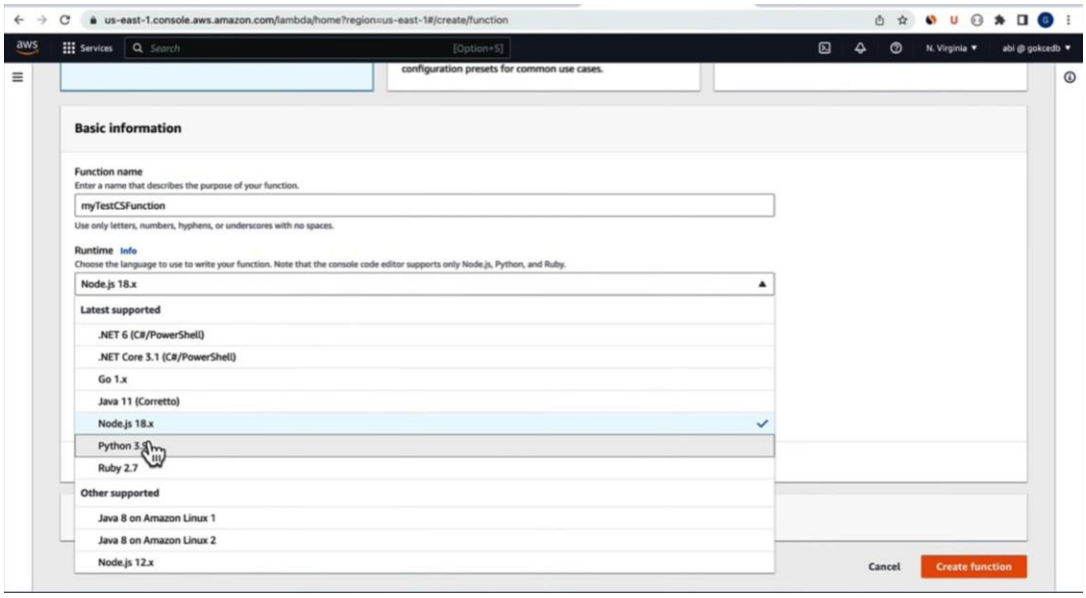

s3.console.aws.amazon.com/s3/buckets/test-gokcedb?region=us-east-1&tab=objects
Amazon S3 > Buckets > test-gokcedb
test-gokcedb Info
Objects
Properties
Permissions
Metrics
Management
Access Points
Objects (2)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
Name
Type
Last modified
Size
Storage class
lambda_function.py.zip
zip
January 3, 2023, 06:06:25 (UTC-05:00)
655.0 B
Standard
signed/
Folder



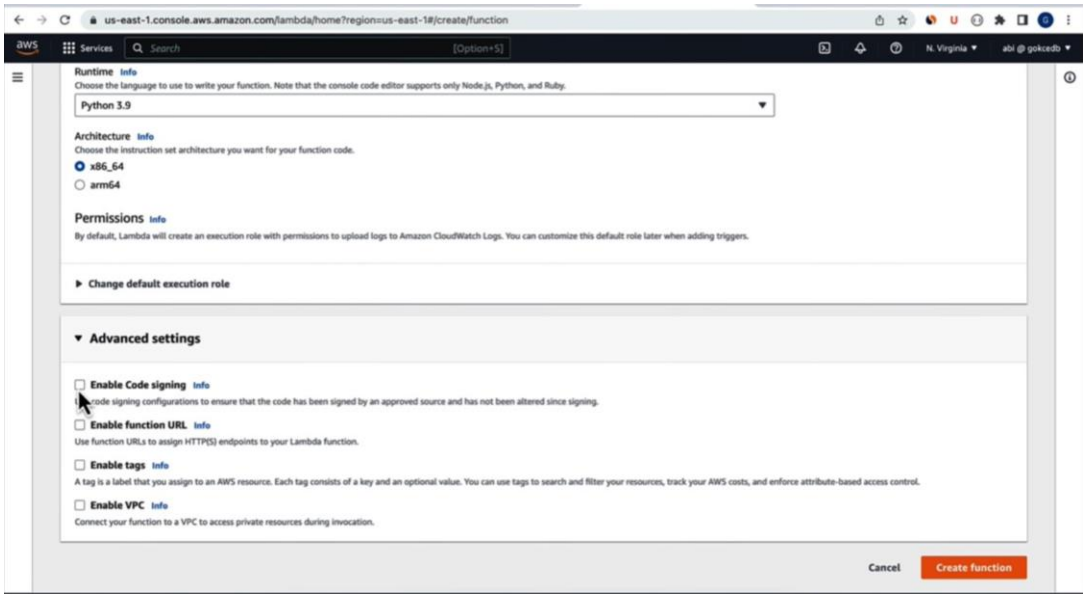
STEP 13: Create A Function in Lambda

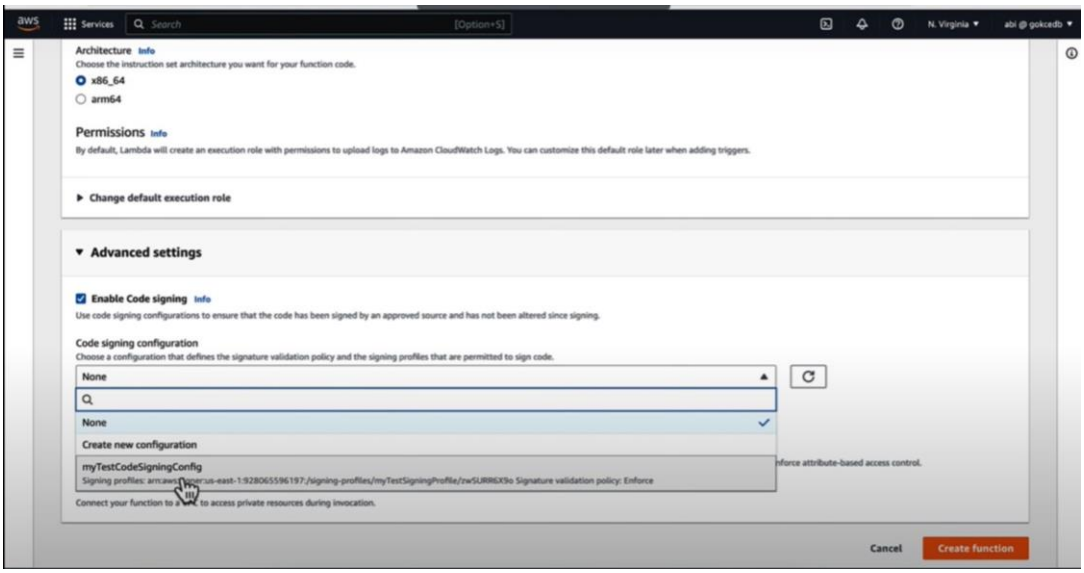


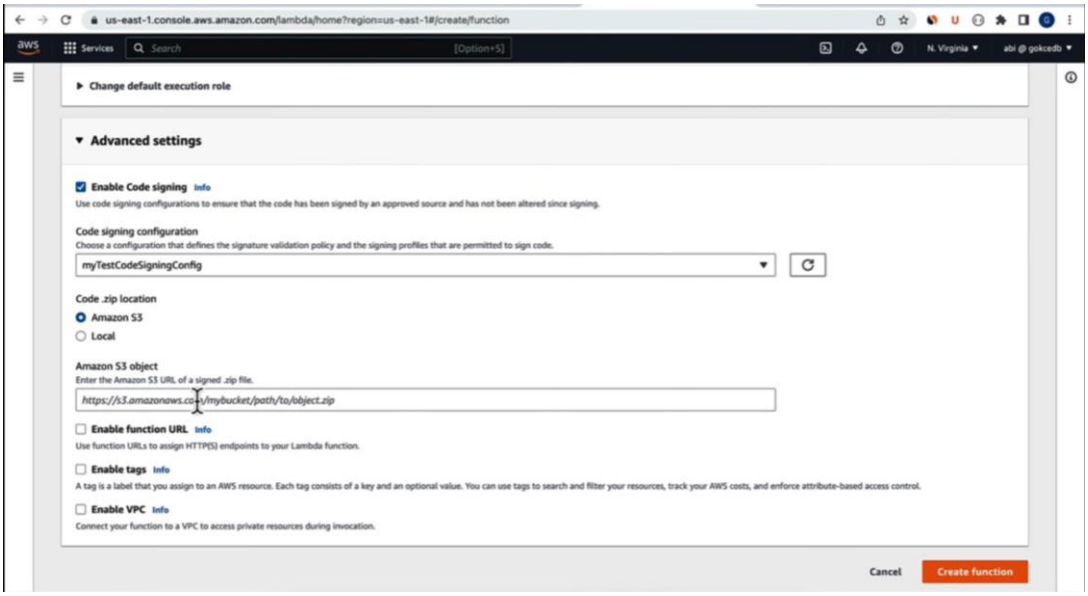

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Replicas
Related AWS resources
Step Functions state machines
Lambda > Functions
Functions (62)
Last fetched now
Actions
Create function
Filter by tags and attributes or search by keyword
Function name
Description
Package type
Runtime
Last modified
sam-listings-BuyHttpApiFunction-UyDZpeGhUDXa
sam-gokcedb-ParseGPSCoordinatesHttpApiFunction-bNXnKxX9kjkR
sam-gokcedb-InsertPropertyHttpApiFunction-eSbUVB6iUN1e
sam-listings-ListingsStatsHttpApiFunction-sfHvk4g9b04I
sam-gokcedb-MonthlySalesReportHttpApiFunction-LPhmxlNL0CCb
sam-gokcedb-AcquisitionReportHttpApiFunction-NkDru4fIfXgn
sam-listings-PriceReductionRecurringHttpApiFunctio-WgwFql4XEbrh
sam-gokcedb-LeadReportHttpApiFunction-KbU13vj2gJ6H
sam-gokcedb-WAFReportHttpApiFunction-tfVf8qcJBiDW
sam-gokcedb-CourseSalesReportHttpApiFunction-8eovtrovav7t
Zip
Python 3.9
2 days ago
6 days ago

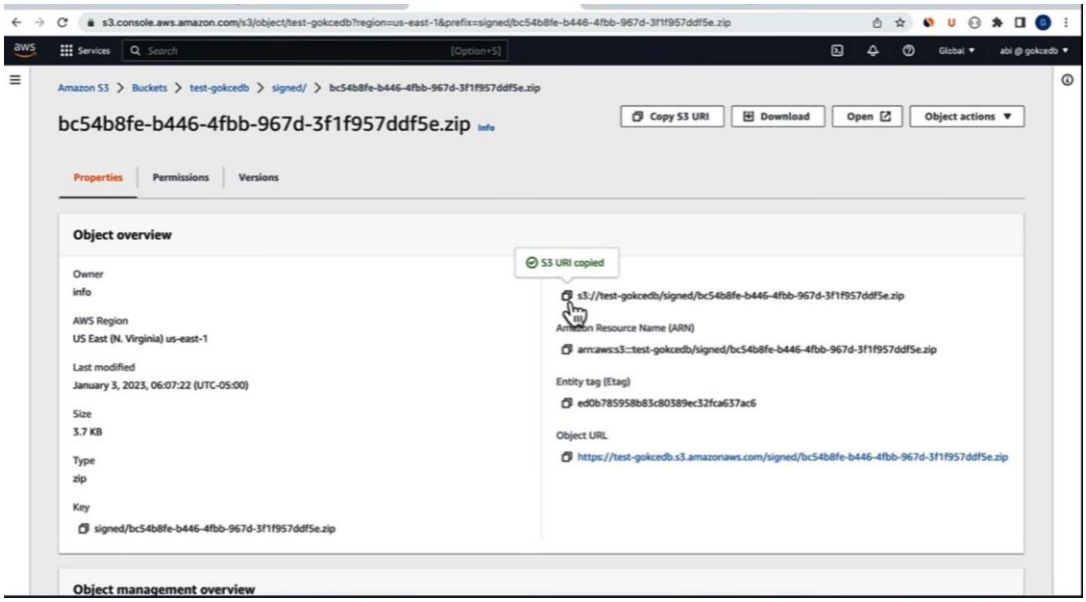

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function
configuration presets for common use cases.
Basic information
Function name
Enter a name that describes the purpose of your function.
myTestCSFunction
Use only letters, numbers, hyphens, or underscores with no spaces.
Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 18.x
Latest supported
.NET 6 (C#/PowerShell)
.NET Core 3.1 (C#/PowerShell)
Go 1.x
Java 11 (Corretto)
Node.js 18.x
Python 3.9
Ruby 2.7
Other supported
Java 8 on Amazon Linux 1
Java 8 on Amazon Linux 2
Node.js 12.x
Cancel
Create function

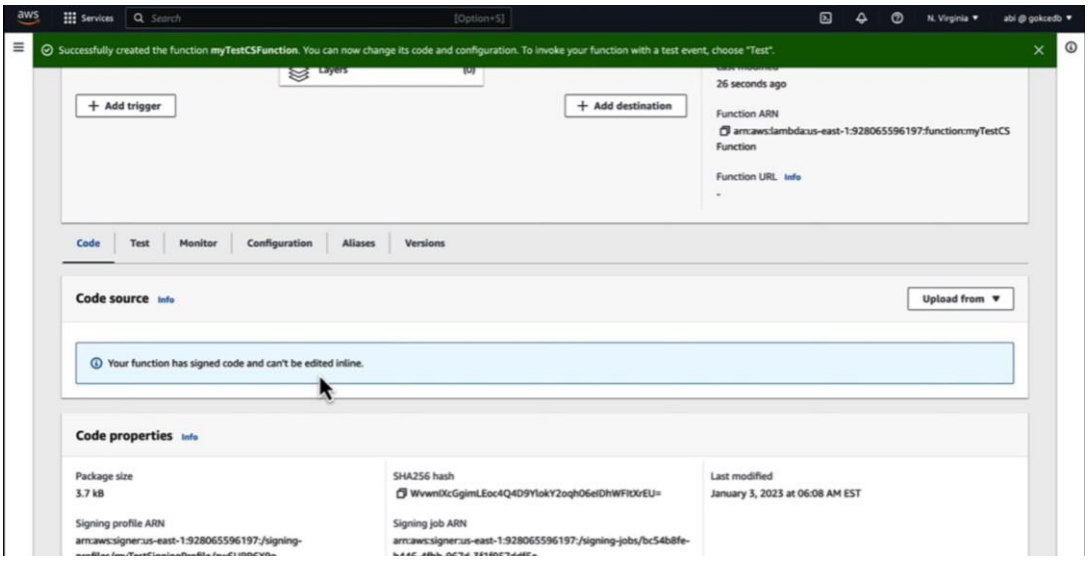
STEP 14: Enable Code Signing



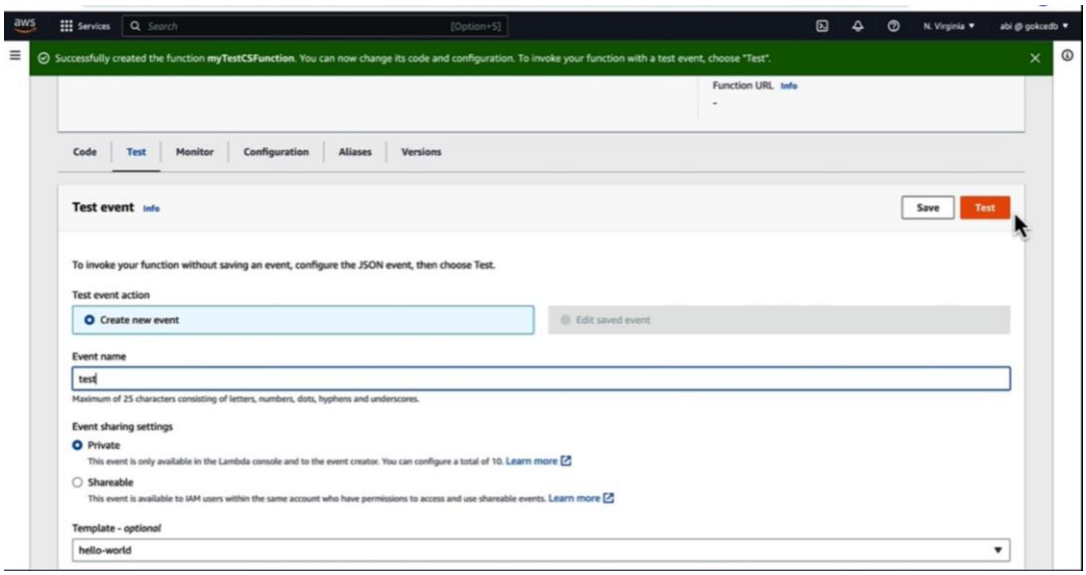


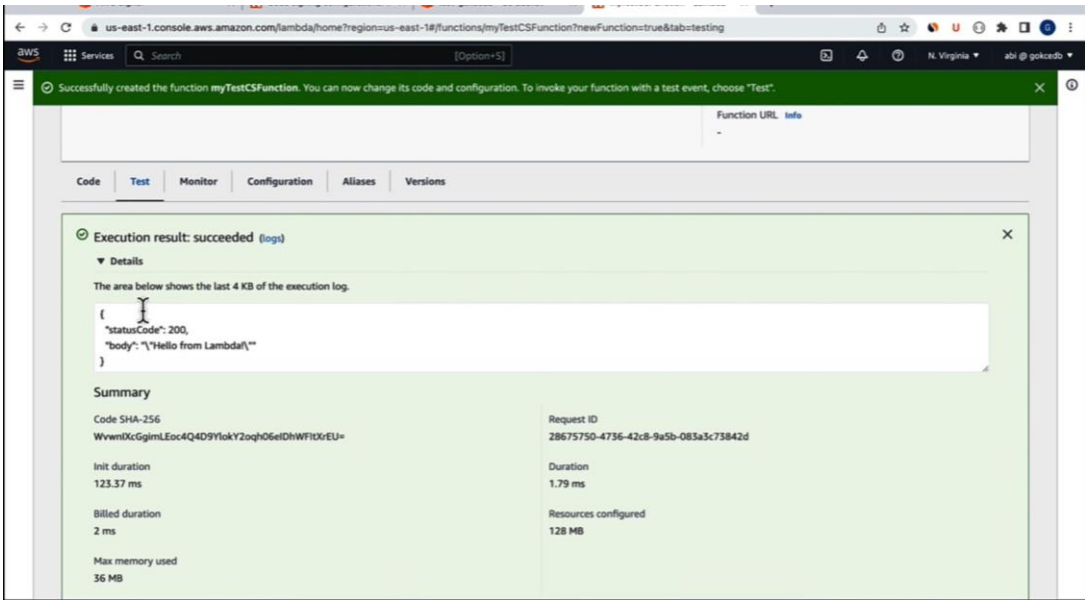

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Change default execution role
Advanced settings
Enable Code signing Info
Use code signing configurations to ensure that the code has been signed by an approved source and has not been altered since signing.
Code signing configuration
Choose a configuration that defines the signature validation policy and the signing profiles that are permitted to sign code.
myTestCodeSigningConfig
Code .zip location
Amazon S3
Local
Amazon S3 object
Enter the Amazon S3 URL of a signed .zip file.
https://s3.amazonaws.com/mybucket/path/to/object.zip
Enable function URL Info
Use function URLs to assign HTTP(S) endpoints to your Lambda function.
Enable tags Info
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources, track your AWS costs, and enforce attribute-based access control.
Enable VPC Info
Connect your function to a VPC to access private resources during invocation.
Cancel
Create function


s3.console.aws.amazon.com/s3/object/test-gokcedb?region=us-east-1&prefix=signed/bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.zip
Services
Search
[Option+S]
Global
abi @ gokcedb
Amazon S3 > Buckets > test-gokcedb > signed/ > bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.zip
bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.zip Info
Copy S3 URI
Download
Open
Object actions
Properties
Permissions
Versions
Object overview
S3 URI copied
Owner
info
AWS Region
US East (N. Virginia) us-east-1
Last modified
January 3, 2023, 06:07:22 (UTC-05:00)
Size
3.7 KB
Type
zip
Key
signed/bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.zip
s3://test-gokcedb/signed/bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.zip
Amazon Resource Name (ARN)
arn:aws:s3:::test-gokcedb/signed/bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.zip
Entity tag (Etag)
ed0b785958b83c80389ec32fca637ac6
Object URL
https://test-gokcedb.s3.amazonaws.com/signed/bc54b8fe-b446-4fbb-967d-3f1f957ddf5e.zip
Object management overview



STEP 15: Test the File




us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/myTestCSFunction?newFunction=true&tab=testing
Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Successfully created the function myTestCSFunction. You can now change its code and configuration. To invoke your function with a test event, choose "Test".
Function URL Info
-
Code
Test
Monitor
Configuration
Aliases
Versions
Execution result: succeeded (logs)
Details
The area below shows the last 4 KB of the execution log.
{
"statusCode": 200,
"body": "\"Hello from Lambda!\""
}
Summary
Code SHA-256
WvwnlXcGgimLEoc4Q4D9YlokY2oqhO6elDhWFltXrEU=
Request ID
28675750-4736-42c8-9a5b-083a3c73842d
Init duration
123.37 ms
Duration
1.79 ms
Billed duration
2 ms
Resources configured
128 MB
Max memory used
36 MB

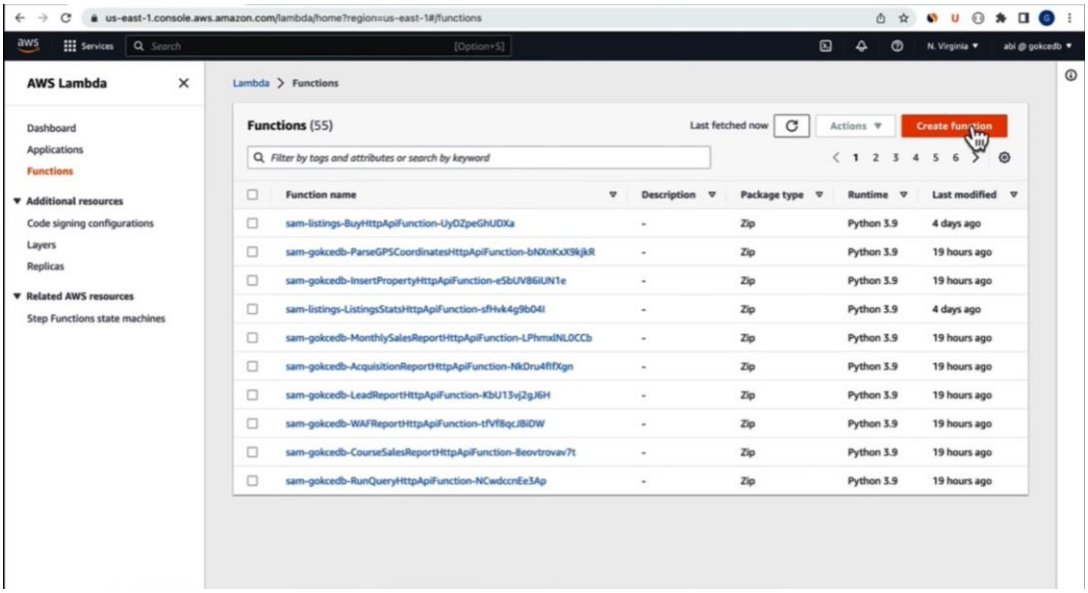
STEP FUNCTIONS STATE MACHINES

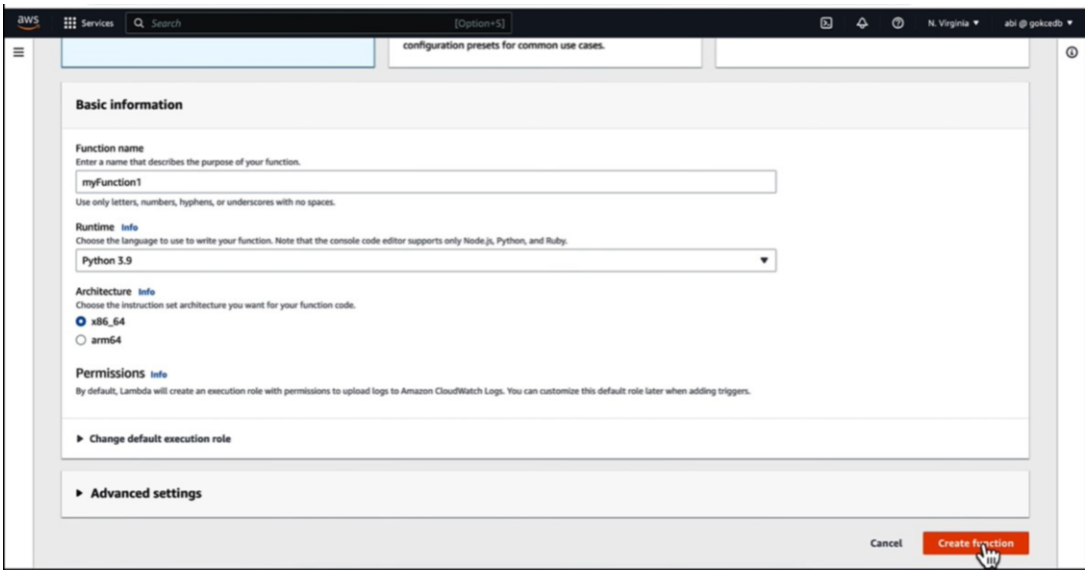
AWS Step Functions enables you to implement a business process as a series of steps that make up a workflow. The individual steps in the workflow can invoke a Lambda function or a container that has some business logic, update a database such as DynamoDB or publish a message to a queue once that step or the entire workflow completes execution.

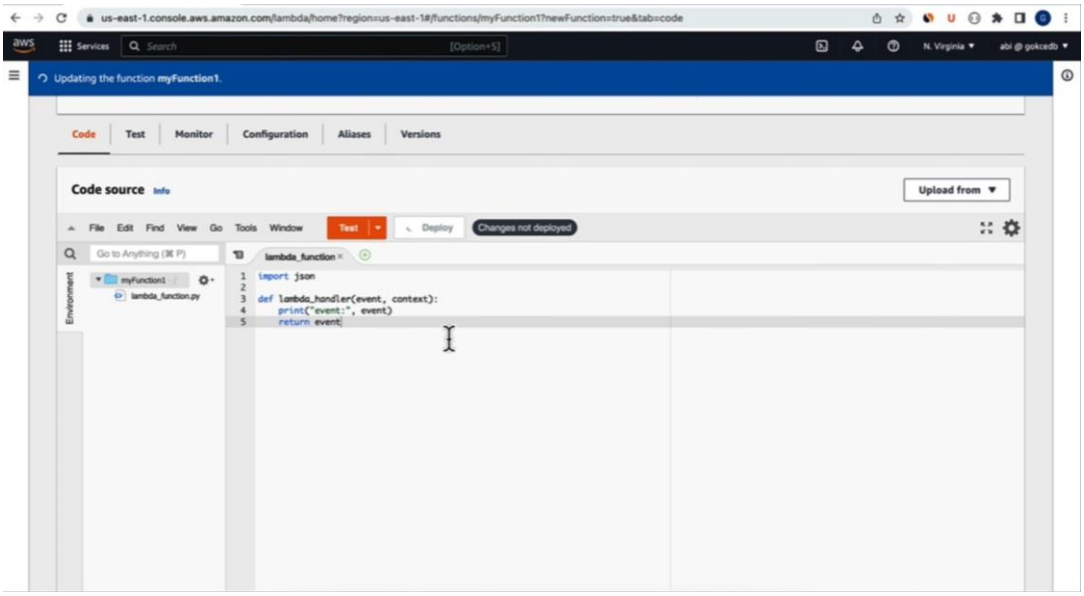
AWS Step Functions has two workflow options - Standard and Express. When your business process is expected to take longer than five minutes for a single execution, you should choose Standard. Some examples of a long running workflow are an ETL orchestration pipeline or when any step in your workflow waits for response from a human to move to the next step.

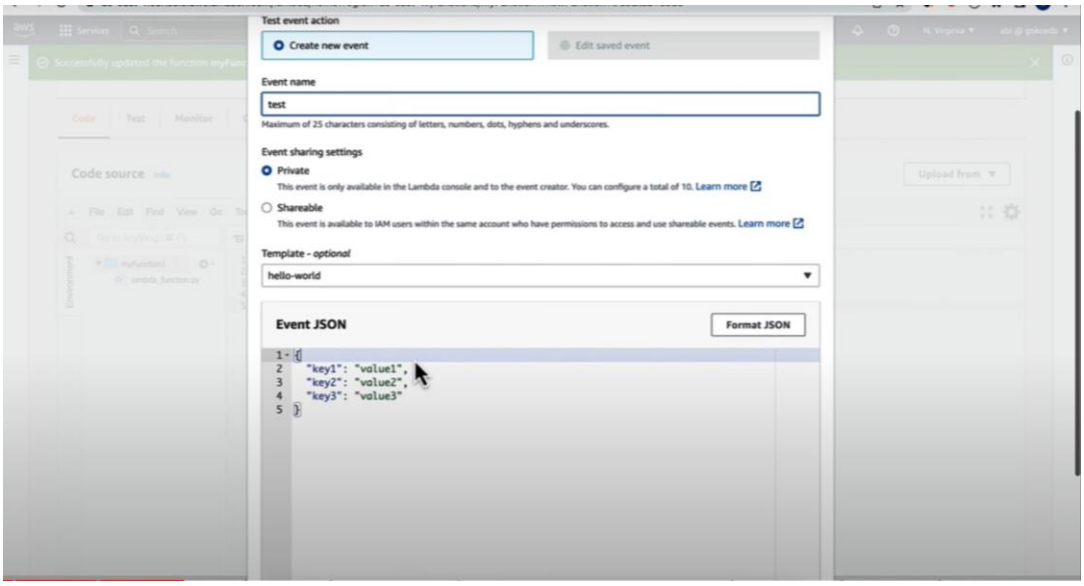
Express workflows are suited for workflows that take less than five minutes and are ideal when you need high execution volume i.e., 100,000 invocations per second. You can use either Standard or Express distinctly or combine them such that a longer Standard workflow triggers multiple shorter Express workflows that execute in parallel.

STEP 1: Create A Function

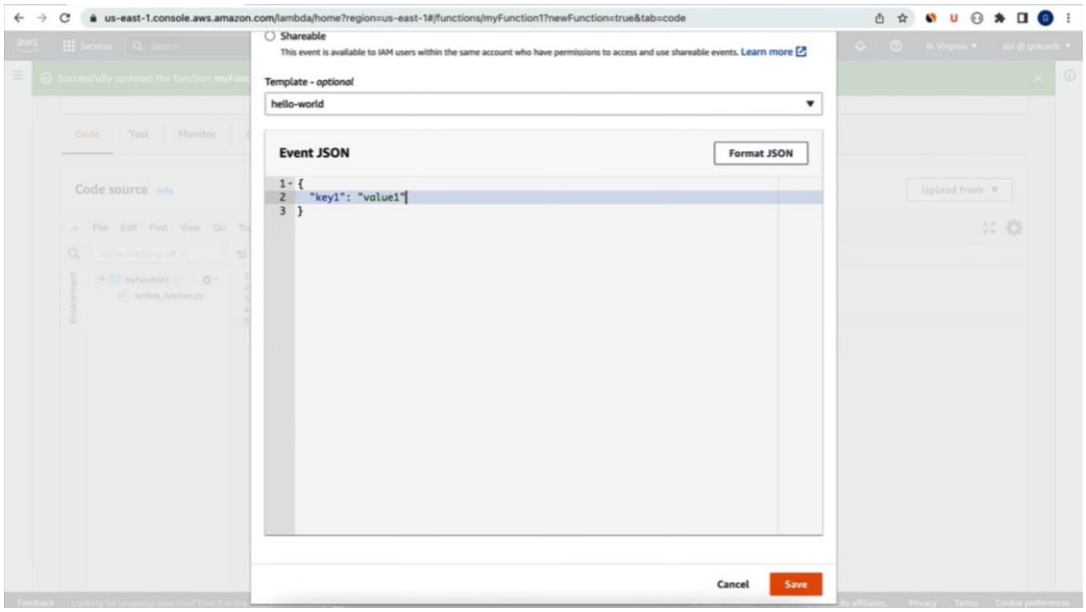


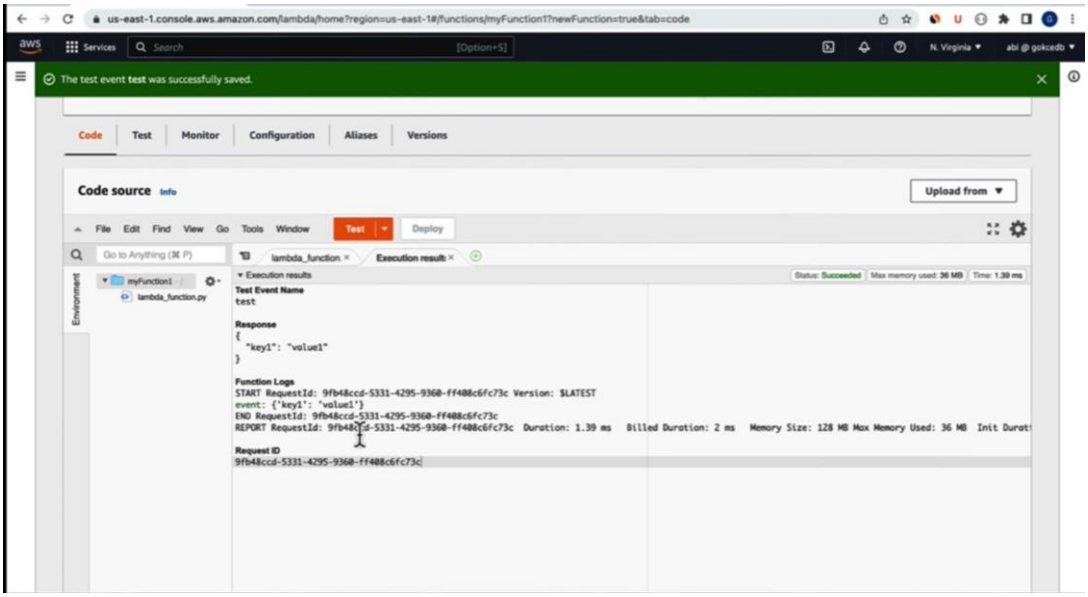



us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/myFunction1?newFunction=true&tab=code
Updating the function myFunction1.
Code
Test
Monitor
Configuration
Aliases
Versions
Code source Info
Upload from
File Edit Find View Go Tools Window
Test
Deploy
Changes not deployed
lambda_function
myFunction1
lambda_function.py
Environment
import json
def lambda_handler(event, context):
print("event:", event)
return event

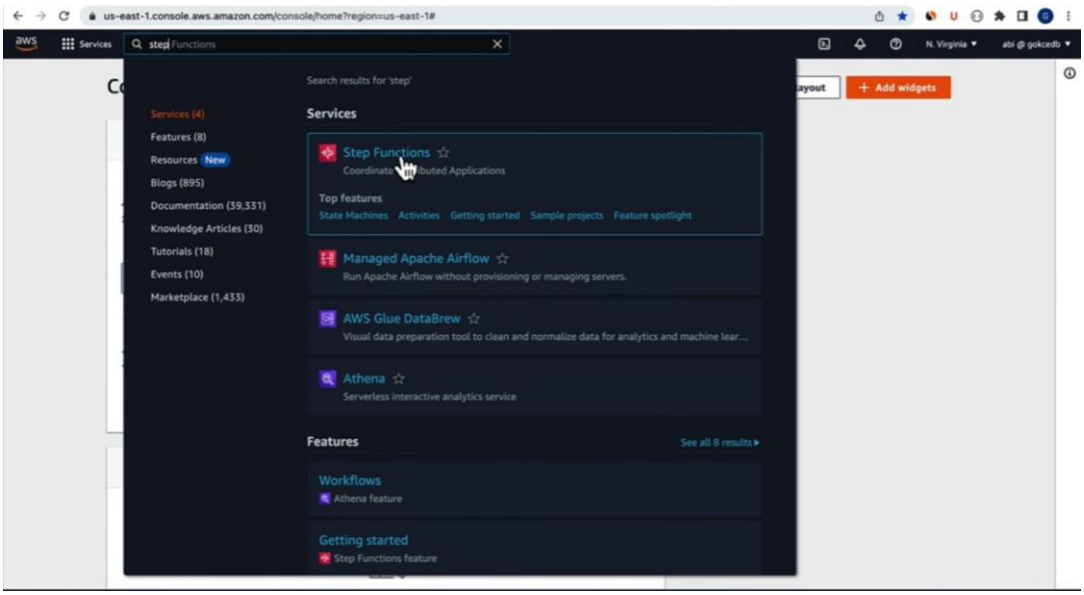

Test event action
Create new event
Edit saved event
Event name
test
Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.
Event sharing settings
Private
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn 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
{
"key1": "value1",
"key2": "value2",
"key3": "value3"
}

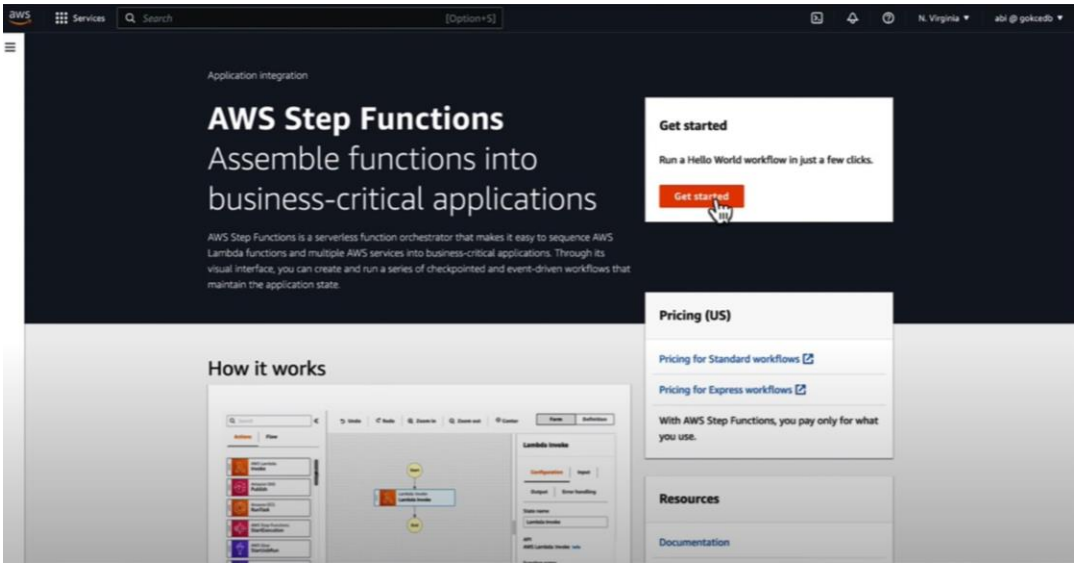
STEP 2: Give A Single Key



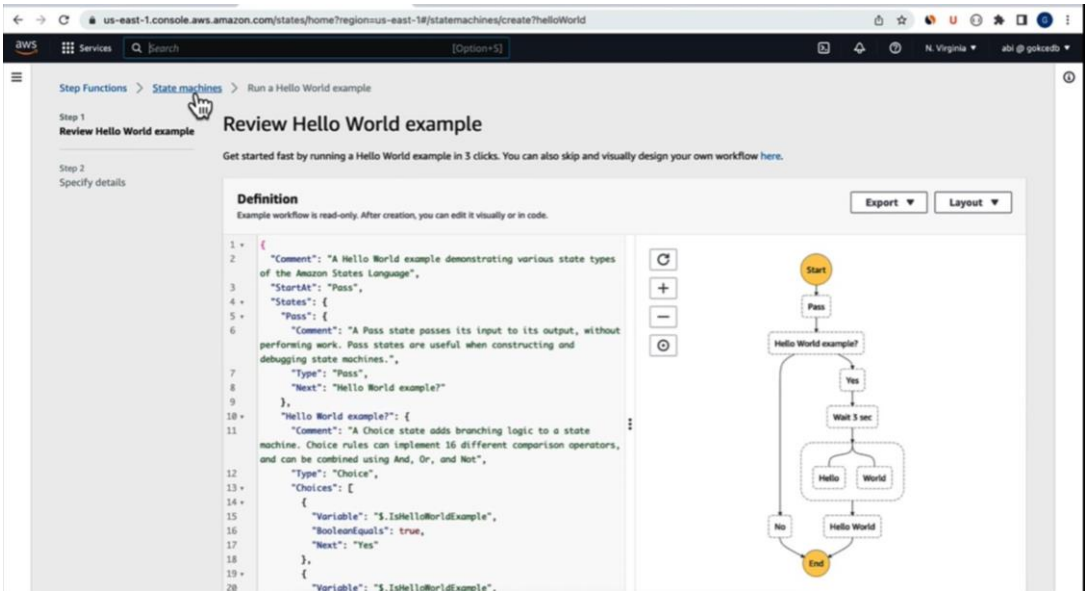


STEP 3: Search Step Functions

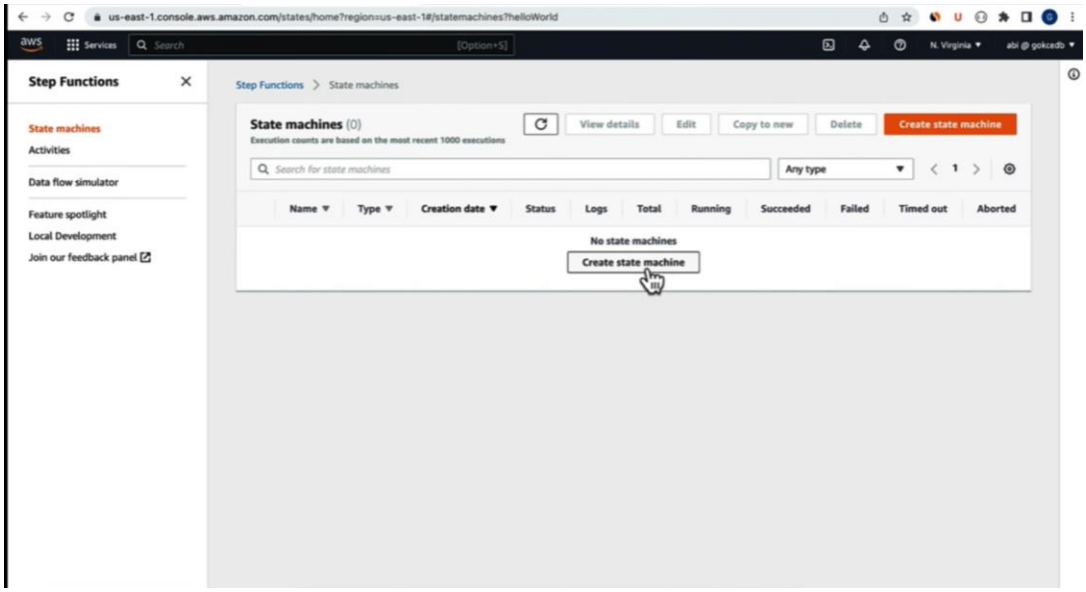


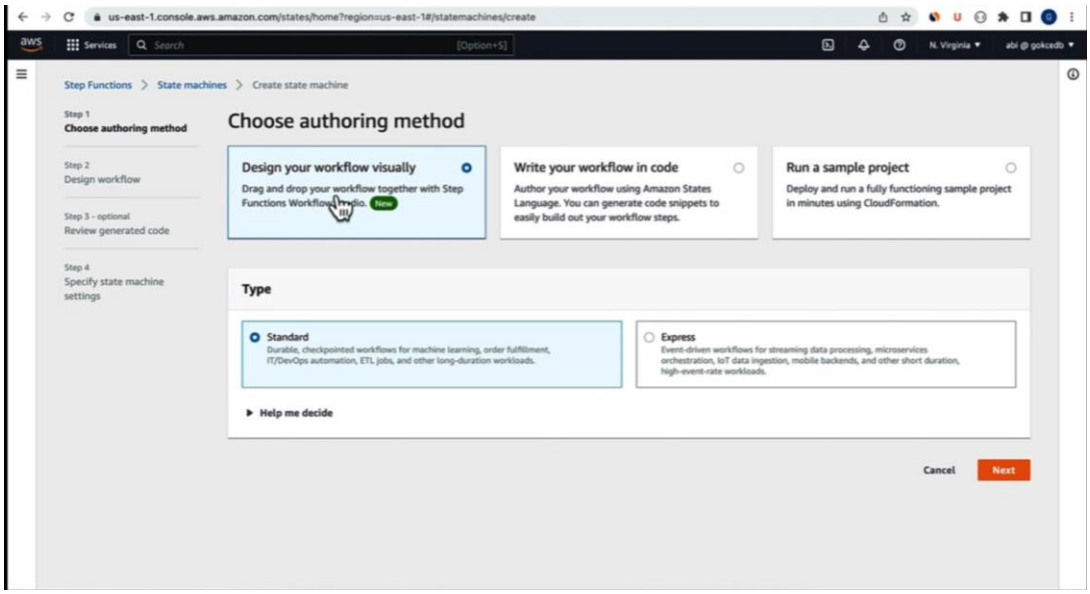


STEP 4: Go to State Machines

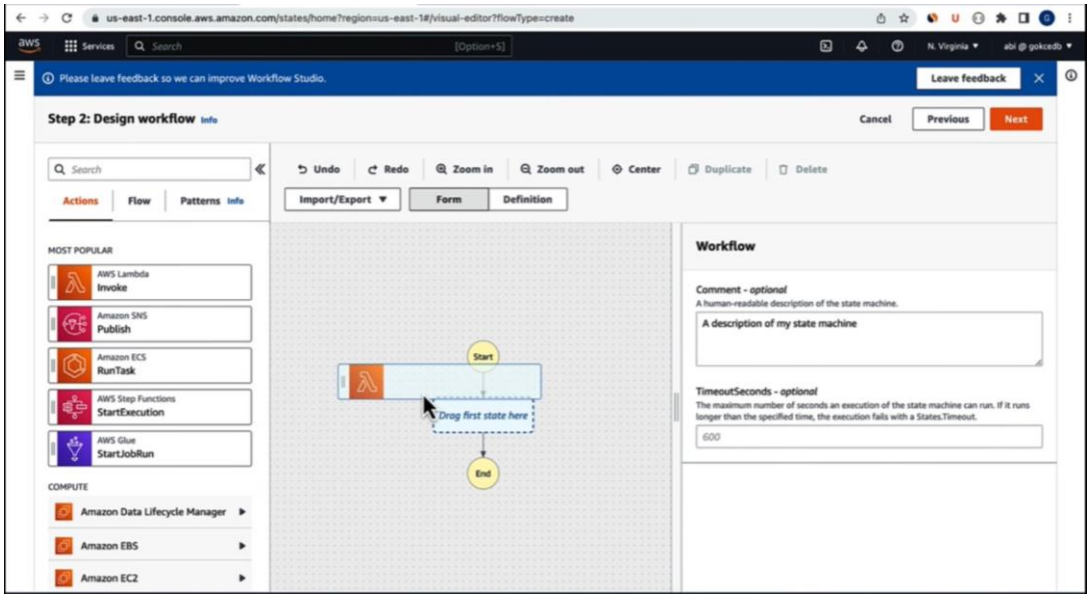


Create A State Machine

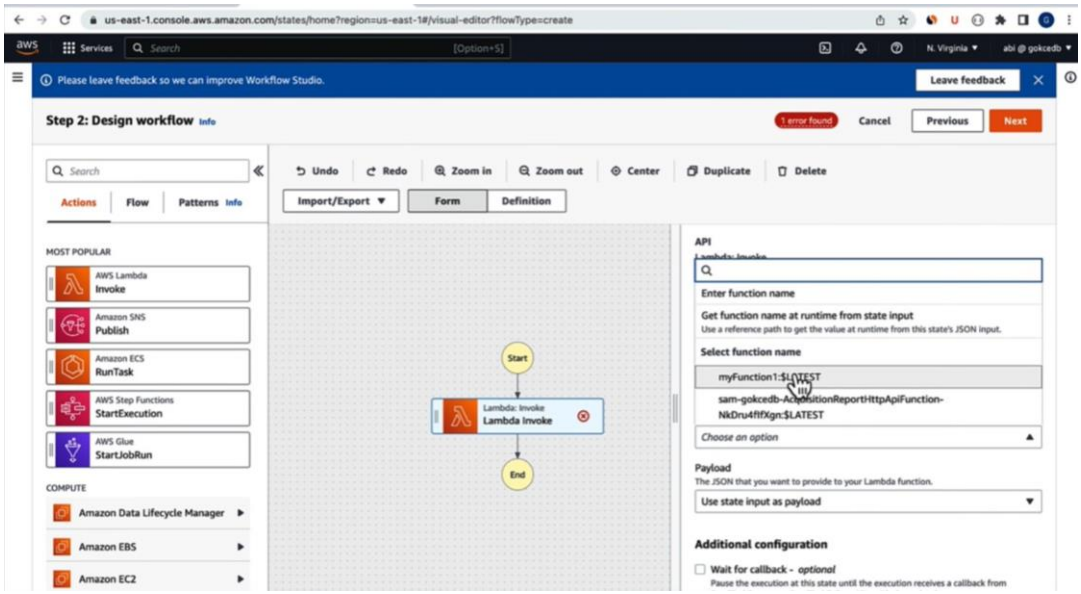


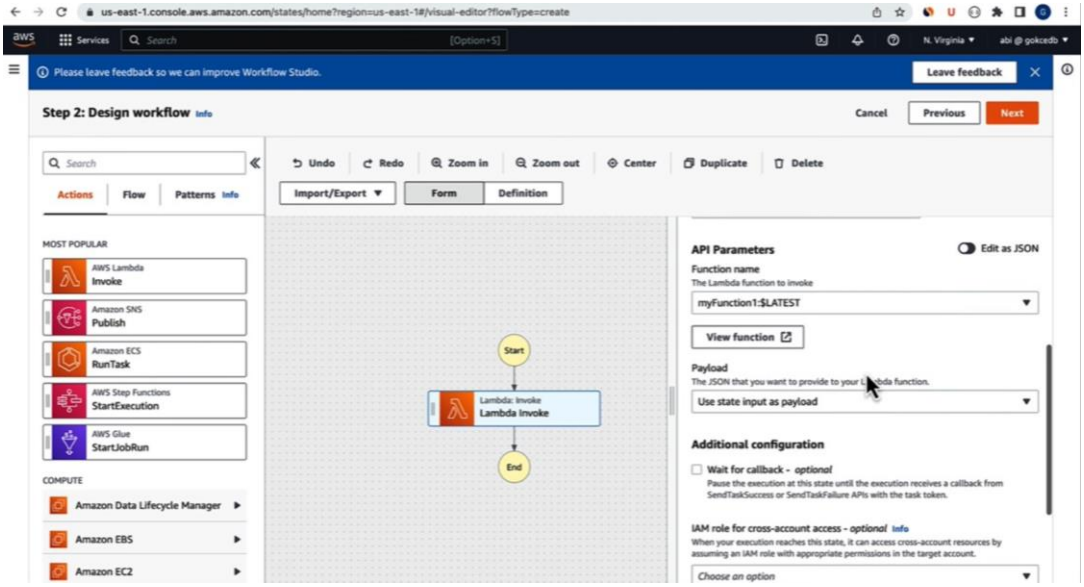


STEP 5: Drag and Drop The Lambda

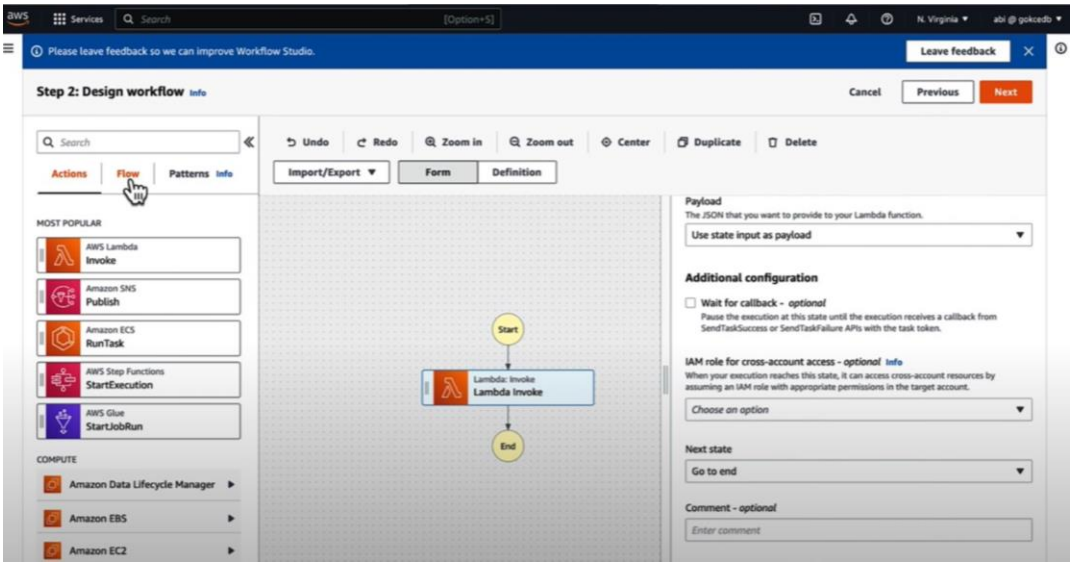


STEP 6: Select the Function Name

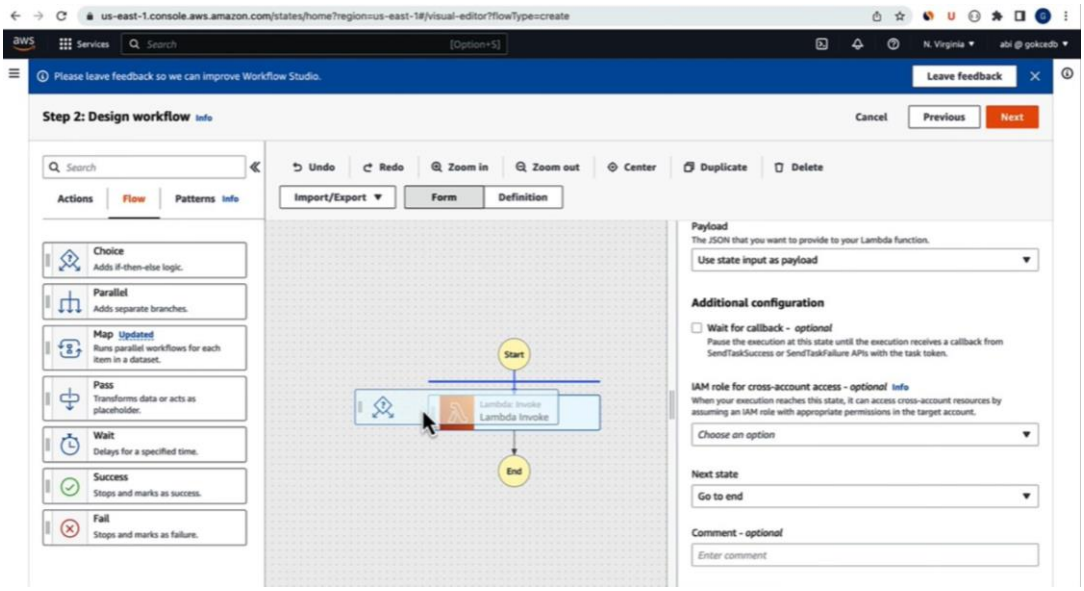




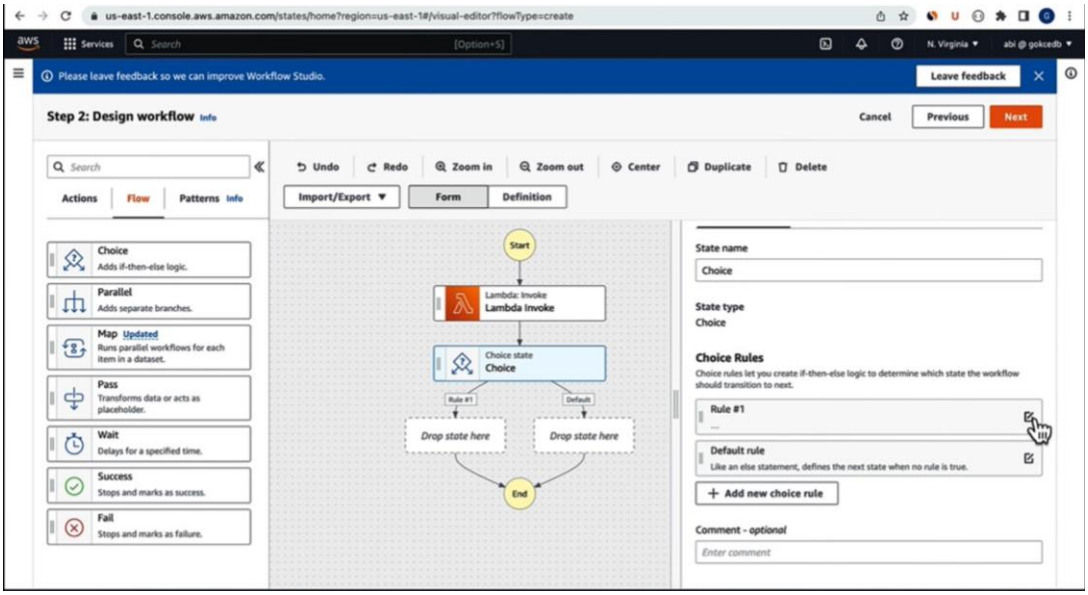
STEP 7: Go to Flow



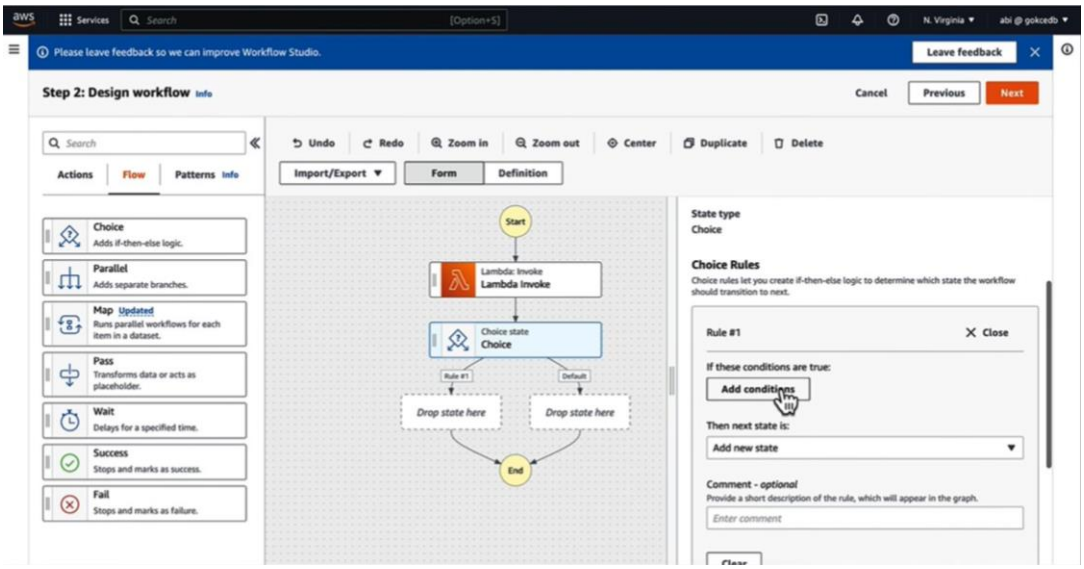
STEP 8: Drag and Drop The Choice

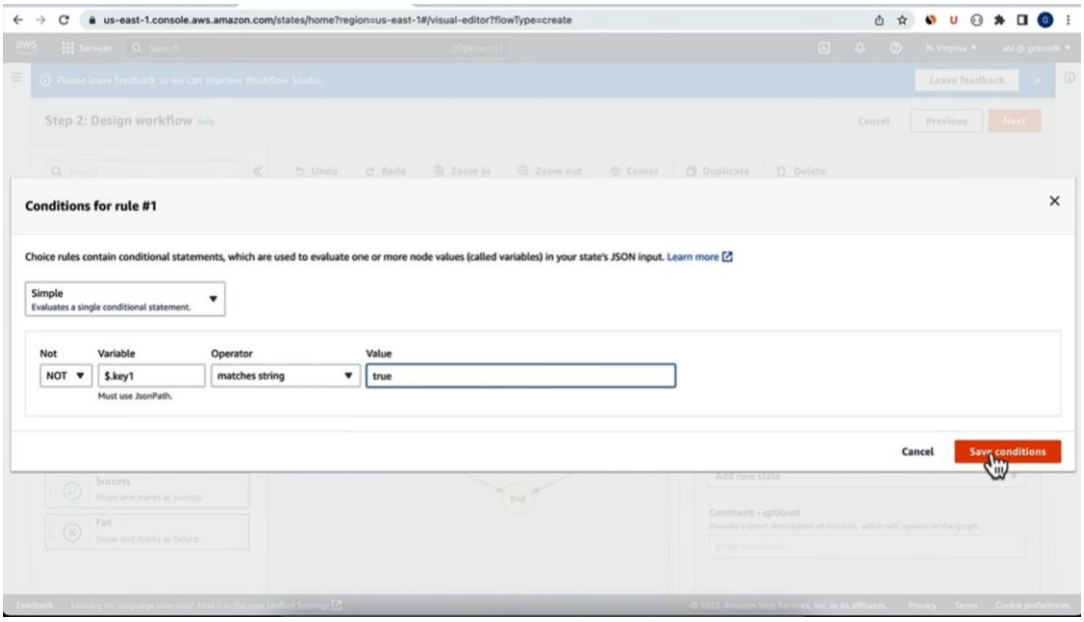


STEP 9: Go to The Choice Rules

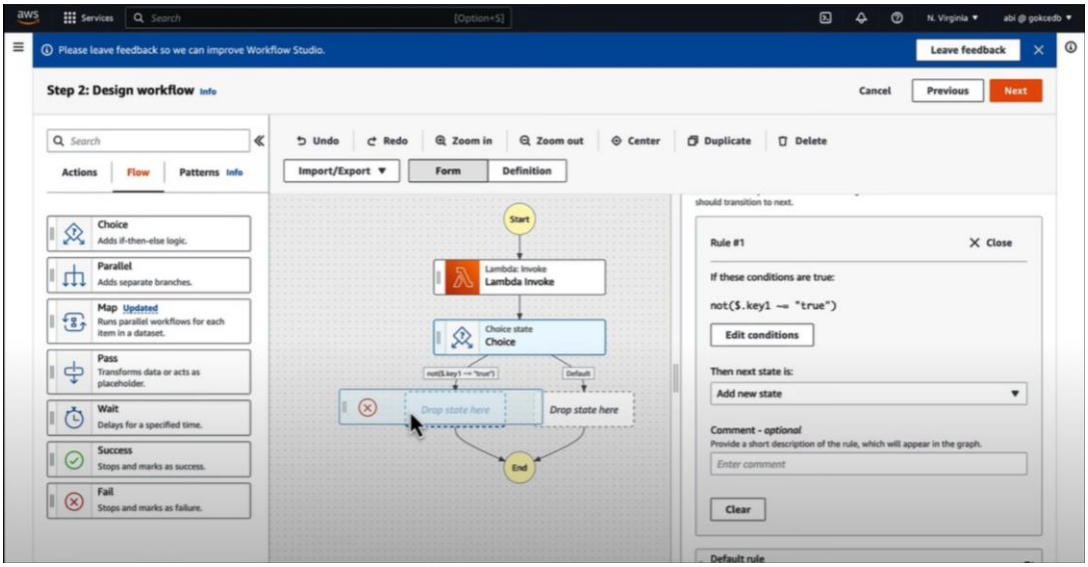


STEP 10: Add Conditions

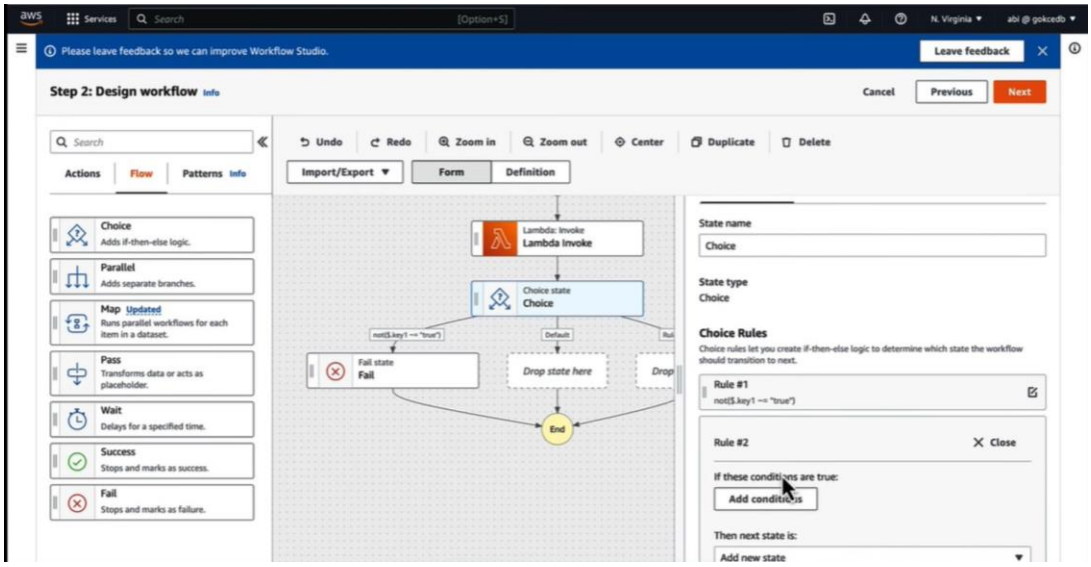


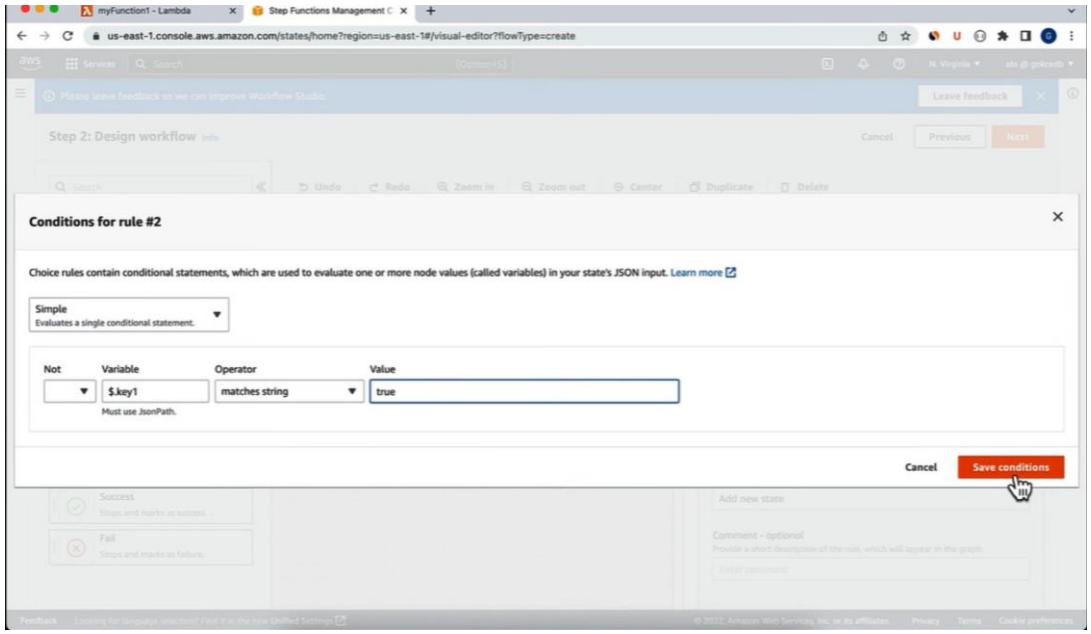


STEP 11: Drag and Drop The Fail

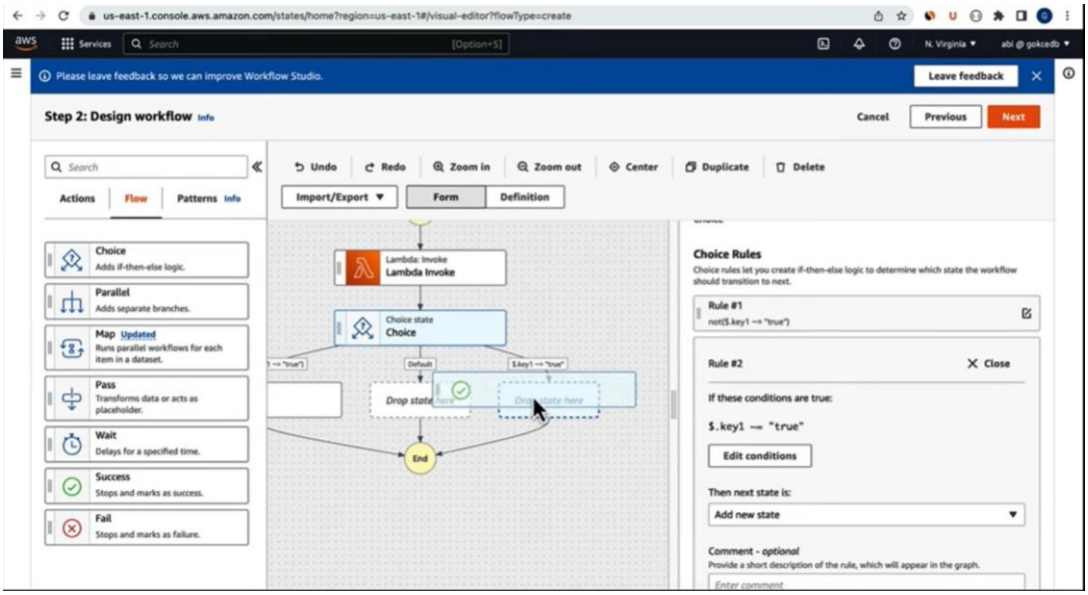


STEP 12: Add Conditions

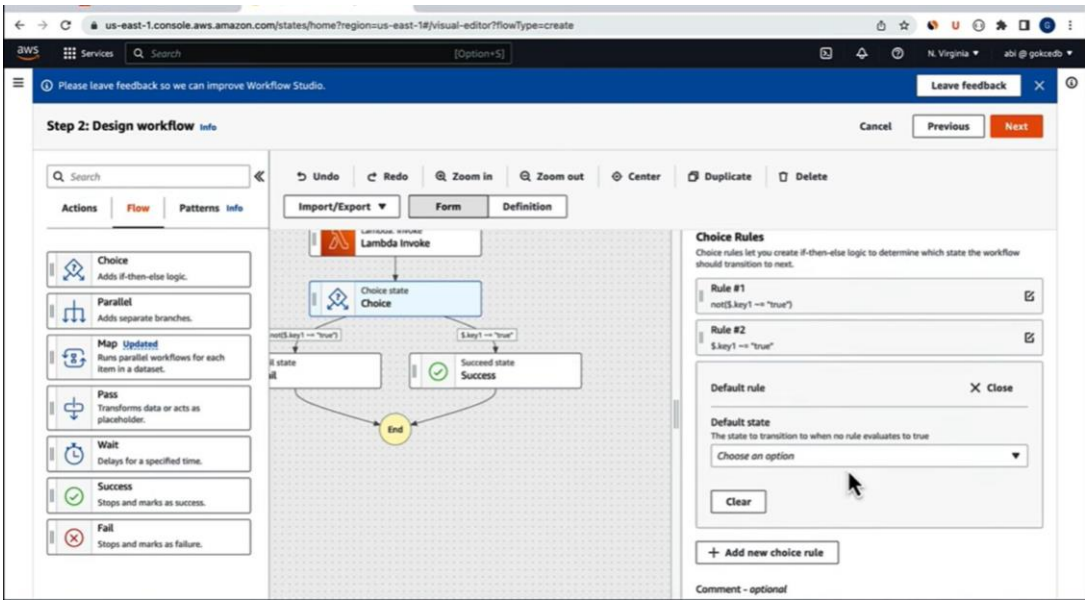




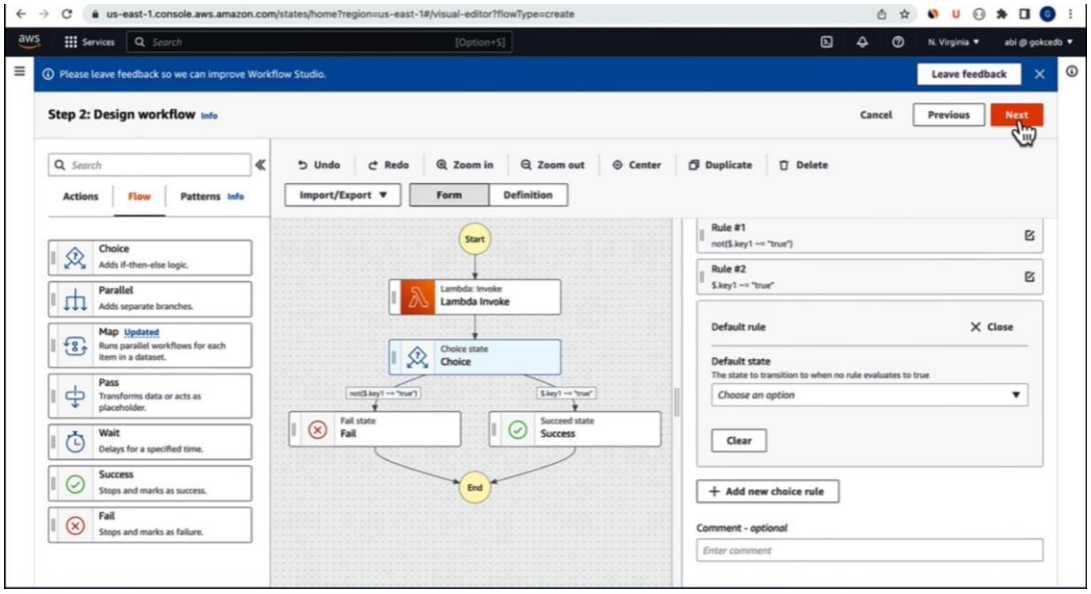
STEP 13: Drag and Drop The Success

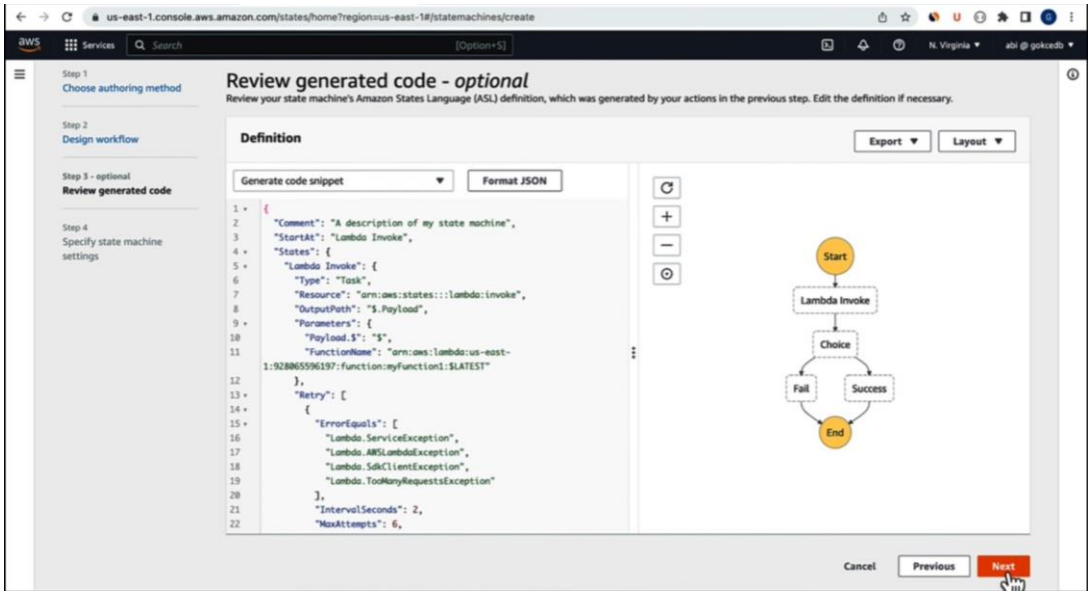


STEP 14: Close Default

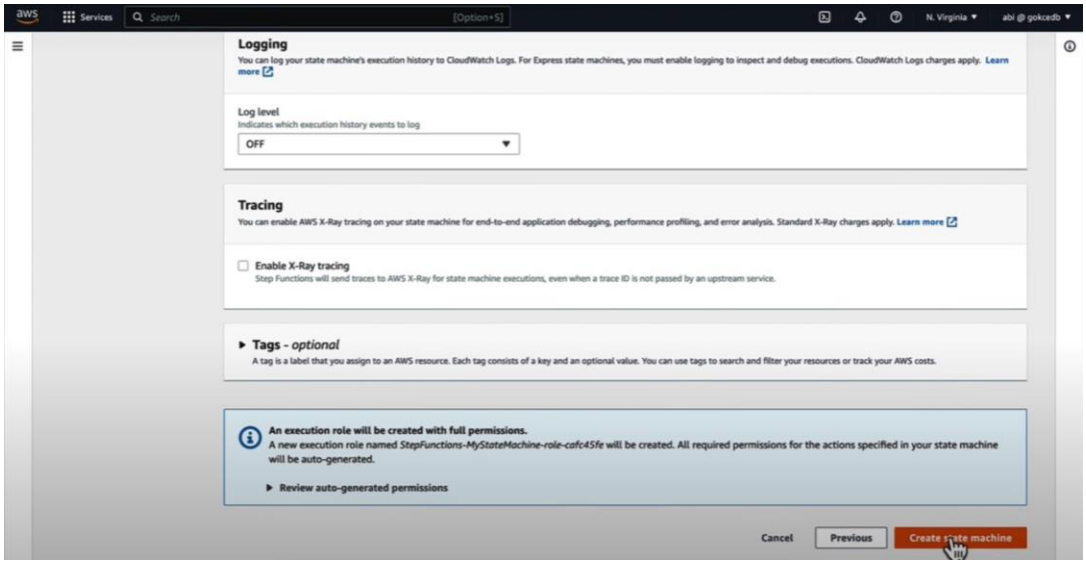


STEP 15: Click Next

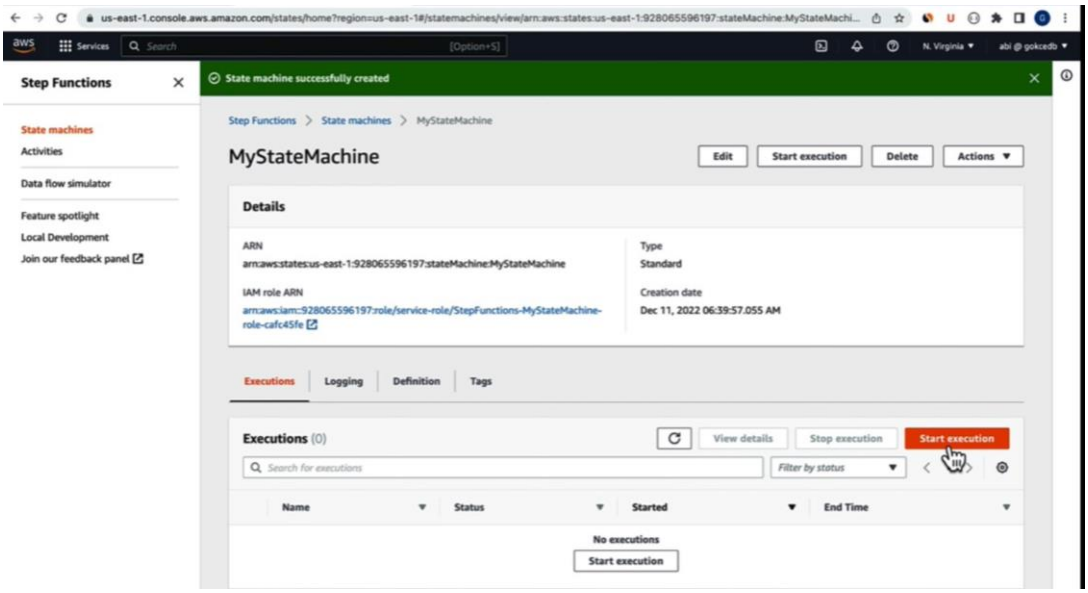




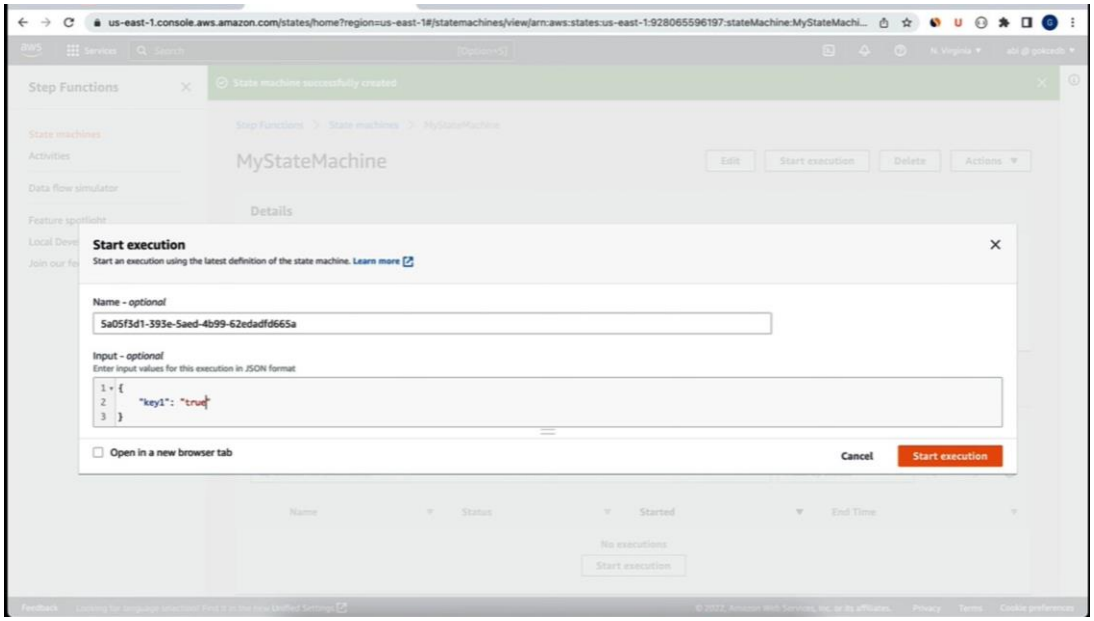
STEP 16: Create State Machine

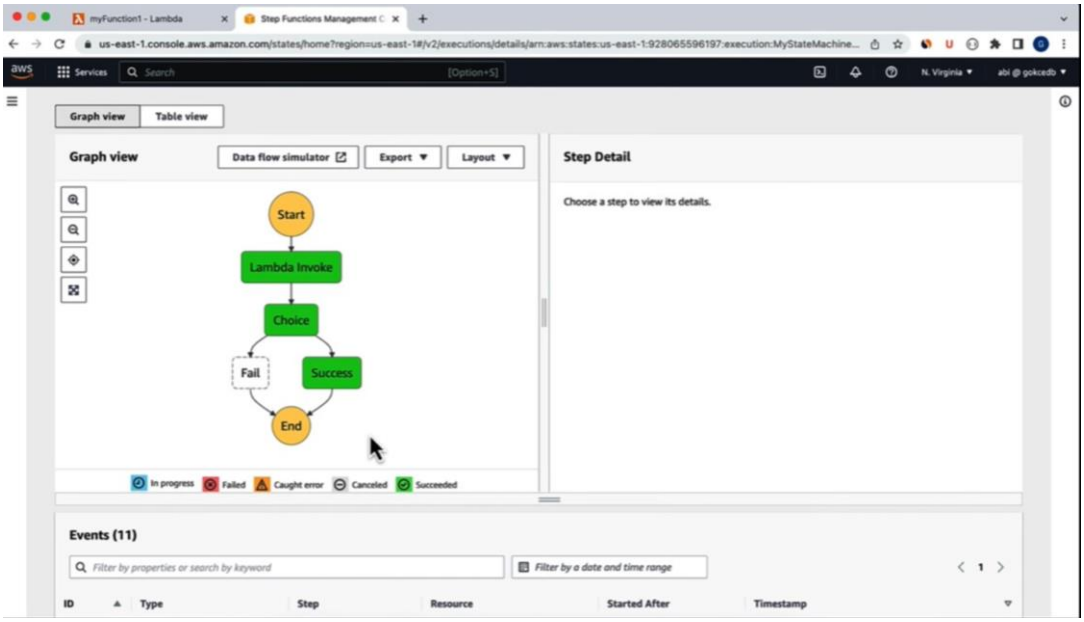


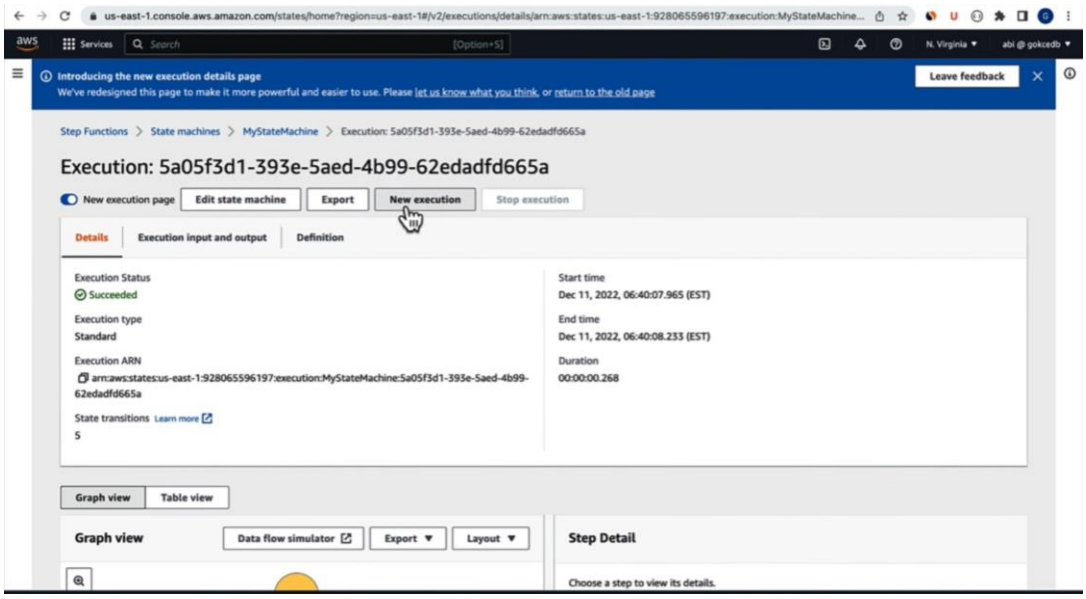
STEP 17: Start Execution



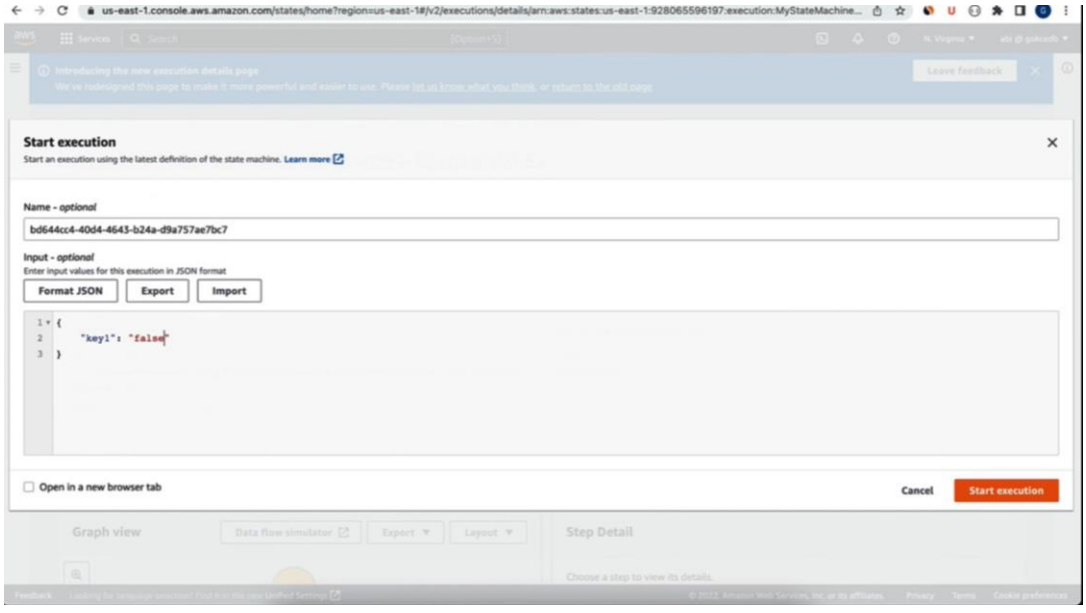
STEP 18: Give the Key Value True And Check

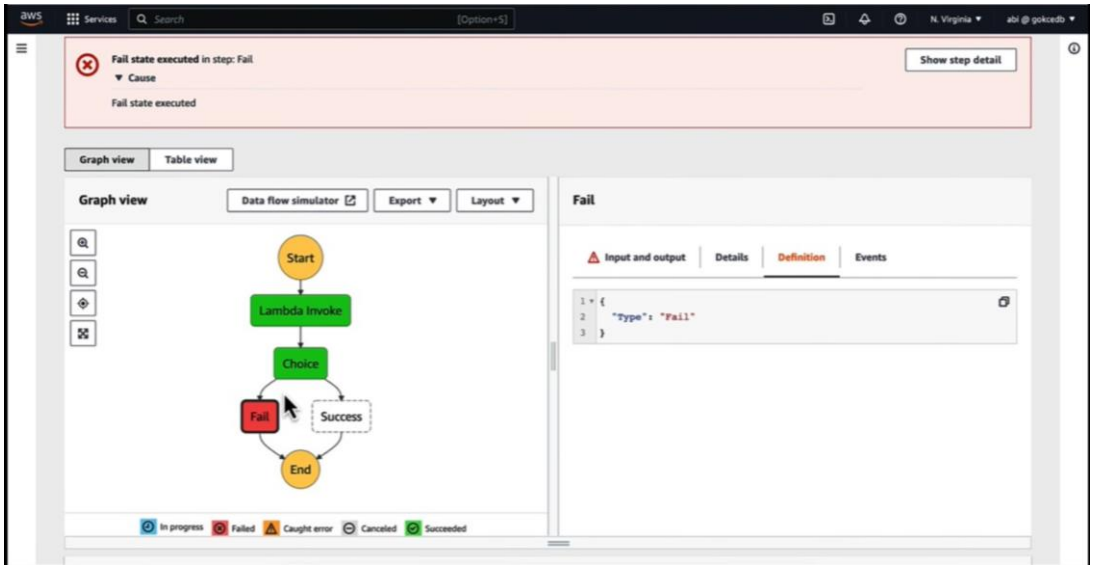






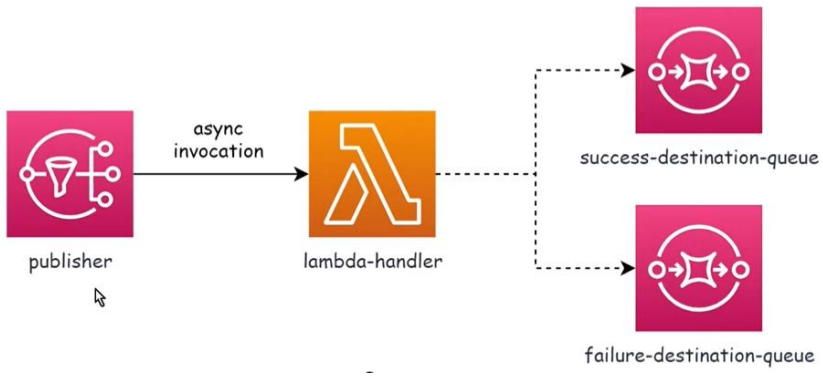
STEP 19: Give the Key Value False



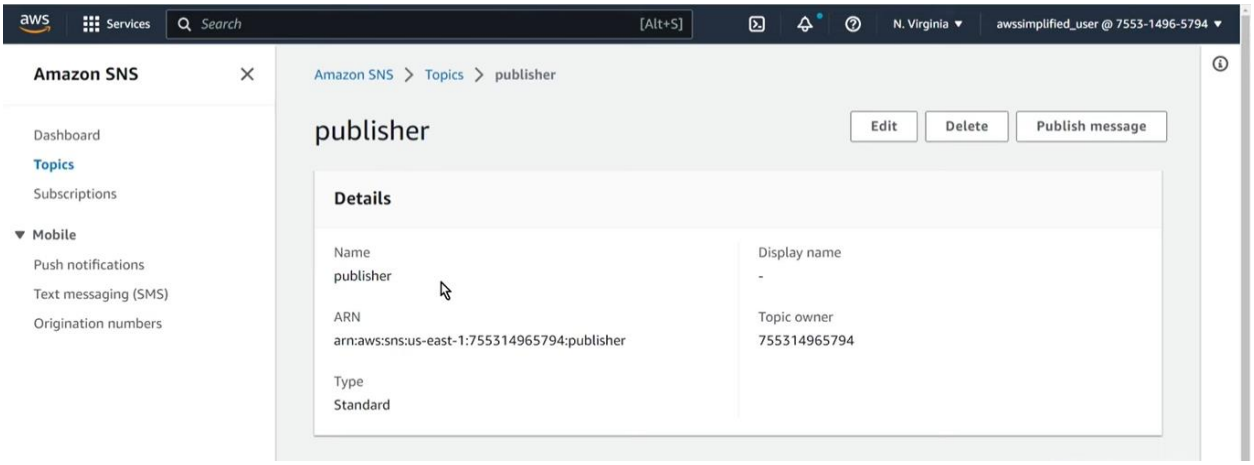


ADD DESINATION IN AWS LAMBDA

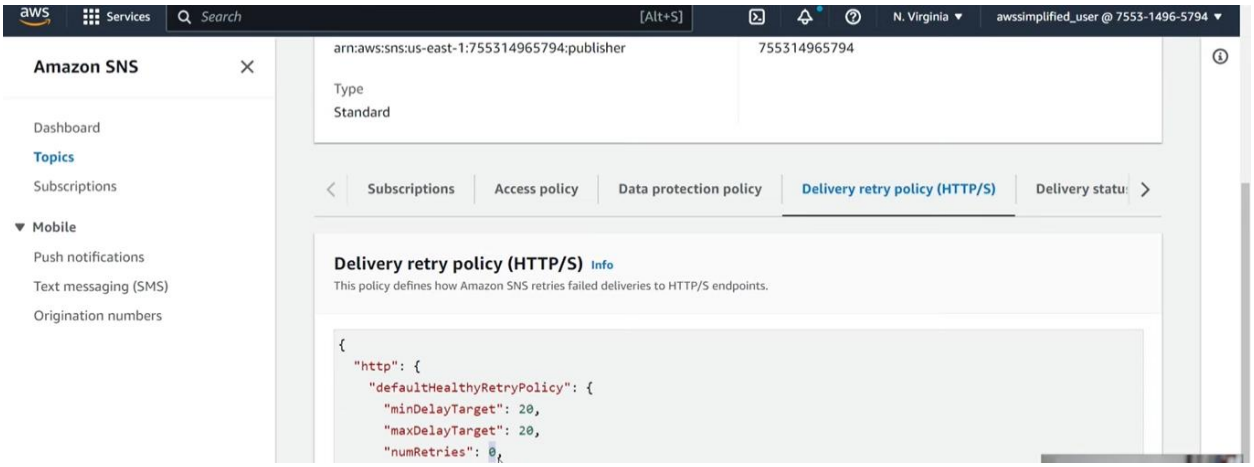
WORKFLOW DIAGRAM



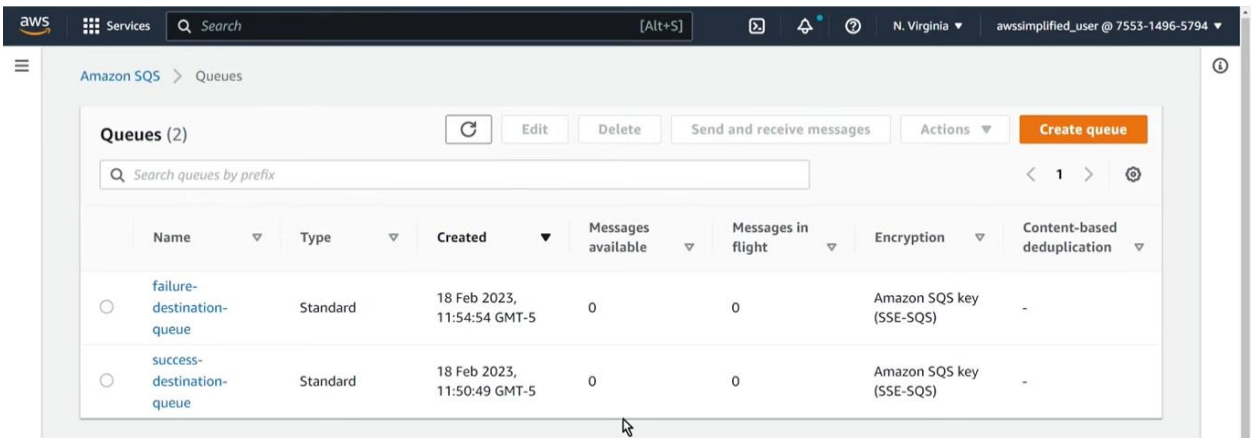
STEP 1: Open SNS service, create topic named as “publisher”.



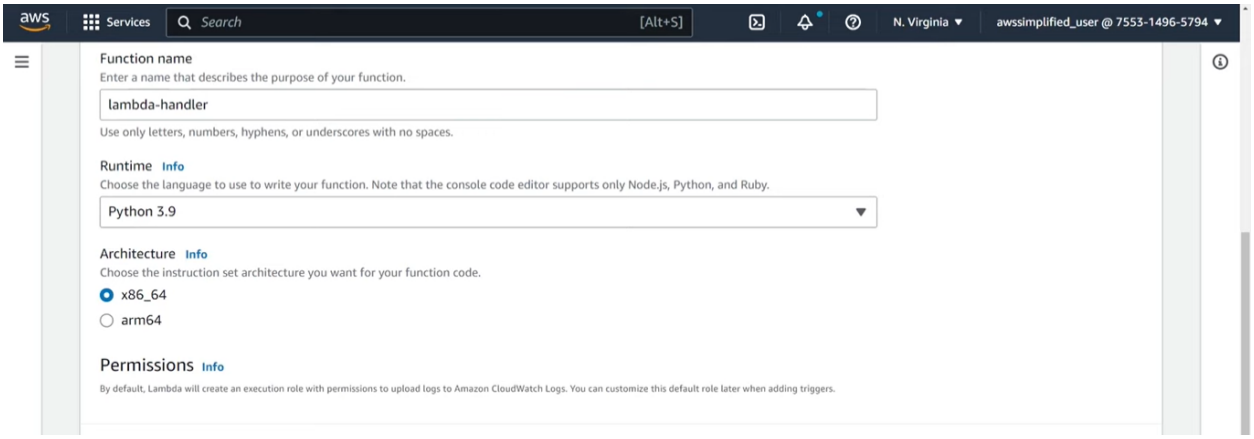
STEP 2: Change “numRetries” to 0.



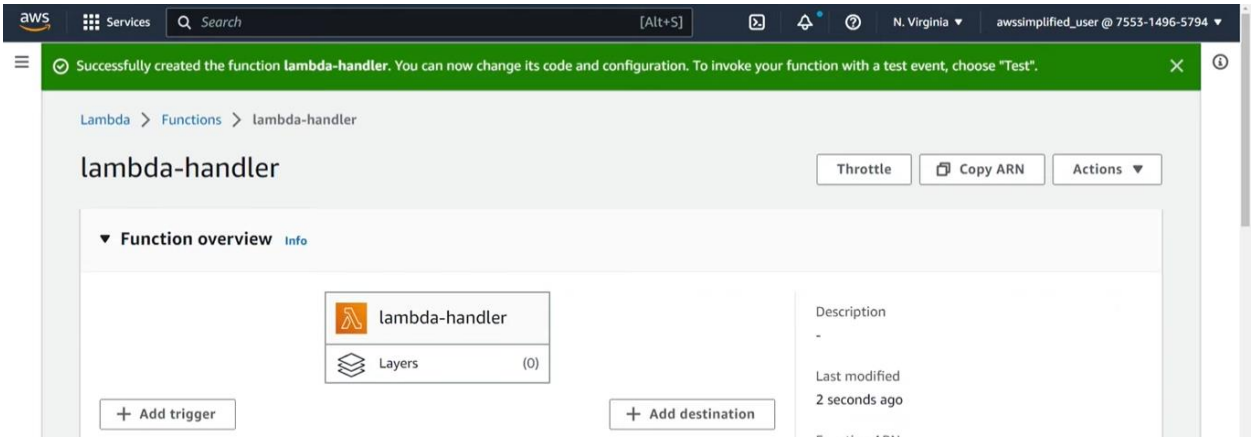
STEP 3: Open SQS service, create two queues namely “success-destination-queue” and “failure-destination-queue”.



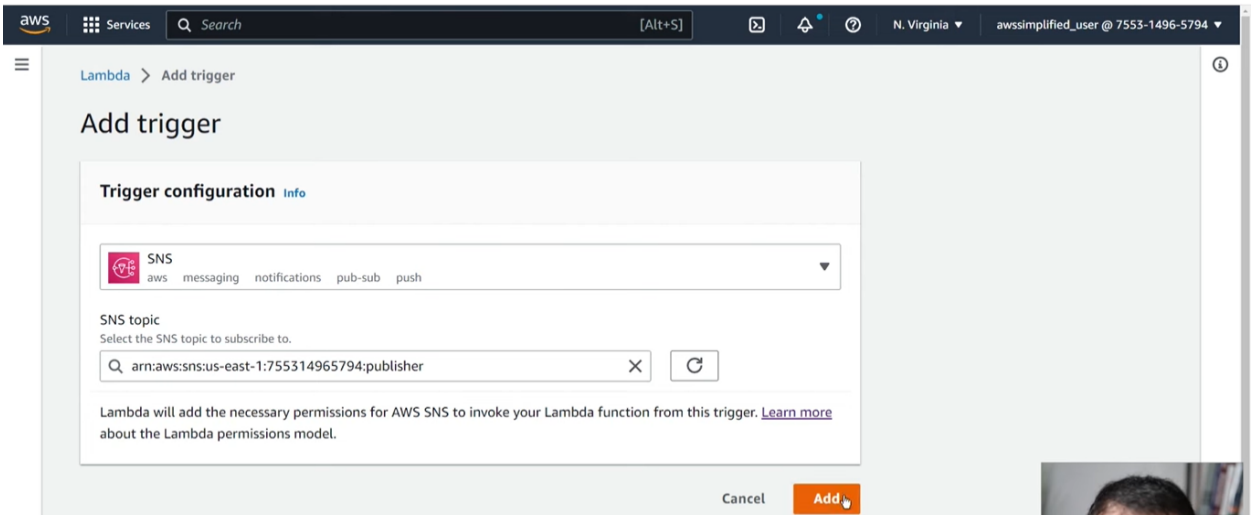
STEP 4: Create lambda function as “lambda-handler”.



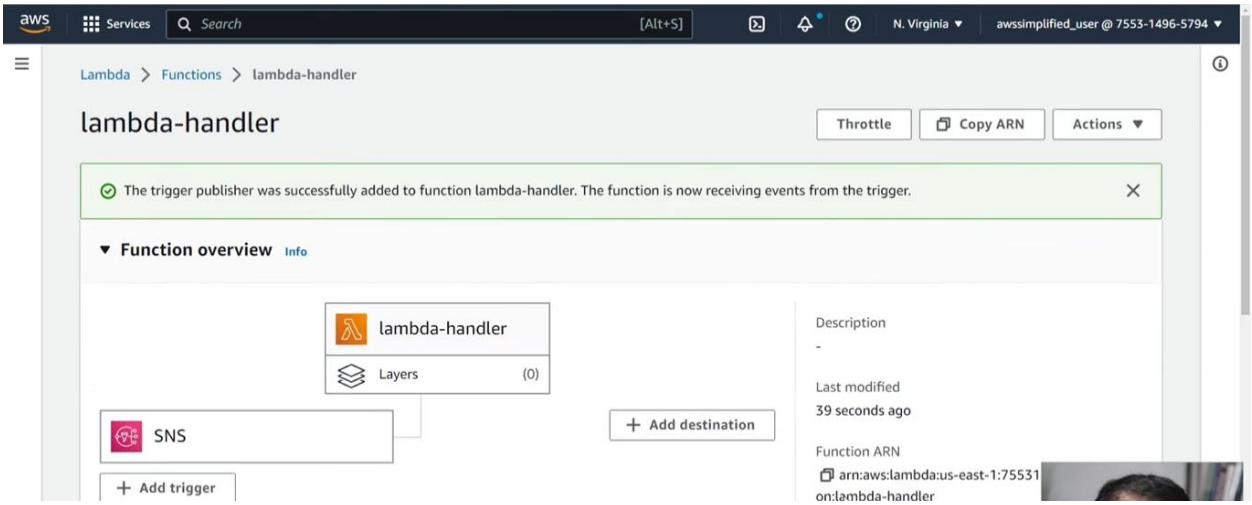
STEP 5: Lambda function created successfully.



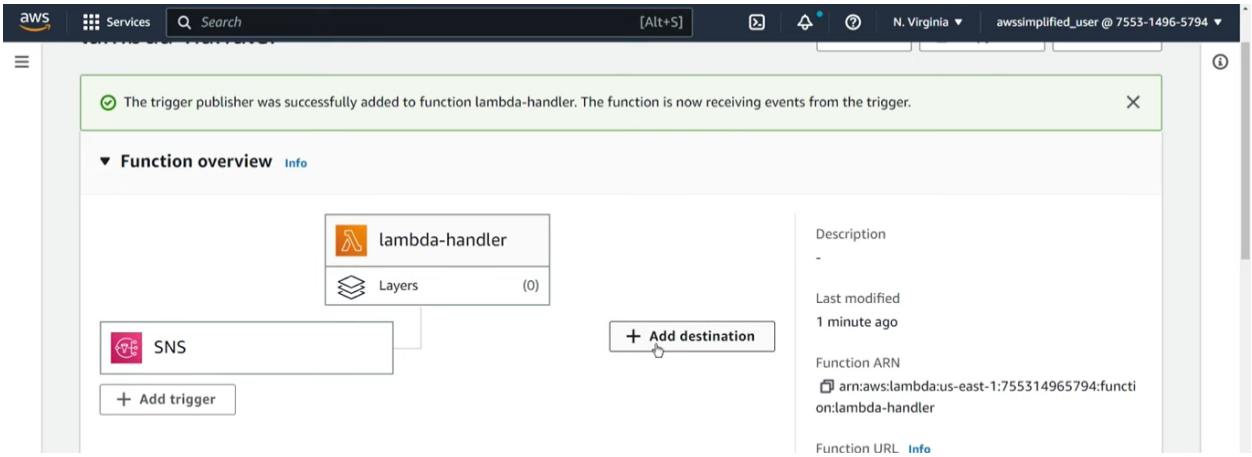
STEP 6: Add trigger as SNS and choose SNS topic as “publisher”.



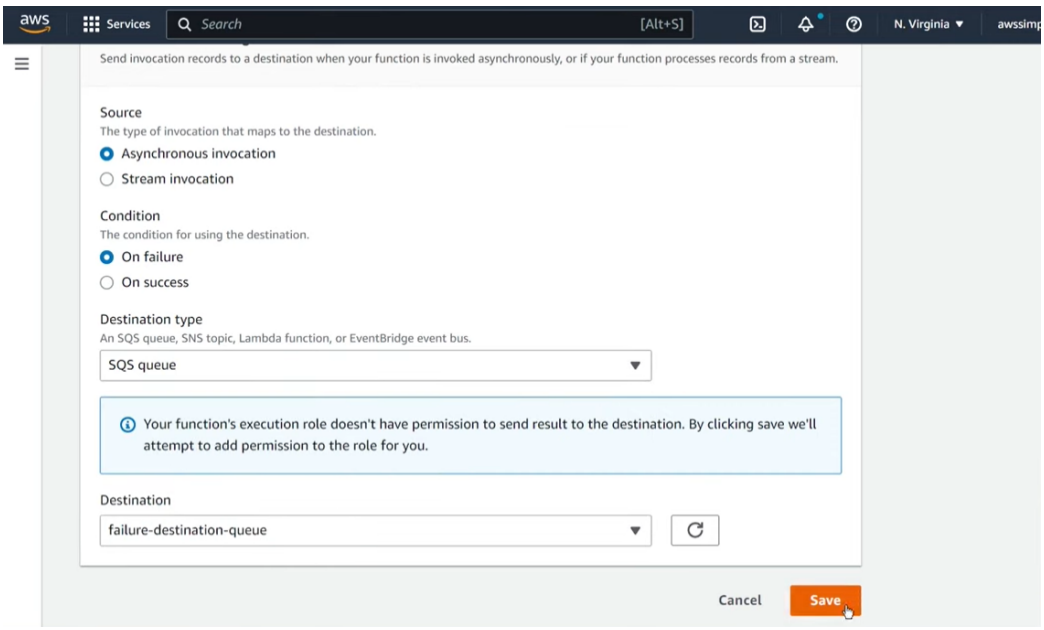
STEP 7: Trigger added successfully.



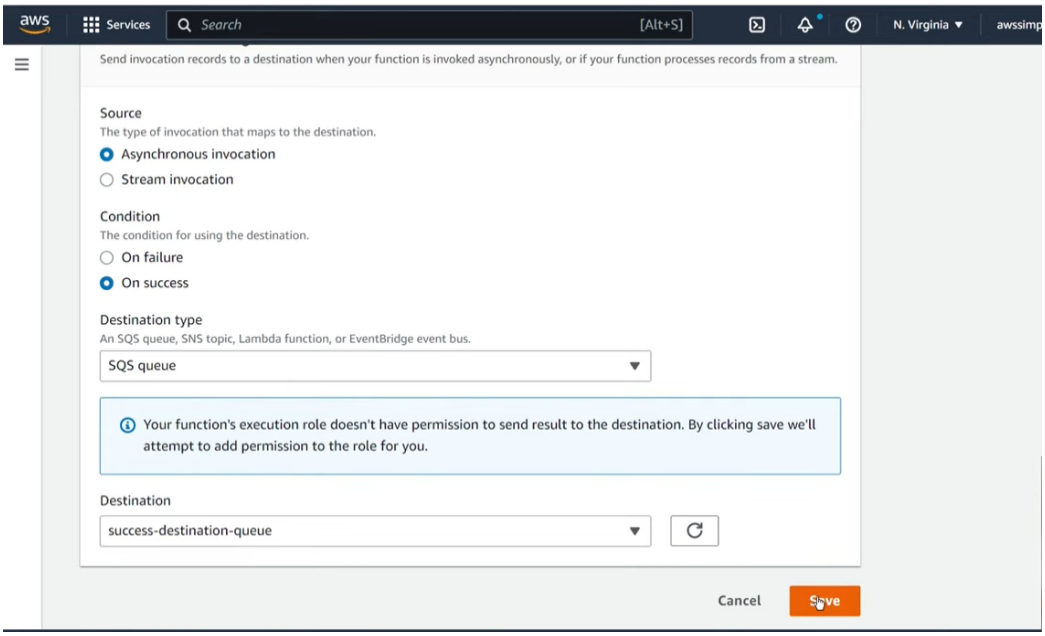
STEP 8: Click “Add destination”.



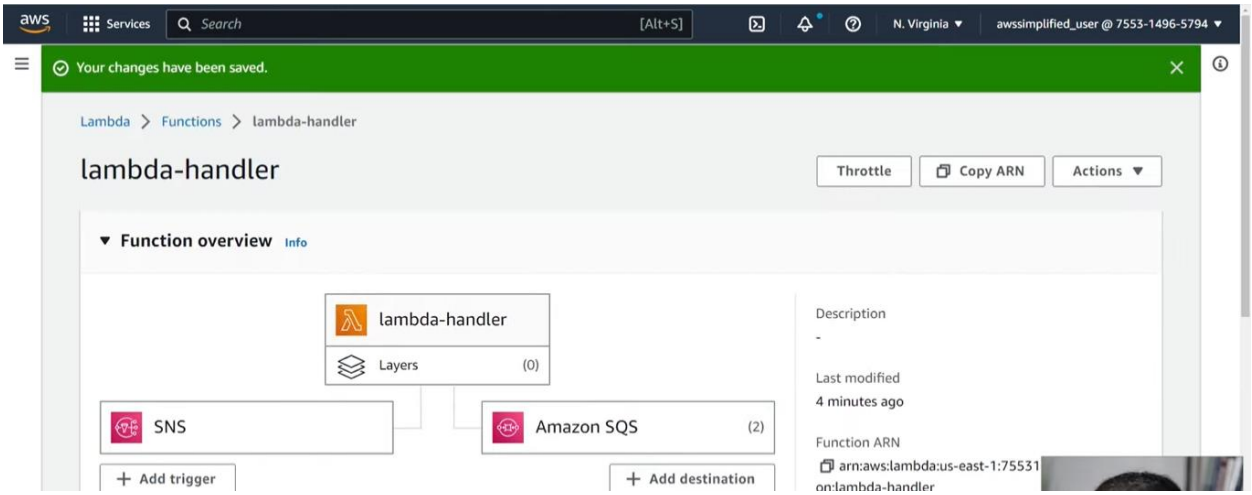
STEP 9: Choose destination type as “SQS queue” and destination as “failure-destination-queue”.



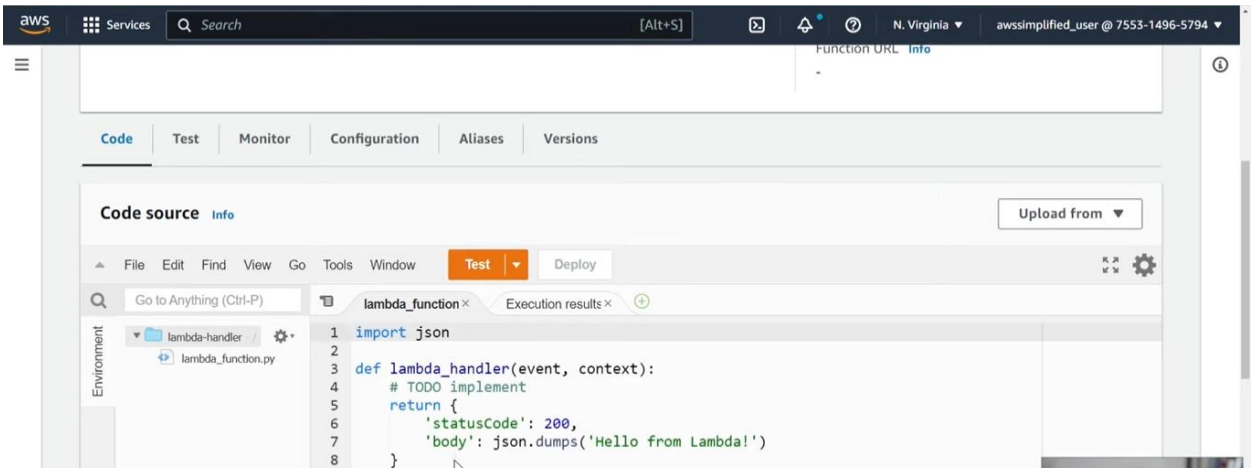
STEP 10: In another destination, Choose destination type as “SQS queue” and destination as “success-destination-queue”.



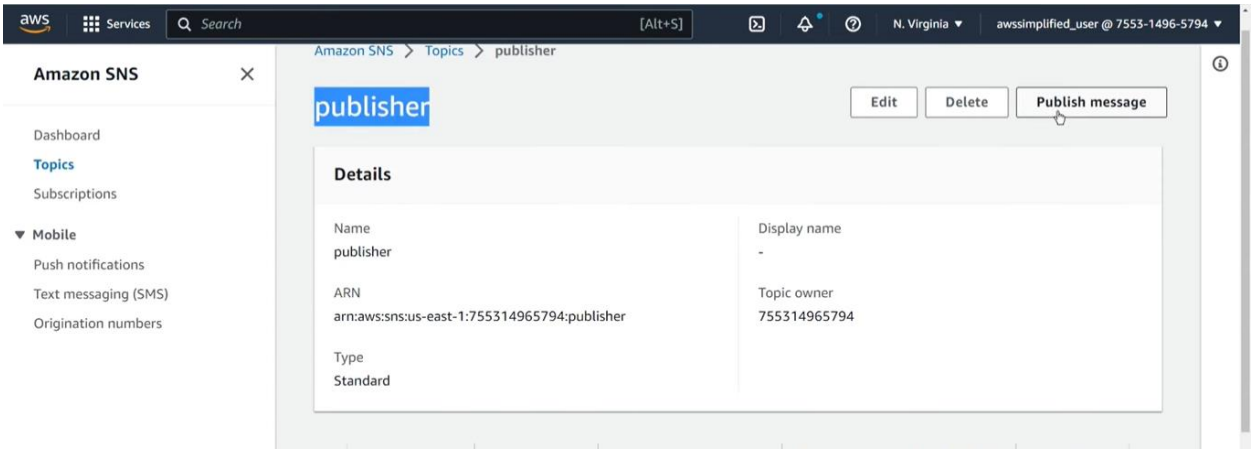
STEP 11: Here trigger and destination added successfully.



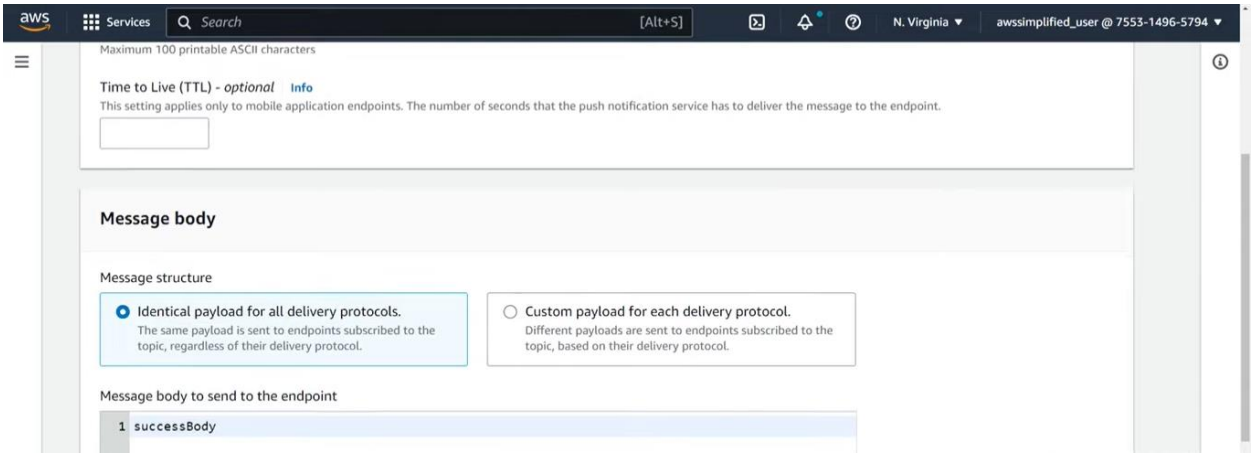
STEP 12: Here I have written a runnable code without error.



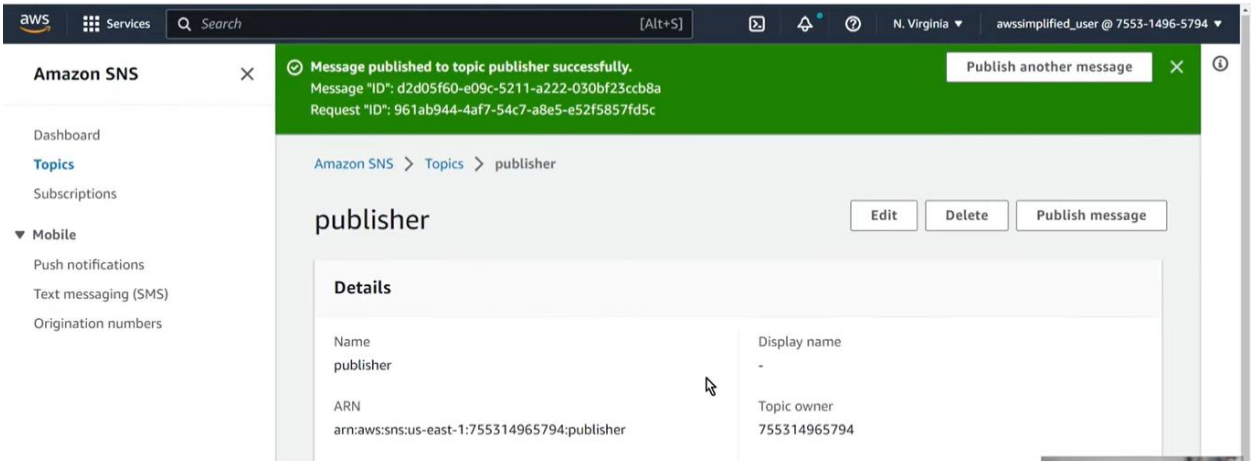
STEP 13: Now go to publisher topic and click “Publish message”.



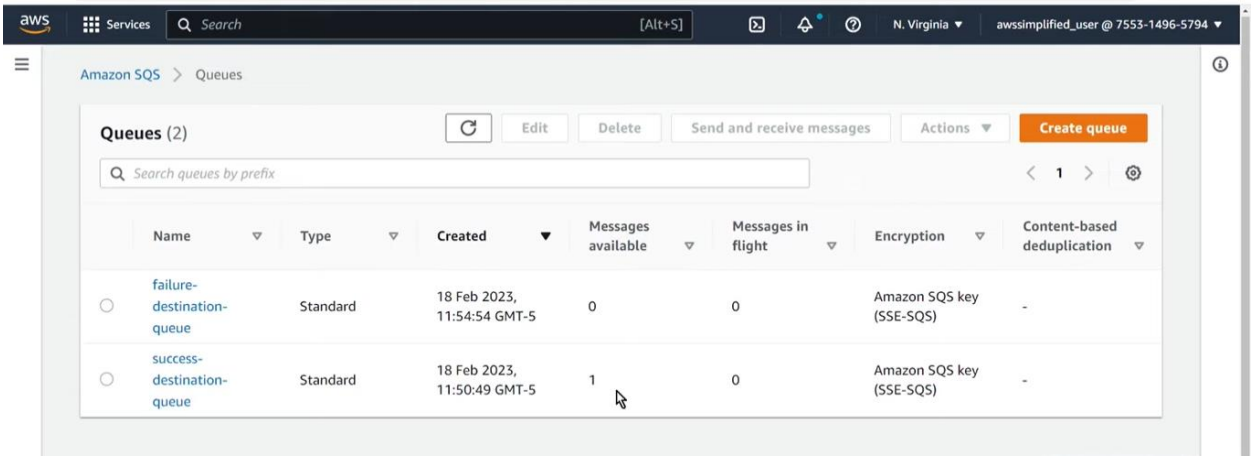
STEP 14: Enter message in a message body and click publish message.



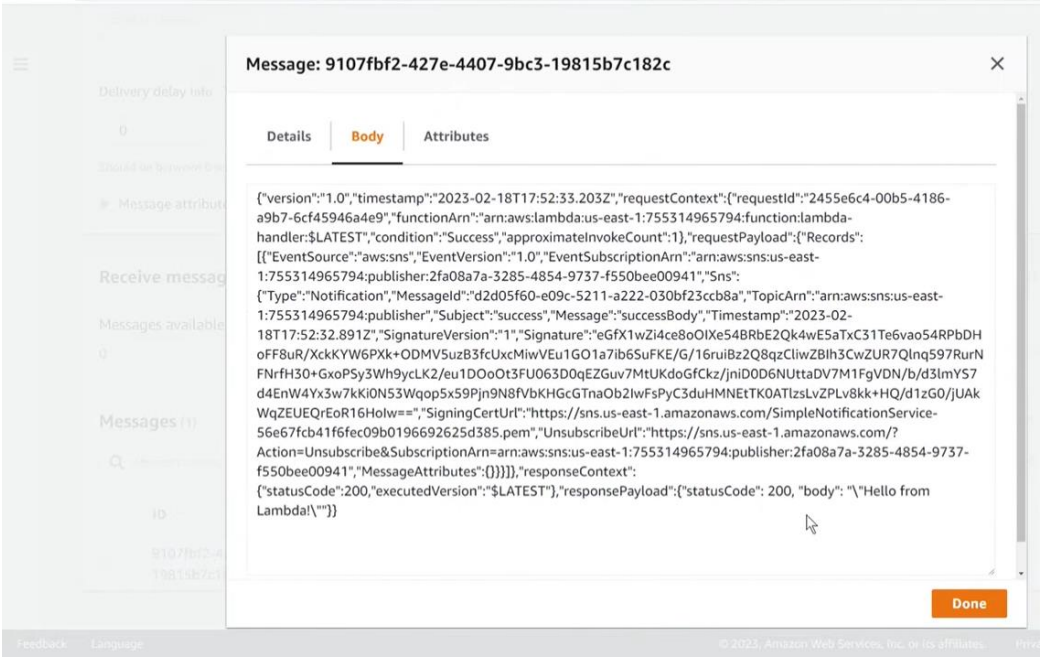
STEP 15: Message published successfully.



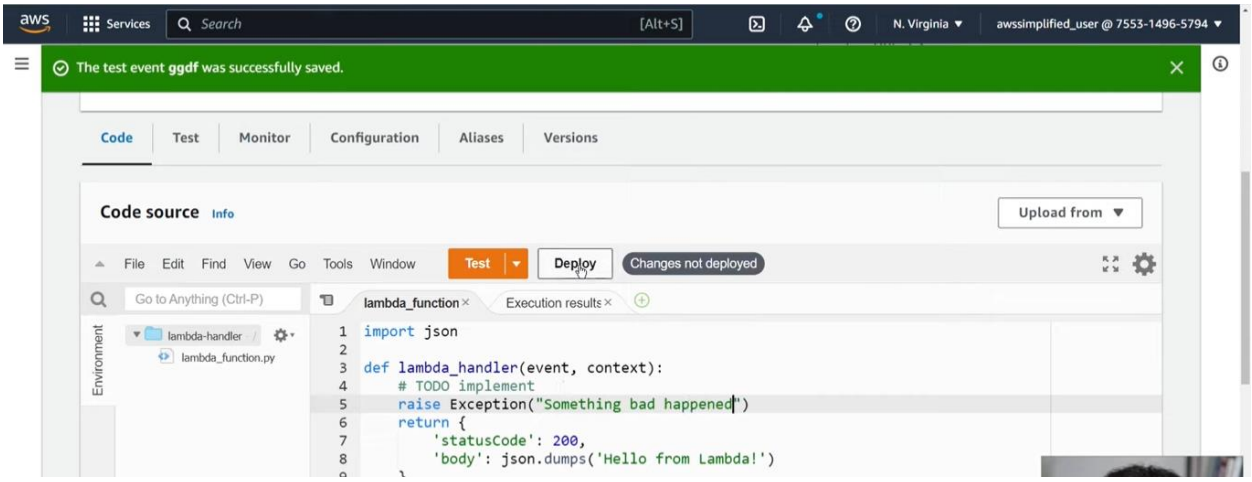
STEP 16: Check the success queue, one message added to the queue.



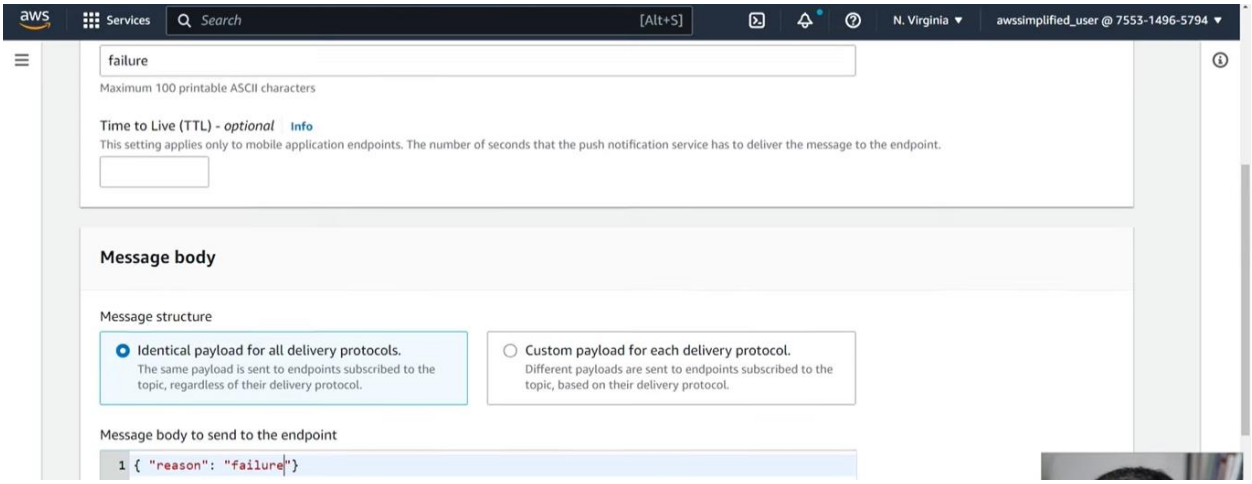
STEP 17: Open success queue and explore the message.



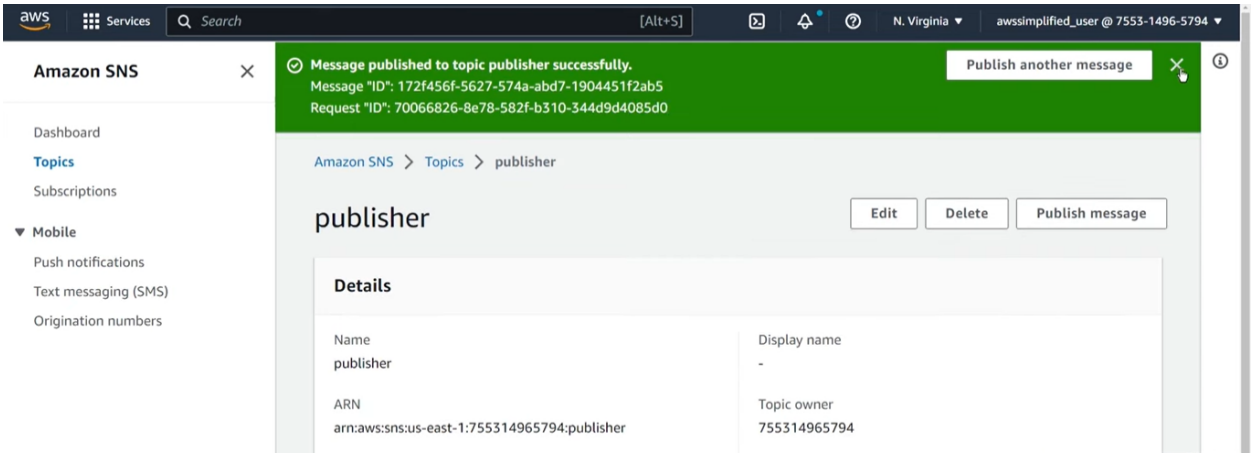
STEP 18: Here, I created an exception.



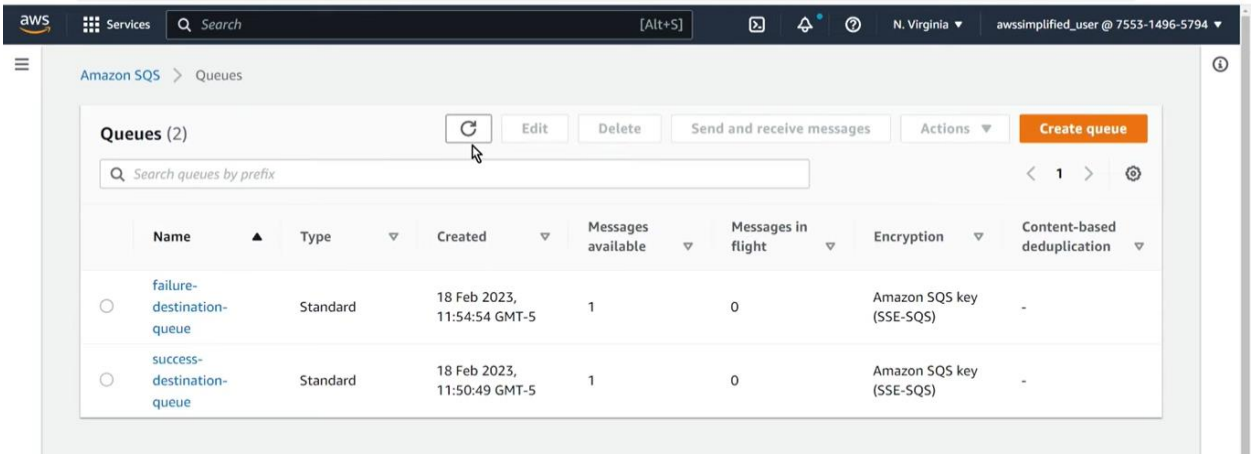
STEP 19: Open publisher topic and send a message again.



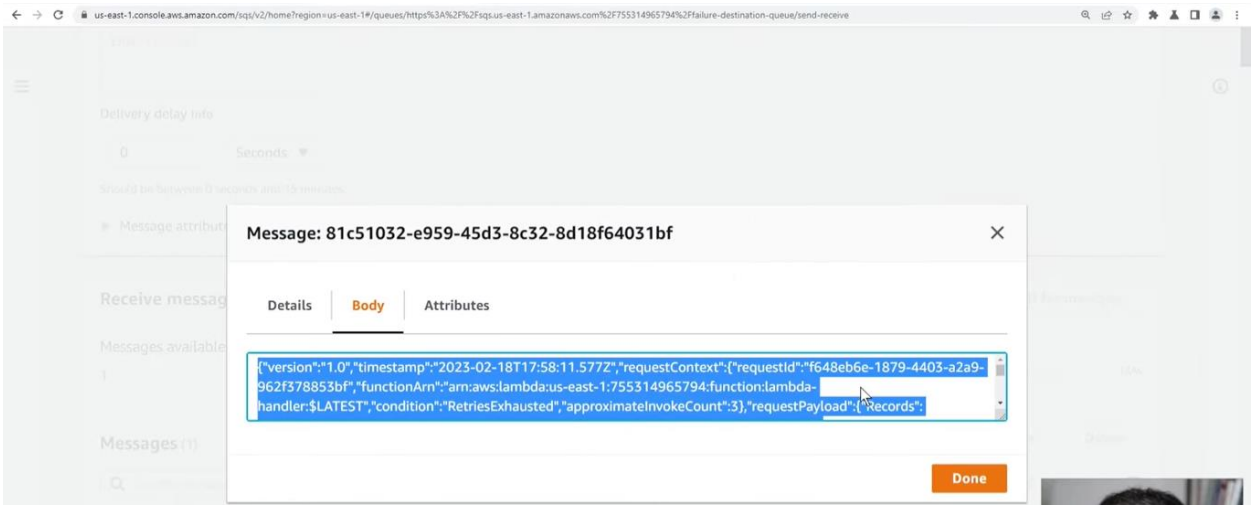
STEP 20: Message published successfully.



STEP 21: Now one message added to failure queue.

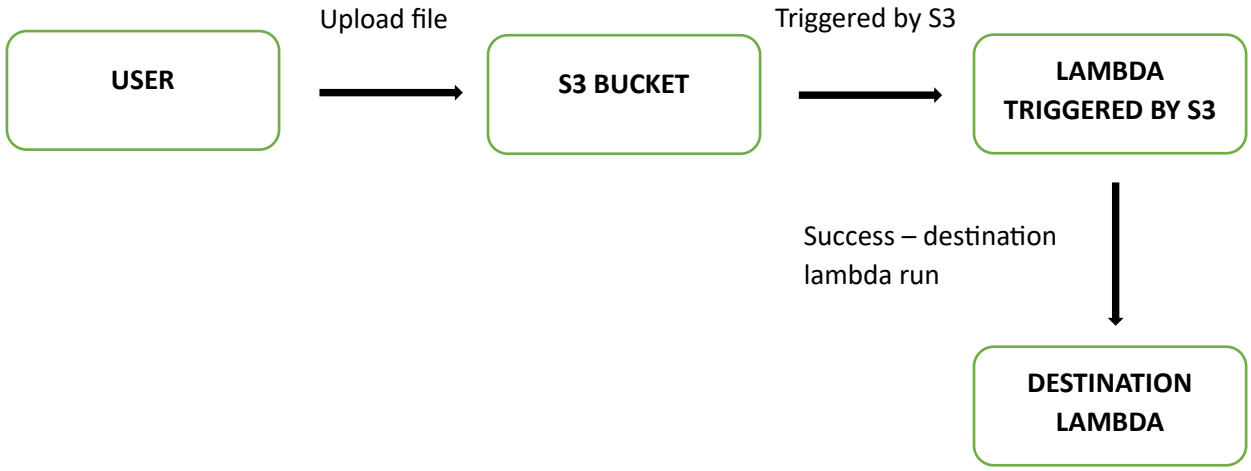


STEP 22: Open and explore failure queue to see the message.

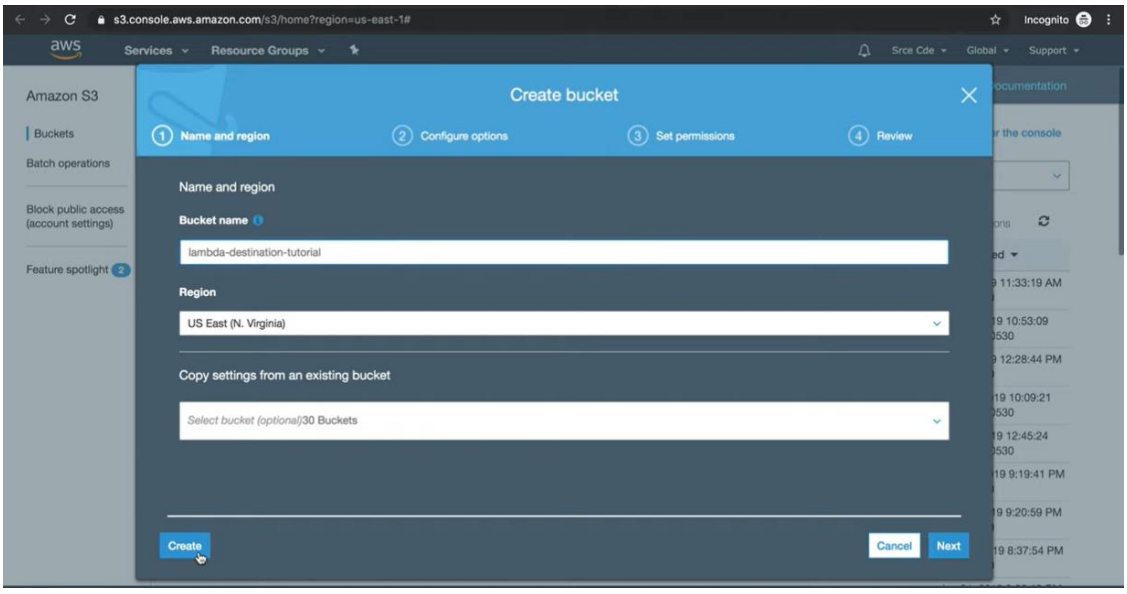


ADD DESTINATION IN AMAZON LAMBDA (WITH S3)

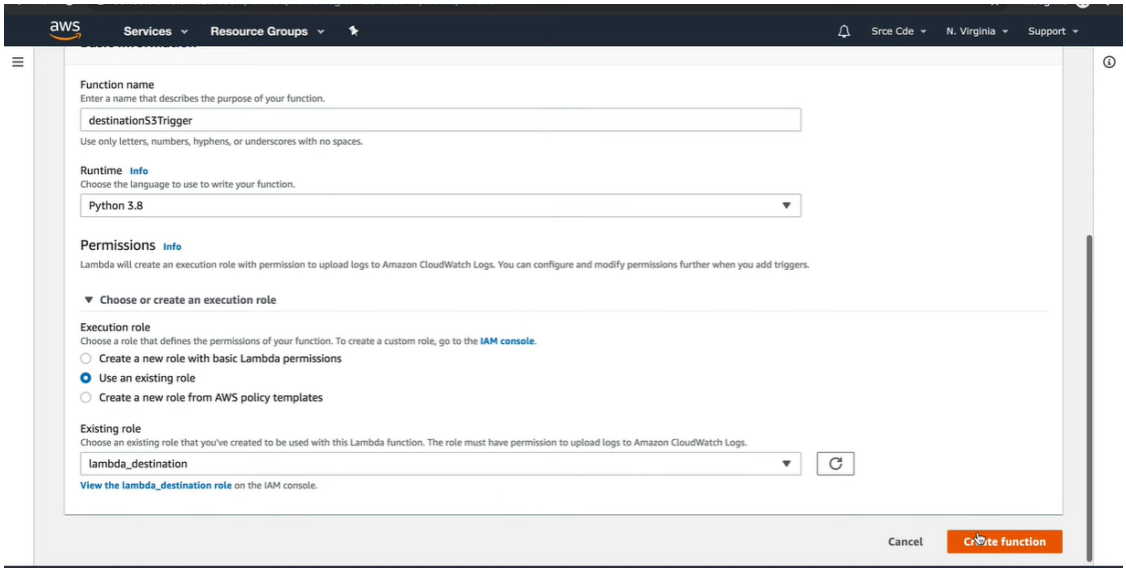
WORKFLOW DIAGRAM



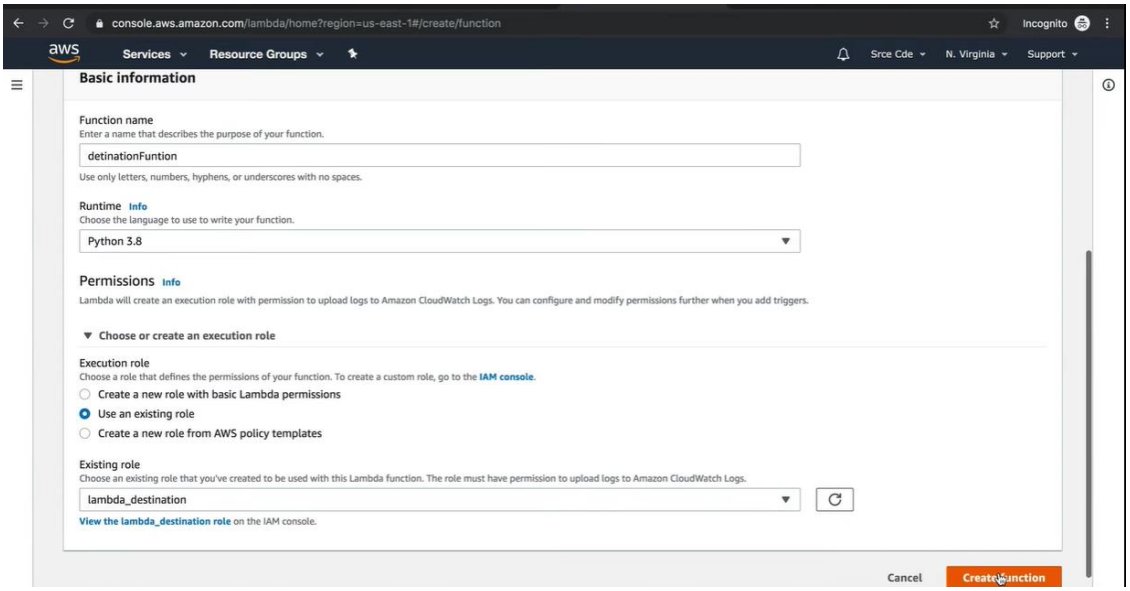
STEP 1: Create a bucket.



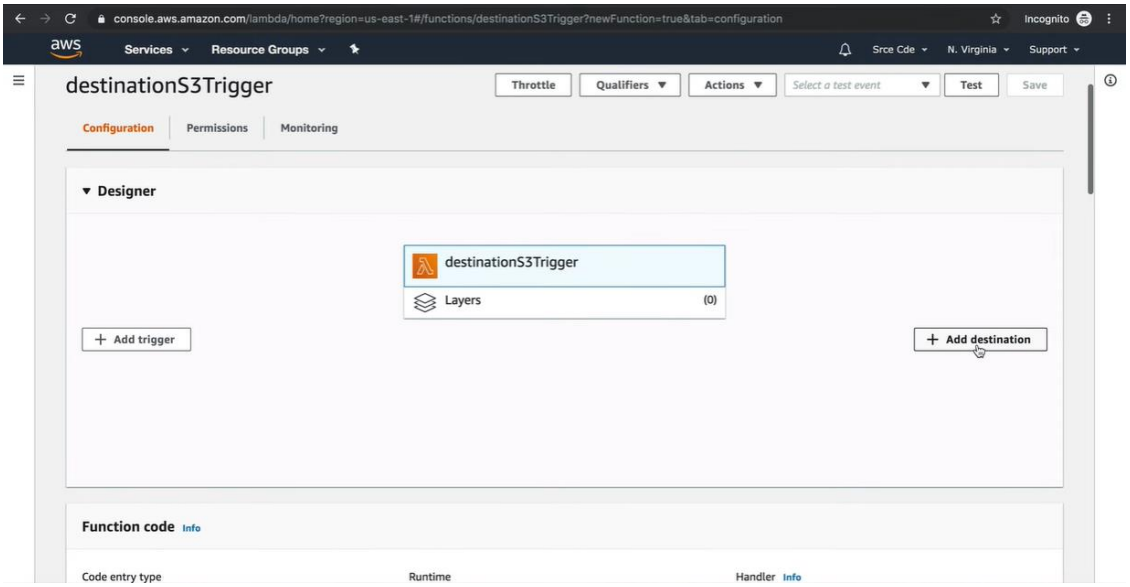
STEP 2: Create a lambda function for triggering.



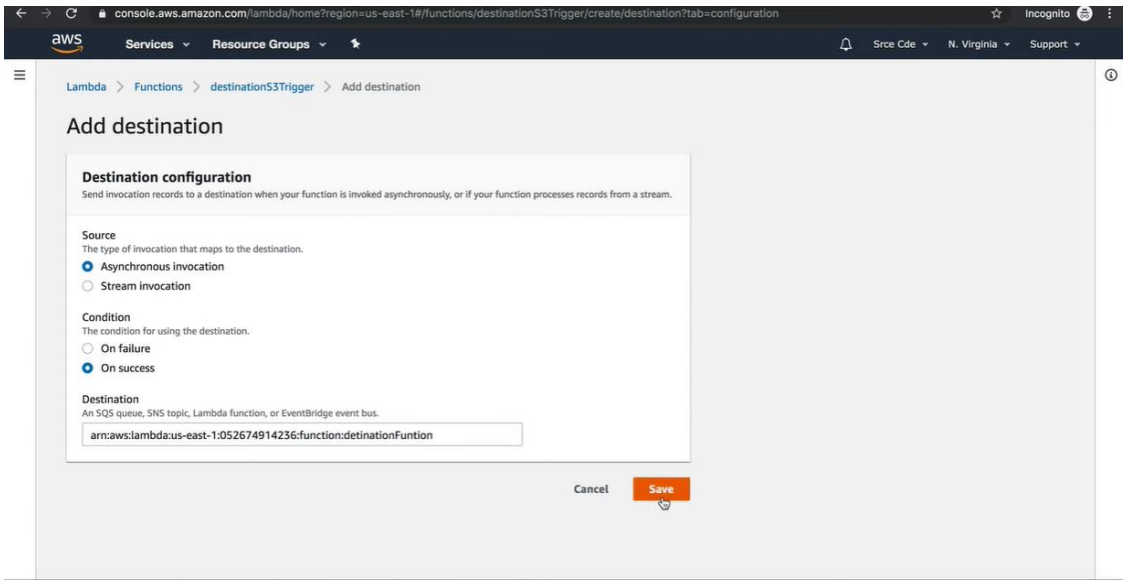
STEP 3: Create another lambda function for destination.



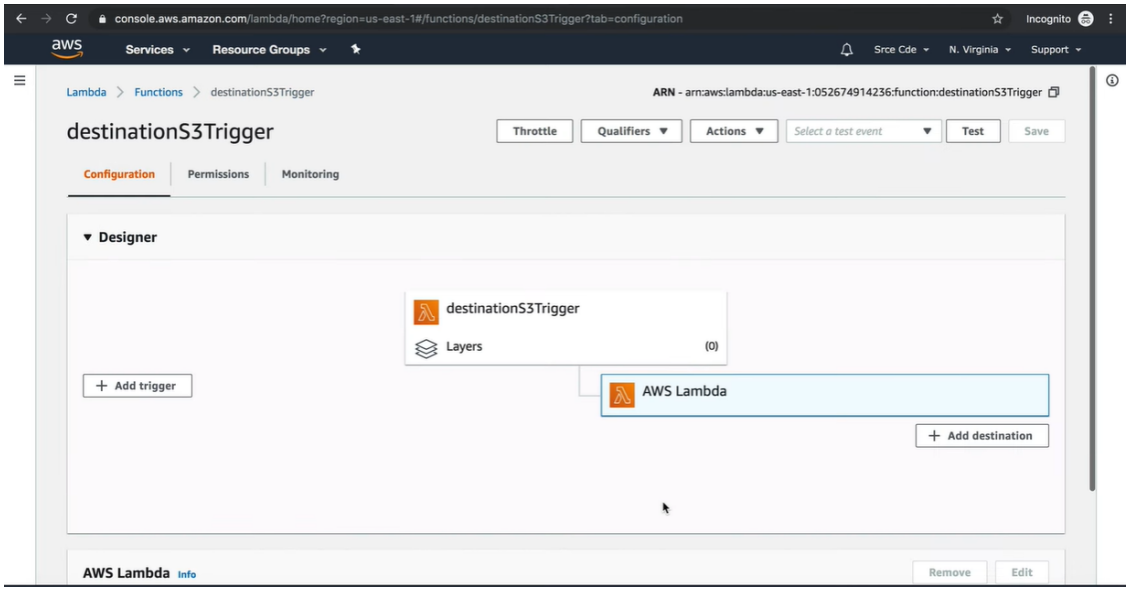
STEP 4: Click “Add destination”.



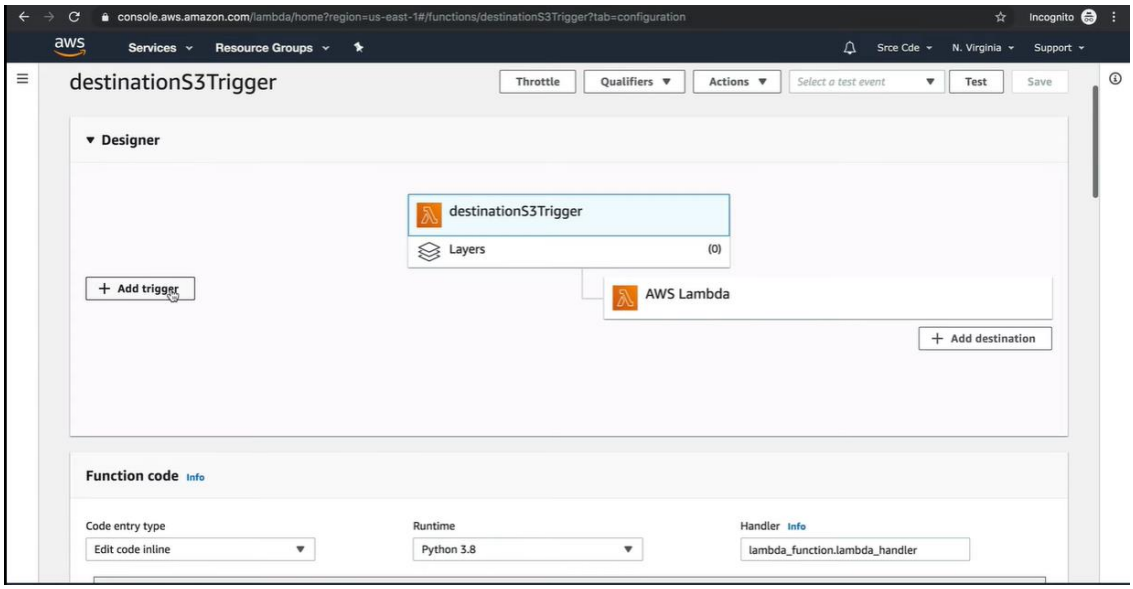
STEP 5: Add ARN for destination.



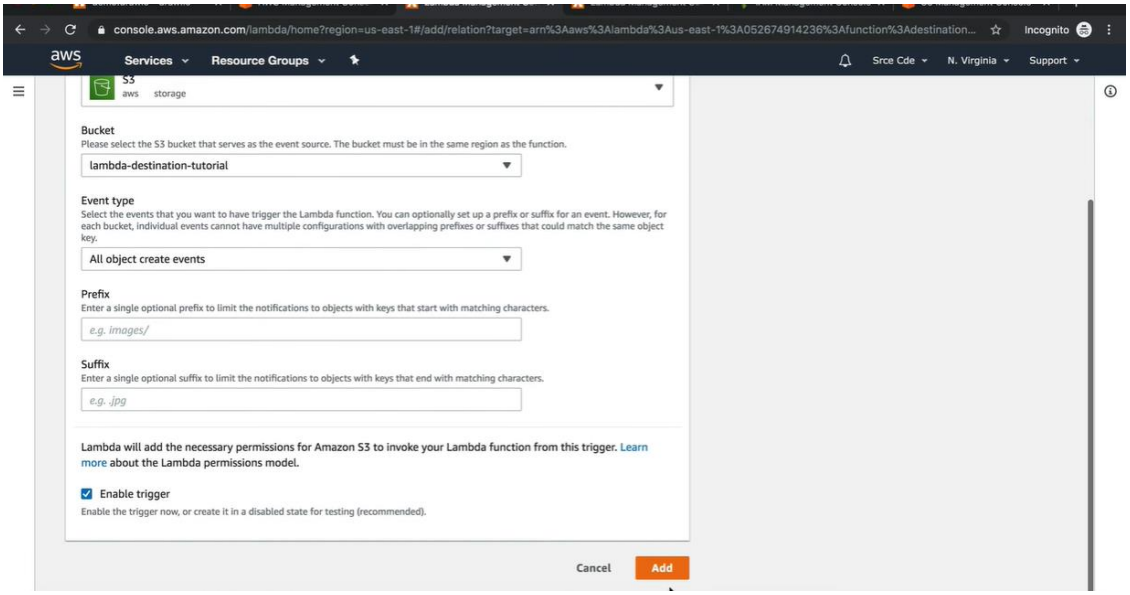
STEP 6: Destination added successfully.



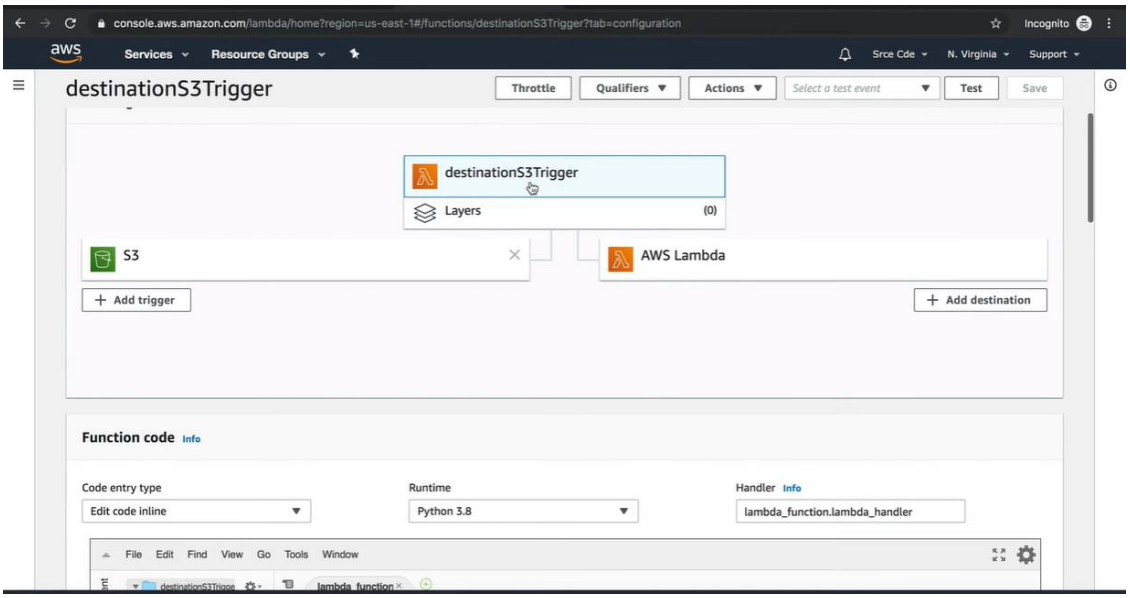
STEP 7: Click “Add trigger”.



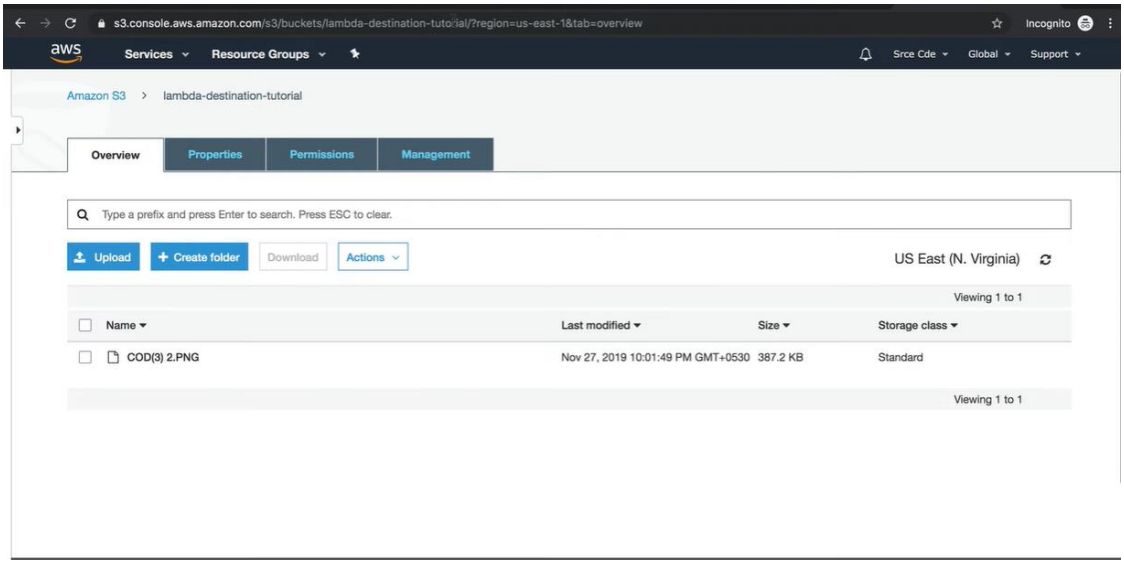
STEP 8: Choose S3 and bucket and click “Add”.



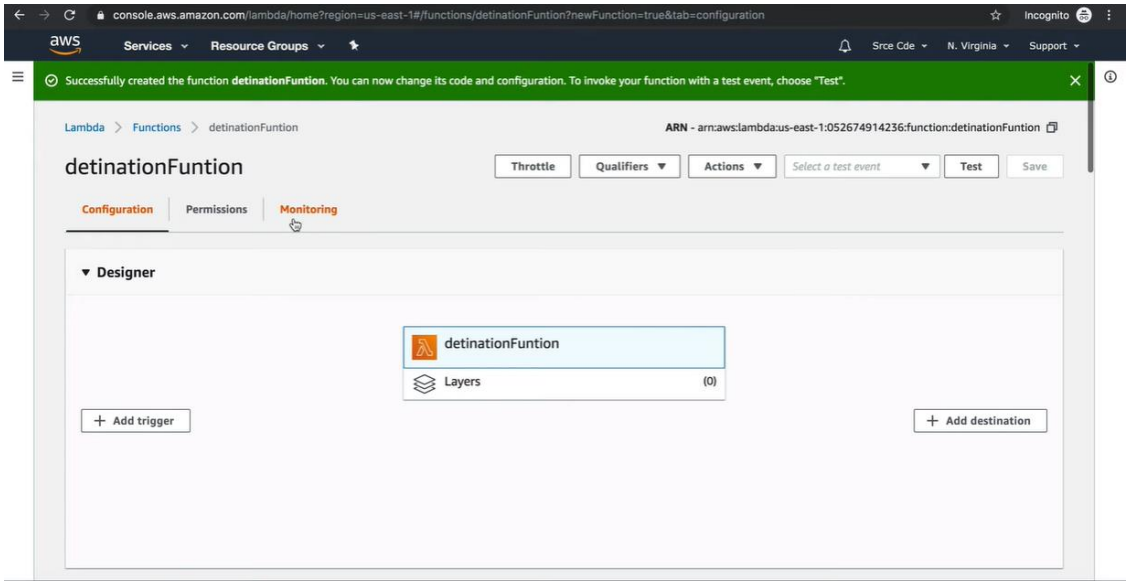
STEP 9: Trigger added successfully.



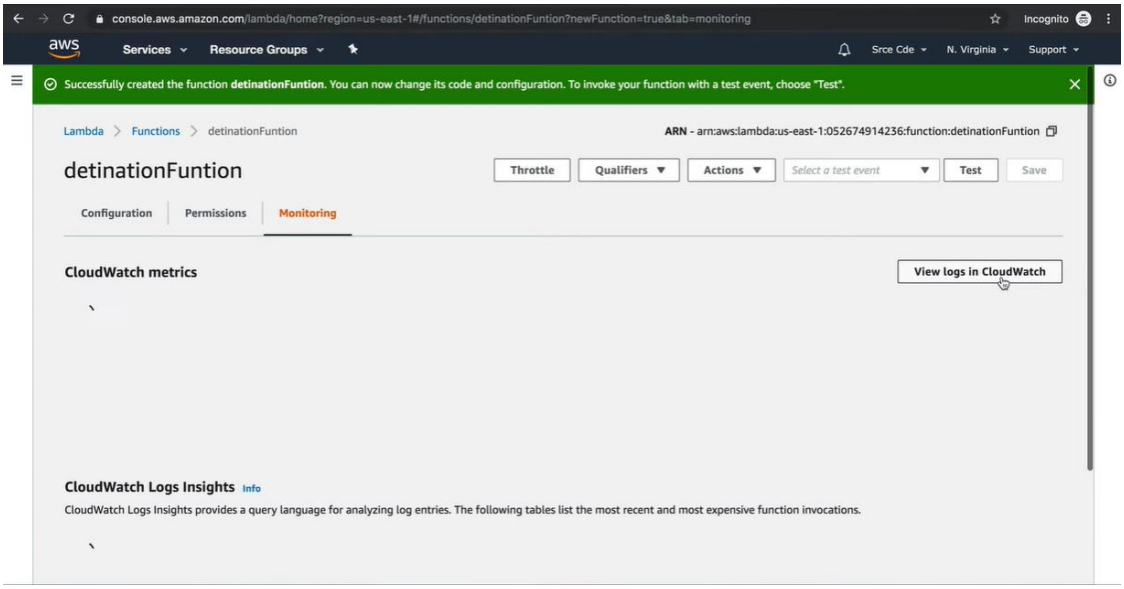
STEP 10: Upload one file in S3 bucket.



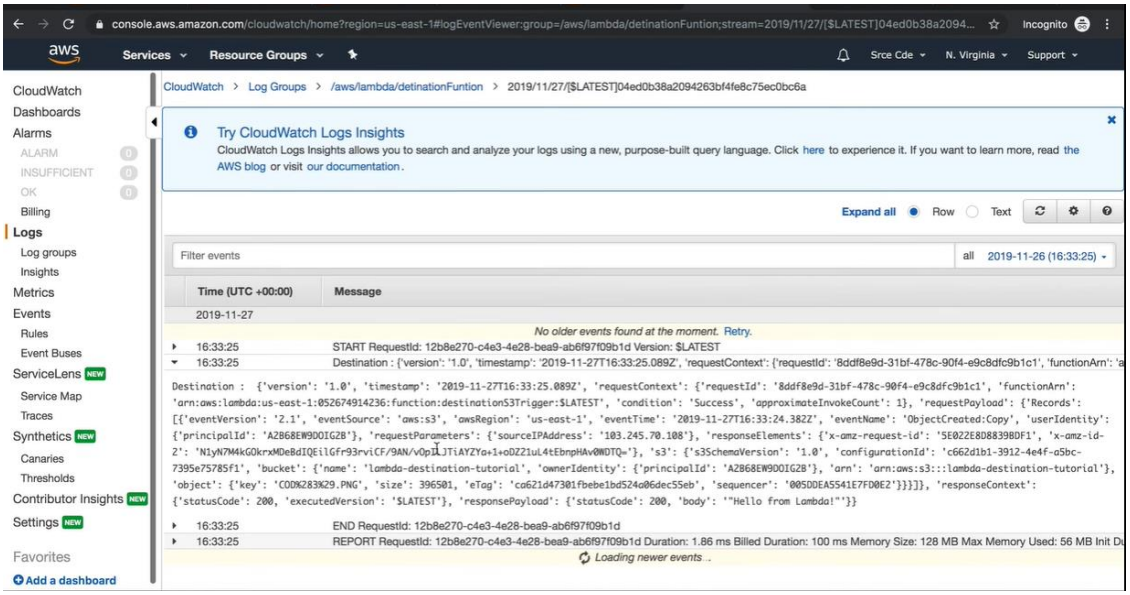
STEP 11: It will trigger the lambda function and destination lambda function will run successfully.



STEP 12: Click “Monitoring” and click “View logs in CloudWatch”.

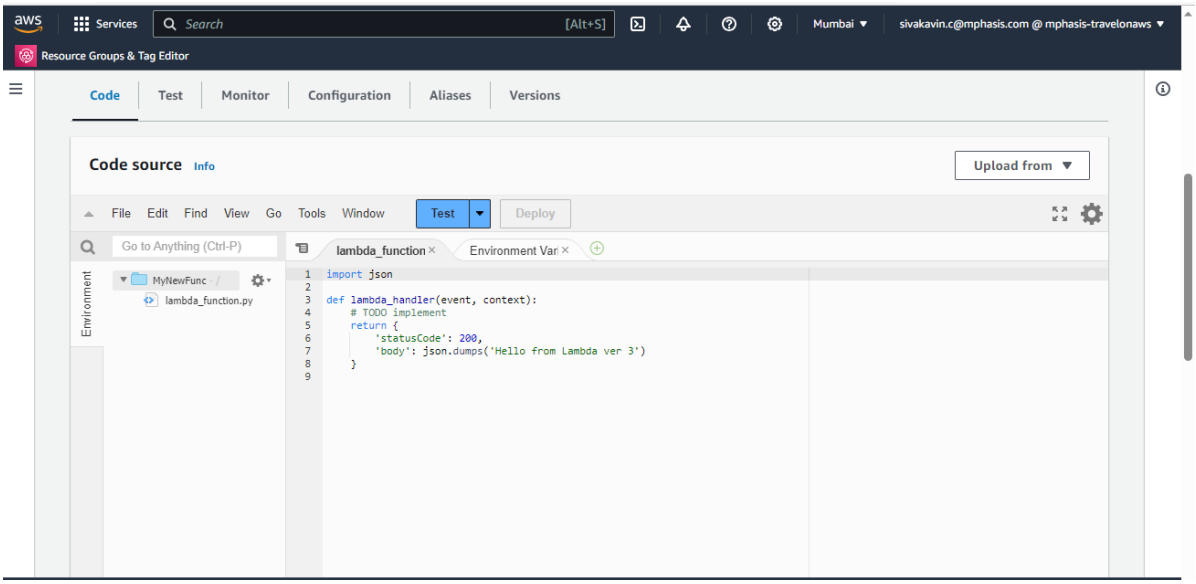


STEP 13: Here destination run successfully.

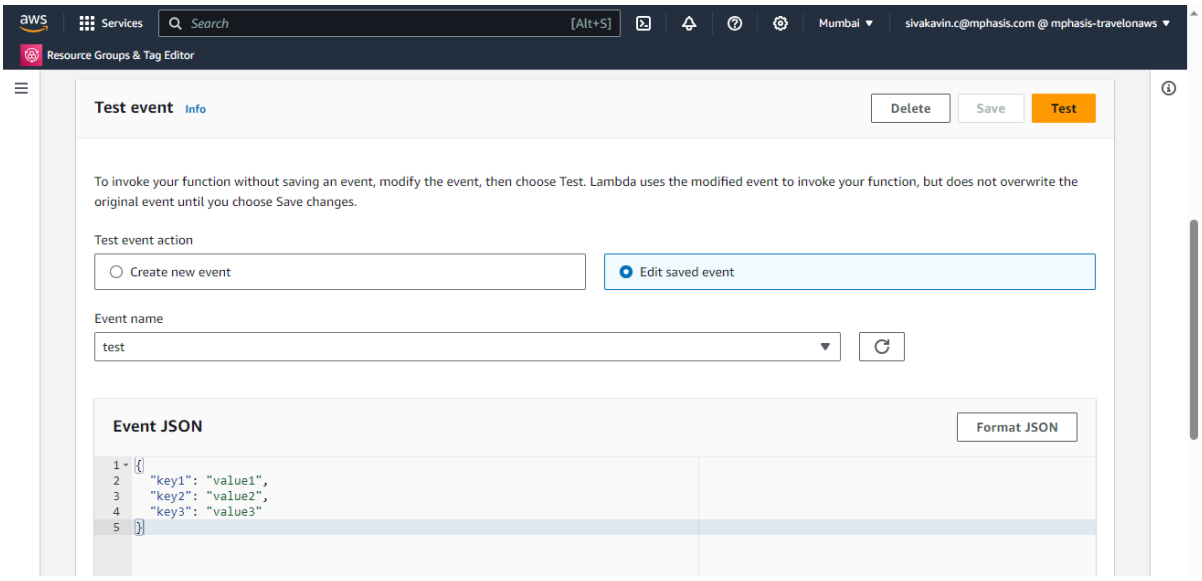


CODE - TEST - MONITOR - CONFIGURATION - ALIASES – VERSIONS

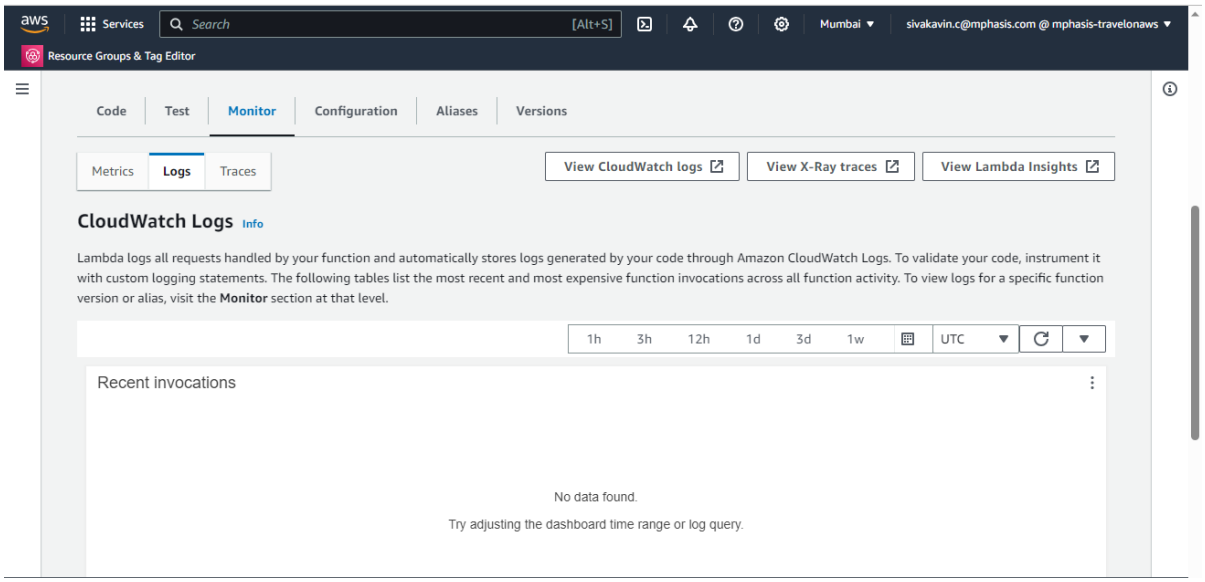
CODE: In coding part, write a code and deploy it.



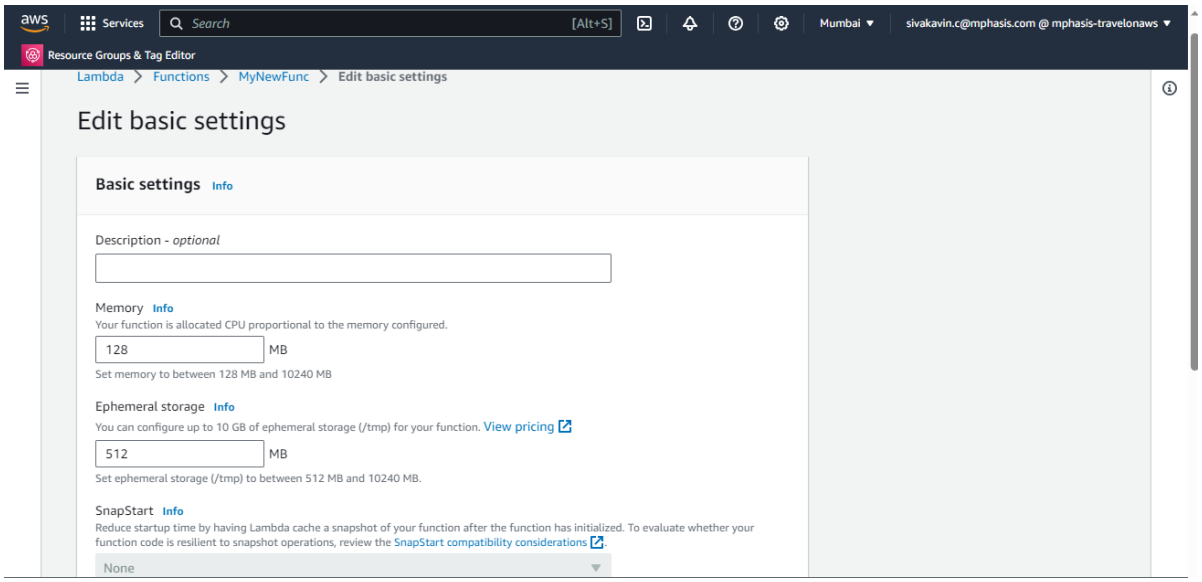
TEST: In test part, enter event name and give test parameters and test.



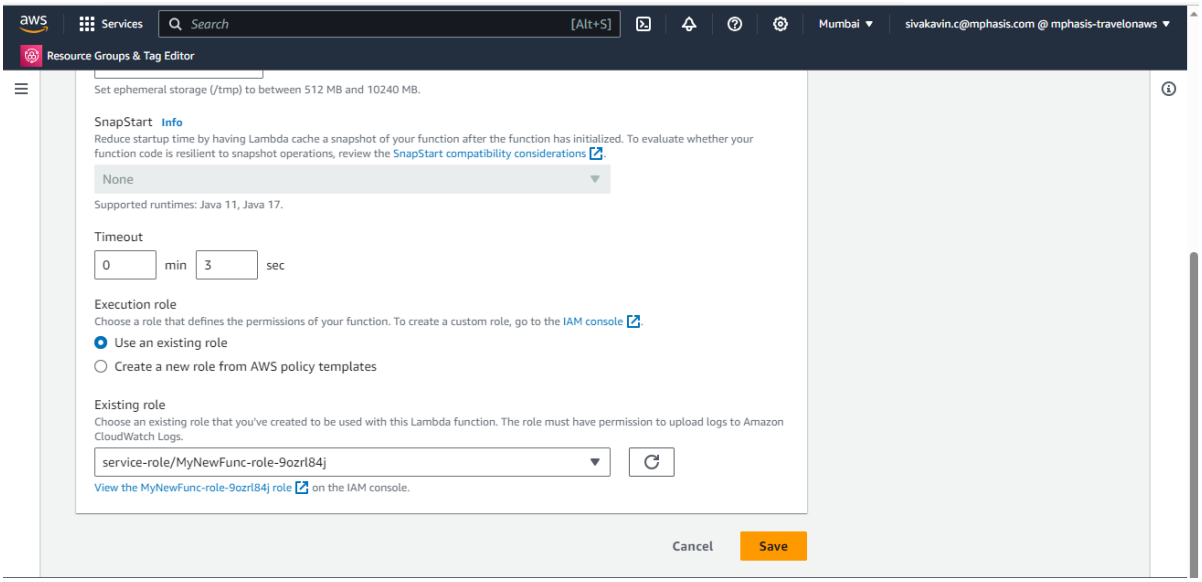
MONITOR: In this part, you can monitor through cloud watch, x-ray traces and lambda insights.



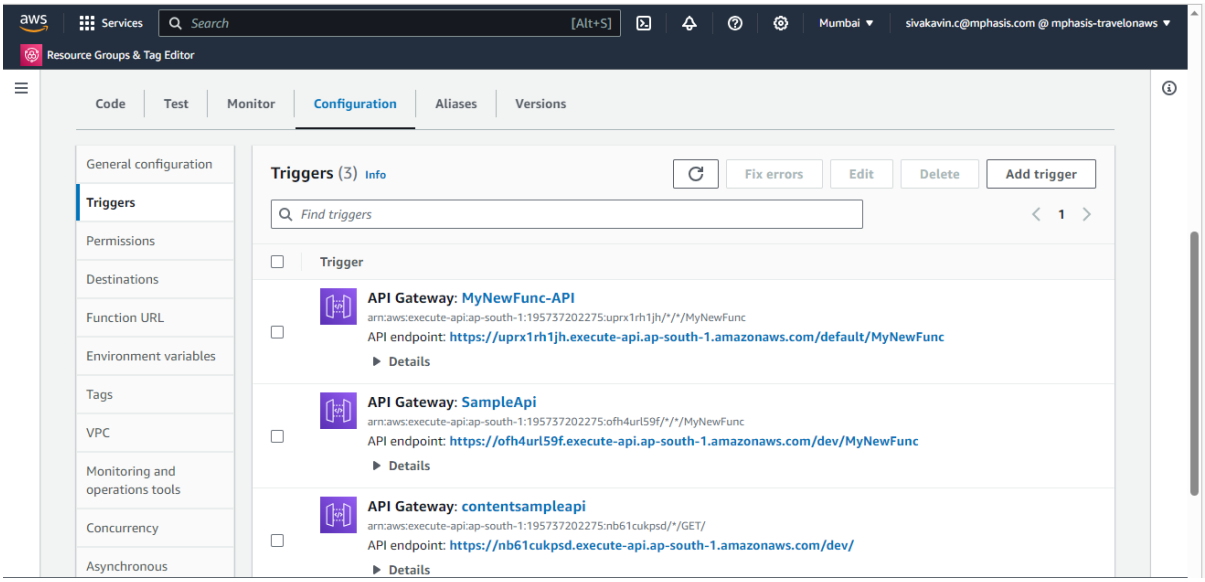
CONFIGURATION: In general permission, we can edit memory size, ephemeral storage, enable snapStart (only appliable for java runtime)



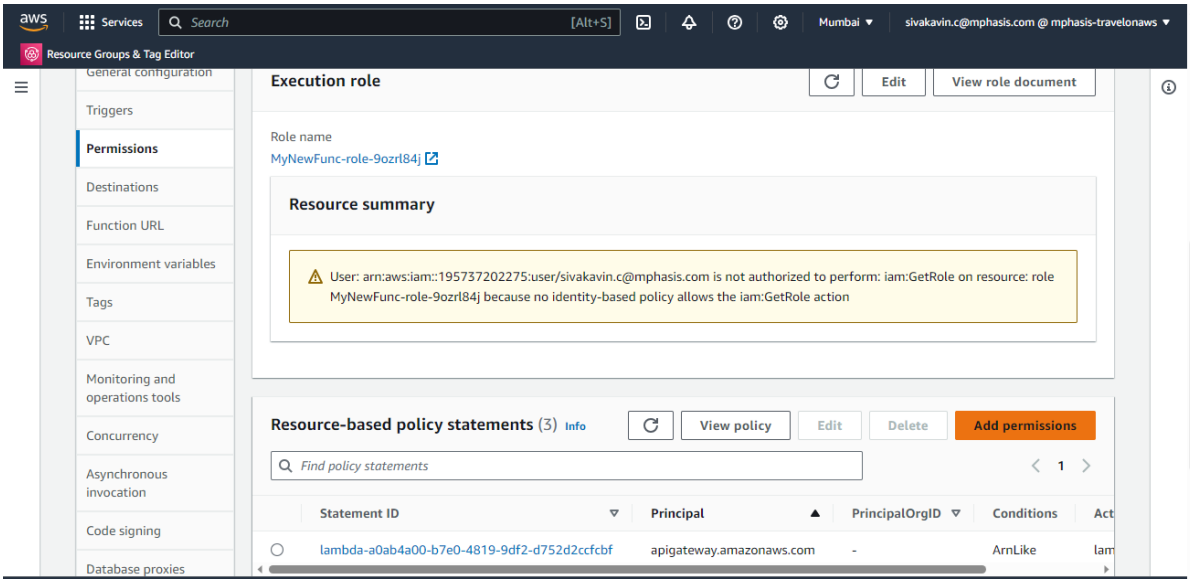
Mention timeout (maximum will be 15 minutes) and create or use existing execution role.



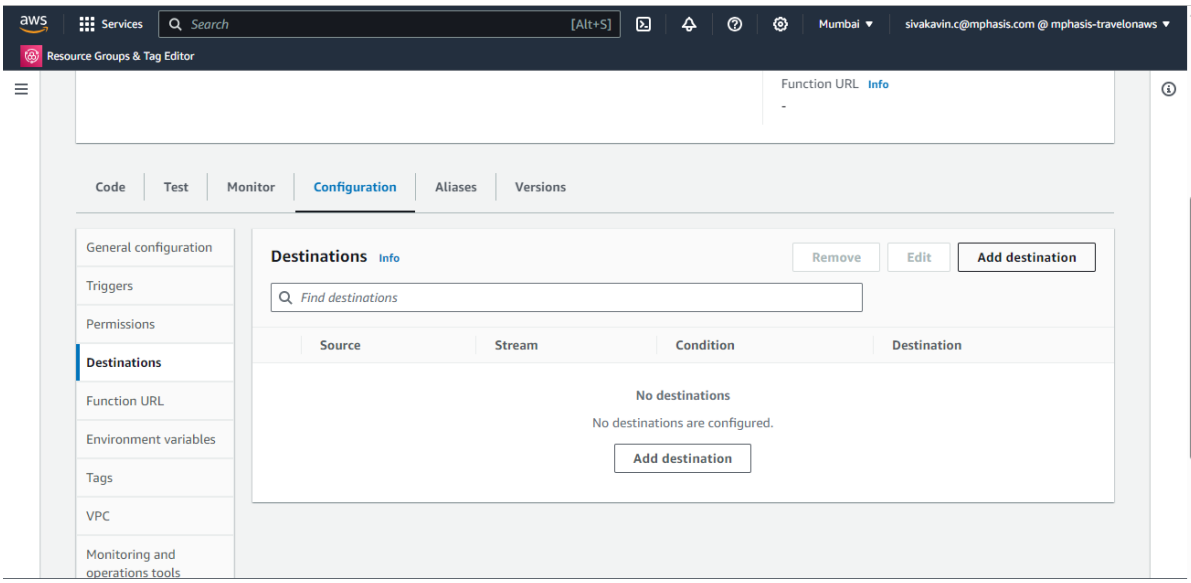
In triggers, you can add new triggers, or you will see the added triggers.



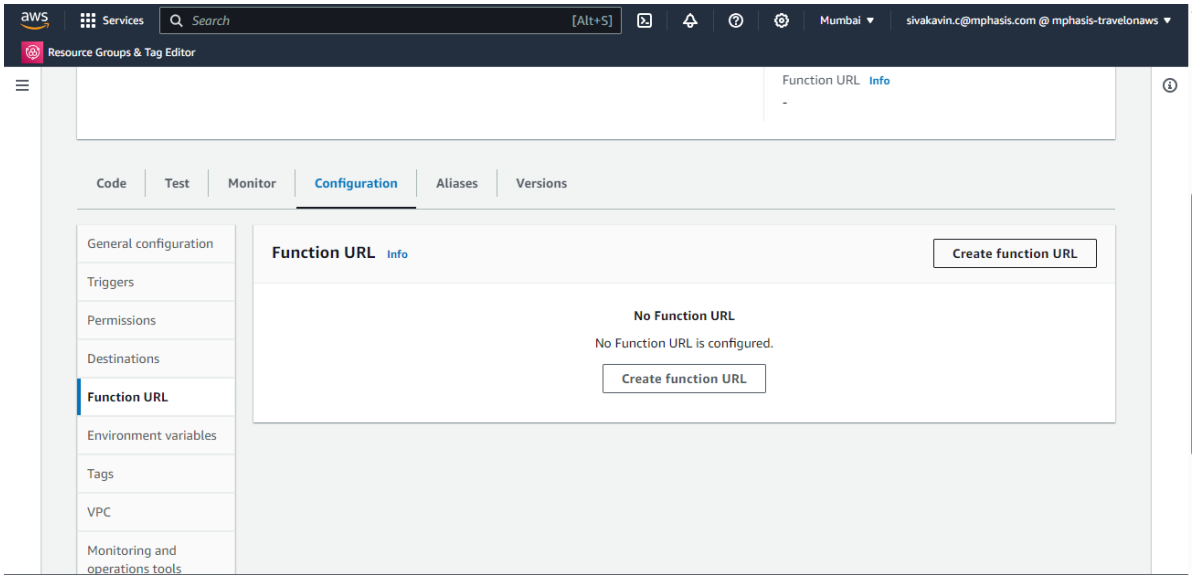
In permissions, if you need to add permissions means you can add it.



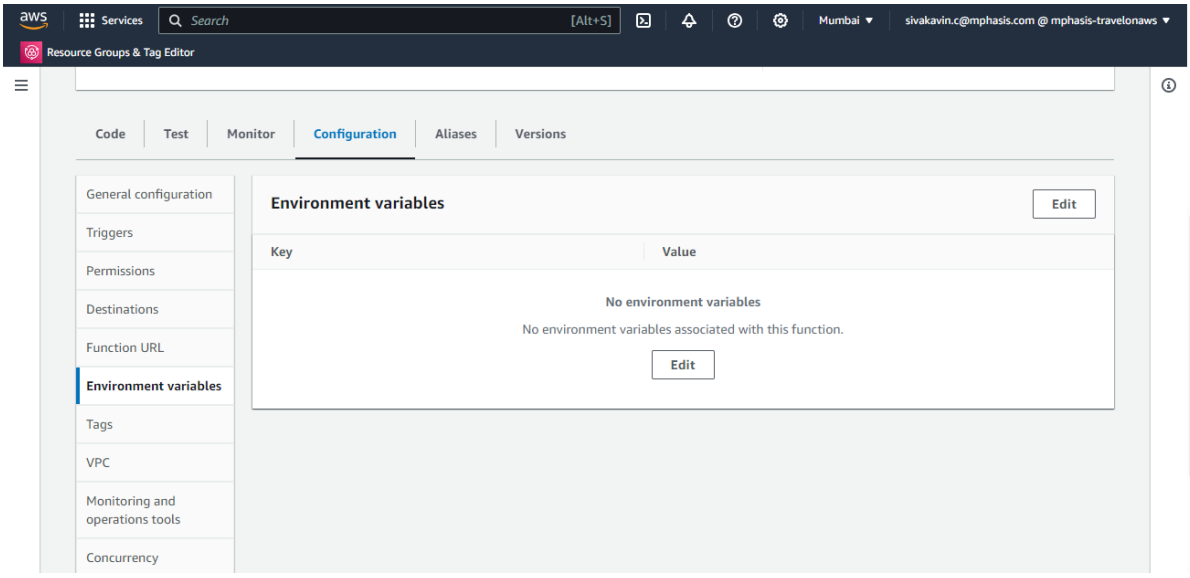
In destination, if you have destination, it will visible here or else you can add here.



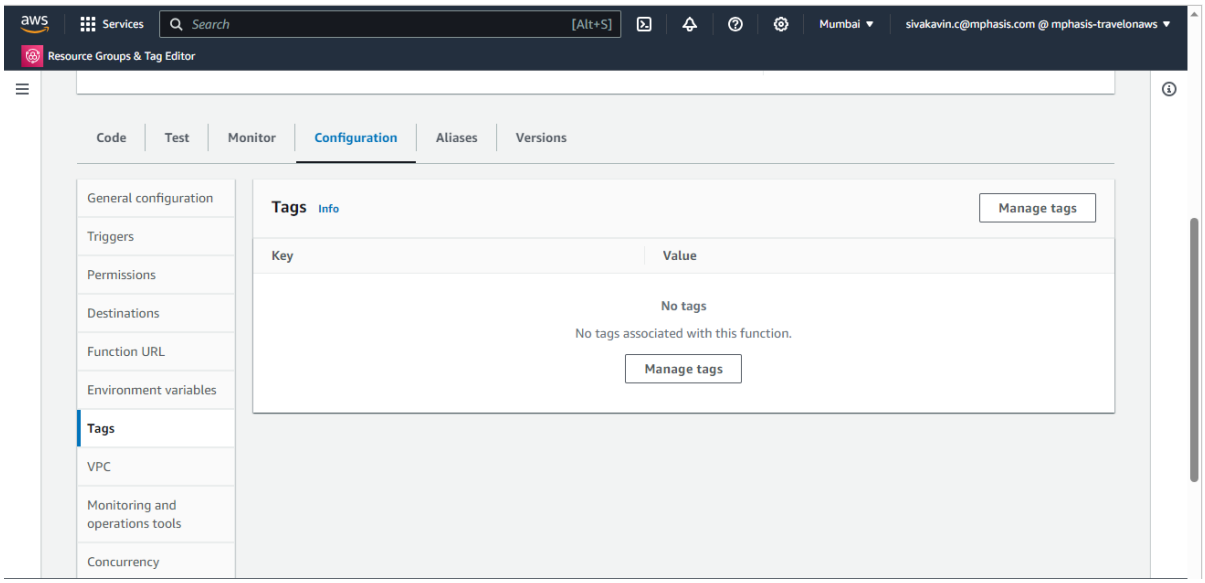
In function URL, if you have URL, it will be visible here or else you can add it here.



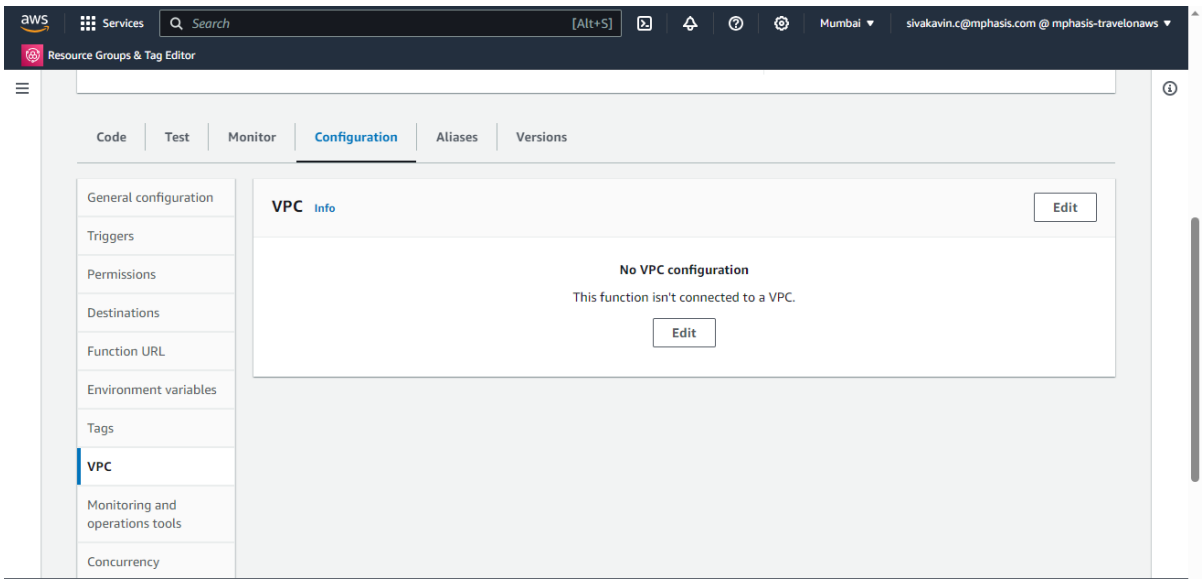
In AWS Lambda, environmental variables, also known as environment variables, are key-value pairs that you can configure and manage for your Lambda function. They allow you to pass configuration information to your function code without making changes to the code itself.



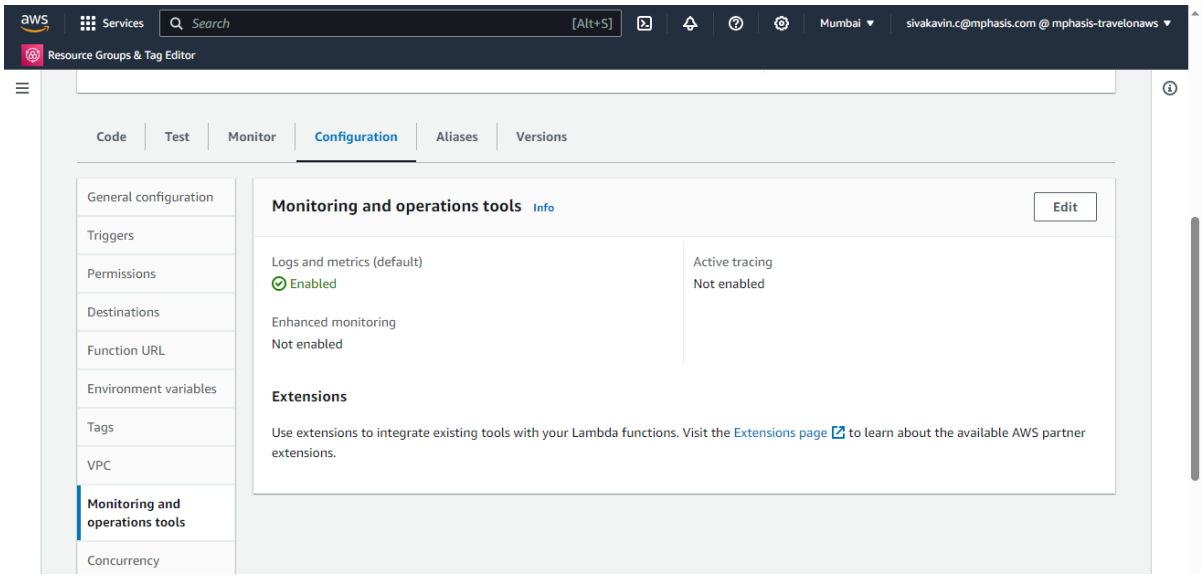
In tags, we can add tags for lambda for identify the lambda functions.



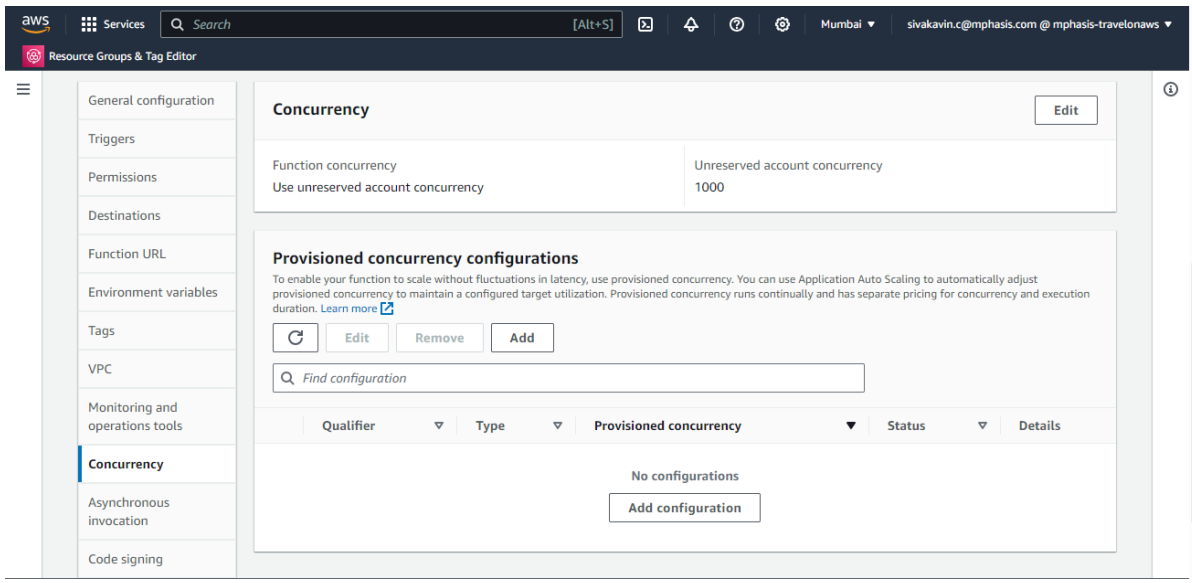
In VPC, you can add VPC if its needed.



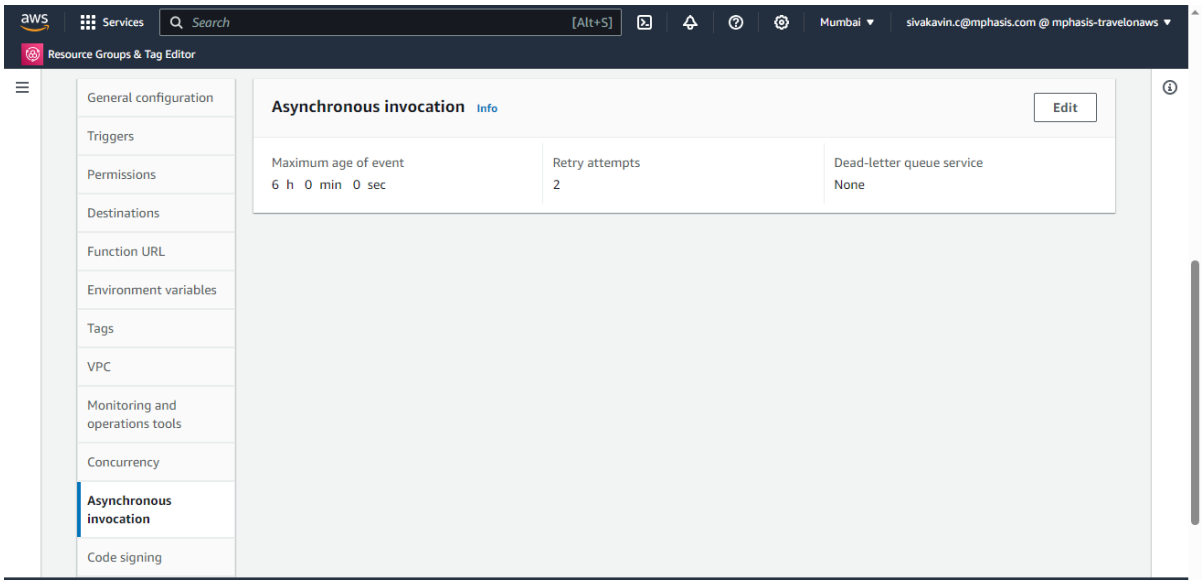
Here, we can see the monitoring details.



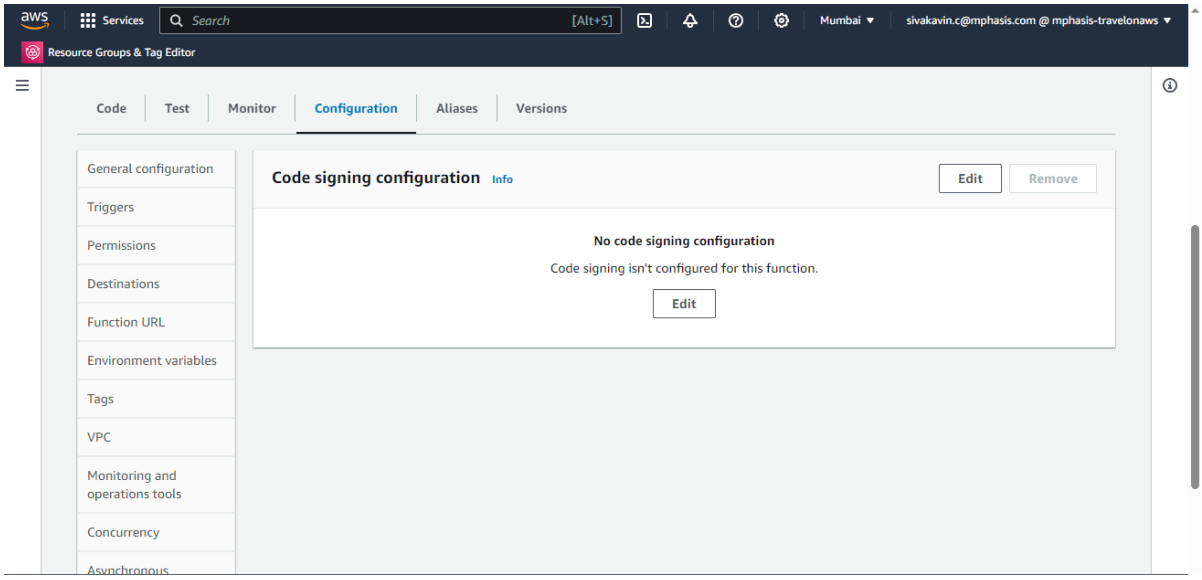
Concurrency in AWS Lambda refers to the number of requests that a function can simultaneously handle. Each concurrent execution of a function is an independent instance.



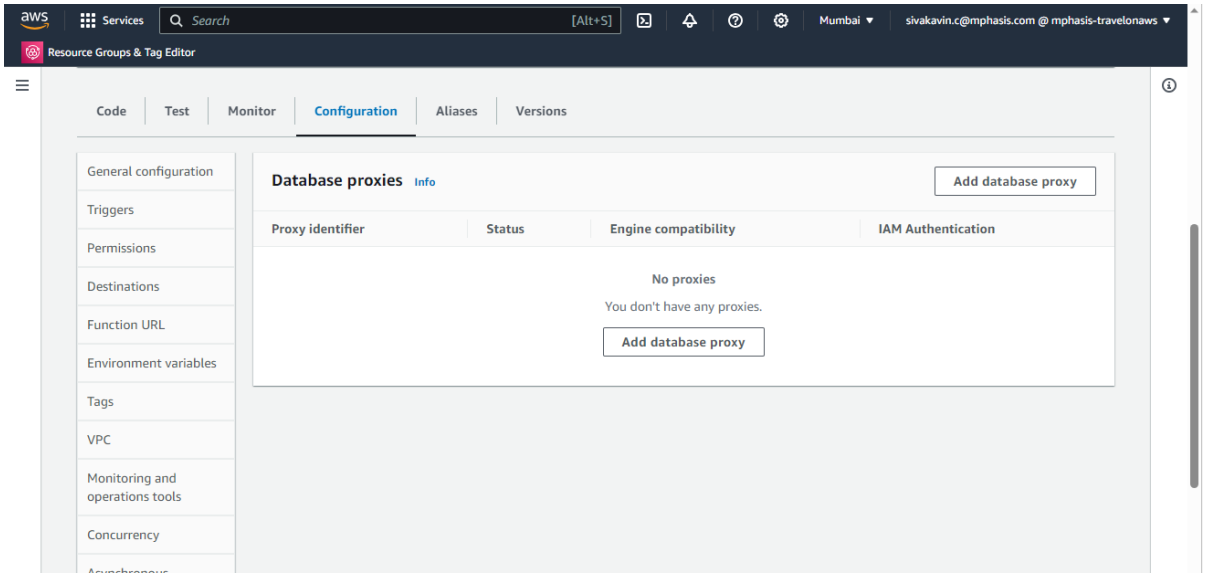
Here, if you add destination, asynchronous invocation will be visible here.



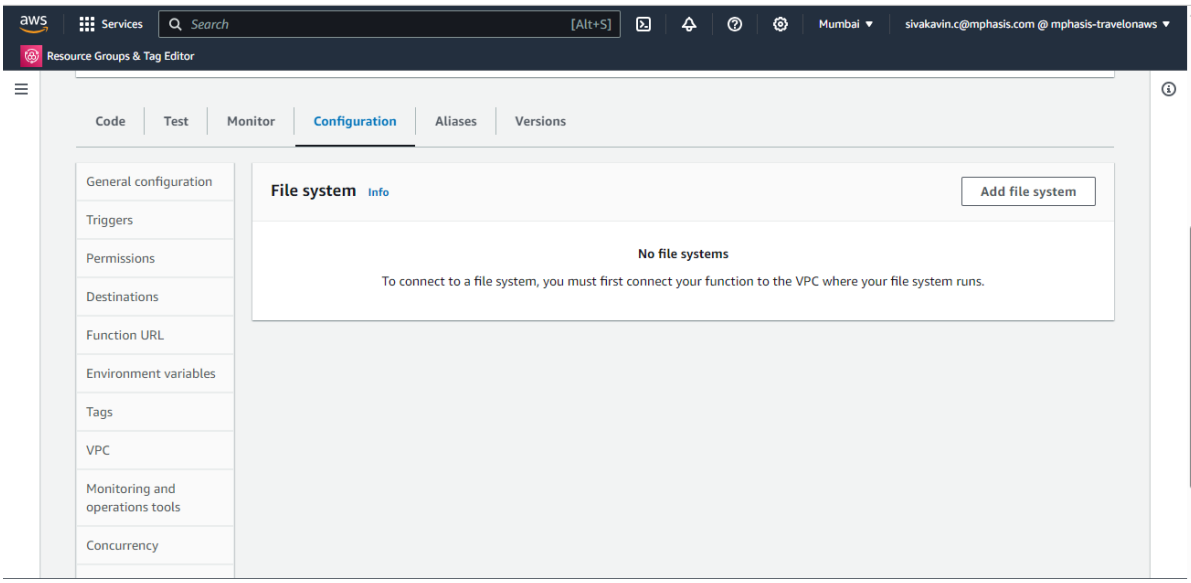
If you added any code signing configurations, it will be visible here.



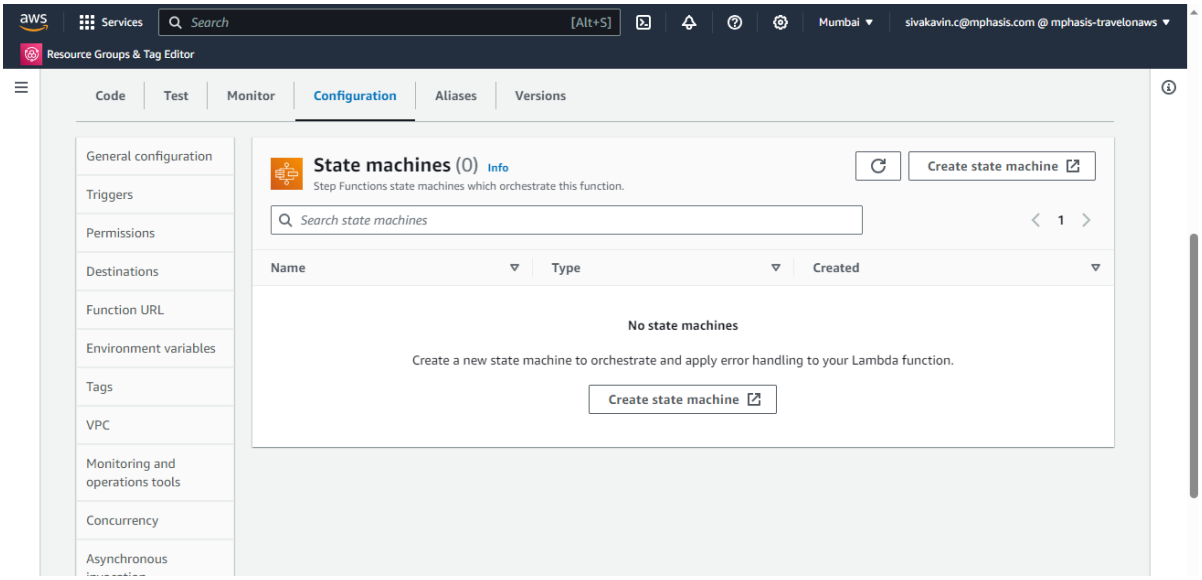
A database proxy is an intermediary layer that sits between an application and a database. It helps manage connections, load balancing, security, and sometimes even caching for database queries.



If you need to add file system, add it here.

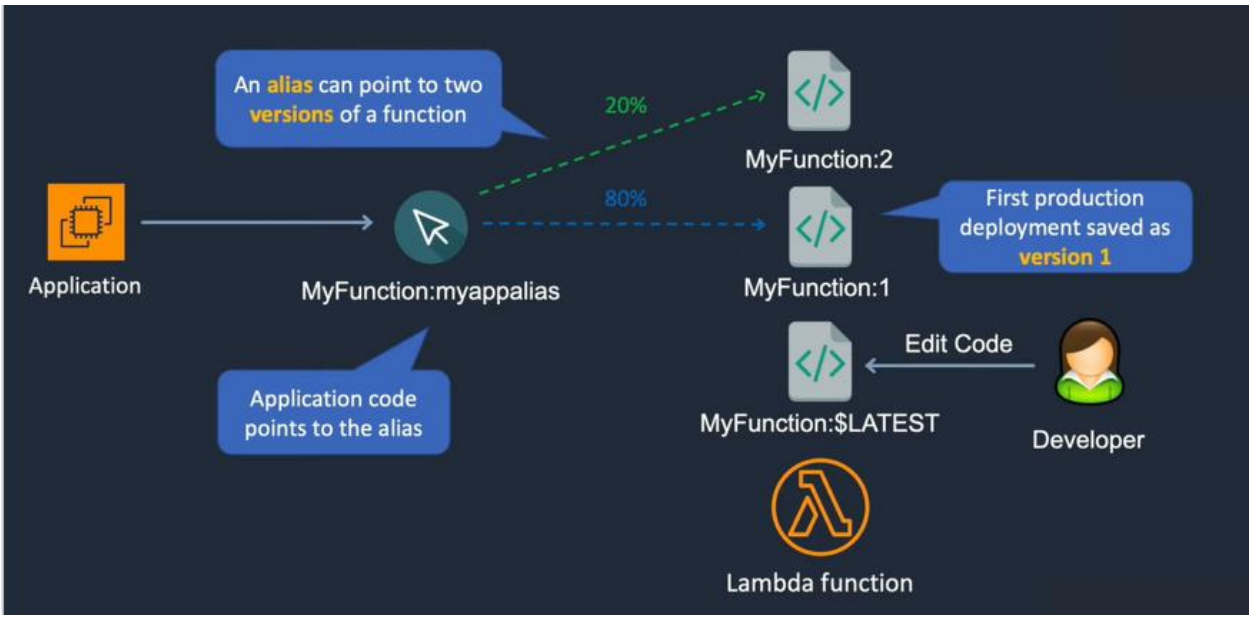


Here, you can create state machine.



VERSIONS

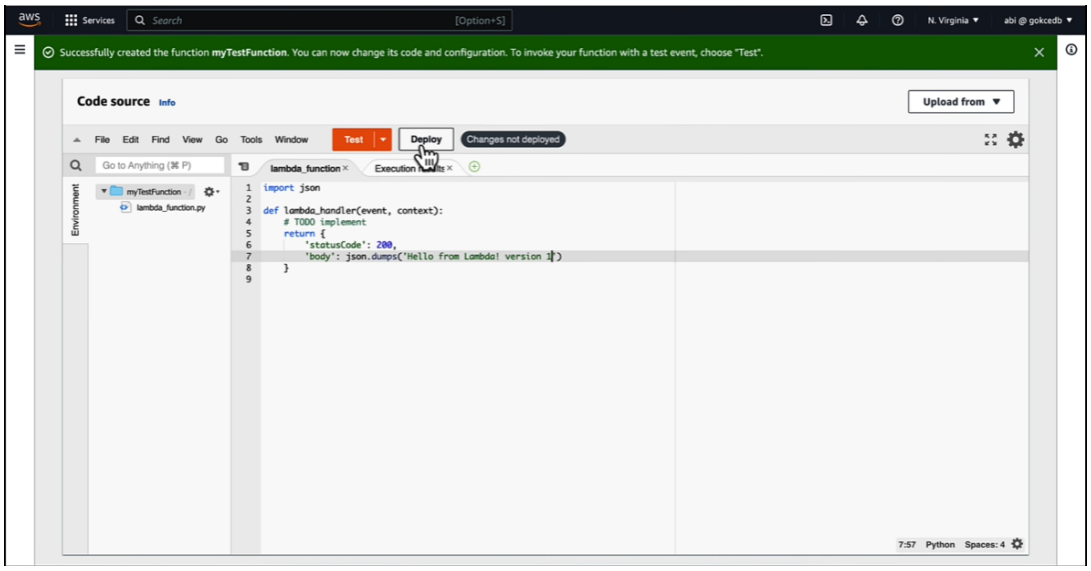
- You can have multiple versions of your function.
- You can use versions to manage the deployment of AWS lambda functions.
- You work on $LATEST which is mutable.
- To publish a lambda function, create a version.
- Versions are immutable.
- Each function has its own ARN.



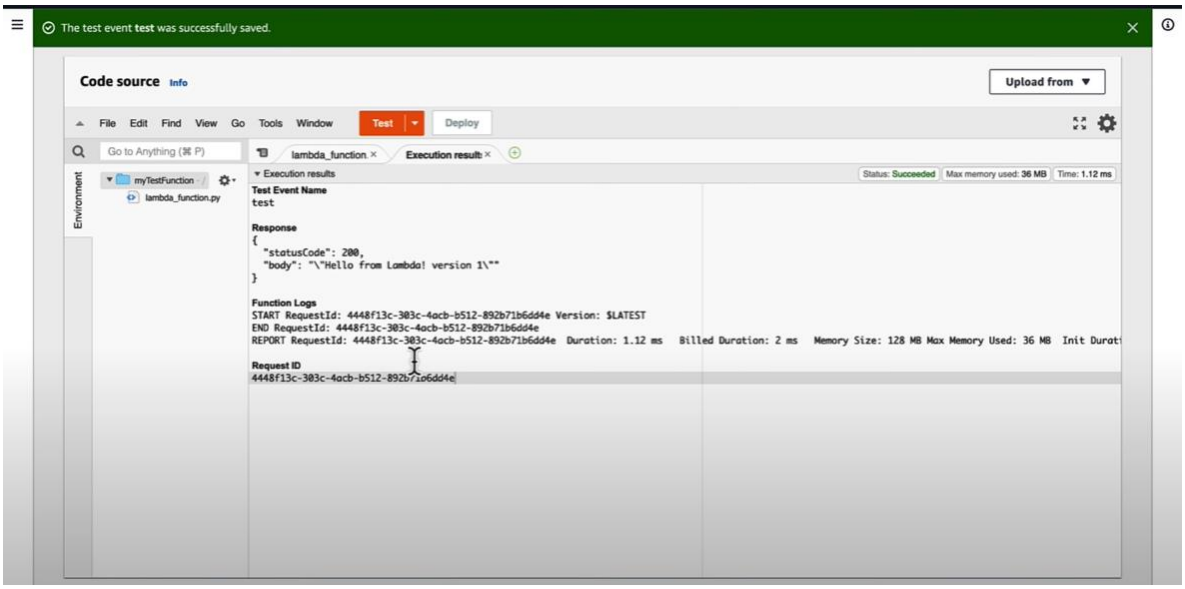
ALIAS

- Using alias, you can invoke a function without having to know which version of the function is being referenced
- Alias is mutable
- Aliases also have static ARNs but can point to any version of the same function
- You must create a version of an alias, you cannot use $LATEST

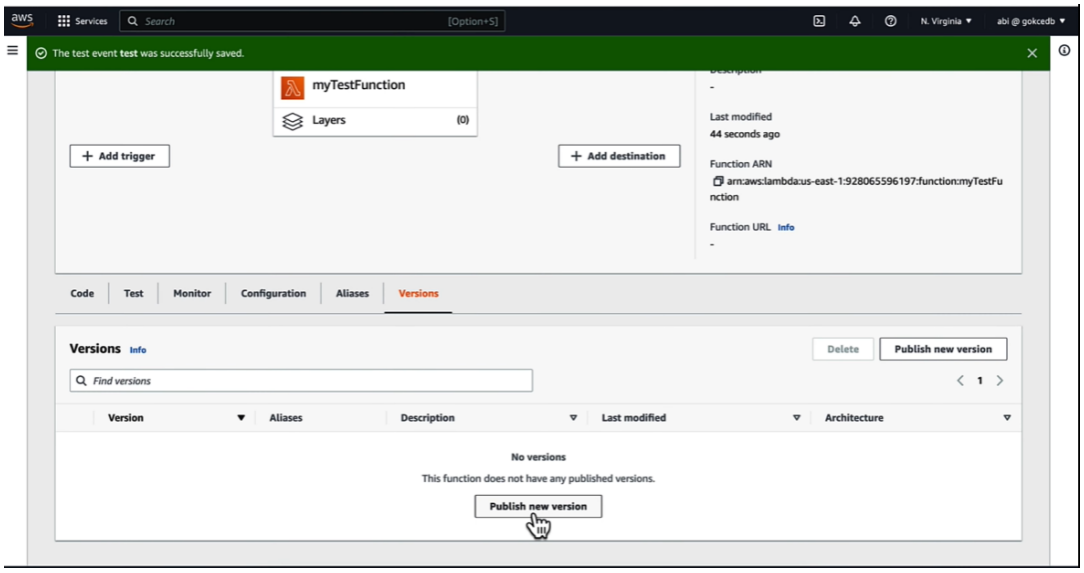
STEP 1: CREATE A FUNCTION (myTestFunction).



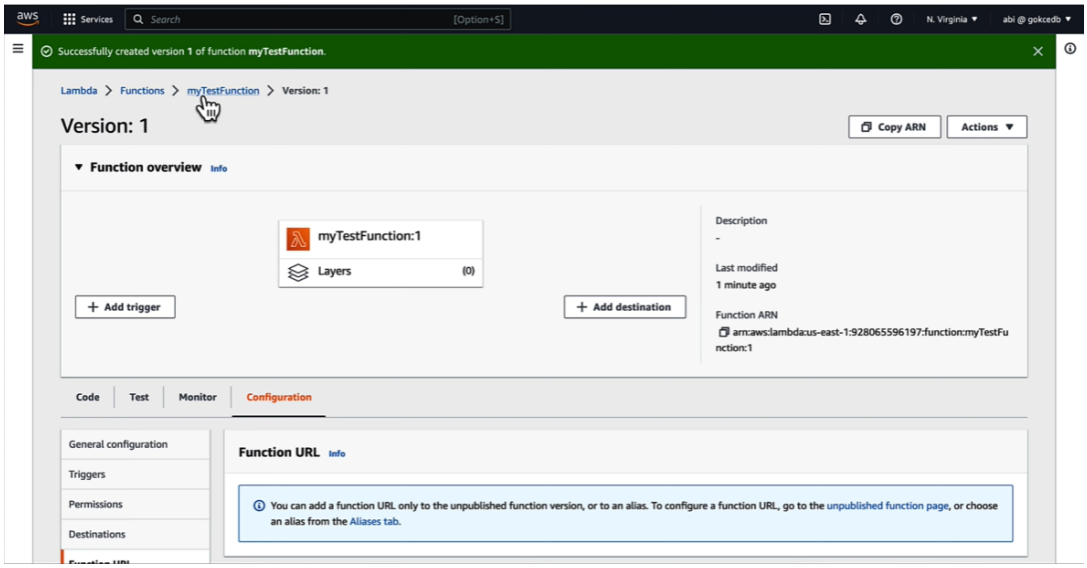
STEP 2: DEPLOY AND TEST THE CODE

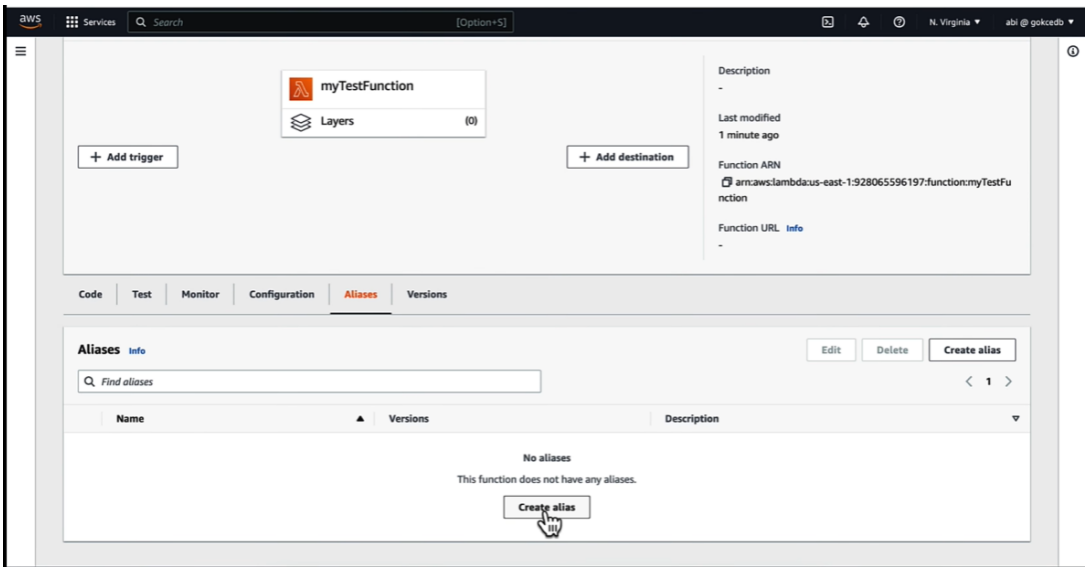


STEP 3: CREATE A NEW VERSION

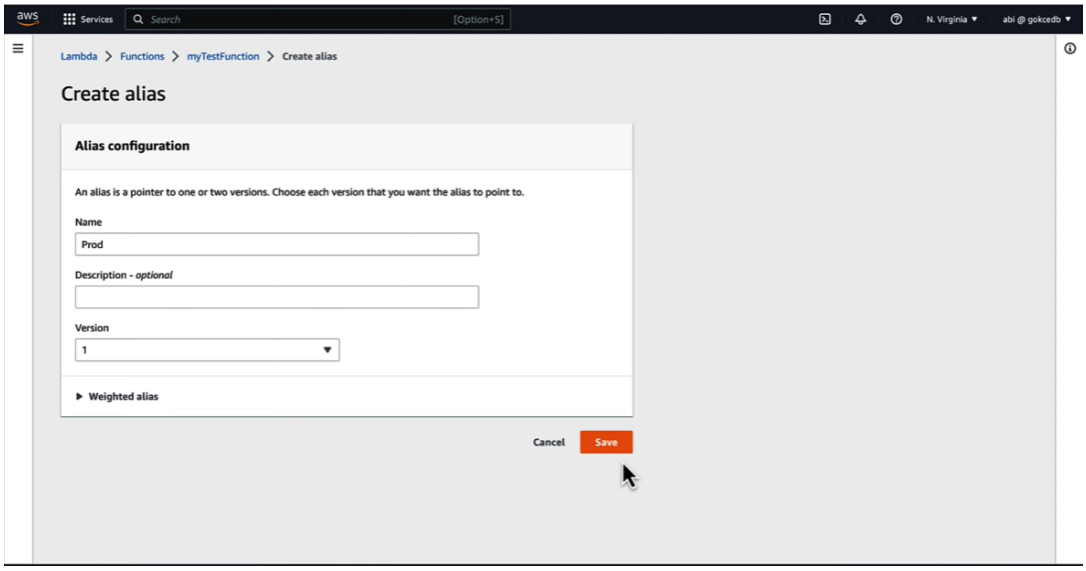


STEP 4: AFTER CREATING VERSION 1, CREATE ALIAS

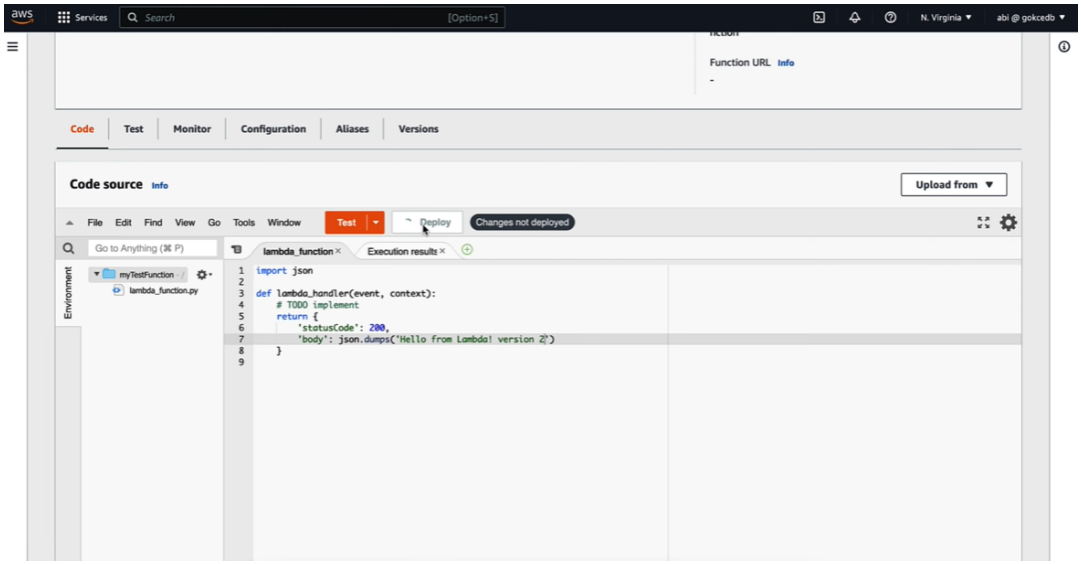




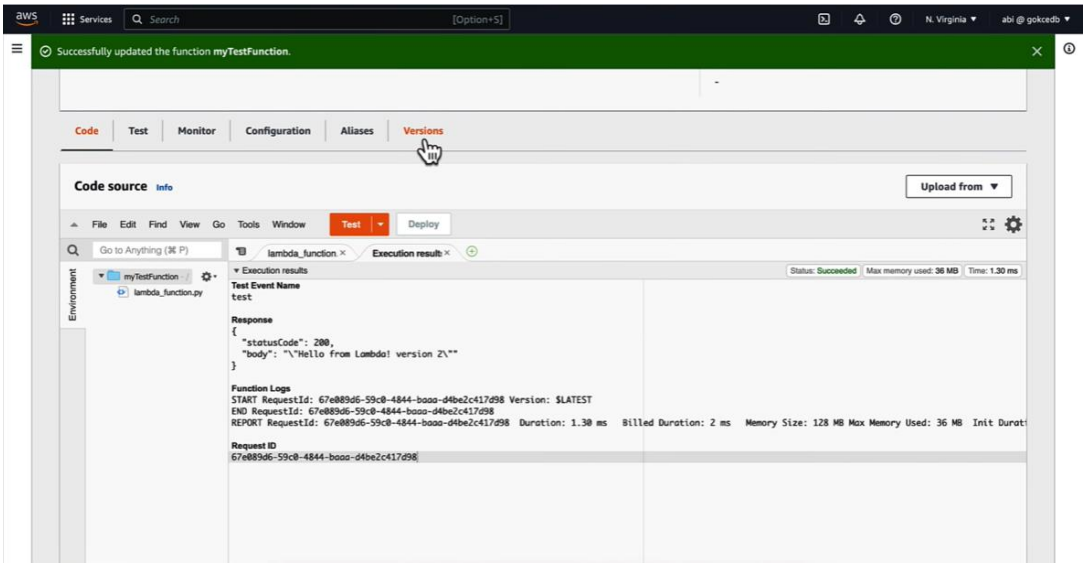
STEP 5: ENTER THE CONFIGURATIONS AND SAVE

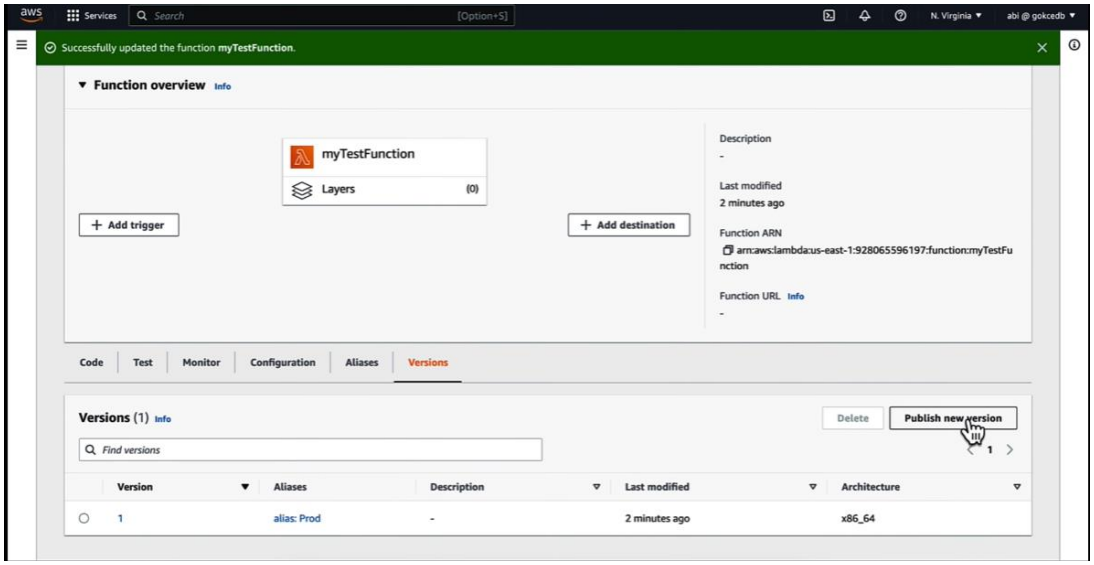


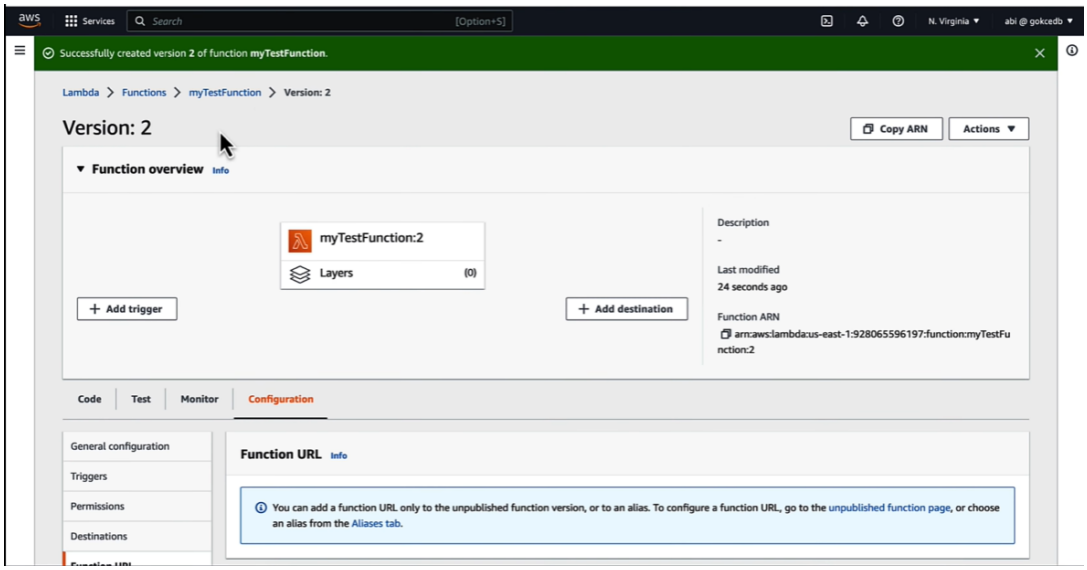
STEP 6: DEPLOY AND TEST THE CODE (VERSION 2)



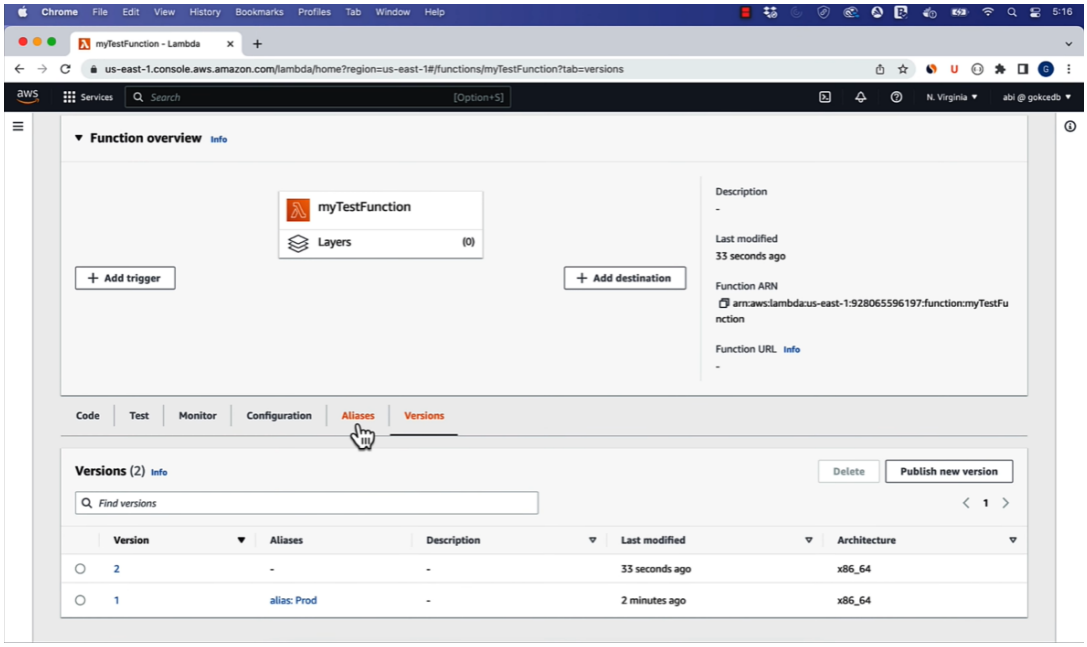
STEP 7: PUBLISH NEW VERSION

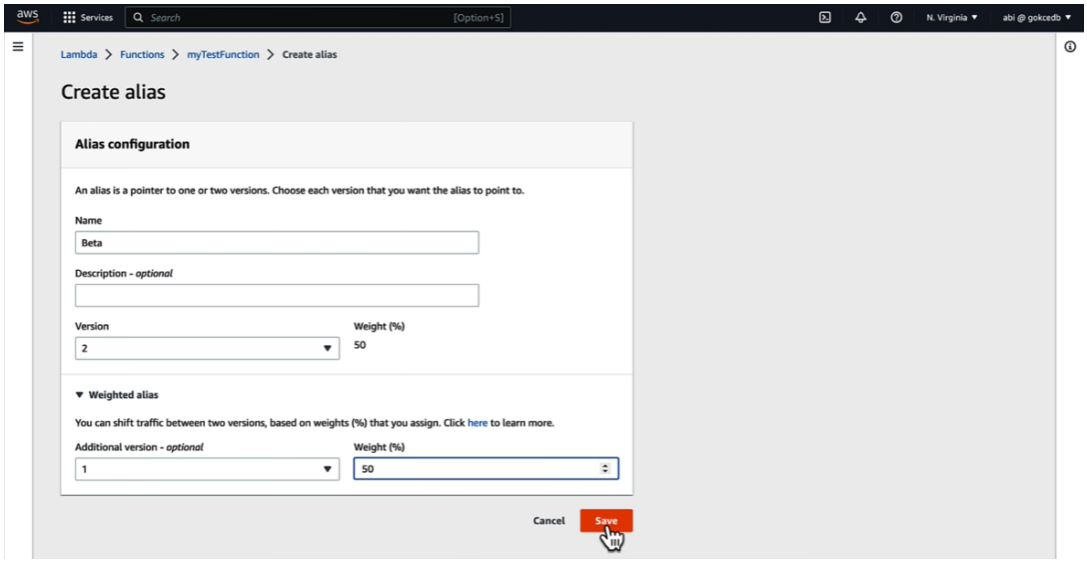


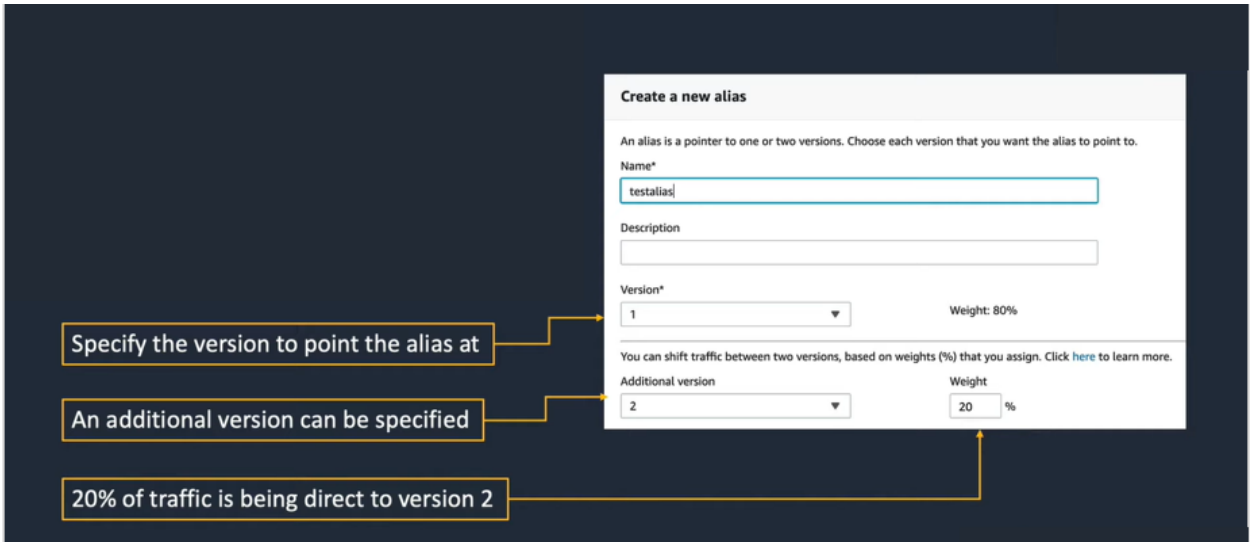

Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Successfully updated the function myTestFunction.
Function overview Info
myTestFunction
Layers (0)
+ Add trigger
+ Add destination
Description
-
Last modified
2 minutes ago
Function ARN
arn:aws:lambda:us-east-1:928065596197:function:myTestFunction
Function URL Info
-
Code
Test
Monitor
Configuration
Aliases
Versions
Versions (1) Info
Find versions
Delete
Publish new version
1
Version
Aliases
Description
Last modified
Architecture
1
alias: Prod
-
2 minutes ago
x86_64


Services
Search
[Option+S]
N. Virginia
abi @ gokcedb
Successfully created version 2 of function myTestFunction.
Lambda > Functions > myTestFunction > Version: 2
Version: 2
Copy ARN
Actions
Function overview Info
myTestFunction:2
Layers (0)
+ Add trigger
+ Add destination
Description
-
Last modified
24 seconds ago
Function ARN
arn:aws:lambda:us-east-1:928065596197:function:myTestFunction:2
Code
Test
Monitor
Configuration
General configuration
Triggers
Permissions
Destinations
Function URL Info
You can add a function URL only to the unpublished function version, or to an alias. To configure a function URL, go to the unpublished function page, or choose an alias from the Aliases tab.

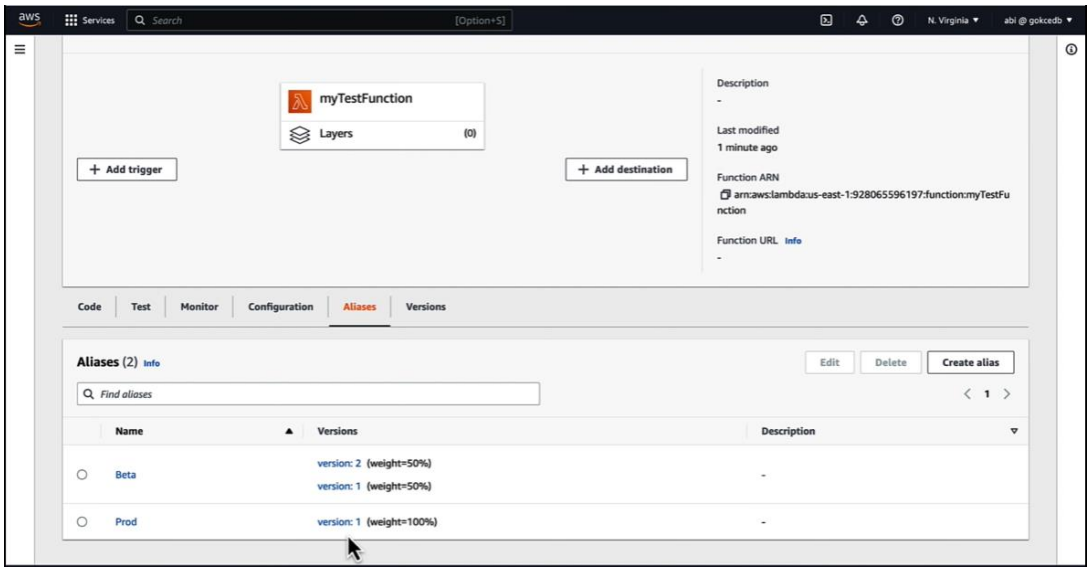
STEP 8: CREATE ALIAS FOR VERSION 2



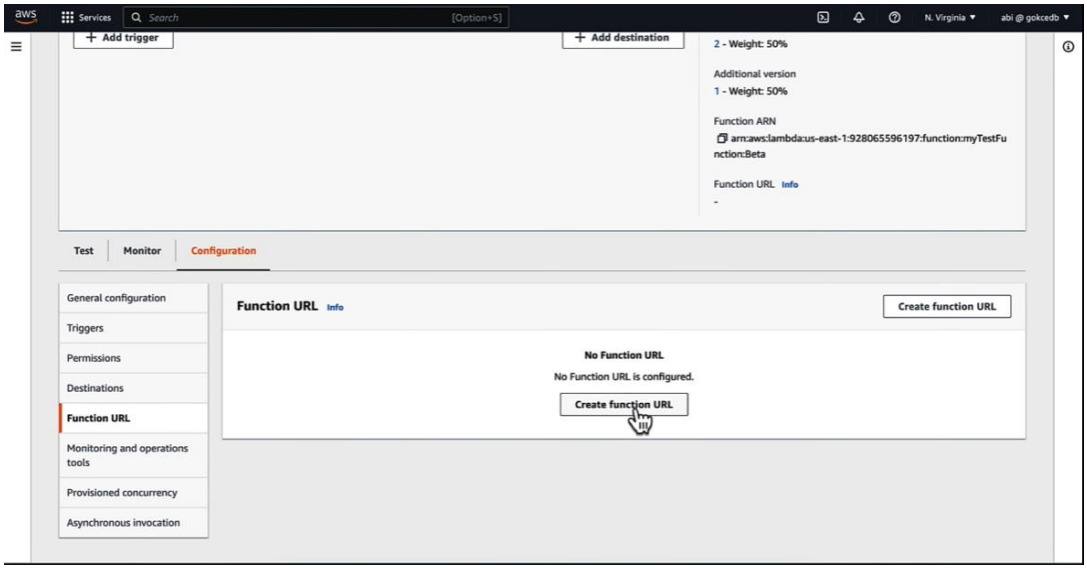


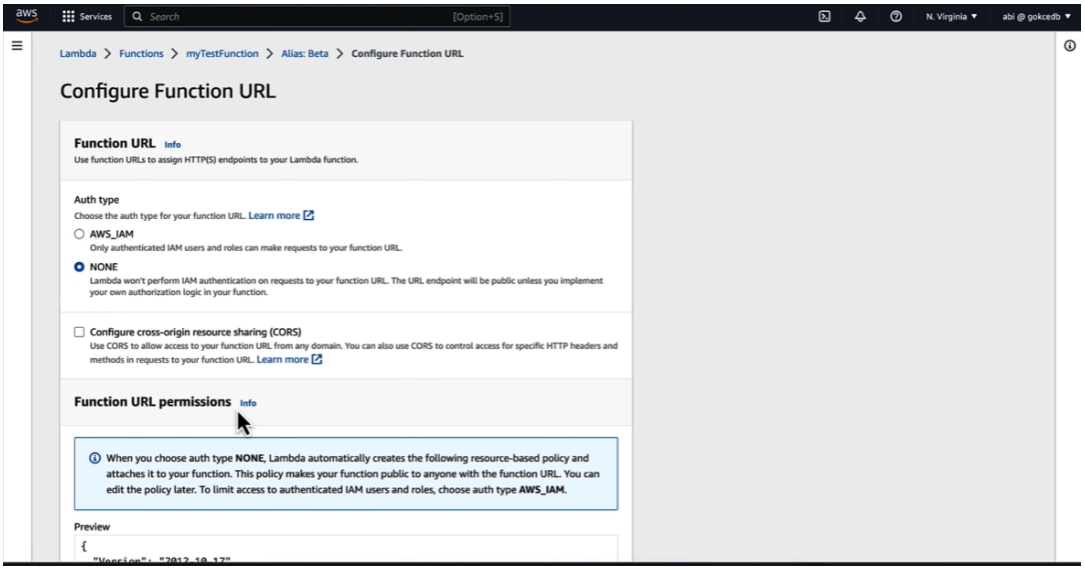


STEP 9: TWO ALIAS CREATED

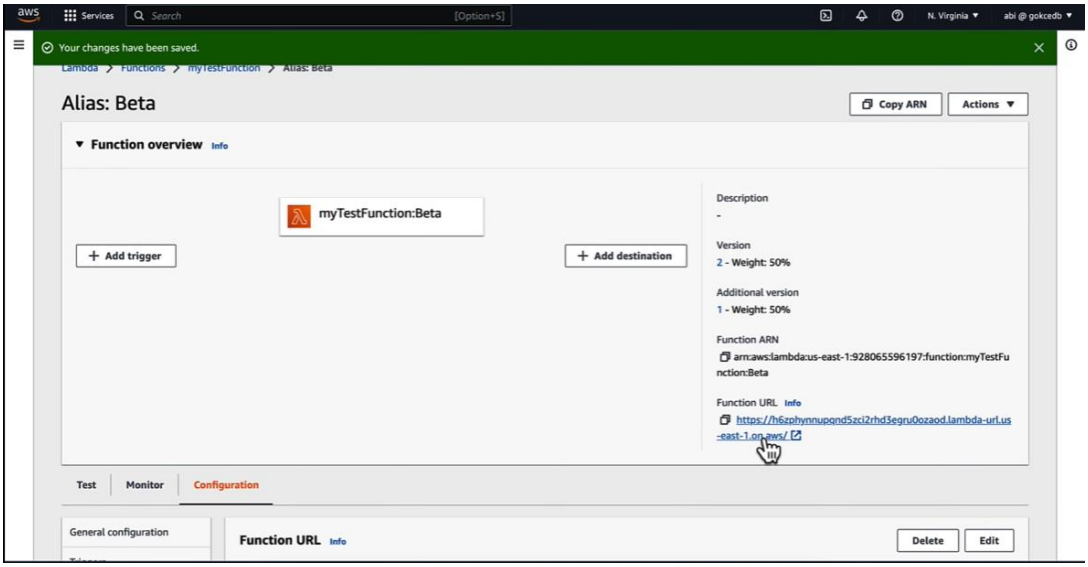


STEP 10: CREATE FUNCTION URL FOR BETA

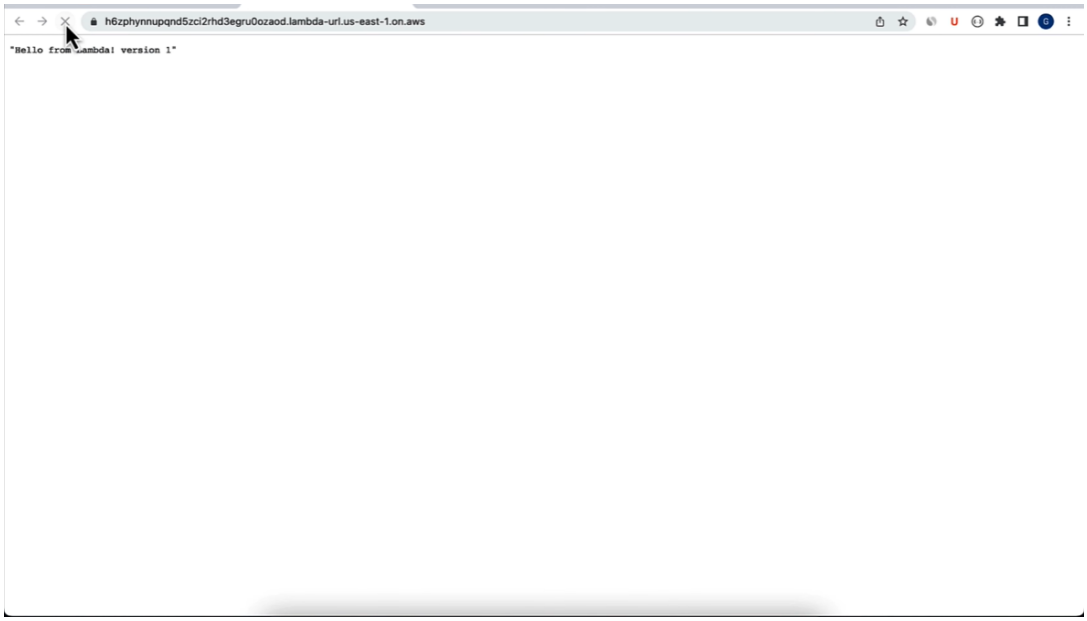


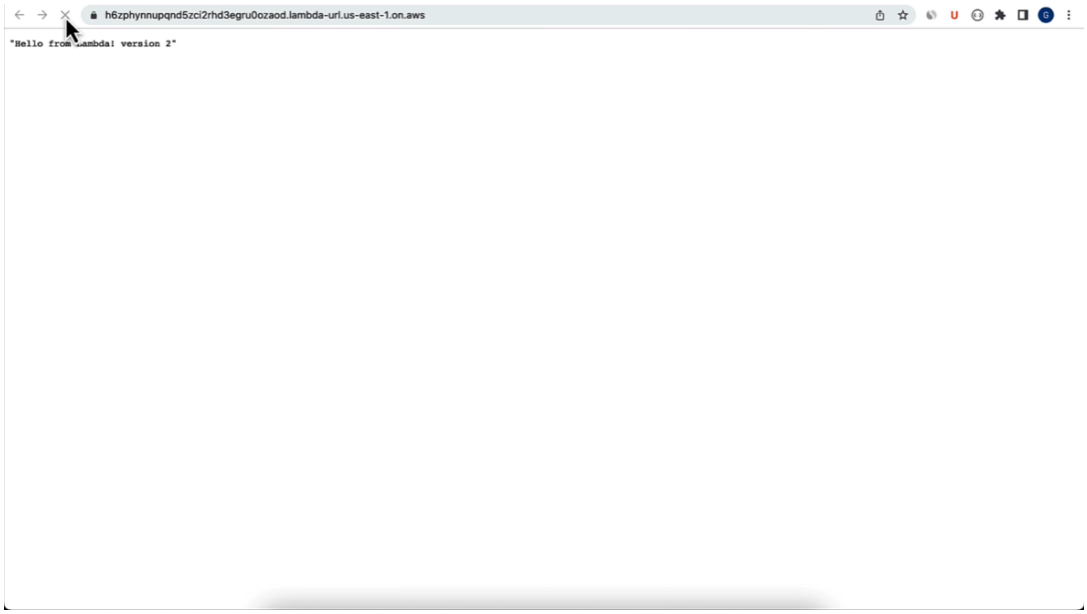


STEP 11: CLICK THE FUNCTION URL AND CHECK THE OUTPUT



STEP 12: REFRESH THE PAGE




h6zphynnupqnd5zci2rhd3egru0ozaod.lambda-url.us-east-1.on.aws
"Hello from Lambda! version 2"