

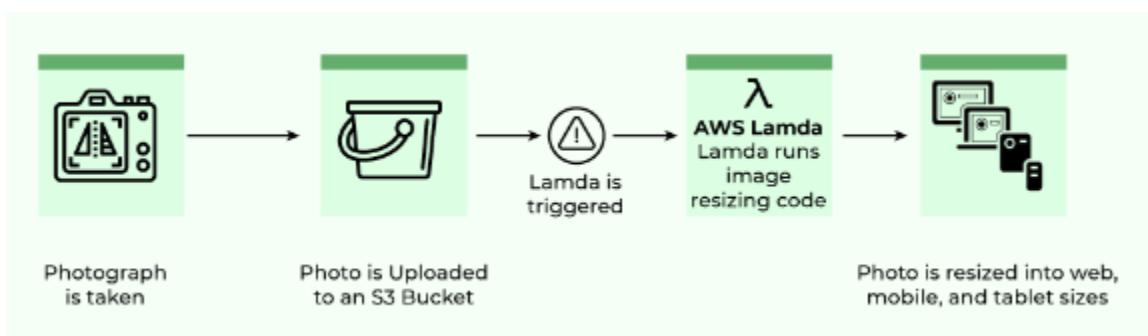
Contents

INTRODUCTION.....	2
HISTORY OF AWS LAMBDA.....	3
COMPUTE SERVICES IN AWS	5
WHAT IS LAMBDA?.....	7
WHY DO WE USE LAMBDA?.....	8
WHEN TO USE AWS LAMBDA	10
WHERE TO USE AWS LAMBDA?.....	10
KEY FEATURES OF AWS LAMBDA	11
USE CASES OF AWS LAMBDA	13
PRICING IN AWS LAMBDA.....	14
ADVANTAGES OF AWS LAMBDA.....	16
DISADVANTAGES OF AWS LAMBDA	17
STEPS TO CREATE AWS LAMBDA	18
INTEGRATING LAMBDA WITH API GATEWAY	23
INTEGRATING LAMBDA WITH DYNAMODB	31
OTHER WAY OF INTEGRATION WITH DYNAMODB	38
REAL TIME DATA PROCESSING WITH AWS KINESIS, LAMBDA AND DYNAMO DB	41
COPY FILES FROM ONE S3 BUCKET TO ANOTHER S3 BUCKET	53
INTEGRATING LAMBA WITH S3.....	53
INTEGRATING LAMBDA WITH SQS.....	60
AUTOMATE SENDING NOTIFICATIONS WITH USING AMAZON SNS AND AWS LAMBDA.....	70
DASHBOARD AND APPLICATIONS.....	83
LAYERS.....	87
CODE SIGNING CONFIGURATIONS.....	97
STEP FUNCTIONS STATE MACHINES.....	116
ADD DESINATION IN AWS LAMBDA	133
ADD DESTINATION IN AMAZON LAMBDA (WITH S3).....	145
CODE - TEST - MONITOR - CONFIGURATION - ALIASES – VERSIONS.....	152

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 Outposts
- 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

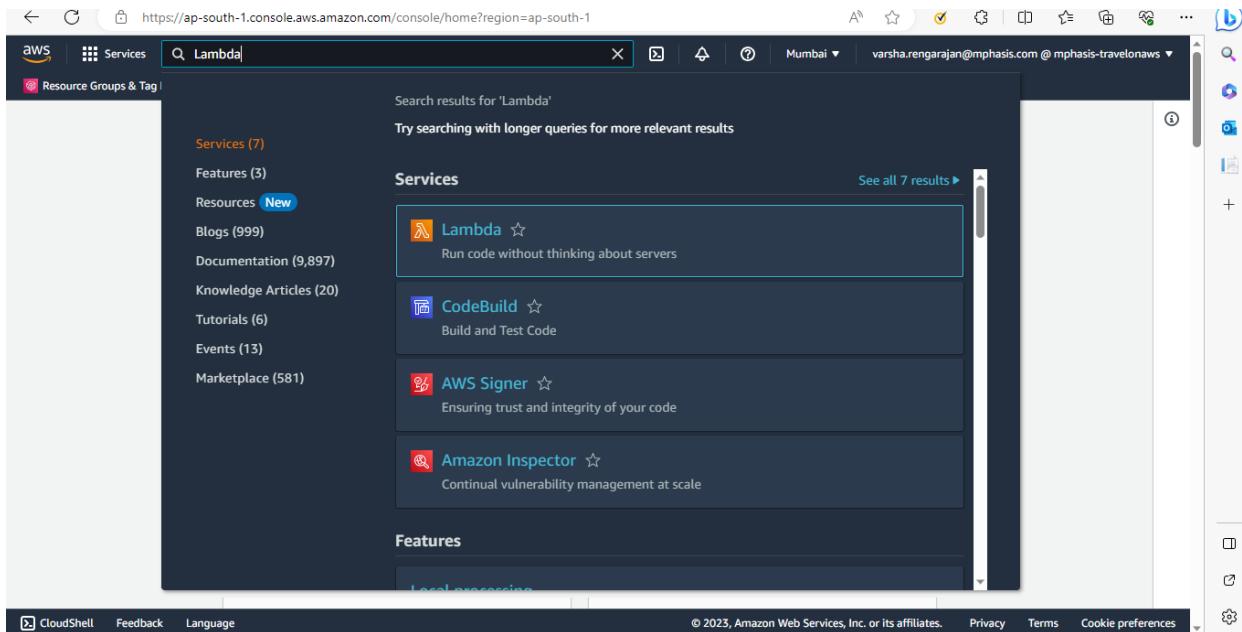
ADVANTAGES OF AWS LAMBDA

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

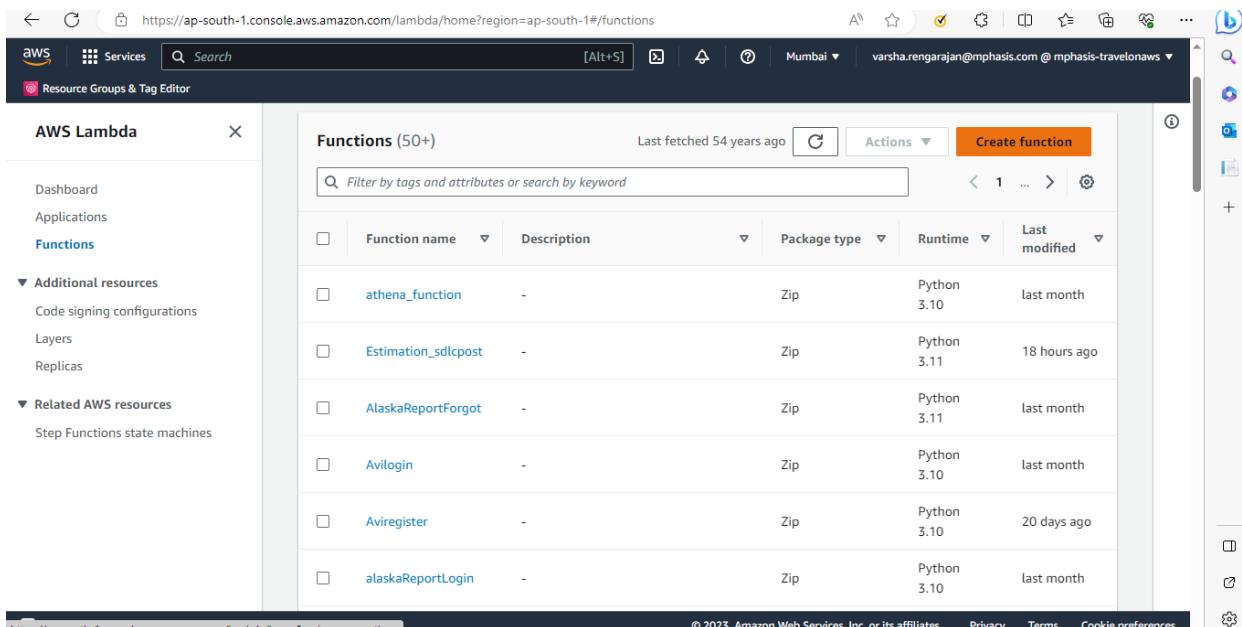
1. **Latency while starting:** While AWS lambda is going to be activated after a long gap it will take some time to initialize the service which is required to deploy the application at that time end users will face latency issues.
2. **Limited control of infrastructure:** Behalf of your lambda function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



The screenshot shows the AWS Lambda search results page. The search bar at the top contains the query 'Lambda'. The results are categorized under 'Services' and 'Features'. Under 'Services', there are seven items: Lambda, CodeBuild, AWS Signer, and Amazon Inspector. Under 'Features', there are three items: Local processing, AWS Step Functions, and AWS Lambda VPC. The Lambda item is highlighted with a blue border.

Log in to AWS account and choose AWS Lambda



The screenshot shows the AWS Lambda home page. The left sidebar has a 'Functions' section selected. The main area displays a table of existing Lambda functions:

Function name	Description	Package type	Runtime	Last modified
athena_function	-	Zip	Python 3.10	last month
Estimation_sdlcpost	-	Zip	Python 3.11	18 hours ago
AlaskaReportForgot	-	Zip	Python 3.11	last month
Avilogin	-	Zip	Python 3.10	last month
Aviregister	-	Zip	Python 3.10	20 days ago
alaskaReportLogin	-	Zip	Python 3.10	last month

Click “Create function” button.

Lambda > Functions > Create function

Create function Info

AWS Serverless Application Repository applications have moved to [Create application](#).

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Container image
Select a container image to deploy for your function.

Choose the way of creating a code template.

Basic information

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

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

Architecture Info
Choose the instruction set architecture you want for your function code.
 x86_64
 arm64

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

Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

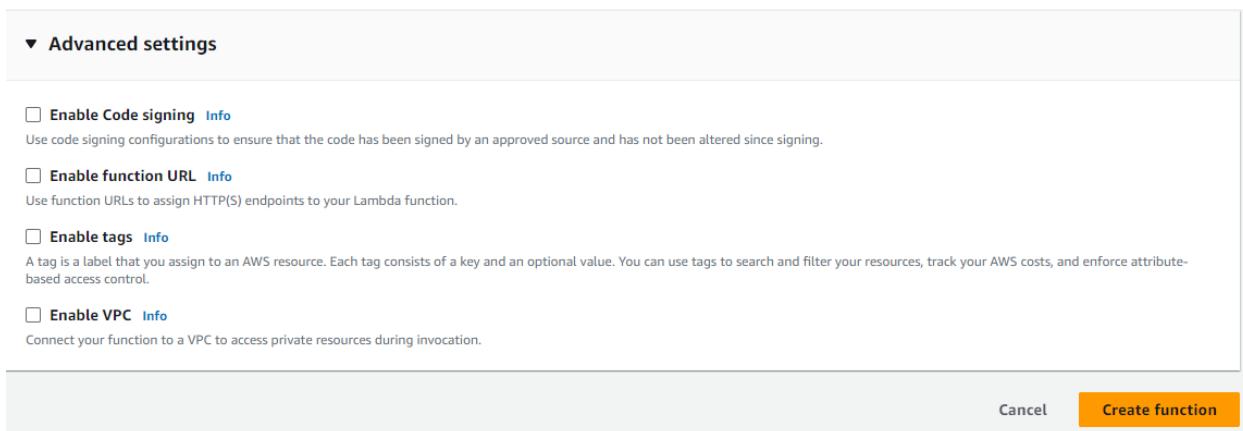
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#) .

Create a new role with basic Lambda permissions
 Use an existing role
 Create a new role from AWS policy templates

i Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named <myFunctionName>-role-sni8g87s, with permission to upload logs to Amazon CloudWatch Logs.

Choose the permissions based on your needs.



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

The screenshot shows the AWS Lambda function details page for 'sampleFunction'. A green success message at the top states: 'Successfully created the function sampleFunction. You can now change its code and configuration. To invoke your function with a test event, choose "Test".' The main area displays the function name 'sampleFunction', a placeholder icon for triggers, and a placeholder '(0)' for layers. Buttons for 'Throttle', 'Copy ARN', and 'Actions' are visible. On the right, detailed information is shown: Description (empty), Last modified (15 seconds ago), Function ARN (arn:aws:lambda:ap-south-1:195737202275:function:sampleFunction), and Function URL (empty). A sidebar on the left shows the navigation path: Lambda > Functions > sampleFunction.

Function created successfully.

The screenshot shows the AWS Lambda code editor interface. At the top, there are tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Code tab is selected. Below the tabs, there's a toolbar with File, Edit, Find, View, Go, Tools, Window, Test, Deploy, and a gear icon. The main area has sections for Environment and Go to Anything (Ctrl-P). A file tree shows a sampleFunction folder containing lambda_function.py. The code editor window displays the following Python code:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

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

The screenshot shows the AWS Lambda test event configuration screen. At the top, there are tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Test tab is selected. Below the tabs, there's a toolbar with Save and Test buttons. The main area has sections for Test event and Info. It says "To invoke your function without saving an event, configure the JSON event, then choose Test." Under Test event action, there are two options: "Create new event" (selected) and "Edit saved event". The "Create new event" section includes fields for Event name (set to "testEvent") and Event sharing settings (Private or Shareable). The "Private" setting is selected, with a note that it's available in the Lambda console and to the event creator. The "Shareable" setting is also present with its own note.

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

The screenshot shows the AWS Lambda console interface. At the top, there's a navigation bar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', a 'Test' dropdown menu, and a 'Deploy' button. To the right of the dropdown is an 'Upload from' button with a downward arrow. Below the navigation bar, there's a search bar labeled 'Go to Anything (Ctrl-P)' and a sidebar titled 'Environment' which lists 'sampleFunction' and 'lambda_function.py'. The main area is titled 'Execution results' and contains tabs for 'Execution results', 'Environment Var', and 'Execution result'. The 'Execution result' tab is active, showing the following details:

- Test Event Name:** (unsaved) test event
- Response:**

```
{ "statusCode": 200, "body": "\"Hello from Lambda!\""}  
}
```
- Function Logs:**

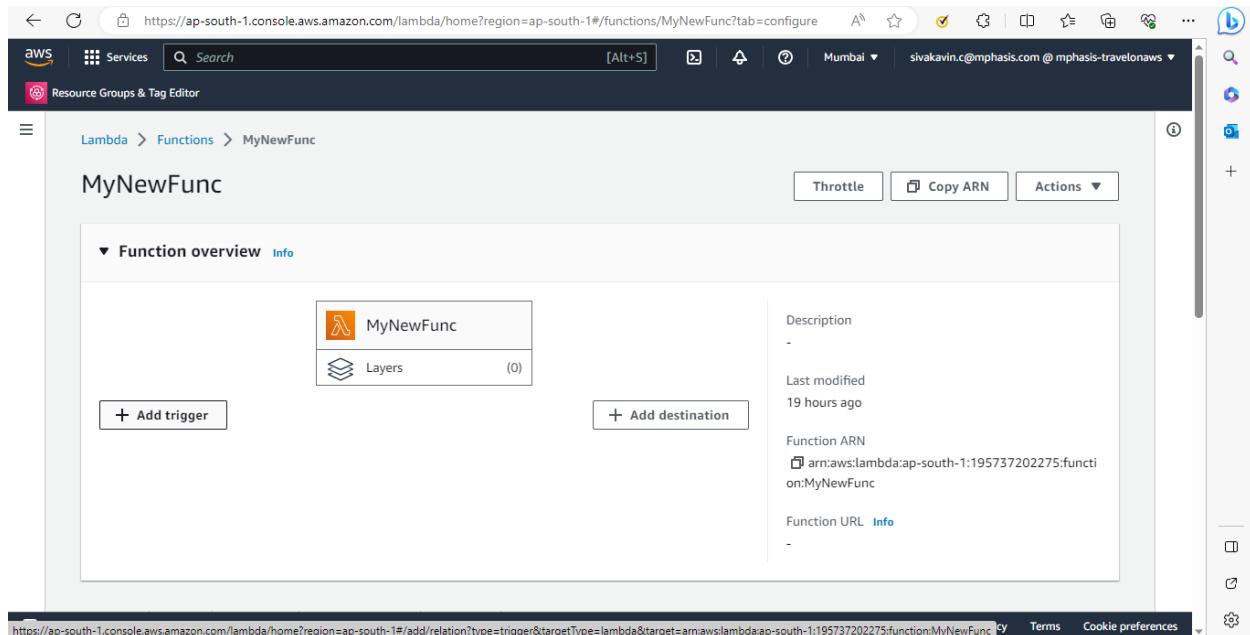
```
START RequestId: 19d40ae3-1d19-4a54-ac57-8b4d00c41416 Version: $LATEST  
END RequestId: 19d40ae3-1d19-4a54-ac57-8b4d00c41416  
REPORT RequestId: 19d40ae3-1d19-4a54-ac57-8b4d00c41416 Duration: 1.51 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 40 MB  
Request ID: 19d40ae3-1d19-4a54-ac57-8b4d00c41416
```

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

INTEGRATION

INTEGRATING LAMBDA WITH API GATEWAY

STEP 1: Click “Add trigger”



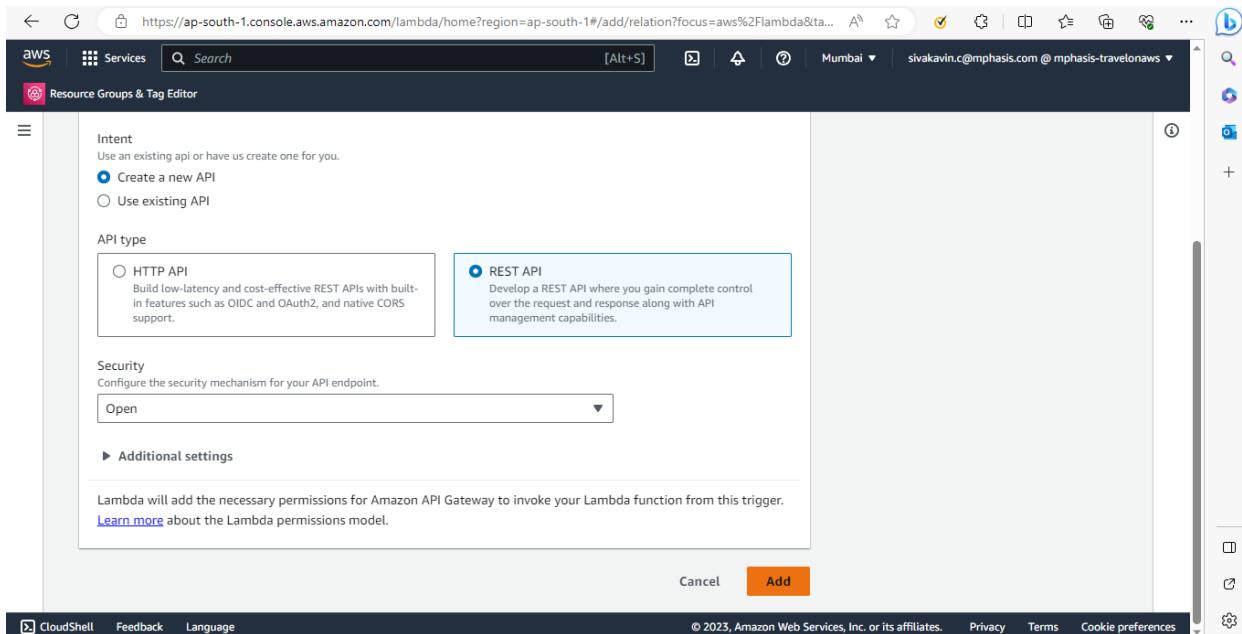
STEP 2: Select the source as “API Gateway.”

The screenshot shows the AWS Lambda console with the URL <https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/add/relation?focus=aws%2Flambda&ta...>. The page title is "Lambda > Add trigger". The main section is titled "Trigger configuration" with a "Info" link. A search bar at the top says "Select a source". Below the search bar, a list of triggers is shown under "APIs/interactive/web". The "API Gateway" option is highlighted with a blue border, indicating it is selected. Other options include Alexa, Application Load Balancer, CodeCommit, and Cognito Sync Trigger. At the bottom of the screen, there are links for CloudShell, Feedback, Language, and a footer with copyright information.

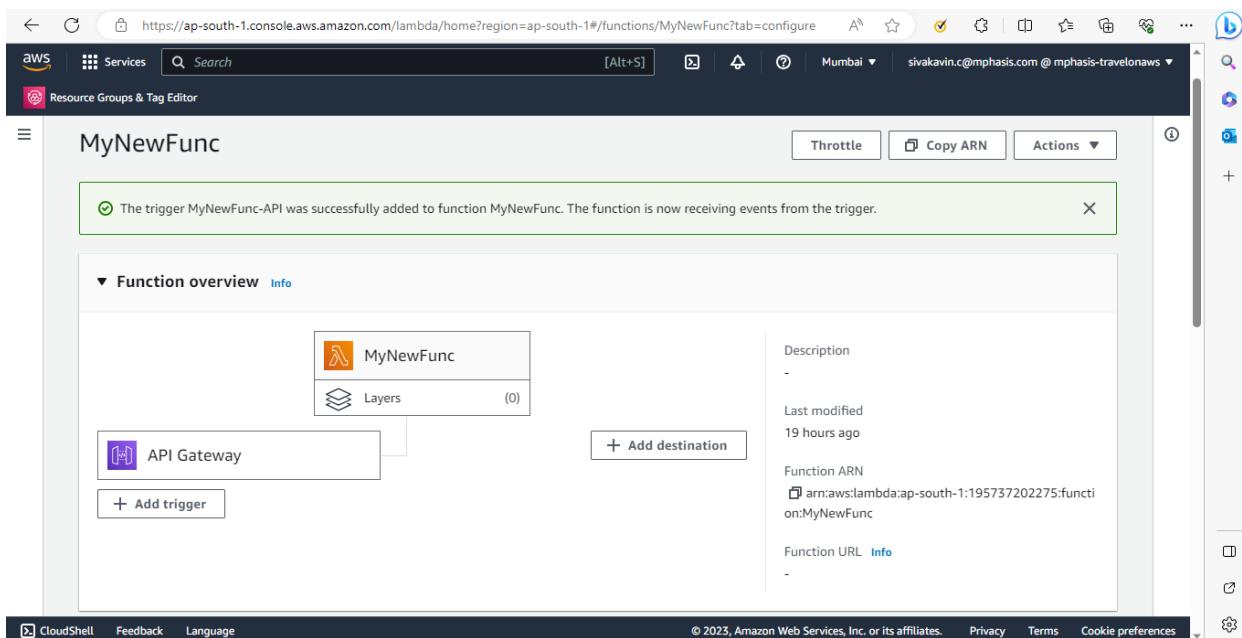
STEP 3: Choose “Create a new API” and choose API type as “REST API” and security type as “Open.”

The screenshot shows the "Trigger configuration" page for the selected "API Gateway" source. The "Intent" section has "Create a new API" selected. The "API type" section has "REST API" selected and highlighted with a blue border. Other options like "HTTP API" and "IAM" are also listed. The page includes instructions to "Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)". At the bottom, there are links for CloudShell, Feedback, Language, and a footer with copyright information.

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



STEP 5: Trigger added successfully.



STEP 6: Trigger details are shown below

The screenshot shows the AWS Lambda function configuration page for 'MyNewFunc'. The left sidebar has 'Triggers' selected. The main area displays a single trigger named 'API Gateway: MyNewFunc-API'. The 'Details' section shows the following information:

- API type: REST
- Authorization: NONE
- Method: ANY
- Resource path: /MyNewFunc
- Service principal: apigateway.amazonaws.com
- Stage: default
- Statement ID: lambda-a0ab4a00-b7e0-4819-9df2-d752d2ccfcfb

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

The screenshot shows the AWS API Gateway service. The left sidebar has 'APIs' selected. The main area displays a table titled 'APIs (14)' with one entry:

Name	Description	ID	Protocol	Endpoint type	Created
MyNewFunc-API	Created by AWS Lambda	uprx1rh1jh	REST	Regional	2023-08-31

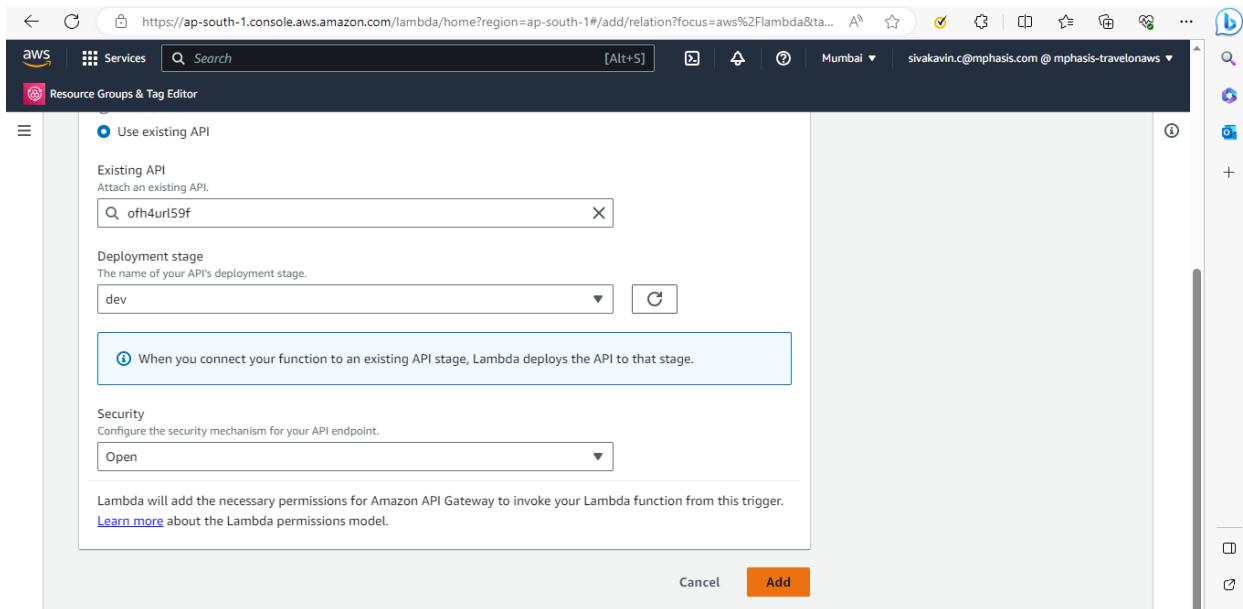
STEP 8: Resource also created by default.

The screenshot shows the AWS API Gateway console. The URL is https://ap-south-1.console.aws.amazon.com/apigateway/home?region=ap-south-1#/apis/uprx1rh1jh/resources/5482s2wui. The navigation path is APIs > MyNewFunc-API (uprx1rh1jh) > Resources > / (5482s2wui). On the left sidebar, under the API 'MyNewFunc-API' section, the 'Resources' tab is selected. In the main pane, the path '/MyNewFunc' is expanded, showing an 'ANY' method. A message states 'No methods defined for the resource.' The right sidebar contains various AWS service icons.

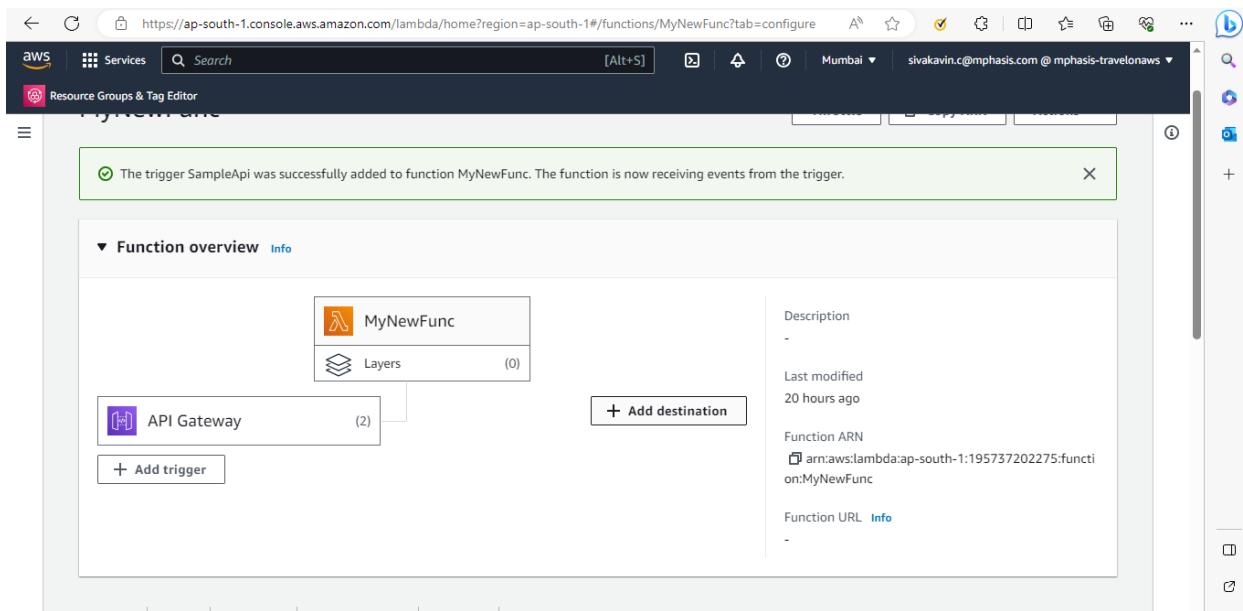
STEP 9: Instead of choosing “Create a new API”, choose “Use existing API” (If you have an existing API)

The screenshot shows the AWS Lambda console. The URL is https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/add/relation?focus=aws%2Flambda&t... . The navigation path is Lambda > Add trigger. The main title is 'Add trigger'. Under 'Trigger configuration', the 'API Gateway' option is selected from a dropdown menu. Below it, there's a note: 'Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)'. There are two radio buttons for 'Intent': 'Create a new API' (unchecked) and 'Use existing API' (checked). Below this, there's a section for 'Existing API' with a search bar containing 'ofh4url59f'. At the bottom, there are links for 'CloudShell', 'Feedback', 'Language', and copyright information: '© 2023, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

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.

The screenshot shows the AWS Lambda console with the URL <https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#functions/MyNewFunc?tab=config>. The left sidebar is titled 'Triggers' and includes options like Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, Asynchronous invocation, Code signing, and Database proxies. The main pane displays a list of triggers under the 'Trigger' section. Two triggers are listed: 'API Gateway: MyNewFunc-API' and 'API Gateway: SampleApi'. The 'SampleApi' trigger is expanded, showing its details: API type: REST, Authorization: AWS_IAM, Method: ANY, Resource path: /MyNewFunc, Service principal: apigateway.amazonaws.com, Stage: dev, and Statement ID: lambda-0c69d48a-46e8-4257-9055-8cecf44a5cc1.

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

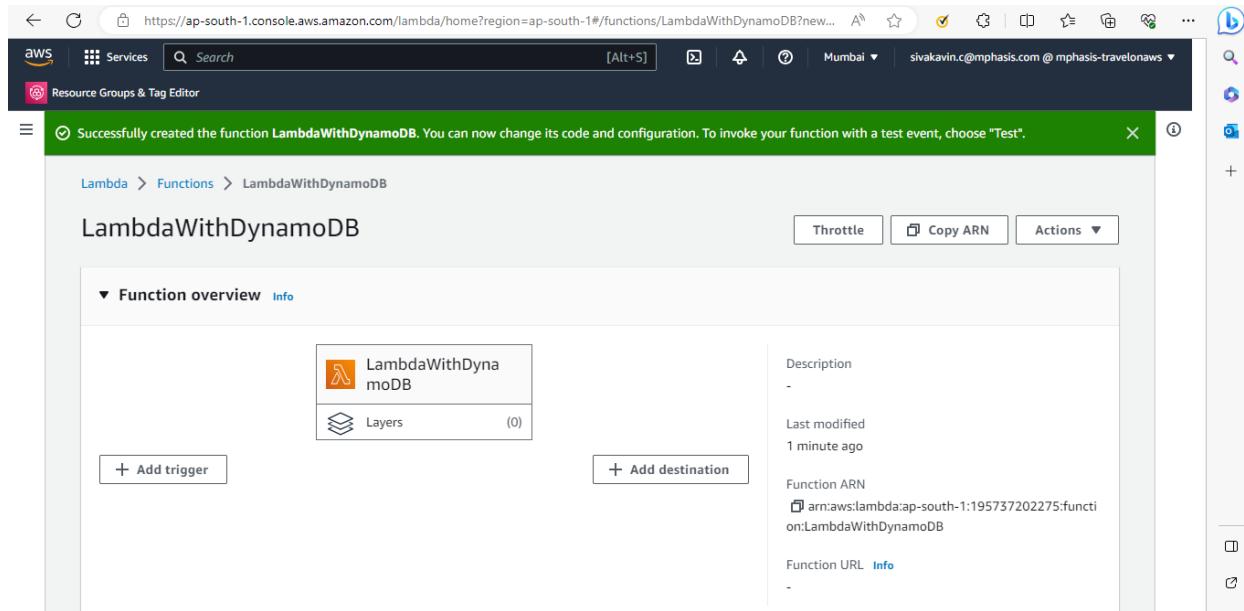
The screenshot shows the AWS API Gateway console with the URL <https://ap-south-1.console.aws.amazon.com/apigateway/main/apis?region=ap-south-1>. The left sidebar is titled 'API Gateway' and includes options like APIs, Custom domain names, and VPC links. The main pane displays a table titled 'APIs (14)'. A search bar at the top has the text 'sa' entered. One API entry is visible: 'SampleApi' (ID: ofh4url59f, Protocol: REST, Endpoint type: Regional, Created: 2023-08-30).

STEP 14: Resource created successfully.

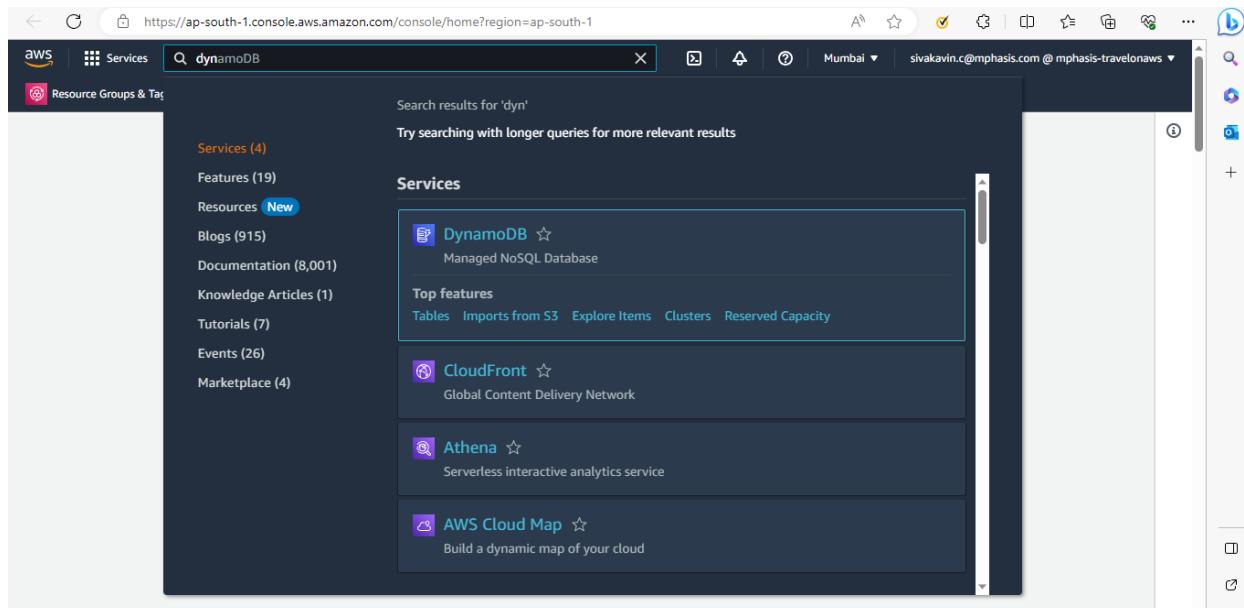
The screenshot shows the AWS API Gateway console at the URL <https://ap-south-1.console.aws.amazon.com/apigateway/home?region=ap-south-1#/apis/ofh4url59f/resources/8v145ugu3h>. The navigation bar includes the AWS logo, Services, a search bar, and account information for sivakavin.c@mphasis.com @ mphasis-travelonaws. The main interface shows the 'Resource Groups & Tag Editor' for the 'Amazon API Gateway'. The left sidebar lists 'APIs', 'Custom Domain Names', and 'VPC Links'. Under 'API: SampleApi', the 'Resources' section is selected, showing a tree structure with a root node '/'. A new child node '/MyNewFunc' has been created under it, with the 'ANY' method selected. The right panel displays the '/ Methods' section, which currently says 'No methods defined for the resource.' There are also tabs for 'Actions' and a 'Show hints' button.

INTEGRATING LAMBDA WITH DYNAMODB

STEP 1: Create the “lambda function.”



STEP 2: Click the “DynamoDB” service.



STEP 3: Click “Create table”.

The screenshot shows the AWS DynamoDB Dashboard. On the left, there's a sidebar with options like 'Tables', 'Update settings', 'Explore items', etc. A 'DAX' section is expanded, showing 'Clusters', 'Subnet groups', and 'Parameter groups'. The main area is titled 'Dashboard' and contains sections for 'Alarms (0)' and 'DAX clusters (0)'. To the right, a 'Create resources' panel is open, featuring a large orange 'Create table' button. Below it, there's a brief description of Amazon DynamoDB Accelerator (DAX) and a 'Create DAX cluster' button. The URL in the browser is <https://ap-south-1.console.aws.amazon.com/dynamodbv2/home?region=ap-south-1#dashboard>.

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

The screenshot shows the 'Create table' wizard. It's on the 'Table details' step. The 'Table name' field is filled with 'DynamoWithLambda'. The 'Partition key' section shows a field with 'id' and a dropdown set to 'String'. The 'Sort key - optional' section has a field with 'Enter the sort key name' and a dropdown set to 'String'. The URL in the browser is <https://ap-south-1.console.aws.amazon.com/dynamodbv2/home?region=ap-south-1#create-table>.

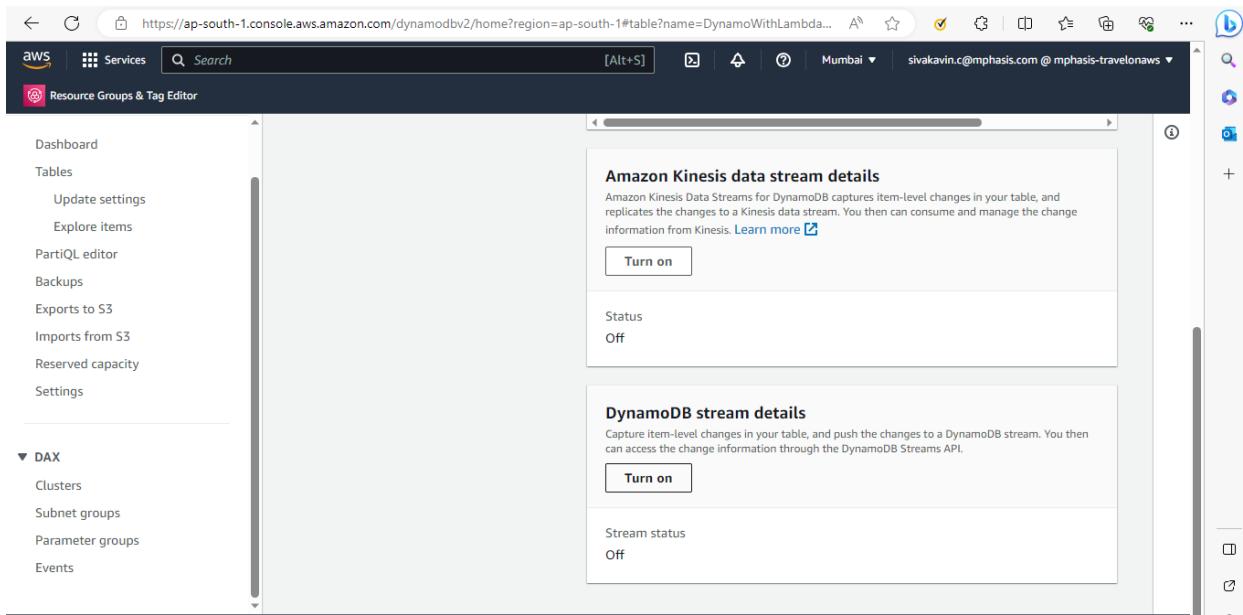
STEP 5: Table is created here.

The screenshot shows the AWS DynamoDB console interface. On the left, there's a navigation sidebar with options like 'Dashboard', 'Tables', 'Update settings', etc. The main area is titled 'Tables (32) Info'. A search bar at the top has 'DynamoW' typed into it. Below the search bar is a table with columns: Name, Status, Partition key, Sort key, Indexes, Deletion protection, and Read capacity. One row in the table is highlighted with a blue background and shows the table name 'DynamoWithLambda', status 'Active', partition key 'id (S)', sort key ' - ', 0 indexes, deletion protection 'Off', and provisioned read capacity. There are also 'Actions' and 'Delete' buttons above the table.

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

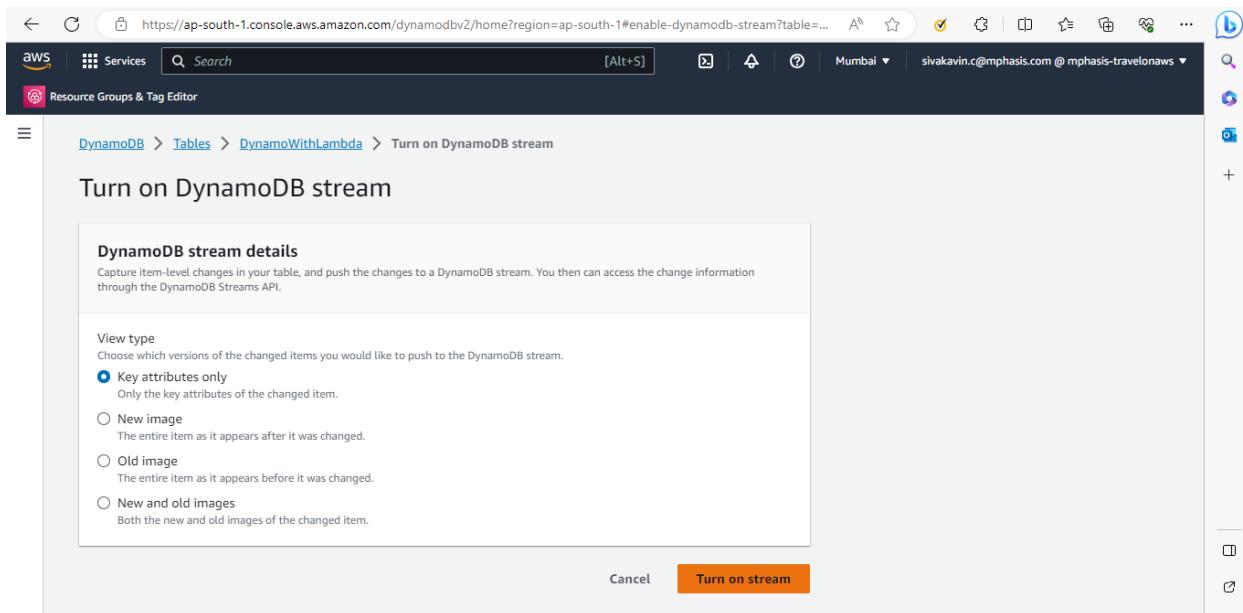
This screenshot shows the same AWS DynamoDB console as the previous one, but with a different view. It's focused on the 'DynamoWithLambda' table. The navigation path is 'DynamoDB > Tables > DynamoWithLambda'. The main content area has tabs at the top: 'Tables', 'Backups', 'Exports and streams' (which is highlighted in blue), and 'Additional settings'. Below these tabs, there's a callout box with an info icon that says 'Protect your DynamoDB table from accidental writes and deletes'. It explains that turning on Point-in-time Recovery (PITR) backs up table data automatically for restoration. There's also a 'Edit PITR' button. At the bottom, there's a section titled 'General information' with fields for 'Partition key' and 'Sort key'.

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



The screenshot shows the AWS DynamoDB console for a table named "DynamoWithLambda". On the left, there's a sidebar with options like Dashboard, Tables, Update settings, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Reserved capacity, Settings, Clusters, Subnet groups, Parameter groups, and Events. The main area is divided into two sections: "Amazon Kinesis data stream details" and "DynamoDB stream details". Both sections have a "Turn on" button and a "Status: Off" indicator. The URL in the browser is https://ap-south-1.console.aws.amazon.com/dynamodbv2/home?region=ap-south-1#table?name=DynamoWithLambda...

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



The screenshot shows the "Turn on DynamoDB stream" dialog. At the top, it says "DynamoDB > Tables > DynamoWithLambda > Turn on DynamoDB stream". The main content area is titled "DynamoDB stream details" and includes a description: "Capture item-level changes in your table, and push the changes to a DynamoDB stream. You then can access the change information through the DynamoDB Streams API." Below this is a "View type" section with the following options:

- Key attributes only: Only the key attributes of the changed item.
- New image: The entire item as it appears after it was changed.
- Old image: The entire item as it appears before it was changed.
- New and old images: Both the new and old images of the changed item.

At the bottom of the dialog are "Cancel" and "Turn on stream" buttons.

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

The screenshot shows the AWS DynamoDB console. On the left, there's a sidebar with options like Dashboard, Tables, Update settings, etc. The main area shows a table named 'DynamoWithLambda'. A green banner at the top says 'Successfully turned on streaming to the DynamoDB stream.' Below it, there's a search bar and a table view with one item selected. On the right, there are tabs for Monitor, Global tables, Backups, and Exports and streams, with 'Exports and streams' being the active tab. It shows a section for 'Exports to S3 (0)' with a 'View details' button and an 'Export to S3' button.

STEP 10: Click “Add trigger”.

The screenshot shows the AWS Lambda console. The left sidebar has 'Lambda' and 'Functions' selected. The main area shows a function named 'LambdaWithDynamoDB'. Under the 'Function overview' section, there's a box for triggers and a box for destinations. A button labeled '+ Add trigger' is visible in the triggers section. Other buttons include 'Throttle', 'Copy ARN', and 'Actions'. To the right, there are sections for Description, Last modified (22 minutes ago), Function ARN (with a link), and Function URL (with a link).

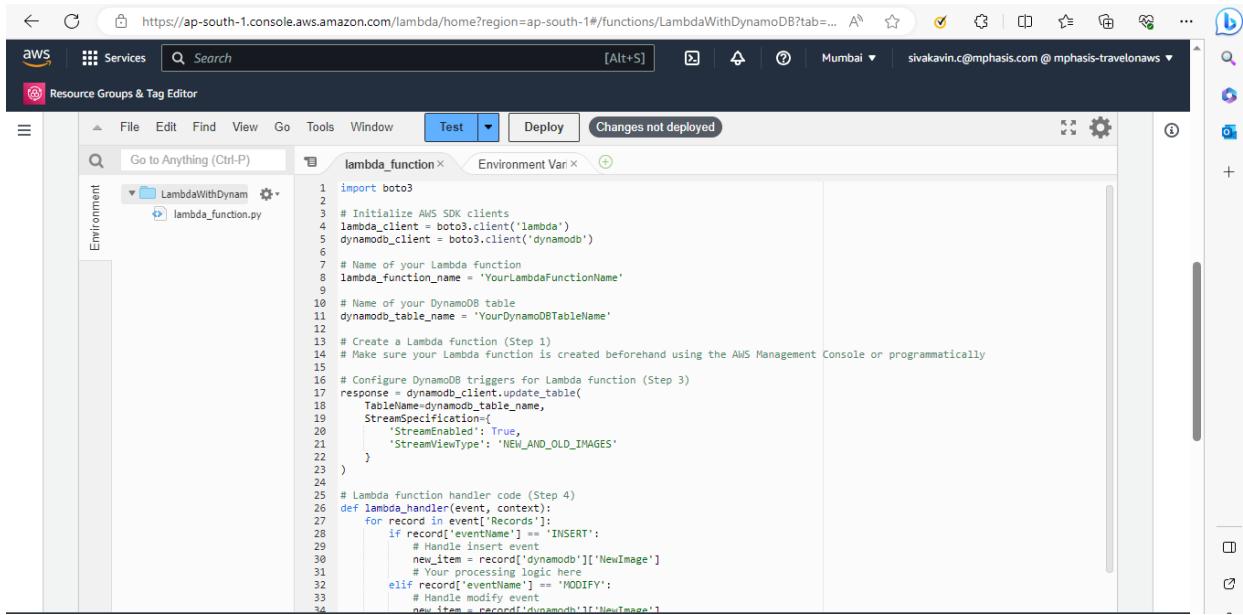
STEP 11: Choose “DynamoDB” as source, “DynamoDB Table name”, Batch size and click “Add”.

The screenshot shows the AWS Lambda Trigger configuration page. In the 'Trigger configuration' section, a dropdown menu is open under 'Source' with 'DynamoDB' selected. Below it, a search bar contains the ARN: arn:aws:dynamodb:ap-south-1:195737202275:table/DynamoWithLambda. Under 'Batch size', the value '100' is entered. The 'Starting position' dropdown is set to 'Latest'. The 'Batch window - optional' field is empty. A checkbox for 'Activate trigger' is unchecked. A note says: 'Select to activate the trigger now. Keep unchecked to create the trigger in a deactivated state for testing (recommended).'

STEP 12: Trigger is added successfully.

The screenshot shows the AWS Lambda Functions page. The function 'LambdaWithDynamoDB' is selected. At the top right, there are buttons for 'Throttle', 'Copy ARN', and 'Actions'. A green success message box states: 'The trigger DynamoWithLambda was successfully added to function LambdaWithDynamoDB. The trigger is in a disabled state.' In the 'Function overview' section, the function name 'LambdaWithDynamoDB' is shown with a small icon. Below it, there's a 'Layers' section with '(0)' and a '+ Add destination' button. Under 'Triggers', a 'DynamoDB' entry is listed with a '+ Add trigger' button. On the right side, there are fields for 'Description', 'Last modified' (54 seconds ago), 'Function ARN' (arn:aws:lambda:ap-south-1:195737202275:function:LambdaWithDynamoDB), and 'Function URL'.

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



The screenshot shows the AWS Lambda function editor interface. The top navigation bar includes the AWS logo, Services, Search, and Mumbai. The main area displays a Python script named `lambda_function.py` under the `LambdaWithDynam` function. The code implements a Lambda function that interacts with AWS SDK clients for Lambda and DynamoDB. It defines variables for function name and table name, configures a stream specification for the DynamoDB table, and defines a handler function that processes events based on their event name (e.g., INSERT, MODIFY) and handles insertions or modifications to the dynamodb table.

```
1 import boto3
2
3 # Initialize AWS SDK clients
4 lambda_client = boto3.client('lambda')
5 dynamodb_client = boto3.client('dynamodb')
6
7 # Name of your Lambda function
8 lambda_function_name = 'YourLambdaFunctionName'
9
10 # Name of your DynamoDB table
11 dynamodb_table_name = 'YourDynamoDBTableName'
12
13 # Create a Lambda function (Step 1)
14 # Make sure your Lambda function is created beforehand using the AWS Management Console or programmatically
15
16 # Configure DynamoDB triggers for Lambda function (Step 3)
17 response = dynamodb_client.update_table(
18     TableName=dynamodb_table_name,
19     StreamSpecification={
20         'StreamEnabled': True,
21         'StreamViewType': 'NEW_AND_OLD_IMAGES'
22     }
23 )
24
25 # Lambda function handler code (Step 4)
26 def lambda_handler(event, context):
27     for record in event['Records']:
28         if record['eventName'] == 'INSERT':
29             # Handle insert event
30             new_item = record['dynamodb']['NewImage']
31             # Your processing logic here
32         elif record['eventName'] == 'MODIFY':
33             # Handle modify event
34             new_item = record['dynamodb']['NewImage']
```

OTHER WAY OF INTEGRATION WITH DYNAMODB

STEP 1: Create Lambda function.

The screenshot shows the AWS Lambda console interface. At the top, a green success message states: "Successfully created the function MyLambdaDynamo. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below this, the function name "MyLambdaDynamo" is displayed. The "Function overview" section shows the function icon, a "Layers" section with "(0)", and buttons for "+ Add trigger" and "+ Add destination". On the right, there are fields for "Description", "Last modified" (6 seconds ago), "Function ARN" (arn:aws:lambda:ap-south-1:195737202275:function:MyLambdaDynamo), and "Function URL" (Info). A sidebar on the left lists "Services" like CloudWatch, Lambda, and S3.

STEP 2: Create Dynamo DB Table.

The screenshot shows the AWS DynamoDB console. The left sidebar has a "Tables" section with options like "Update settings", "Explore items", and "Create table". The main area shows a table named "Tables (32)" with a search bar containing "My". A table row for "MyDynamoDB" is listed with the status "Active", partition key "id (S)", sort key "-", and read capacity mode "Provisioned (5)". A "Create table" button is visible at the top right of the table list. The top navigation bar shows the URL https://ap-south-1.console.aws.amazon.com/dynamodbv2/home?region=ap-south-1#tables.

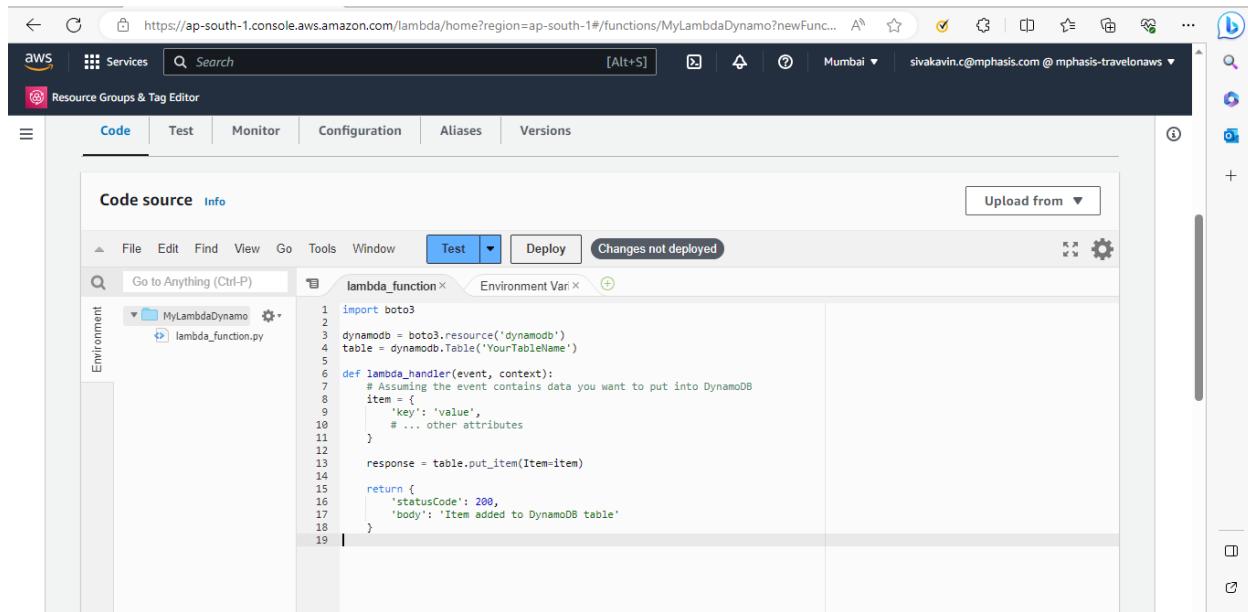
STEP 3: Create an “IAM role” or use existing role with access as “DynamoDBFullAccess”.

The screenshot shows the AWS Lambda function configuration page for a function named 'MyLambdaDynamo'. In the 'Execution role' section, the 'Use an existing role' radio button is selected, and a dropdown menu shows 'tnb-mphasis-lambda-role' is chosen. Below the dropdown, there is a link 'View the tnb-mphasis-lambda-role role on the IAM console.' At the bottom of the page, there are 'Cancel' and 'Save' buttons.

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.

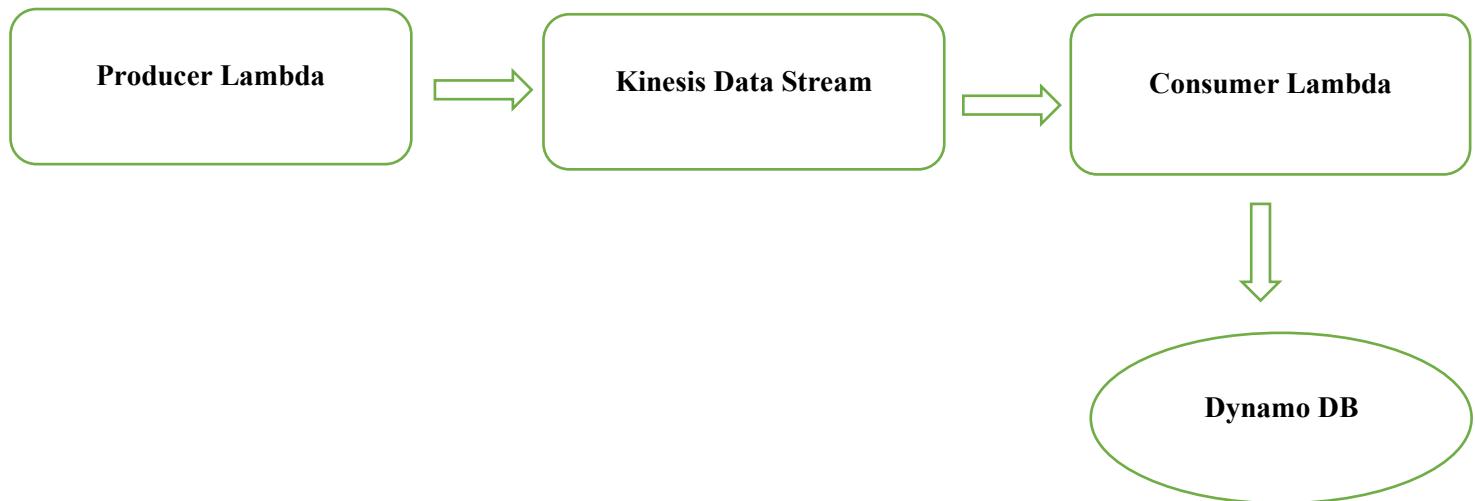


The screenshot shows the AWS Lambda console interface. At the top, there's a navigation bar with links for Services, Search, [Alt+S], Mumbai, and sivakavin.c@mphasis.com @ mphasis-travelonaws. Below the navigation bar, there are tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Code tab is selected. On the left, there's a sidebar with Environment and Resource Groups & Tag Editor. The main area is titled "Code source" and "Info". It has tabs for File, Edit, Find, View, Go, Tools, Window, Test, Deploy, and Changes not deployed. A search bar says "Go to Anything (Ctrl-P)". There's a "lambda_function" folder expanded, containing a file named "lambda_function.py". The code in the file is:

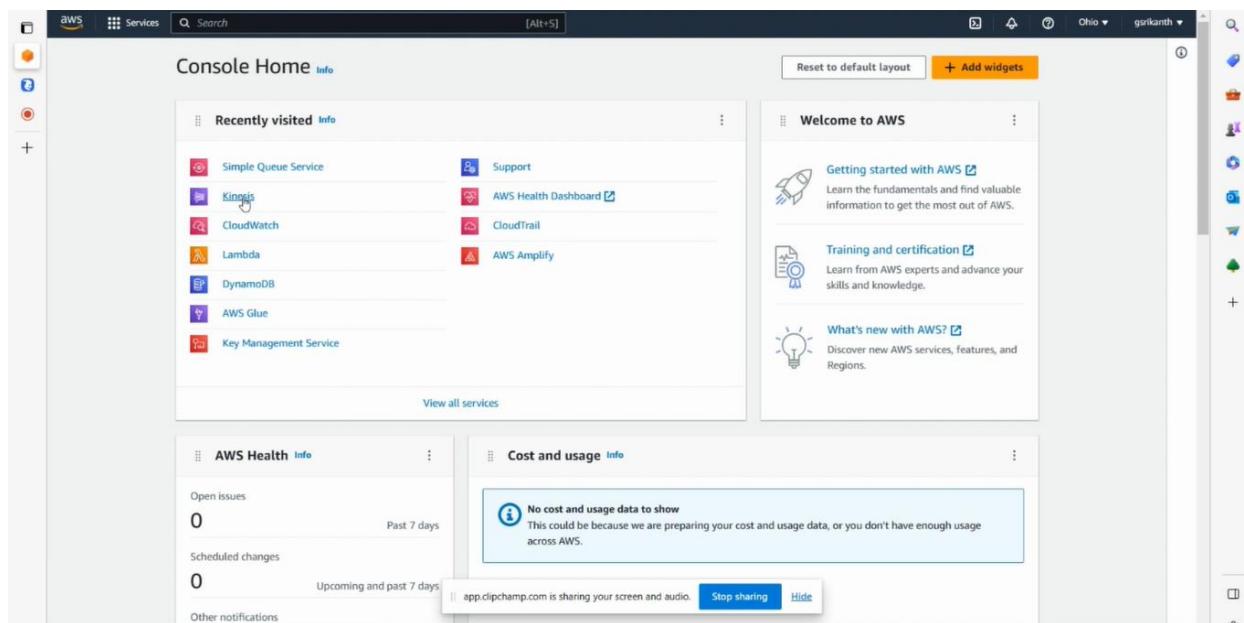
```
1 import boto3
2
3 dynamodb = boto3.resource('dynamodb')
4 table = dynamodb.Table('YourTableName')
5
6 def lambda_handler(event, context):
7     # Assuming the event contains data you want to put into DynamoDB
8     item = {
9         "key": "value",
10        # ... other attributes
11    }
12
13 response = table.put_item(Item=item)
14
15 return {
16     'statusCode': 200,
17     'body': 'Item added to DynamoDB table'
18 }
```

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

WORKFLOW DIAGRAM



STEP 1: Choose the service “Kinesis”.



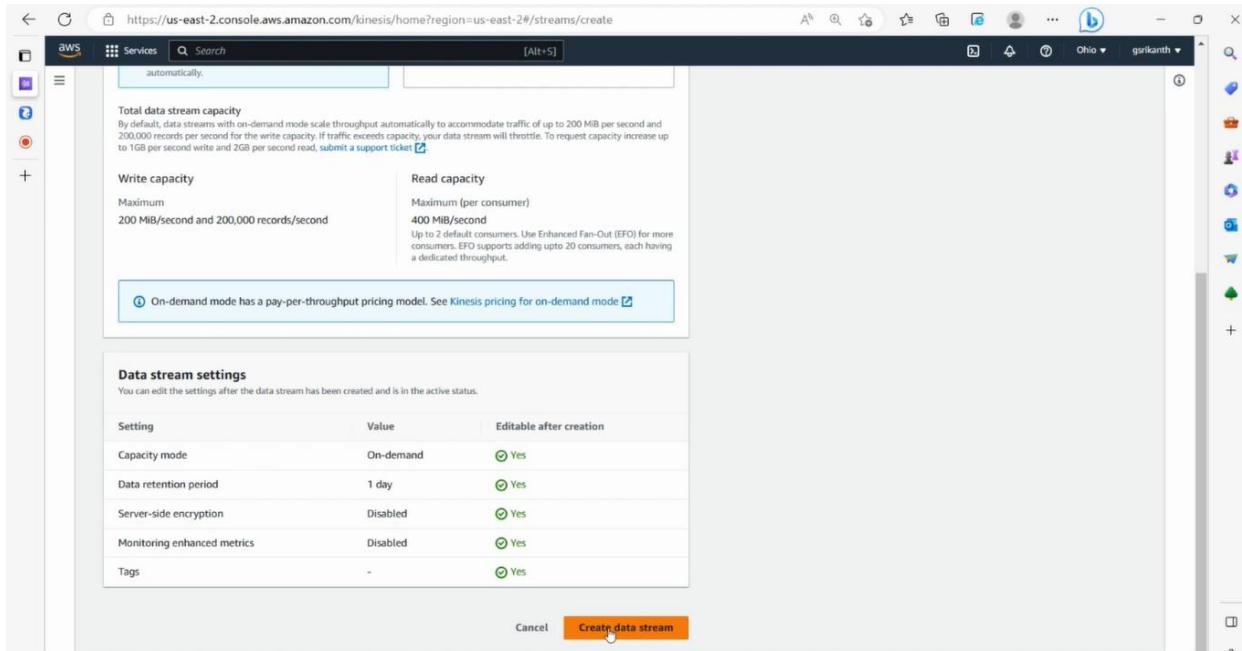
STEP 2: Click “Create data streams”.

The screenshot shows the AWS Kinesis Data Streams list page. On the left, there's a navigation sidebar with 'Amazon Kinesis' selected. The main area displays a table titled 'Data streams (0) Info'. A prominent button at the top right of the table header is labeled 'Create data stream'. Above the table, a message box states 'New on-demand mode for Kinesis data streams' and describes its benefits. The table has columns for Name, Status, Capacity mode, Provisioned shards, Data retention period, Encryption, and Consumers with enhanced fan-out. A note at the bottom of the table says 'No data streams'.

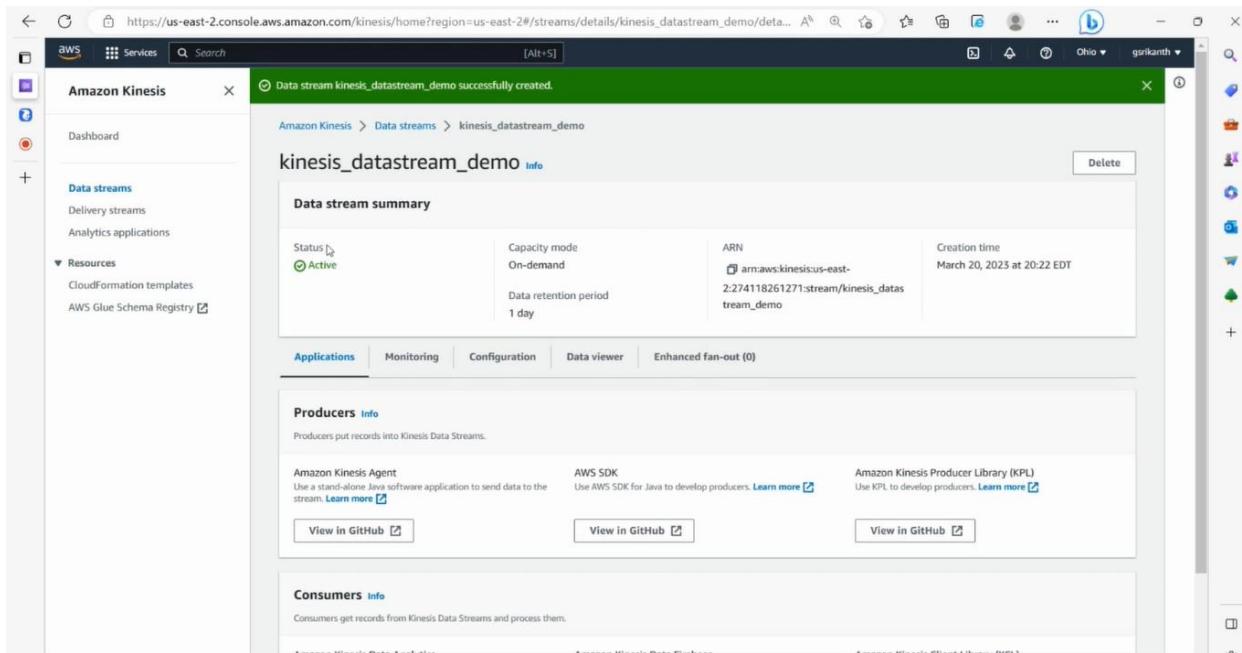
STEP 3: Mention name for data stream and choose data stream capacity.

The screenshot shows the 'Create data stream' configuration page. In the 'Data stream configuration' section, the 'Data stream name' field contains 'kinesis_datastream_demo'. Below it, a note specifies acceptable characters: uppercase and lowercase letters, numbers, underscores, hyphens, and periods. In the 'Data stream capacity' section, the 'Capacity mode' dropdown is set to 'On-demand', which is described as a mode for unpredictable and variable throughput. The 'Total data stream capacity' section notes that throughput scales automatically between 200 MB per second and 200,000 records per second. The 'Write capacity' section shows a maximum of 200 MB per second and 200,000 records per second. The 'Read capacity' section shows a maximum of 400 MB per second for up to 2 default consumers. A note at the bottom right indicates that a dedicated consumer can be used with Enhanced Fan-Out (EFO).

STEP 4: Click “Create data stream”.



STEP 5: Kinesis data stream created successfully.



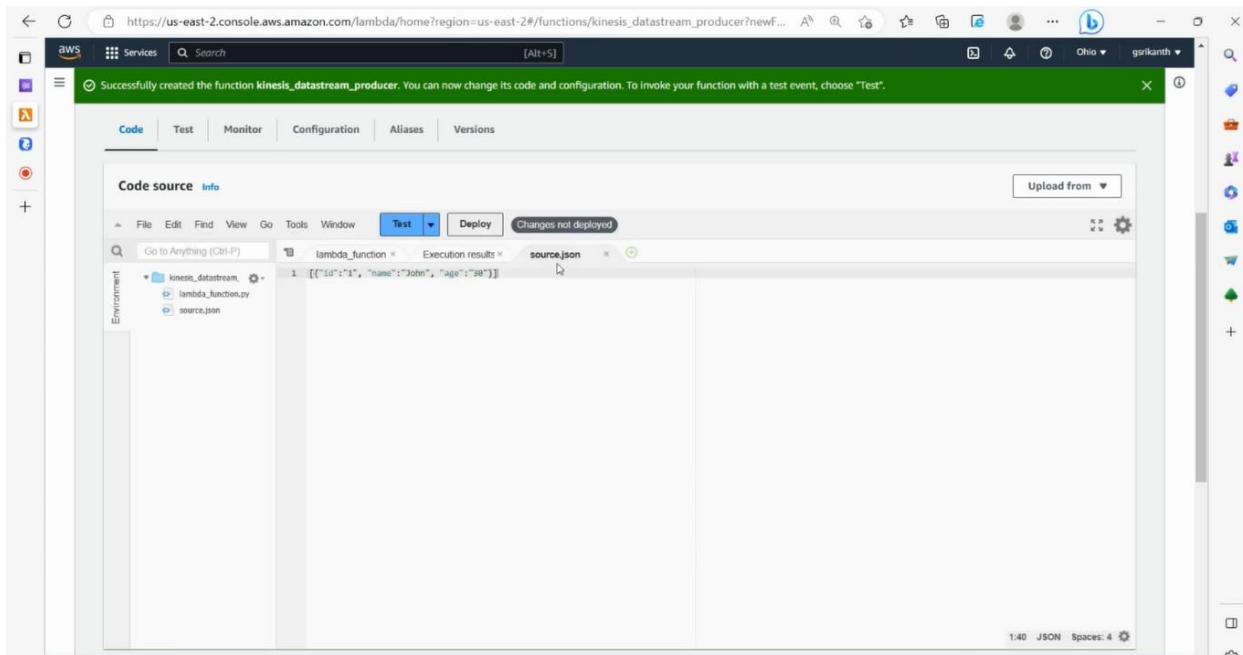
STEP 6: Create lambda function.

The screenshot shows the AWS Lambda Functions page. On the left, there's a sidebar with options like Dashboard, Applications, Functions, Additional resources, and Related AWS resources. The main area is titled 'Functions (6)' and lists six existing Lambda functions: 'Display_movie_details_from_dynamodb', 'MovieDetails_Func', 'lambda_map_inline_demo', 'sqS_trigger_demo', 'sqS_msg_processing', and 's3_trigger_glue_job'. Each function entry includes columns for Function name, Description, Package type, Runtime, and Last modified. A prominent orange 'Create function' button is located at the top right of the table.

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

The screenshot shows the 'Create function' wizard. Step 1: Set runtime and architecture. It asks for a function name ('kinesis_datostream_producer') and runtime ('Python 3.9'). It also includes sections for architecture (choose between x86_64 or arm64) and permissions (choose an execution role). The 'Advanced settings' section is partially visible at the bottom. A large orange 'Create function' button is at the bottom right.

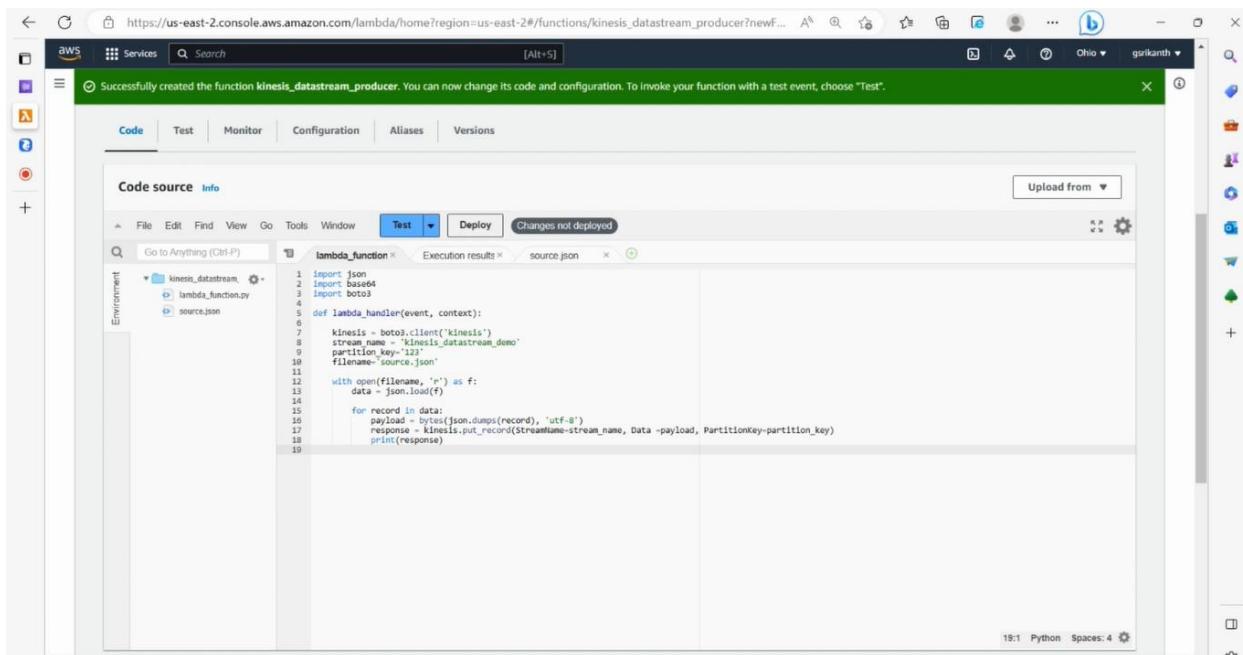
STEP 8: Just created source.json and add particular details.



The screenshot shows the AWS Lambda console interface. At the top, a green banner says "Successfully created the function kinesis_datastream_producer. You can now change its code and configuration. To invoke your function with a test event, choose "Test". The main area is titled "Code source" and shows a file tree. Inside the "lambda_function" folder, there are three files: "lambda_function.py", "source.json", and "source.json" (a duplicate). The "source.json" file is open in the editor, displaying the following JSON content:

```
[{"id": "1", "name": "John", "age": "30"}]
```

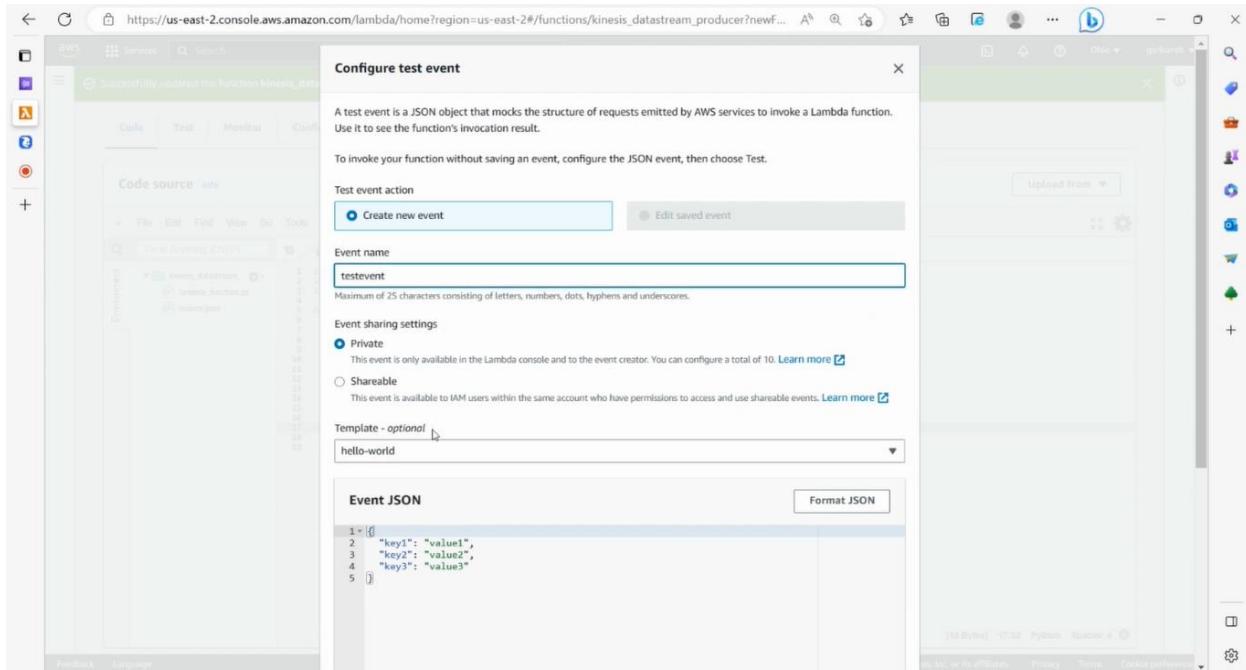
STEP 9: Write a code for sending a data to kinesis data stream.



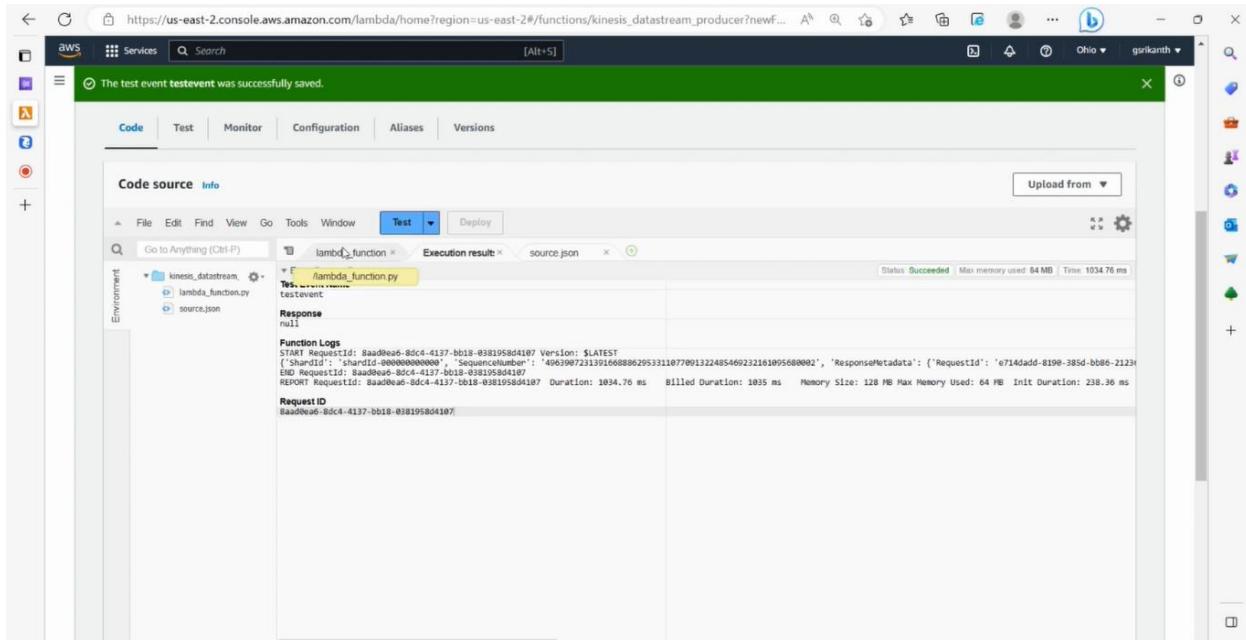
The screenshot shows the AWS Lambda console interface. At the top, a green banner says "Successfully created the function kinesis_datastream_producer. You can now change its code and configuration. To invoke your function with a test event, choose "Test". The main area is titled "Code source" and shows a file tree. Inside the "lambda_function" folder, there are three files: "lambda_function.py", "source.json", and "source.json" (a duplicate). The "lambda_function.py" file is open in the editor, displaying the following Python code:

```
1 import json
2 import base64
3 import botocore
4
5 def lambda_handler(event, context):
6     kinesis = botocore.client('kinesis')
7     stream_name = 'kinesis_datastream_demo'
8     partition_key = 'partition_key'
9     filename='source.json'
10
11     with open(filename, 'r') as f:
12         data = json.load(f)
13
14         for record in data:
15             payload = bytes(json.dumps(record), 'utf-8')
16             response = kinesis.put_record(StreamName=stream_name, Data =payload, PartitionKey=partition_key)
17             print(response)
18
```

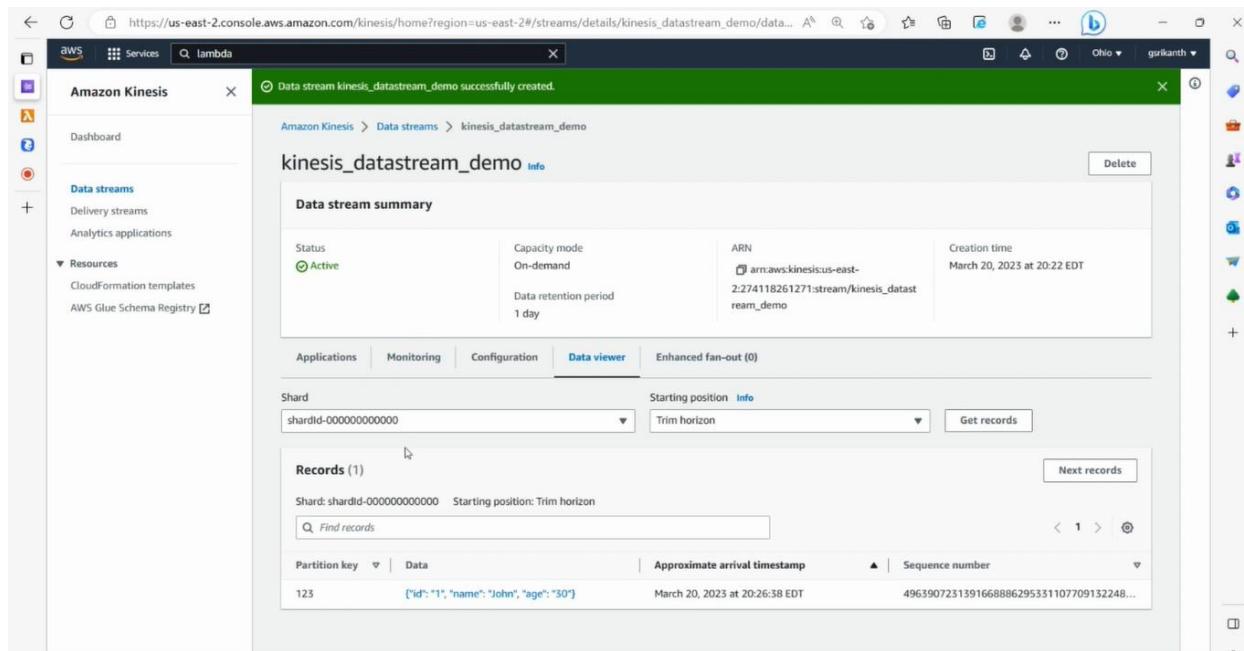
STEP 10: Create test event and save it.



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



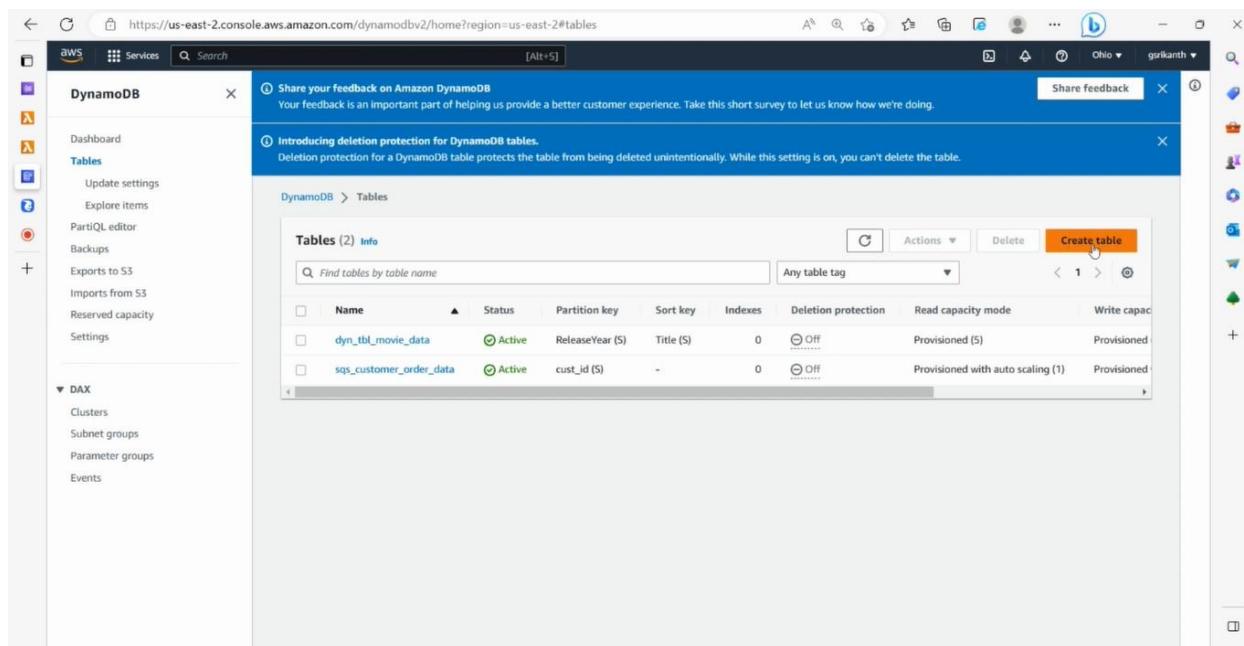
STEP 12: After testing the producer function, open kinesis console. In that, click “Data viewer” and it showing the data is in the kinesis data stream.



The screenshot shows the Amazon Kinesis Data Stream Data Viewer interface. At the top, a green banner indicates "Data stream kinesis_dynamodb_demo successfully created." Below this, the "kinesis_dynamodb_demo" data stream is selected. The "Data viewer" tab is active, showing a summary table with details like Status (Active), Capacity mode (On-demand), ARN (arn:aws:kinesis:us-east-2:274118261271:stream/kinesis_dynamodb_demo), and Creation time (March 20, 2023 at 20:22 EDT). Below the summary, a "Shard" dropdown is set to "shardid-000000000000" and a "Starting position" dropdown is set to "Trim horizon". A "Get records" button is present. The main area displays a table titled "Records (1)" with one record listed:

Partition key	Data	Approximate arrival timestamp	Sequence number
123	[{"id": "1", "name": "John", "age": "30"}]	March 20, 2023 at 20:26:38 EDT	4963907231591668886295331107709152248...

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



The screenshot shows the Amazon DynamoDB Tables page. The left sidebar includes options like Dashboard, Tables (selected), Update settings, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Reserved capacity, and Settings. The main area displays a table titled "Tables (2) Info" with two entries:

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode
dyn_tbl_movie_data	Active	ReleaseYear (\$)	Title (\$)	0	Off	Provisioned (5)	Provisioned
sqs_customer_order_data	Active	cust_id (\$)	-	0	Off	Provisioned with auto scaling (1)	Provisioned

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

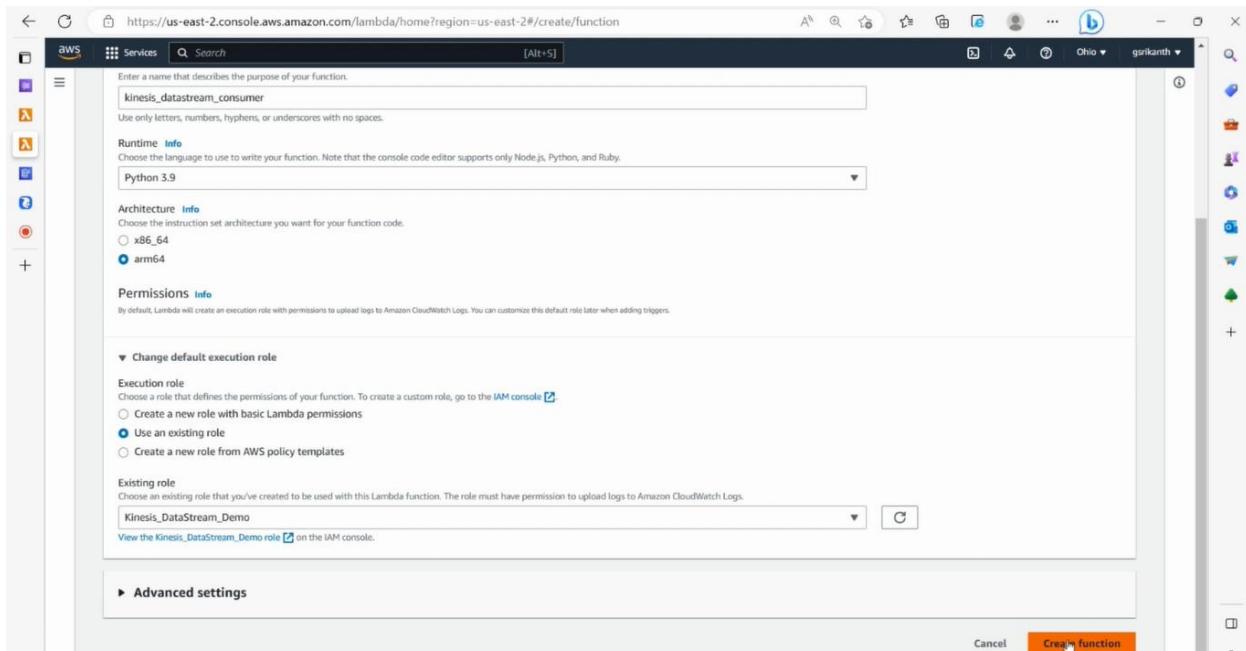
The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The 'Table details' step is active. In the 'Table name' section, the name 'kinesis_datastream_customer_demo' is entered. In the 'Partition key' section, the key 'id' is defined as a String type. A 'Sort key - optional' section is also present. The browser address bar shows the URL <https://us-east-2.console.aws.amazon.com/dynamodbv2/home?region=us-east-2#create-table>.

STEP 15: Table created successfully.

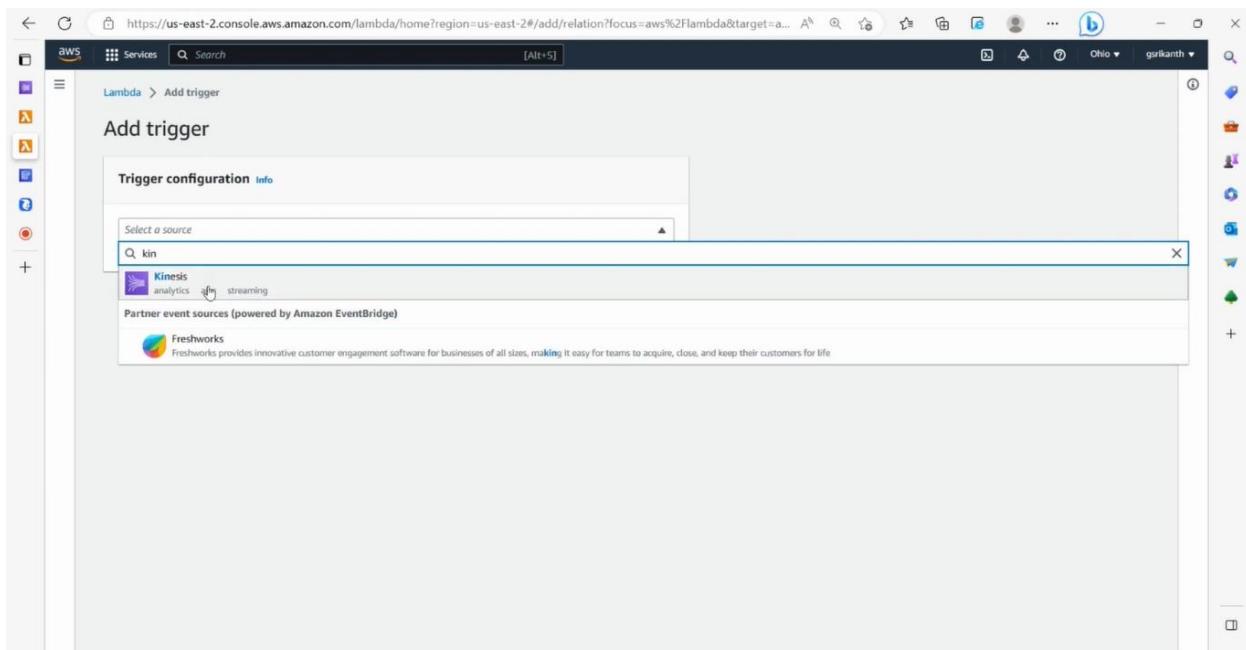
The screenshot shows the 'Tables' page in the AWS DynamoDB console. The table 'kinesis_datastream_customer_demo' is listed, showing it was created successfully. The table has 3 items and is in the 'Active' status. The browser address bar shows the URL <https://us-east-2.console.aws.amazon.com/dynamodbv2/home?region=us-east-2#tables>.

Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode
dyn_tbl_movie_data	Active	ReleaseYear (\$)	Title (\$)	0	Off	Provisioned (5)
kinesis_datastream_customer_demo	Active	id (\$)	-	0	Off	Provisioned with auto scaling (5)
sqs_customer_order_data	Active	cust_id (\$)	-	0	Off	Provisioned with auto scaling (1)

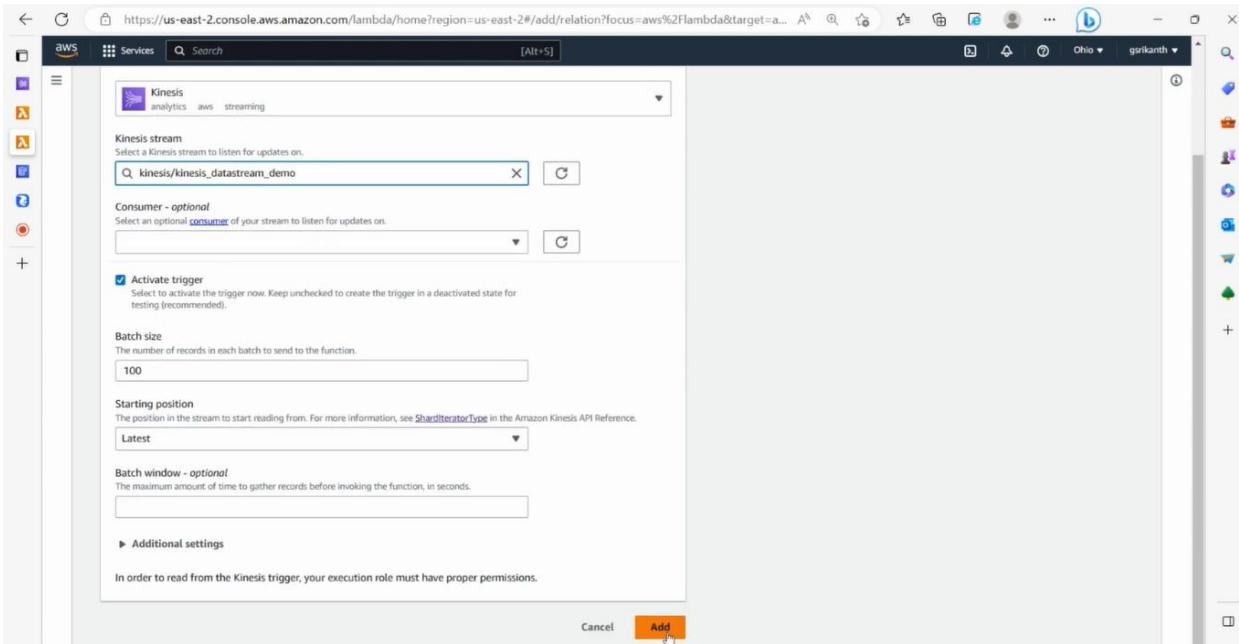
STEP 16: Now create another lambda function as consumer and fill the necessary details here.



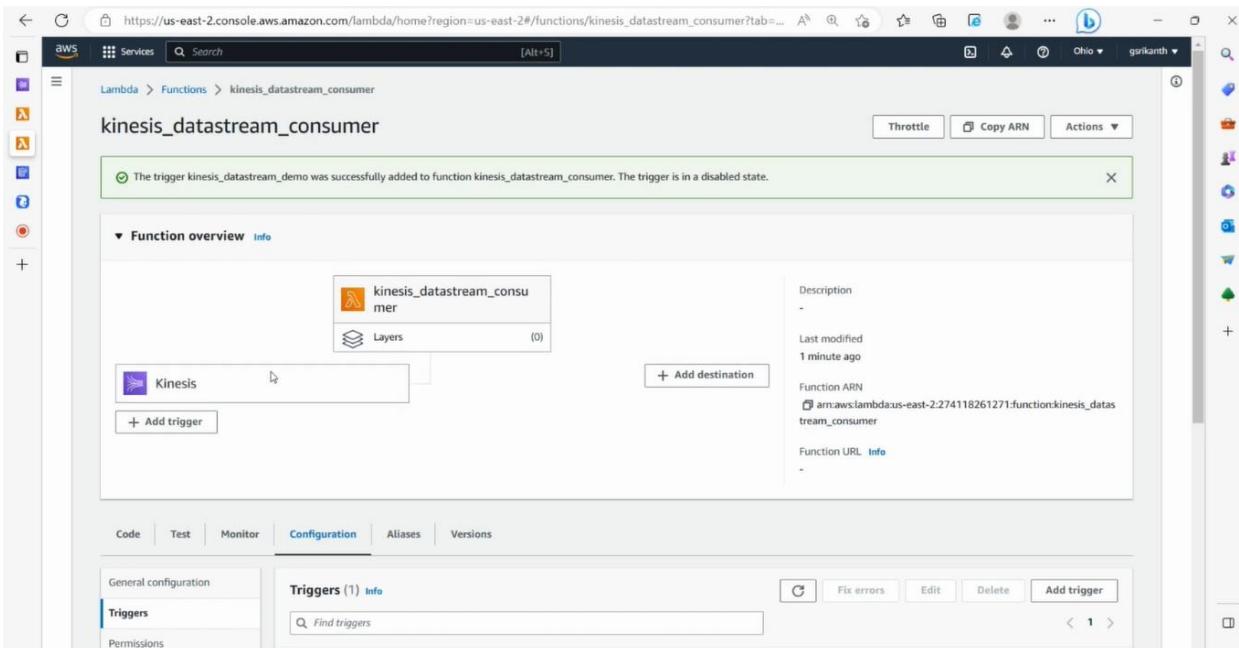
STEP 17: In this consumer lambda, add a trigger (Kinesis).



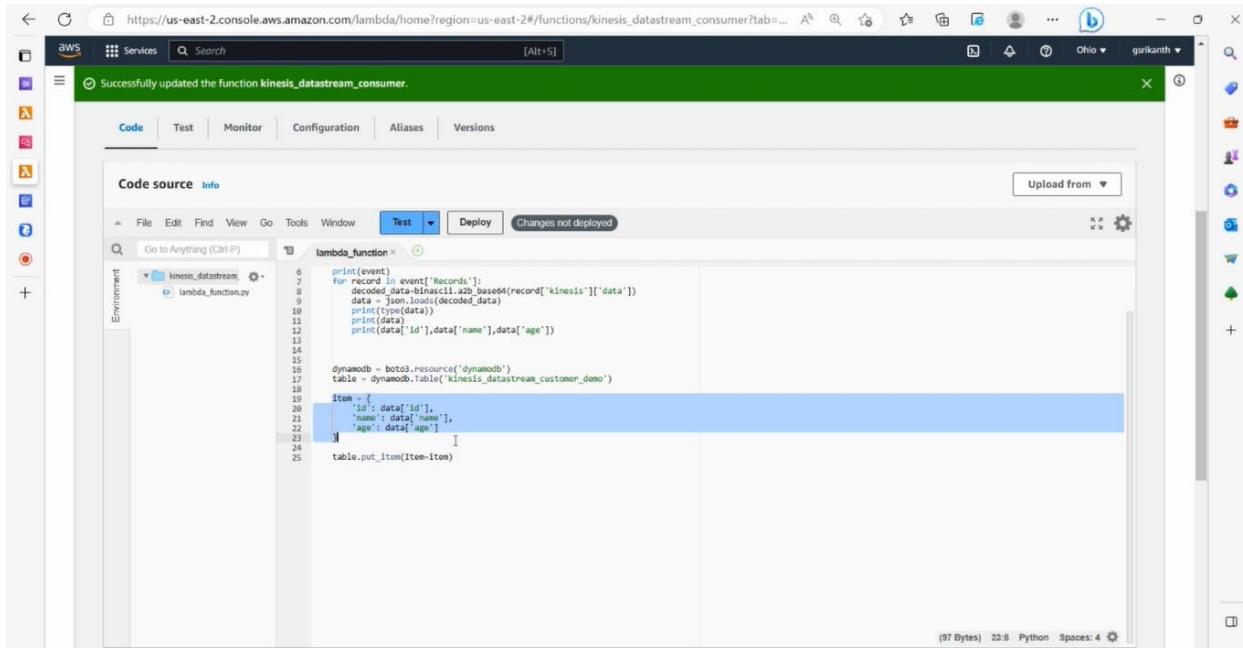
STEP 18: Choose Kinesis stream (already created) and click add.



STEP 19: Trigger added successfully.



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



The screenshot shows the AWS Lambda console interface. A green success message at the top states "Successfully updated the function kinesis_datastream_consumer." Below it, the "Code source" tab is selected, showing the Python code for the lambda_function. The code reads records from a Kinesis stream, decodes them, loads them into a JSON object, prints the data, and then inserts it into a DynamoDB table named "kinesis_datastream_customer_demo". The code is as follows:

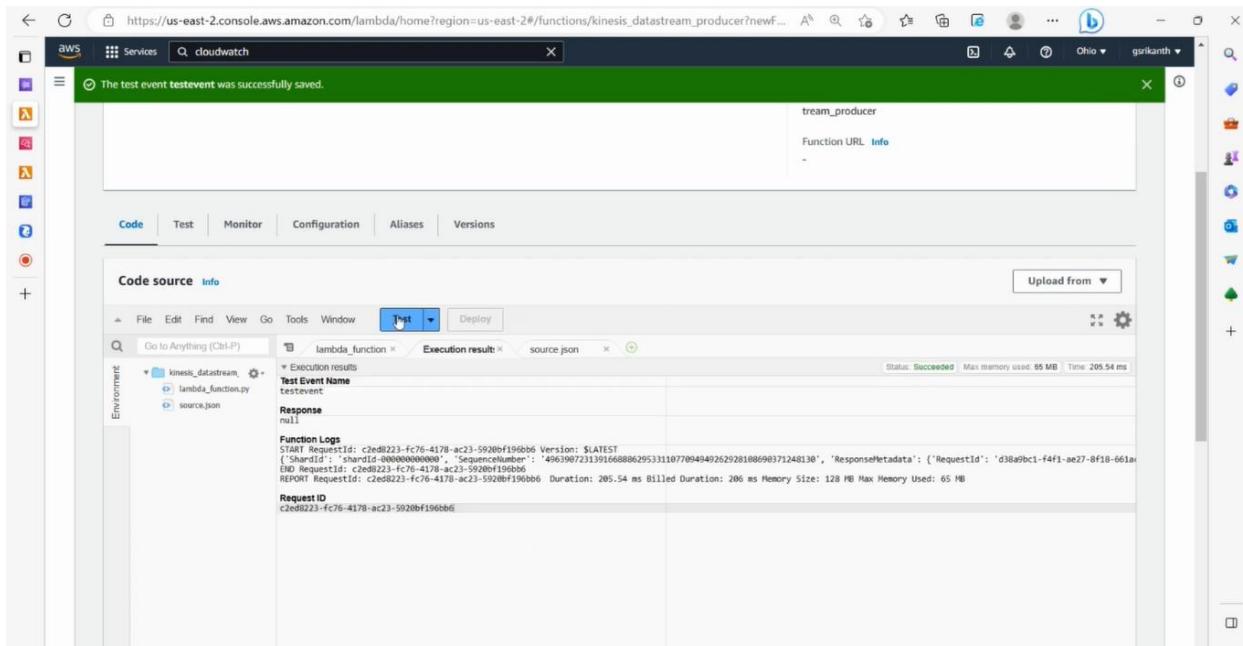
```
print(event)
for record in event['Records']:
    decoded_data=base64.b64decode(record['kinesis']['data'])
    data = json.loads(decoded_data)
    print(data)
    print(data['id'],data['name'],data['age'])

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('kinesis_datastream_customer_demo')

item = {
    'id': data['id'],
    'name': data['name'],
    'age': data['age']
}
table.put_item(Item=item)
```

At the bottom right, status information is displayed: "(97 Bytes) 23.6 Python Spaces: 4".

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



The screenshot shows the AWS Lambda console interface. A green success message at the top states "The test event testevent was successfully saved." Below it, the "Code source" tab is selected, showing the Python code for the lambda_function. The code is a simple test event named "testevent" with a null response. The "Test" tab is selected, showing the execution results. The test event name is "testevent". The response is null. The function logs show a successful execution with a Request ID of "c2ed8223-fc76-4178-ac23-5920bf190bb6". The status is "Succeeded" with a max memory used of 65 MB and a duration of 205.54 ms.

```
Execution result: source.json
Test Event Name: testevent
Response: null
Function Logs
START RequestId: c2ed8223-fc76-4178-ac23-5920bf190bb6 Version: $LATEST
('ShardId': 'shardId-000000000000', 'SequenceNumber': '4963907231391668886295311077094949262928108690371248130', 'ResponseMetadata': {'RequestId': 'd38a9bc1-f4f1-ae27-8f18-661a43e33333', 'HTTPStatusCode': 200, 'HTTPHeaders': {'Content-Type': 'application/json', 'Content-Length': '113', 'Date': 'Tue, 10 Jul 2018 10:45:20 UTC', 'X-Amzn-RequestID': 'c2ed8223-fc76-4178-ac23-5920bf190bb6'}, 'RetryAttempts': 0}, 'END RequestId: c2ed8223-fc76-4178-ac23-5920bf190bb6 Duration: 205.54 ms Billed Duration: 206 ms Memory Size: 128 MB Max Memory Used: 65 MB
Request ID: c2ed8223-fc76-4178-ac23-5920bf190bb6
```

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

The screenshot shows the AWS CloudWatch service dashboard. On the left, the navigation pane is expanded to show 'Logs' under 'CloudWatch'. Under 'Logs', 'Log groups' is selected, and 'aws/lambda/kinesis_datastream_consumer' is chosen. The main area displays 'Log events' for the date '2023/03/21'. A single log event is visible:

```
2023-03-20T20:34:36.019-04:00
INIT_START Runtime Version: python:3.9.v17 Runtime Version ARN: arn:aws:lambda:us-east-2::runtime:3e0b3830c8f6c1f1dc330174bd86f065c014...
2023-03-20T20:34:36.256-04:00
START RequestId: cc07b5a4-2fdd-4d5c-9208-0dfe6aa6afab Version: $LATEST
("Records": [{"kinesis": {"kinesisSchemaVersion": "1.0", "partitionKey": "123", "sequenceNumber": "4963907231391668886295331107709494926..."}, "class": "dict"}, {"id": "1", "name": "John", "age": "30"}])
2023-03-20T20:34:36.256-04:00
1 John 30
2023-03-20T20:34:37.754-04:00
END RequestId: cc07b5a4-2fdd-4d5c-9208-0dfe6aa6afab Duration: 1498.40 ms Billed Duration: 1499 ms Memory Size: 128 MB Max Memory Used...
2023-03-20T20:34:37.754-04:00
REPORT RequestId: cc07b5a4-2fdd-4d5c-9208-0dfe6aa6afab Duration: 1498.40 ms Billed Duration: 1499 ms Memory Size: 128 MB Max Memory Used...
No newer events at this moment. Auto retry paused. Resume
```

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

The screenshot shows the AWS DynamoDB service dashboard. On the left, the navigation pane is expanded to show 'Tables' under 'DynamoDB'. A table named 'kinesis_datastream_customer_demo' is selected. The main area shows the results of a 'Scan or query items' operation:

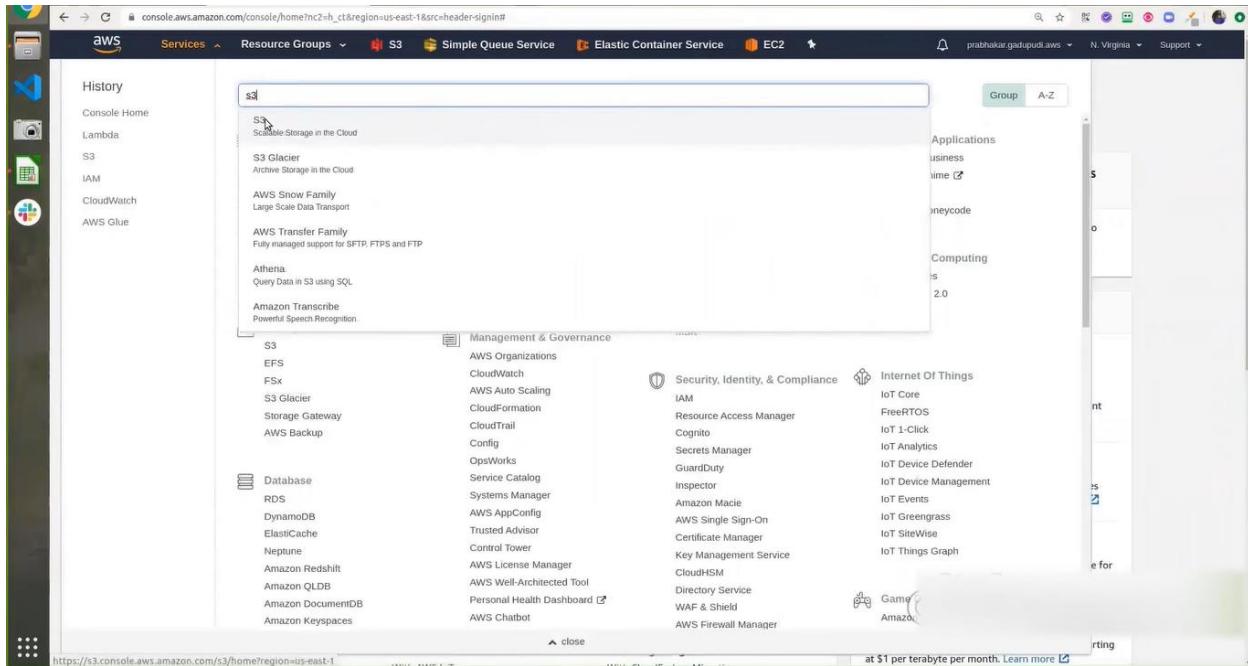
Completed. Read capacity units consumed: 0.5

id	age	name
1	30	John

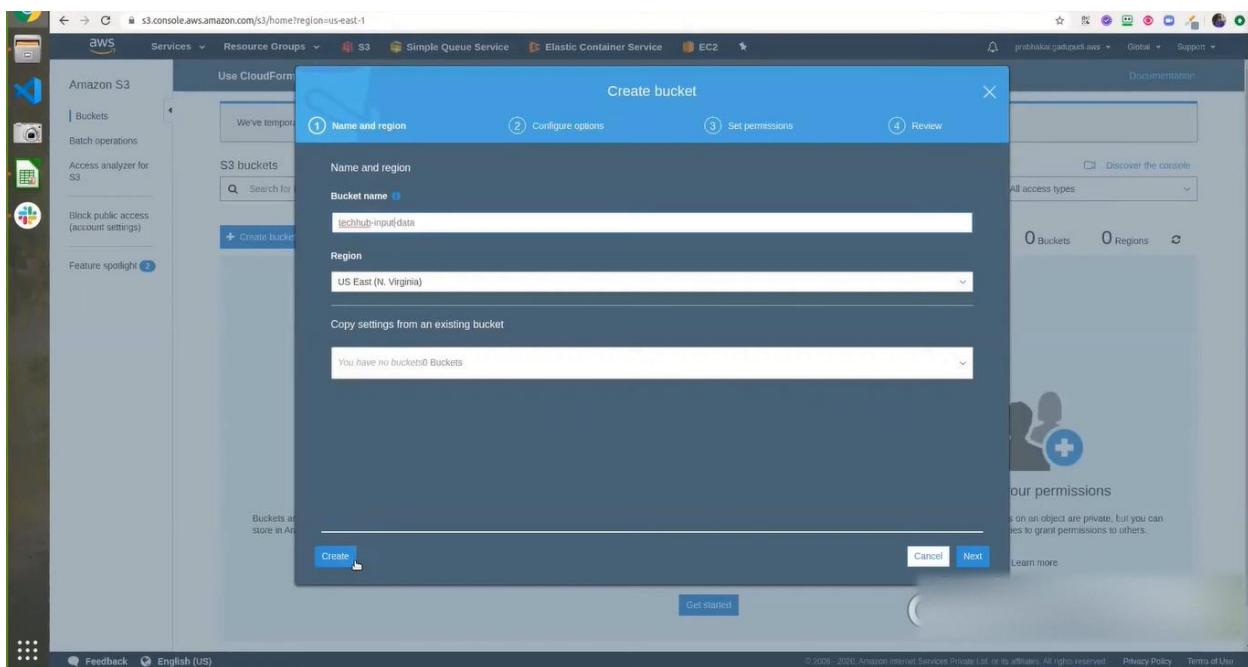
COPY FILES FROM ONE S3 BUCKET TO ANOTHER S3 BUCKET

INTEGRATING LAMBDA WITH S3

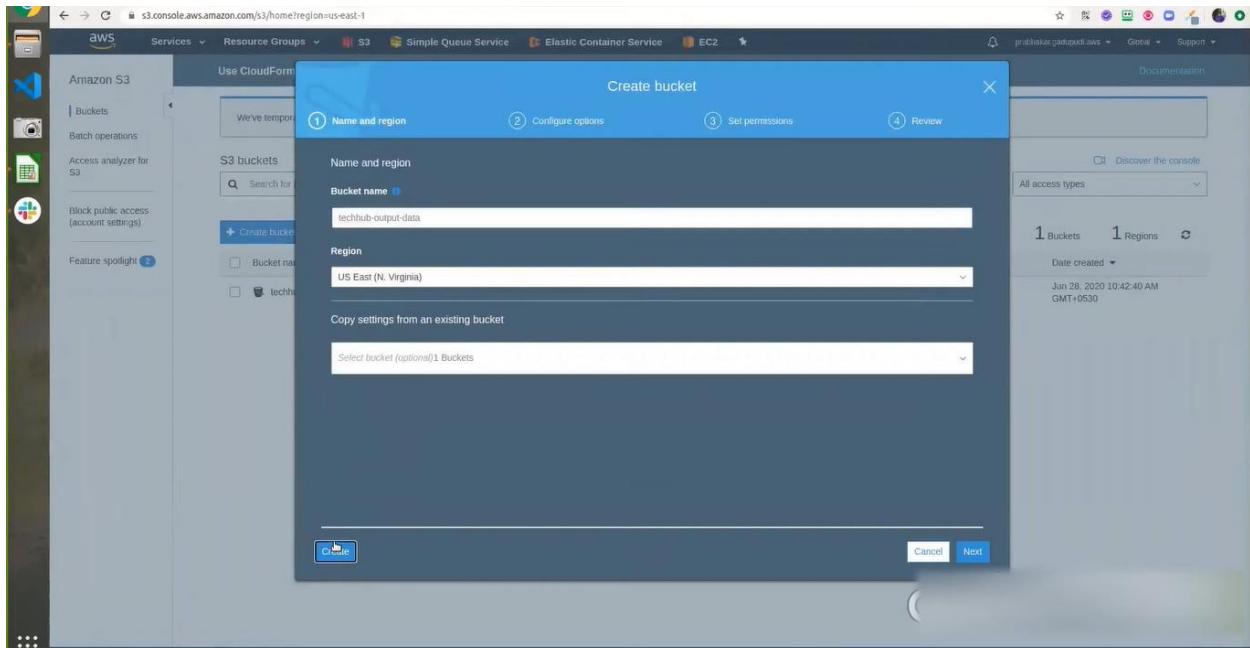
STEP 1: Choose S3 service.



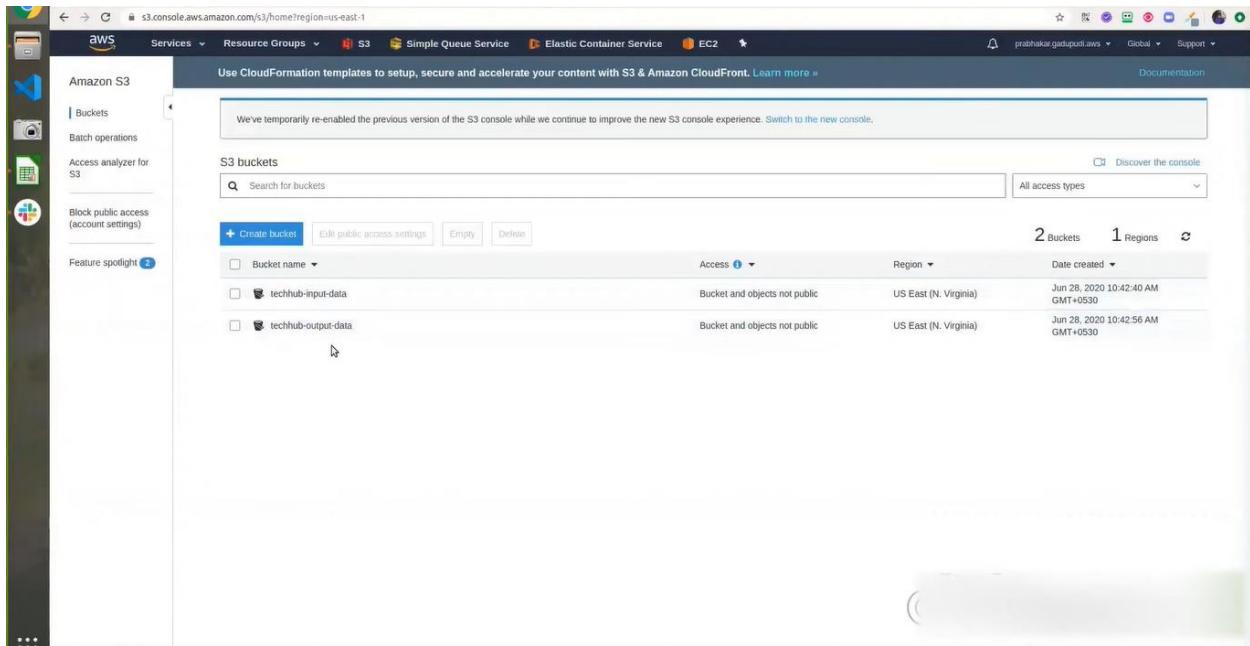
STEP 2: Create first bucket as “input-data”.



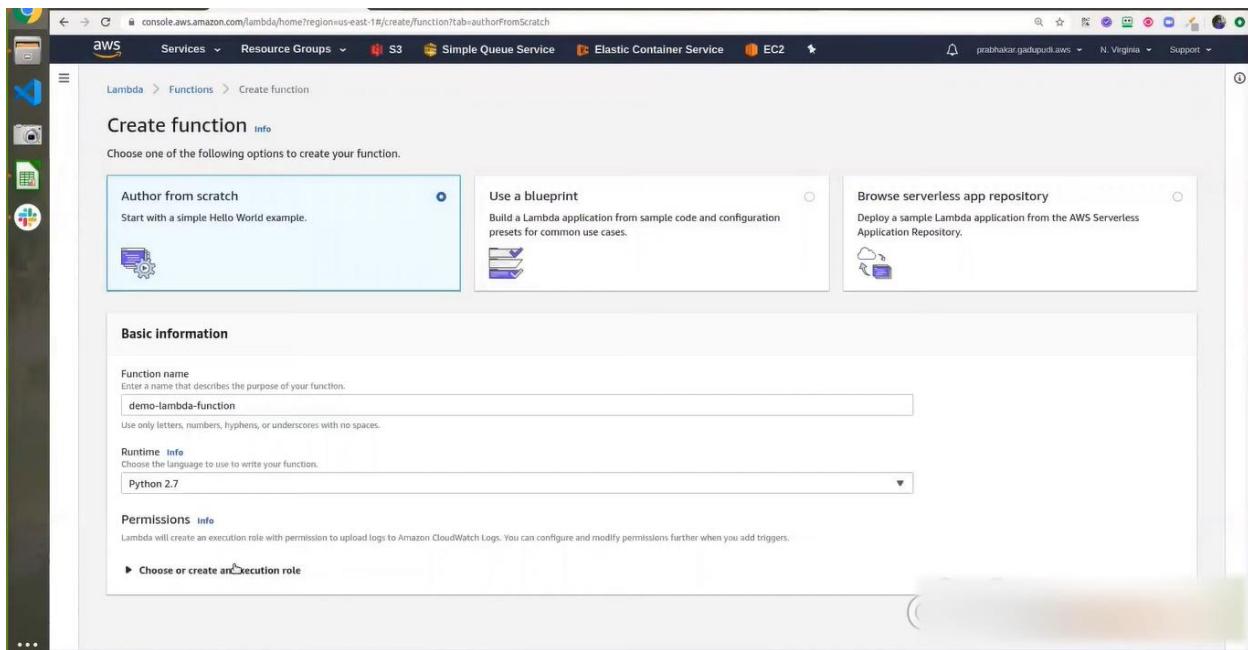
STEP 3: Create another bucket as “output-data”.



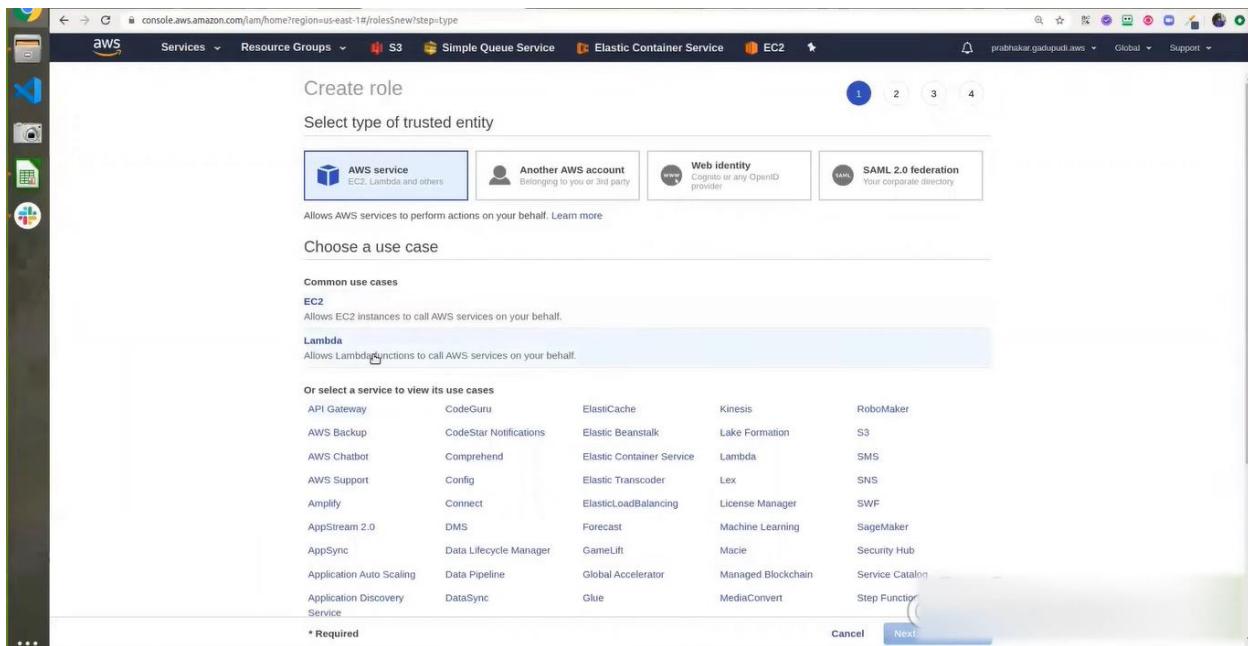
STEP 4: Two buckets created successfully.



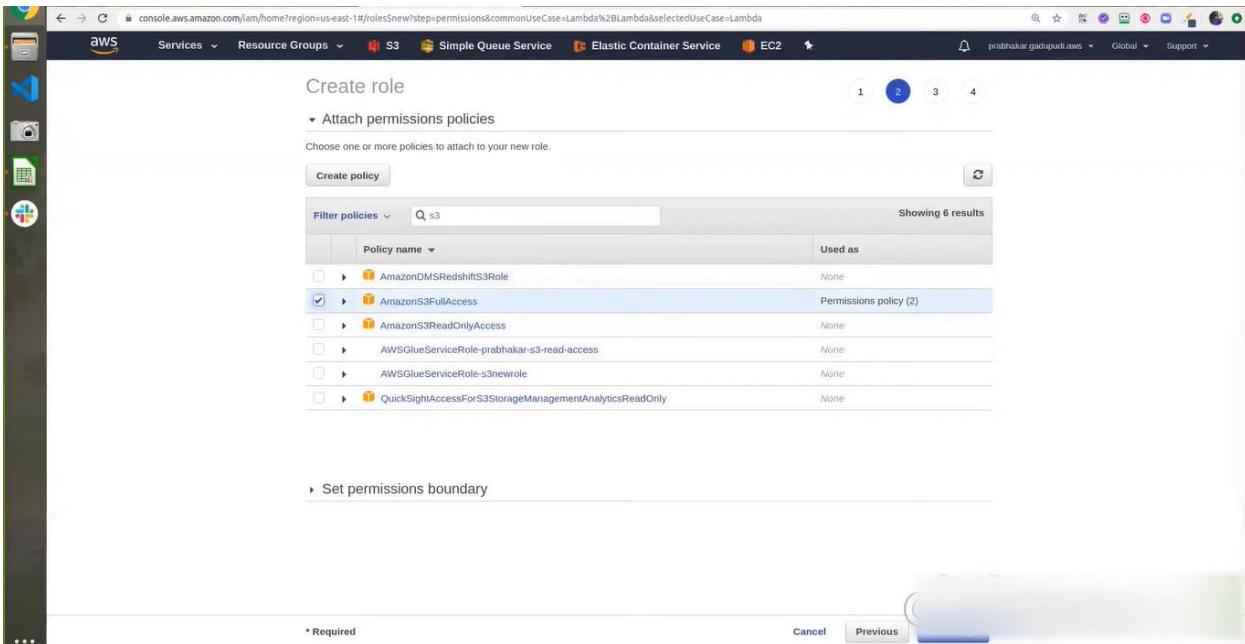
STEP 5: Create lambda function as “demo-lambda-function”.



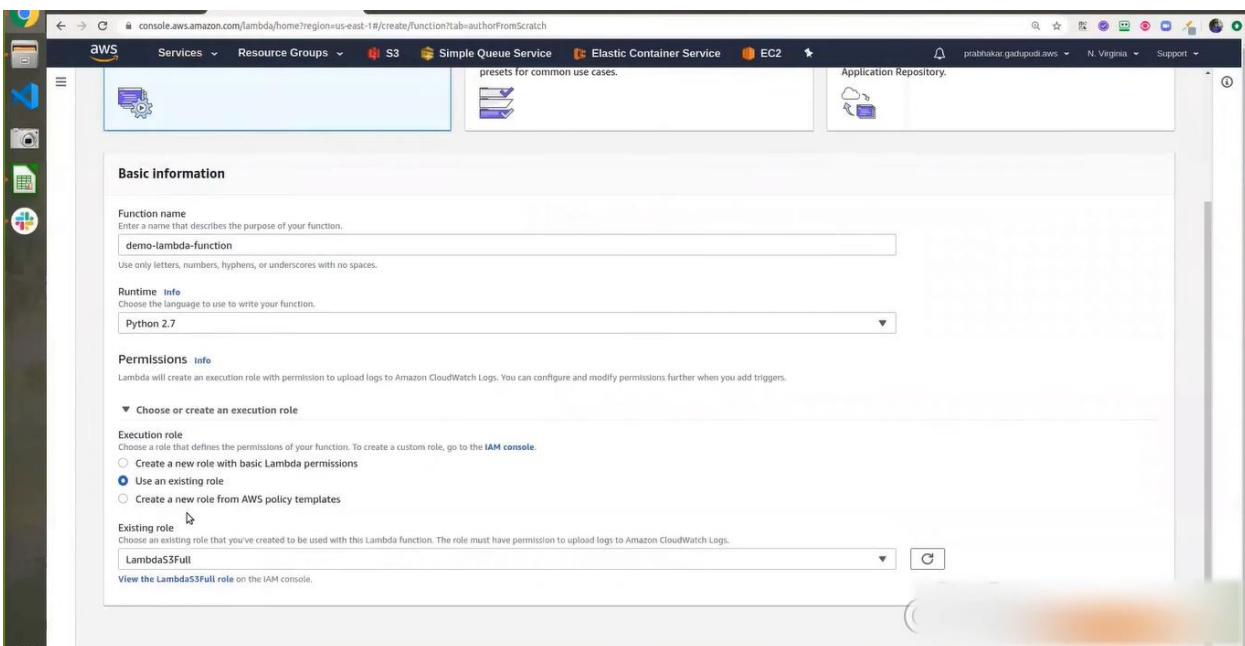
STEP 6: Choose role for lambda function.



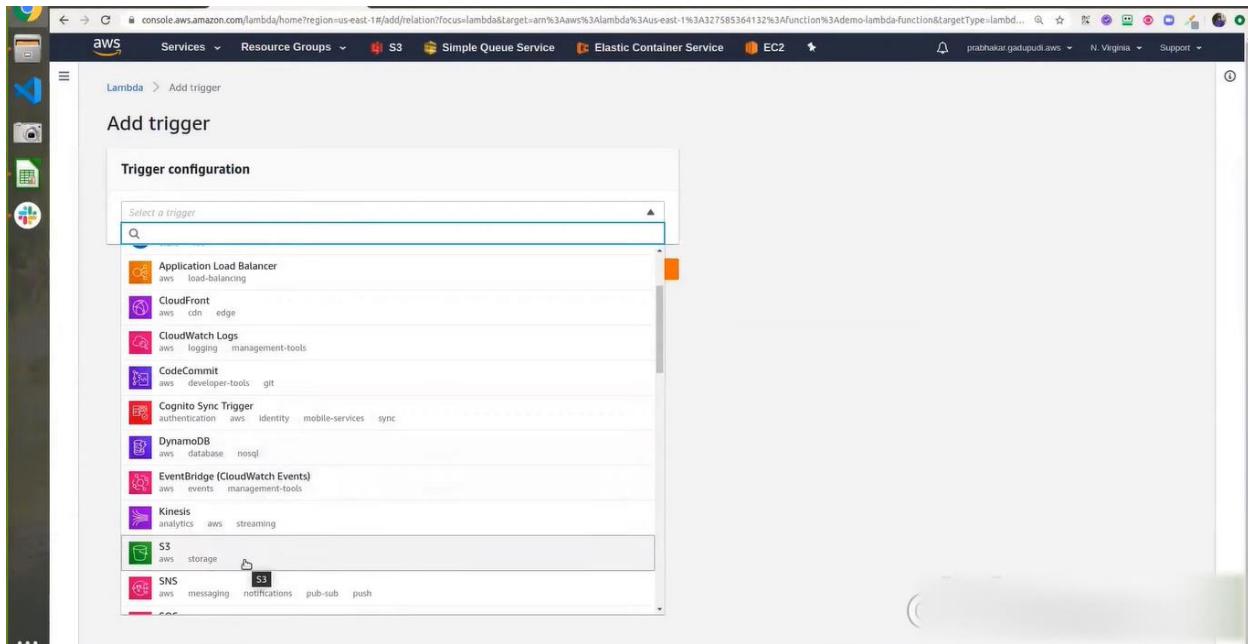
STEP 7: Choose “AmazonS3FullAccess” and create the role.



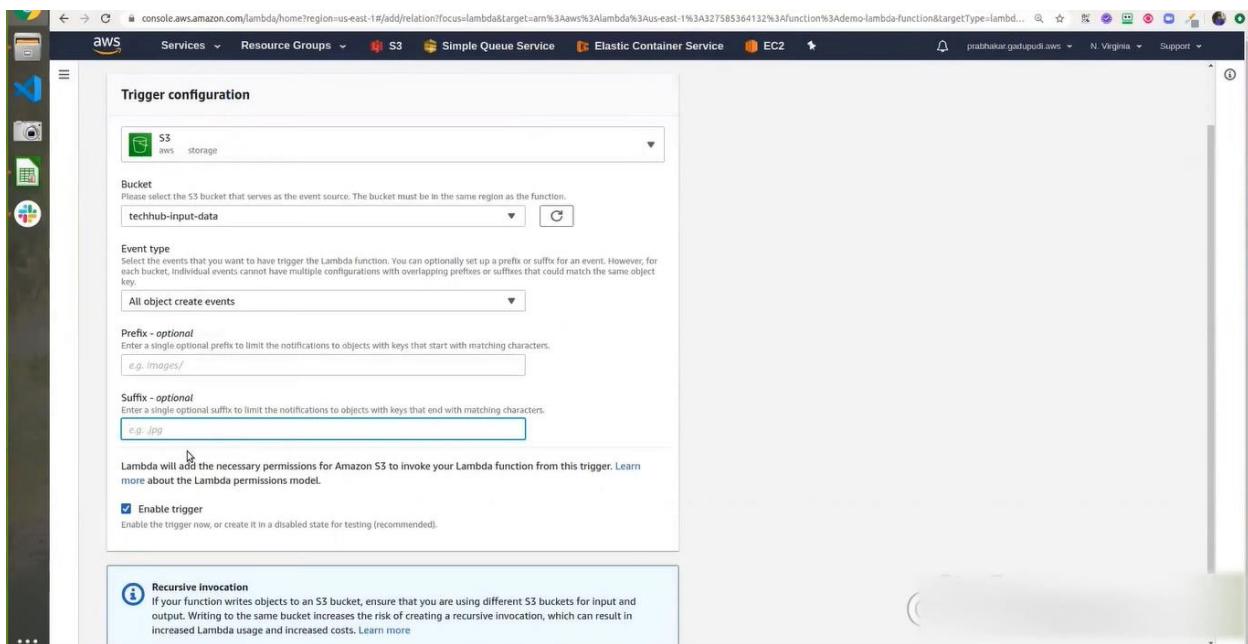
STEP 8: Enter the required details for creating lambda function and click create function.



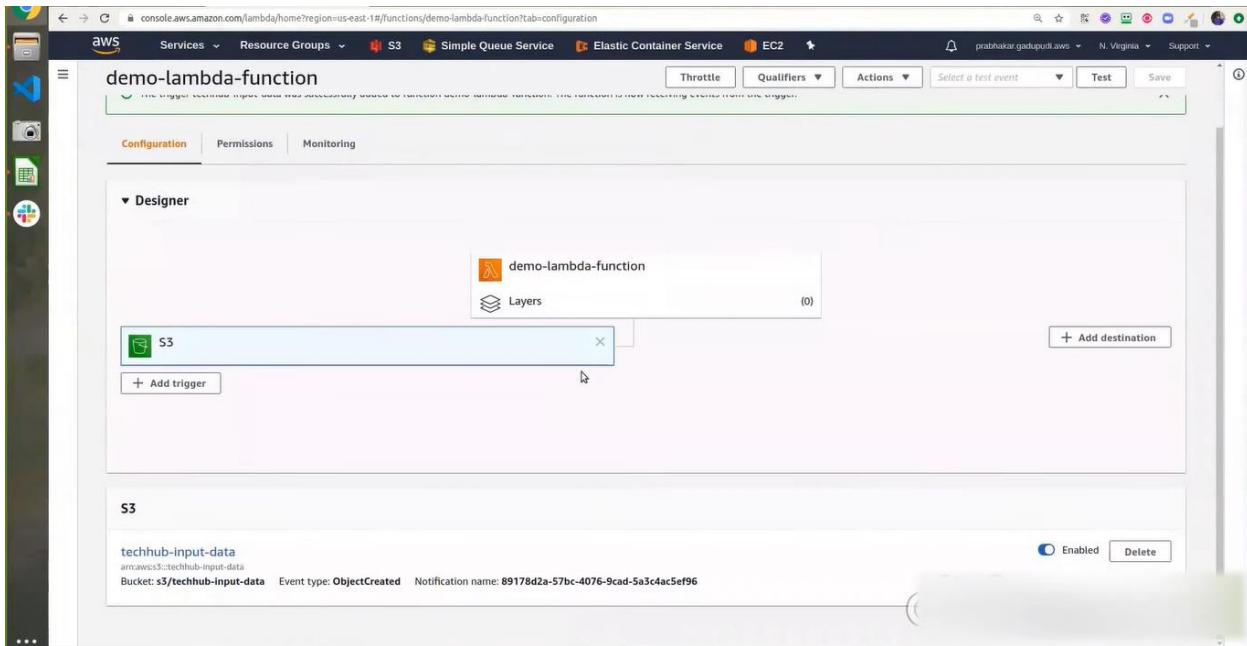
STEP 9: After creating lambda function, click “Add trigger” and choose S3.



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.

```
13
14 def lambda_handler(event, context):
15     # TODO implement
16     source_bucket = event['Records'][0]['s3']['bucket']['name']
17     object_key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key'])
18     target_bucket = 'techhub-output-data'
19     copy_source = {'Bucket': source_bucket, 'Key': object_key}
20     print ("Source bucket : ", source_bucket)
21     print ("Target bucket : ", target_bucket)
22     print ("Log Stream name: ", context.log_stream_name)
23     print ("Log Group name: ", context.log_group_name)
24     print ("Request ID: ", context.aws_request_id)
25     print ("Mem. limits(MB): ", context.memory_limit_in_mb)
26     try:
27         print ("Using waiter to waiting for object to persist through s3 service")
28         waiter = s3.get_waiter('object_exists')
29         waiter.wait(Bucket=source_bucket, Key=object_key)
30         s3.copy_object(Bucket=target_bucket, Key=object_key, CopySource=copy_source)
31         return response['ContentType']
32     except Exception as err:
33         print ("Error - "+str(err))
34     return e
```

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

The screenshot shows two side-by-side browser windows. The left window displays the 'techhub-input-data' bucket in the 'Amazon S3' service. It shows one object named 'employees.csv' with a size of 119.0 B and a last modified date of Jun 28, 2020 1:53:27 PM GMT+0530. The right window displays the 'techhub-output-data' bucket, which is currently empty. It includes instructions for uploading objects, setting properties, and managing permissions.

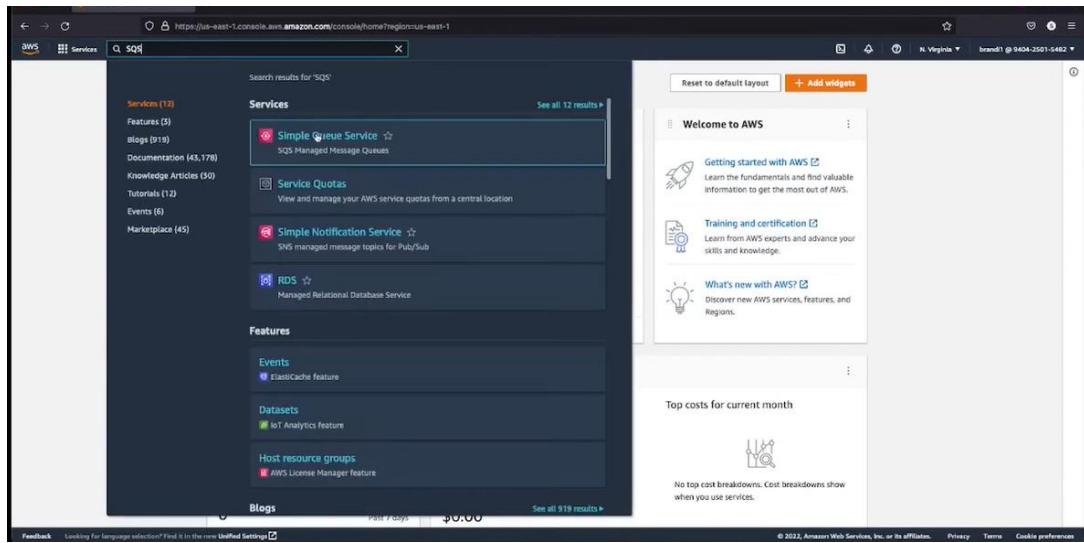
STEP 14: Open the output-data bucket and refresh. A copy of the same file is available in this bucket.

This screenshot shows the same two buckets after a refresh. Both the 'techhub-input-data' and 'techhub-output-data' buckets now contain the 'employees.csv' file, indicating that the Lambda function has successfully copied the file from the input bucket to the output 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.

A screenshot of the 'Create queue' wizard in the AWS Management Console. The URL in the browser is 'https://us-east-1.console.aws.amazon.com/sqsv2/home?region=us-east-1#/create-queue'. The page title is 'Amazon SQS > Queues > Create queue'. The first step is 'Create queue' under 'Details'. It asks to choose a 'Type' between 'Standard' and 'FIFO'. 'Standard' is selected, with a note: 'You can't change the queue type after you create a queue.' 'FIFO' is also shown with a note: 'First-in-first-out delivery, message ordering is preserved'. The next step is 'Name', where the user has entered 'MyQueue'. Below that is 'Configuration' with sections for 'Visibility timeout info' (set to 30 seconds) and 'Message retention period info' (set to 4 days). At the bottom, there are 'Delivery delay info' and 'Maximum message size info' sections.

STEP 3: Choose the configuration details.

Name: demo-queue1

Configuration

visibility timeout info: 30 Seconds

Message retention period info: 4 Days

Delivery delay info: 0 Seconds

Maximum message size info: 256 KB

Receive message wait time info: 0 Seconds

Encryption

Amazon SQS provides in-transit encryption by default. To add at-rest encryption to your queue, enable server-side encryption. Info

Server-side encryption:

- Disabled
- Enabled

STEP 4: Click create queue.

Basic

Use simple criteria to define a basic access policy.

Advanced

Use a JSON object to define an advanced access policy.

JSON (read-only)

```
{ "Version": "2008-10-17", "Id": "..._default_policy_ID", "Statement": [ { "Sid": "..._owner_statement", "Effect": "Allow", "Principal": { "AWS": "arn:aws:iam::123456789012:root" }, "Action": [ "SQS:SendMessage", "SQS:ReceiveMessage" ], "Resource": "arn:aws:sqs:us-east-1:123456789012:demo-queue1" } ] }
```

Redrive allow policy - Optional

Identify which source queues can use this queue as the dead-letter queue. Info

Dead-letter queue - Optional

Send undeliverable messages to a dead-letter queue. Info

Tags - Optional

A tag is a label assigned to an AWS resource. Use tags to search and filter your resources or track your AWS costs. Learn more ▾

Create queue

STEP 5: SQS queue created successfully.

The screenshot shows the AWS SQS Queue details page for 'demo-queue1'. At the top, a green banner says 'Queue demo-queue1 created successfully. You can now send and receive messages.' Below the banner, the queue name is 'demo-queue1'. The 'Details' tab is selected, showing the following information:

Name	Type	ARN
demo-queue1	Standard	arn:aws:sqs:us-east-1:940425015482:demo-queue1
Encryption	Amazon SQS key (SSE-SQS)	URL
		https://sqs.us-east-1.amazonaws.com/940425015482/demo-queue1

Below the details, there are tabs for 'SNS subscriptions', 'Lambda triggers', 'Dead-letter queue', 'Monitoring', 'Tagging', 'Access policy', 'Encryption', and 'Dead-letter queue redrive tasks'. Under 'SNS subscriptions', it says '(0)' and has a 'Subscribe to Amazon SNS topic' button. The 'Subscription region' dropdown is set to 'us-east-1'. There is also a 'Search subscribers' input field and a 'Subscription ARN' dropdown.

STEP 6: Create lambda function with necessary details.

The screenshot shows the AWS Lambda 'Create function' wizard. The first step is 'Basic information'.

Function name: demo-function

Runtime: Node.js 16.x

Architecture: x86_64

Permissions:

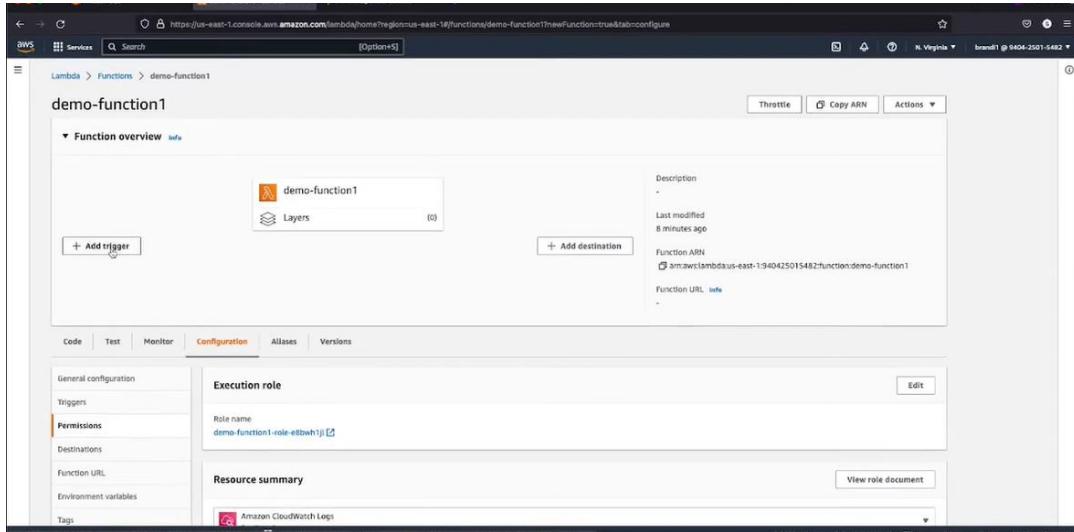
- Change default execution role
- Execution role:
 - Create a new role with basic Lambda permissions (selected)
 - Use an existing role
 - Create a new role from AWS policy templates

A note says: 'Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.'

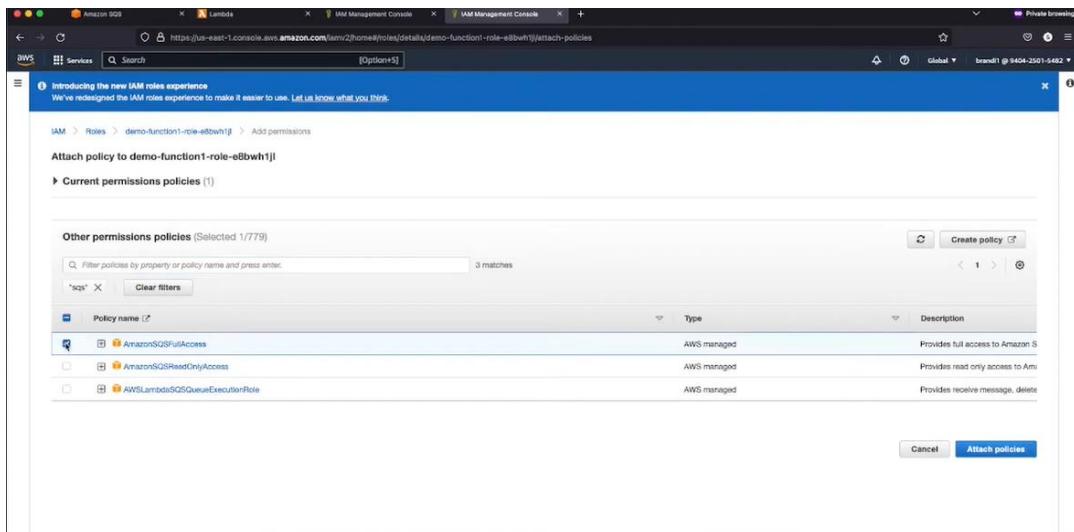
Lambda will create an execution role named 'demo-function1-role-edbwh1j', with permission to upload logs to Amazon CloudWatch Logs.

Advanced settings is collapsed at the bottom.

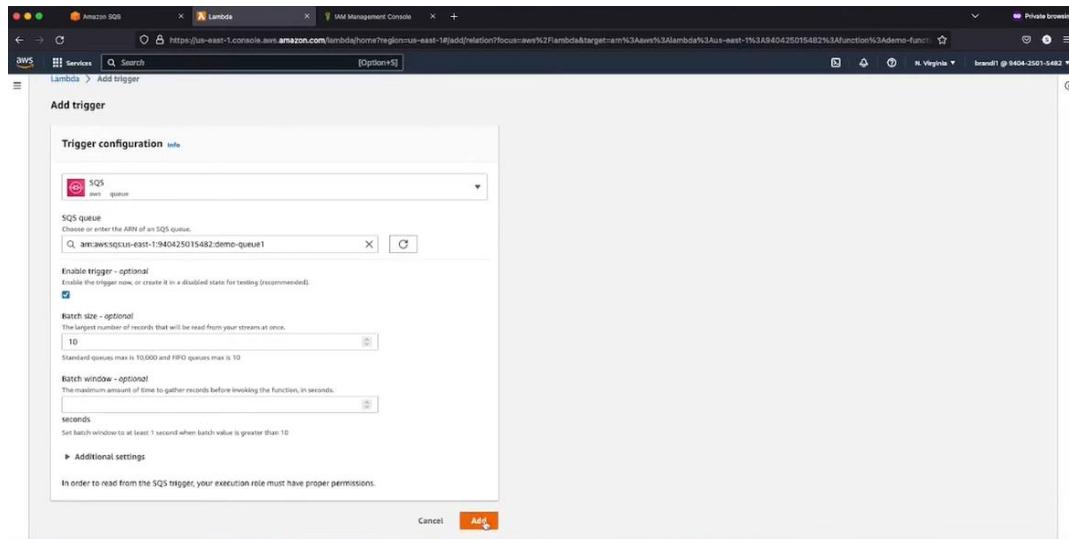
STEP 7: Click “Add trigger”.



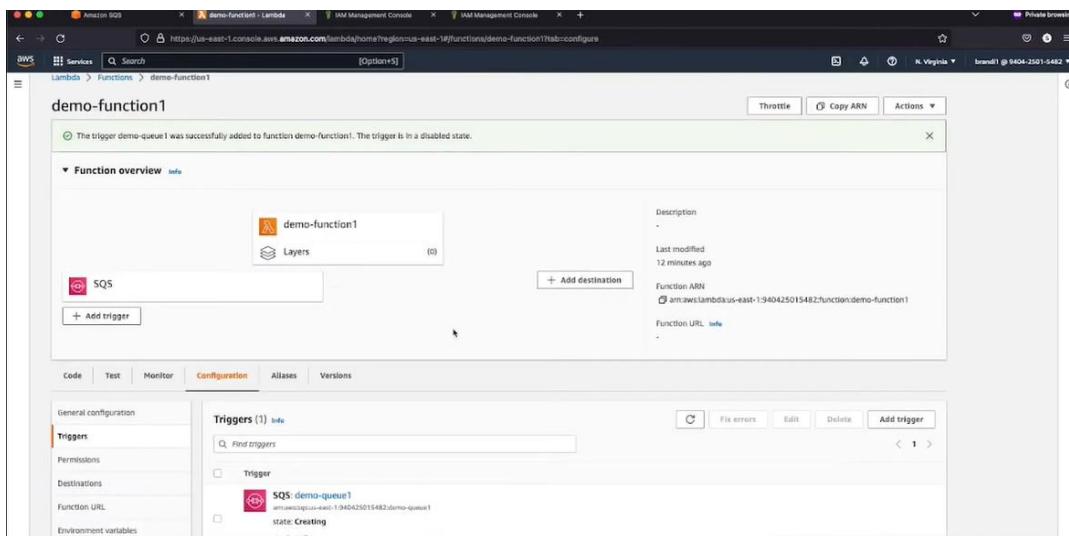
STEP 8: Add a permission in IAM role for SQS – AmazonSQSFullAccess



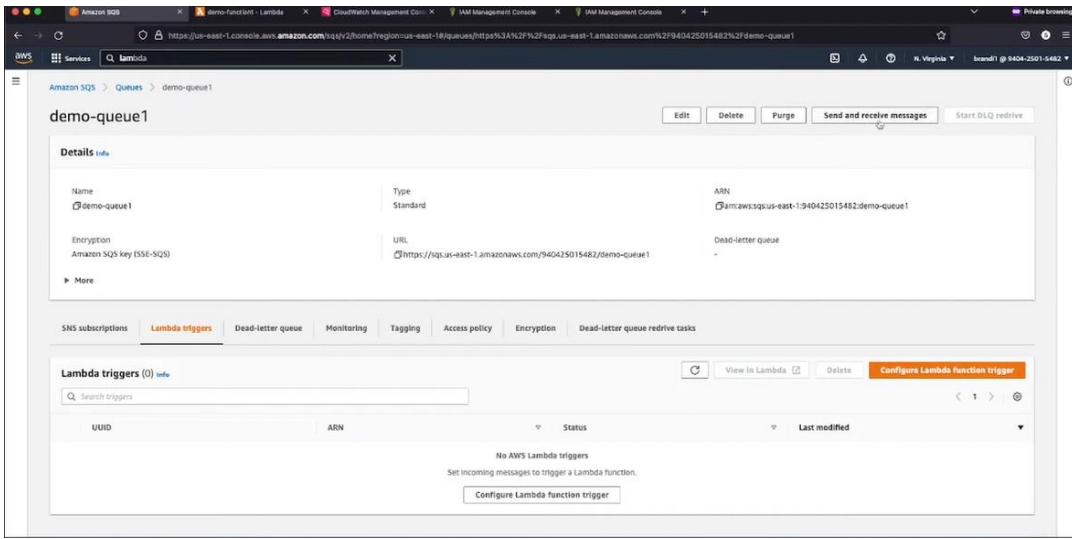
STEP 9: Choose SQS and enter necessary details and click “Add”.



STEP 10: Trigger added successfully.

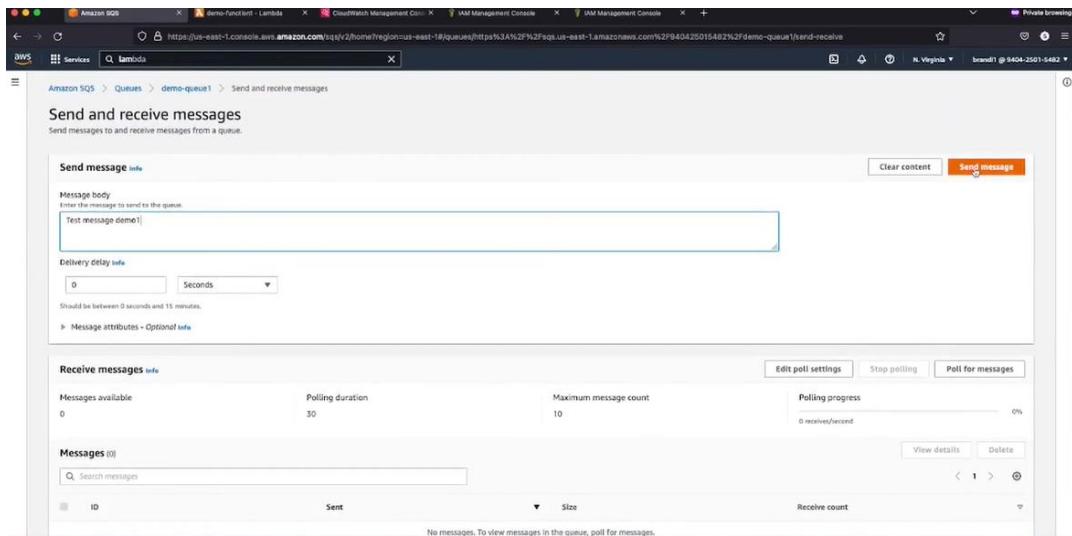


STEP 11: Open SQS service and click “send and receive messages”.



The screenshot shows the AWS Lambda triggers section for the queue 'demo-queue1'. At the top, there are tabs for SNS subscriptions, Lambda triggers (which is selected), Dead-letter queue, Monitoring, Tagging, Access policy, Encryption, and Dead-letter queue redrive tasks. Under the Lambda triggers tab, there is a search bar labeled 'Search triggers' and a table with columns for UUID, ARN, Status, and Last modified. A button labeled 'Configure Lambda function trigger' is located at the bottom of this section.

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



The screenshot shows the 'Send and receive messages' interface for the queue 'demo-queue1'. It has two main sections: 'Send message' and 'Receive messages'. In the 'Send message' section, there is a text area labeled 'Message body' containing the text 'Test message demo!'. Below it is a 'Delivery delay info' section with a dropdown set to 'Seconds'. In the 'Receive messages' section, there is a table with columns for ID, Sent, Size, and Receive count. A note at the bottom says 'No messages. To view messages in the queue, poll for messages.'

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

The screenshot shows the AWS CloudWatch Log Groups interface. The left sidebar navigation includes CloudWatch, Logs, Log groups, Metrics, X-Ray traces, Events, Application monitoring, and Insights. The main content area displays log entries for the 'demo-function' Lambda function. The logs show the following sequence of events:

- 2022-11-09T14:30:49,626+11:00: START RequestId: 5e544fc7-3ff9-4ec1-a047-344566c490de Version: \$LATEST
- 2022-11-09T14:30:49,626+11:00: END RequestId: 5e544fc7-3ff9-4ec1-a047-344566c490de Version: \$LATEST
- 2022-11-09T14:30:49,660+11:00: INFO RequestId: 5e544fc7-3ff9-4ec1-a047-344566c490de INFO event { key1: 'test1', key2: 'test2' }
- 2022-11-09T14:30:49,660+11:00: REPORT RequestId: 5e544fc7-3ff9-4ec1-a047-344566c490de Duration: 37.17 ms Billed Duration: 38 ms Memory Size: 128 MB Max Memory Used: 57 MB Init Duration: 145.94 ms
- 2022-11-09T14:31:53,480+11:00: START RequestId: c11790b2-fc58-5b67-396d-271dbb Version: \$LATEST
- 2022-11-09T14:31:53,480+11:00: END RequestId: c11790b2-fc58-5b67-396d-271dbb Version: \$LATEST
- 2022-11-09T14:32:53,482+11:00: REPORT RequestId: c11790b2-fc58-5b67-396d-271dbb Duration: 2.24 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 57 MB Init Duration: 145.94 ms

No newer events at this moment. Auto retry paused.

STEP 14: Next, choose “DynamoDB” service.

The screenshot shows the AWS Lambda service interface. The left sidebar navigation includes Services, Features, Blogs, Documentation, Tutorials, and Events. The main content area displays search results for 'dynamodb'. The 'DynamnoDB' service is highlighted in the search results list. The right panel shows detailed information for the 'demo-function' Lambda function, including its ARN, last modified time (20 minutes ago), and function URL.

DynamnoDB

Description

Last modified

Function ARN

Function URL

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

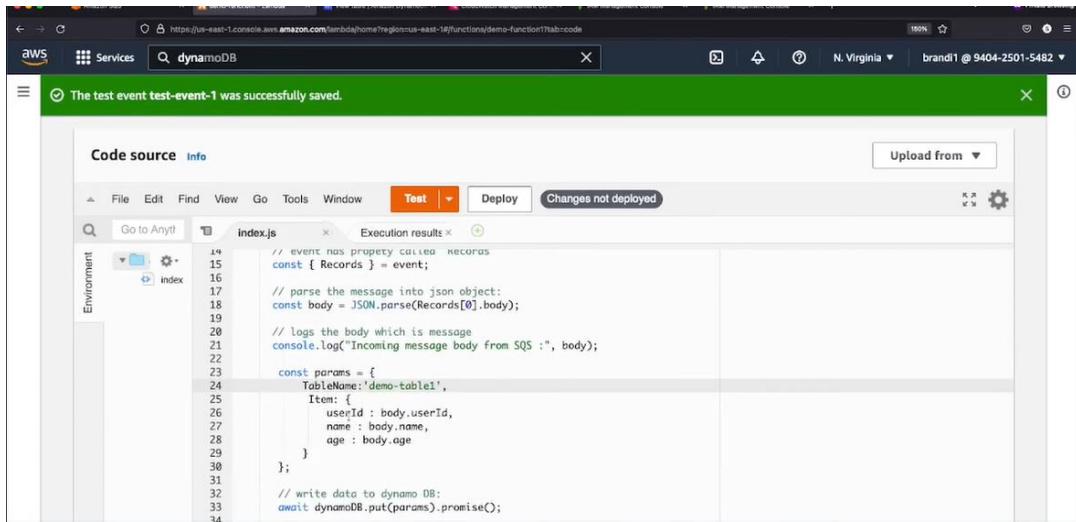
The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. In the 'Table details' section, the table name is set to 'demo-table1'. The 'Partition key' is defined as 'userid' of type 'String'. In the 'Table settings' section, the 'Default settings' radio button is selected. The table currently has two items: 'demo-table1' and 'test-dynamo-table1'.

STEP 16: Table created successfully.

The screenshot shows the 'Tables' page in the AWS DynamoDB console. It lists two tables: 'demo-table1' and 'test-dynamo-table1'. Both tables are active and have a partition key of 'userid'. The table 'demo-table1' was created in step 16.

Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode	Size	Table class
demo-table1	Active	userid (\$)	-	0	Provisioned with auto scaling (5)	Provisioned with auto scaling (5)	0 bytes	DynamoDB Standard
test-dynamo-table1	Active	userid (\$)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	0 bytes	DynamoDB Standard

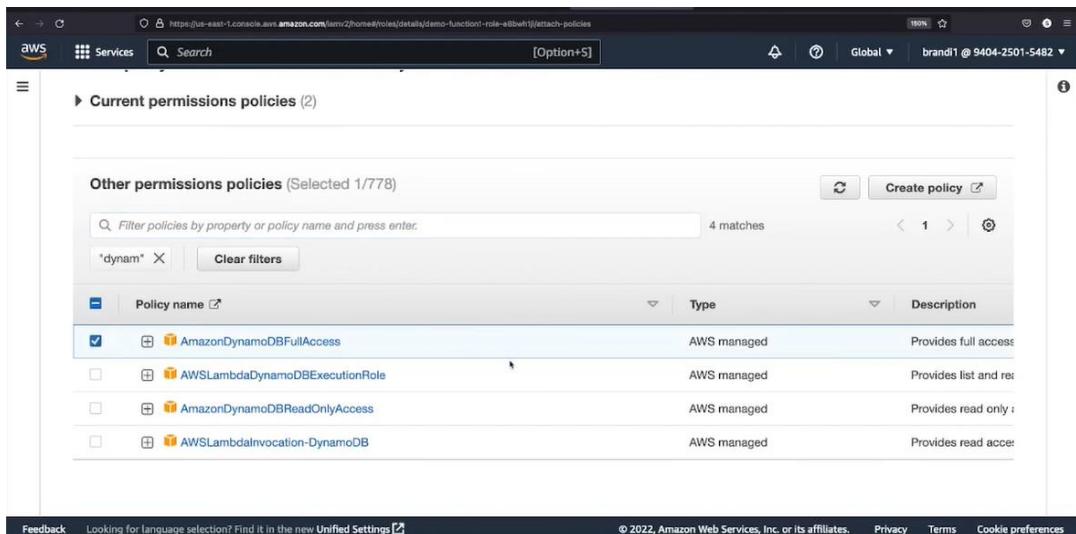
STEP 17: Write a code to store that message in Dynamo DB.



The screenshot shows the AWS Lambda function editor. The code source tab is selected, displaying the `index.js` file. The code is as follows:

```
// event has property called 'Records'  
const { Records } = event;  
  
// parse the message into json object:  
const body = JSON.parse(Records[0].body);  
  
// logs the body which is message  
console.log("Incoming message body from SQS :", body);  
  
const params = {  
    TableName: 'demo-table1',  
    Item: {  
        userId : body.userId,  
        name : body.name,  
        age : body.age  
    }  
};  
  
// write data to dynamo DB:  
await dynamoDB.put(params).promise();
```

STEP 18: Enable permission for DynamoDB – AmazonDynamoDBFullAccess.



The screenshot shows the AWS IAM role permissions page. It displays the current permissions policies and other permissions policies. The `"dynam"` filter is applied to the list of policies.

Policy name	Type	Description
<input checked="" type="checkbox"/>  AmazonDynamoDBFullAccess	AWS managed	Provides full access
<input type="checkbox"/>  AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and re
<input type="checkbox"/>  AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only :
<input type="checkbox"/>  AWSLambdaInvocation-DynamoDB	AWS managed	Provides read acce

STEP 19: Open SQS service and send a message.

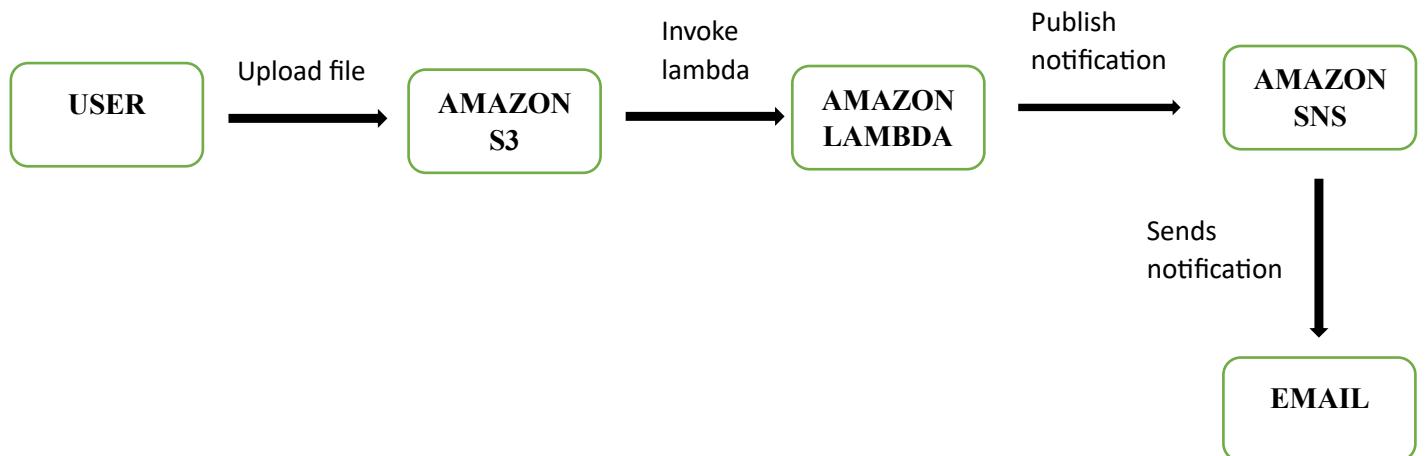
The screenshot shows the AWS SQS 'Send and receive messages' interface. In the 'Send message' section, a JSON message body is entered: `{"userId": "123", "name": "lovetocode", "age": "22"}`. The 'Delivery delay' is set to 0 seconds. Below the message body, there is a link to 'Message attributes - Optional Info'. At the bottom of the page, there are buttons for 'Edit poll settings', 'Stop polling', and 'Poll for messages'.

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

The screenshot shows the AWS DynamoDB 'Items' view for the 'demo-table1' table. The table contains two items: 'demo-table1' and 'test-dynamo-table1'. The 'demo-table1' item has attributes: 'userId' (124), 'age' (30), and 'name' (lovetocode). The 'Completed' status indicates that 0.5 read capacity units were consumed.

AUTOMATE SENDING NOTIFICATIONS WITH USING AMAZON SNS AND AWS LAMBDA

WORKFLOW DIAGRAM



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

The screenshot shows the AWS Amazon S3 service console. The left sidebar includes options like Buckets, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings, Storage Lens, Dashboards, and AWS Organizations settings. The main area displays an **Account snapshot** and a table of existing buckets:

Name	AWS Region	Access	Creation date
tnhb-xml-extraction	Asia Pacific (Mumbai) ap-south-1	Bucket and objects not public	August 17, 2023, 12:55:00 (UTC+05:30)
cf-templates-1xipvxbllfrpn-ap-south-1	Asia Pacific (Mumbai) ap-south-1	Bucket and objects not public	August 9, 2023, 13:27:38 (UTC+05:30)
tnb-aws-glue-practise	Asia Pacific (Mumbai) ap-south-1	Bucket and objects not public	July 14, 2023, 11:28:39 (UTC+05:30)
sagemaker-us-east-1-195737202275	US East (N. Virginia) us-east-1	Insufficient permissions	July 12, 2023, 14:21:27 (UTC+05:30)

A prominent orange **Create bucket** button is located at the top right of the bucket list area.

STEP 2: Fill bucket name and other information.

The screenshot shows the 'Create bucket' wizard in the AWS S3 console. The 'General configuration' step is active. A 'Bucket name' input field contains 'ec2automated'. Below it, a note states: 'Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming.' An 'AWS Region' dropdown is set to 'Asia Pacific (Mumbai) ap-south-1'. A 'Copy settings from existing bucket - optional' section includes a 'Choose bucket' button. At the bottom, there are 'Next Step' and 'Cancel' buttons.

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

The screenshot shows the 'Create bucket' wizard in the AWS S3 console. The 'Encryption type' step is active. It offers three options: 'Server-side encryption with Amazon S3 managed keys (SSE-S3)' (selected), 'Server-side encryption with AWS Key Management Service keys (SSE-KMS)', and 'Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)'. Below this, a 'Bucket Key' section notes that using an S3 Bucket Key for SSE-KMS reduces costs by lowering calls to AWS KMS. It shows 'Disable' (unchecked) and 'Enable' (checked). A 'Advanced settings' section is partially visible. At the bottom, a note says: 'After creating the bucket, you can upload files and folders to the bucket, and configure additional bucket settings.' There are 'Cancel' and 'Create bucket' buttons at the bottom right.

STEP 4: Bucket created successfully.

The screenshot shows the AWS S3 console at s3.console.aws.amazon.com/s3/buckets/ec2automates?region=us-east-1&tab=objects. The left sidebar has 'Buckets' selected. The main area shows the 'ec2automates' bucket with one object named 'automate.sh'. The object details are as follows:

Name	Type	Last modified	Size	Storage class
automate.sh	sh	September 18, 2022, 15:44:26 (UTC+05:30)	148.0 B	Standard

STEP 5: Create lambda function.

The screenshot shows the AWS Lambda console at us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions. The left sidebar has 'Functions' selected. The main area shows two functions: 'InvokeTexttract' and 'ResultTexttract'. The table details are as follows:

Function name	Description	Package type	Runtime	Last modified
InvokeTexttract	-	Zip	Python 3.9	9 days ago
ResultTexttract	-	Zip	Python 3.9	8 days ago

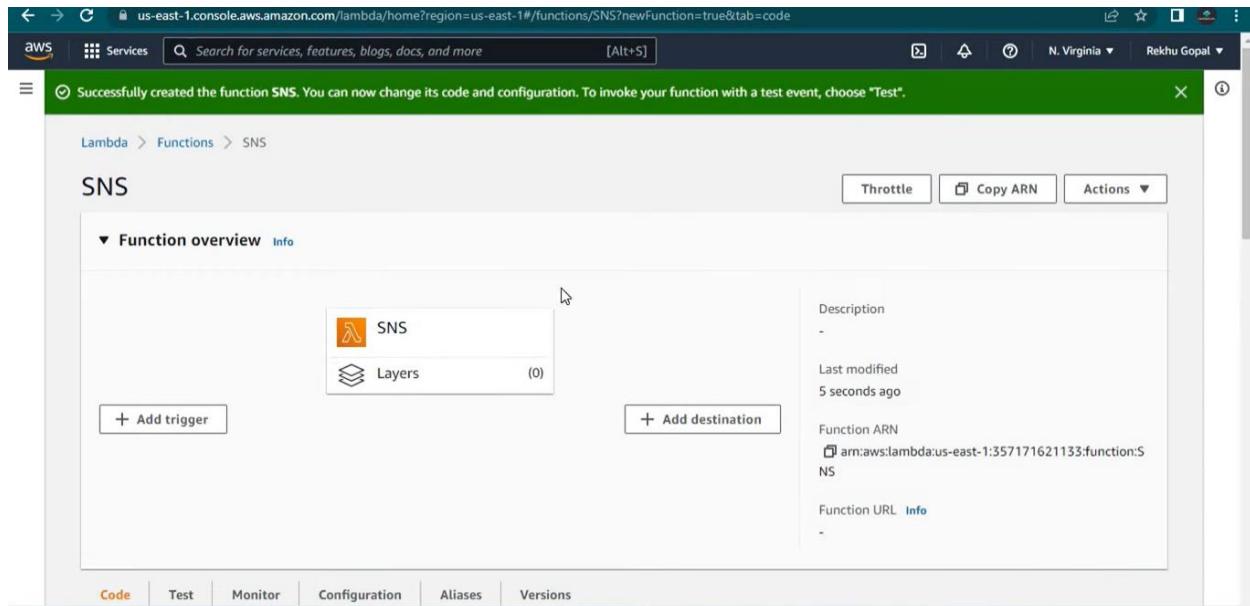
STEP 6: Fill function name, runtime, architecture, and permissions.

The screenshot shows the 'Basic information' section of the AWS Lambda 'Create function' wizard. It includes fields for 'Function name' (SNS), 'Runtime' (Python 3.9), 'Architecture' (x86_64 selected), and 'Permissions' (Change default execution role). A link to 'Advanced settings' is also visible.

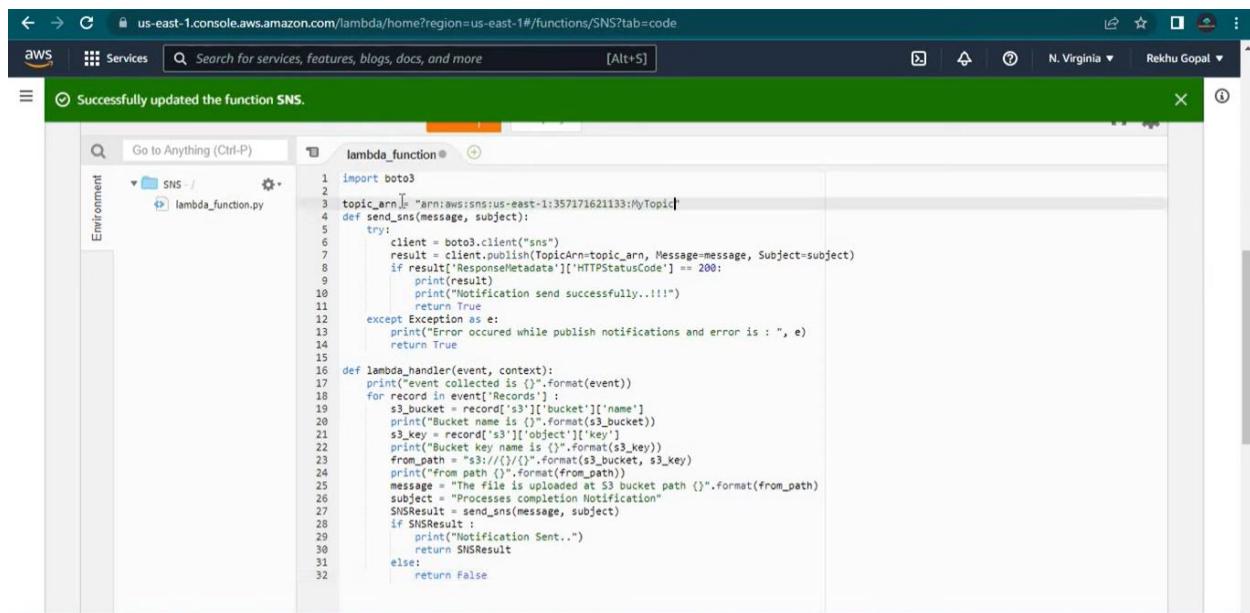
STEP 7: Click “Create function”.

The screenshot shows the 'Execution role' section of the AWS Lambda 'Create function' wizard. It allows selecting 'Create a new role with basic Lambda permissions' (selected), 'Use an existing role' (selected), or 'Create a new role from AWS policy templates'. An existing role named 'ETLLambdaAccessRole' is chosen. The 'Create function' button is highlighted at the bottom right.

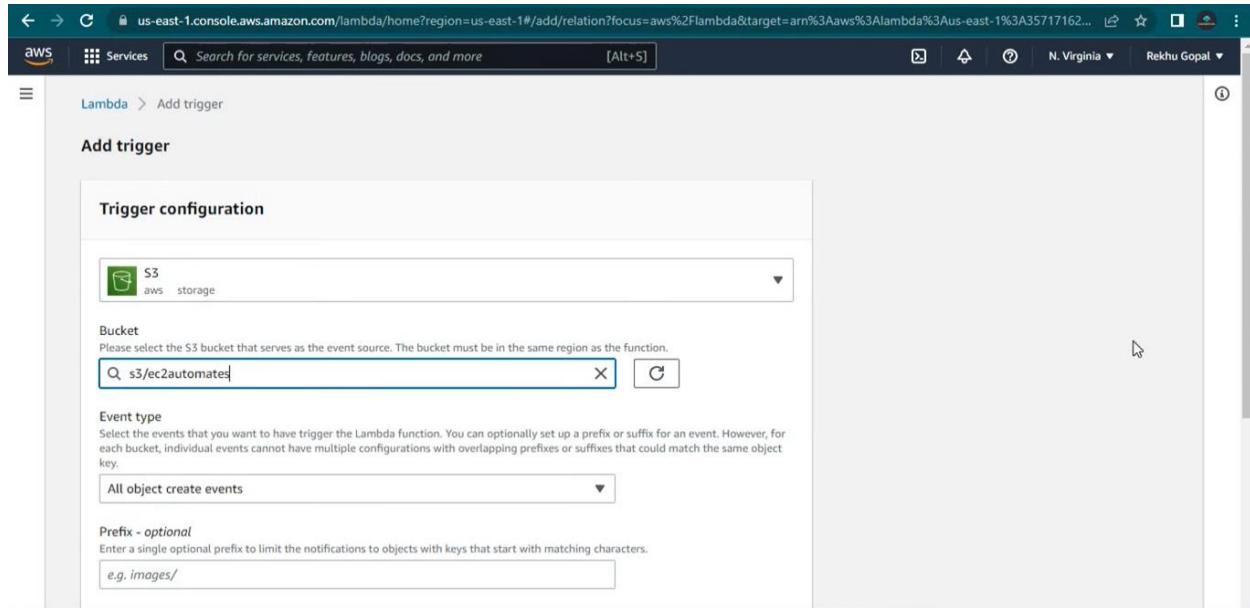
STEP 8: Lambda function created successfully.



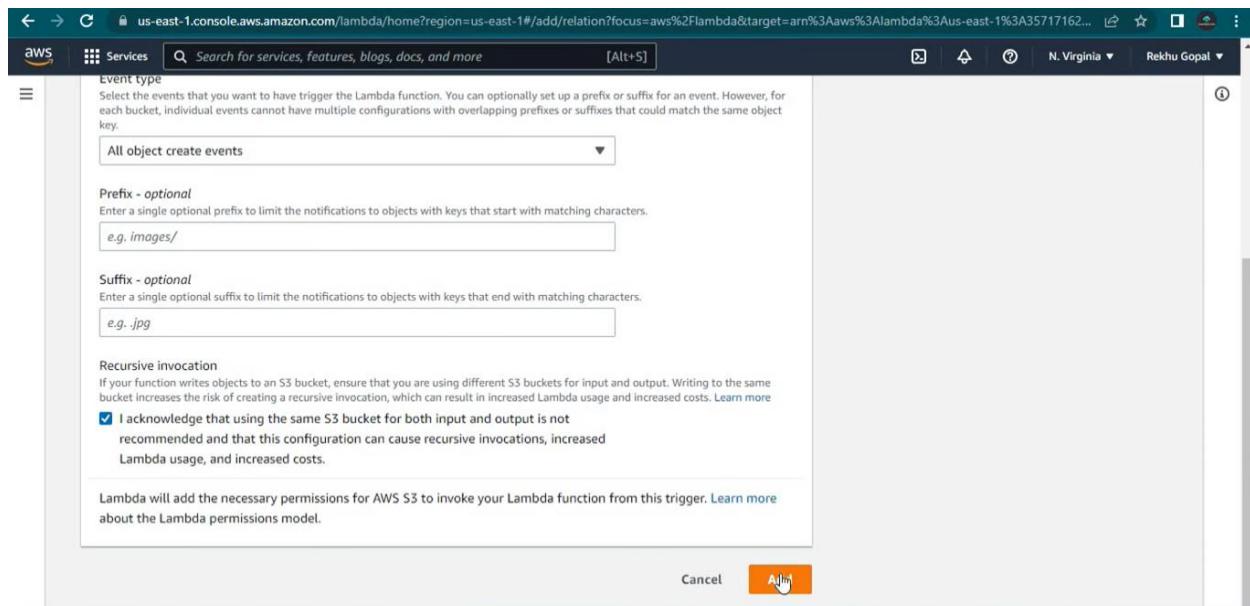
STEP 9: Write a code to get a detail from S3 and push to SNS.



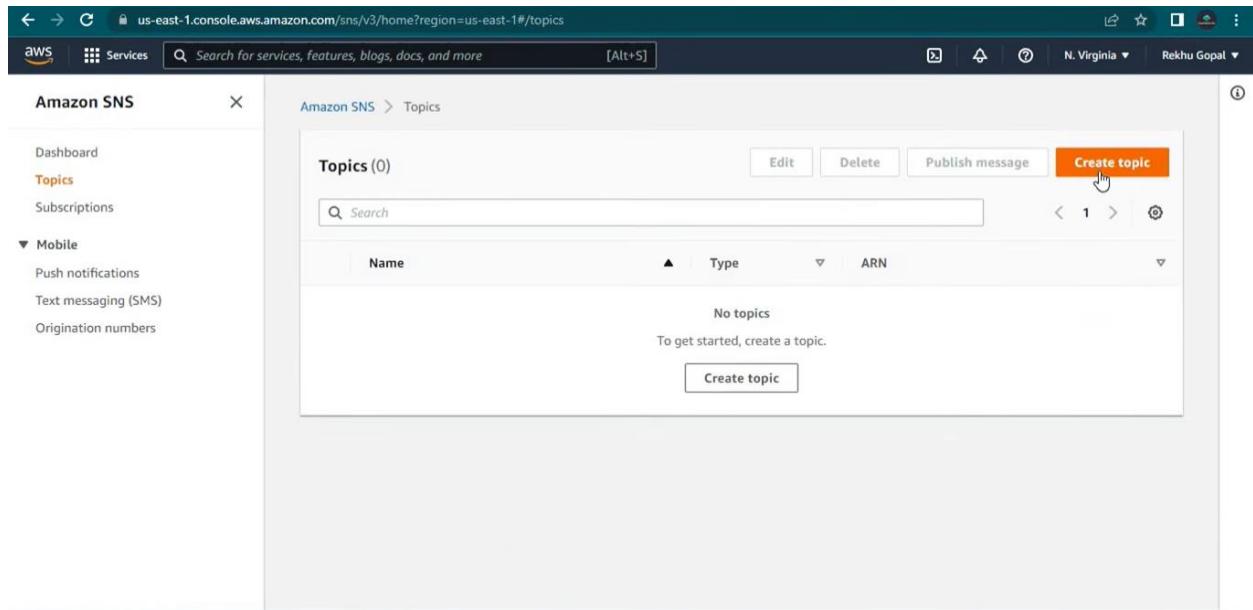
STEP 10: Add trigger with S3 – mention bucket name and event type.



STEP 11: Click “Add”.

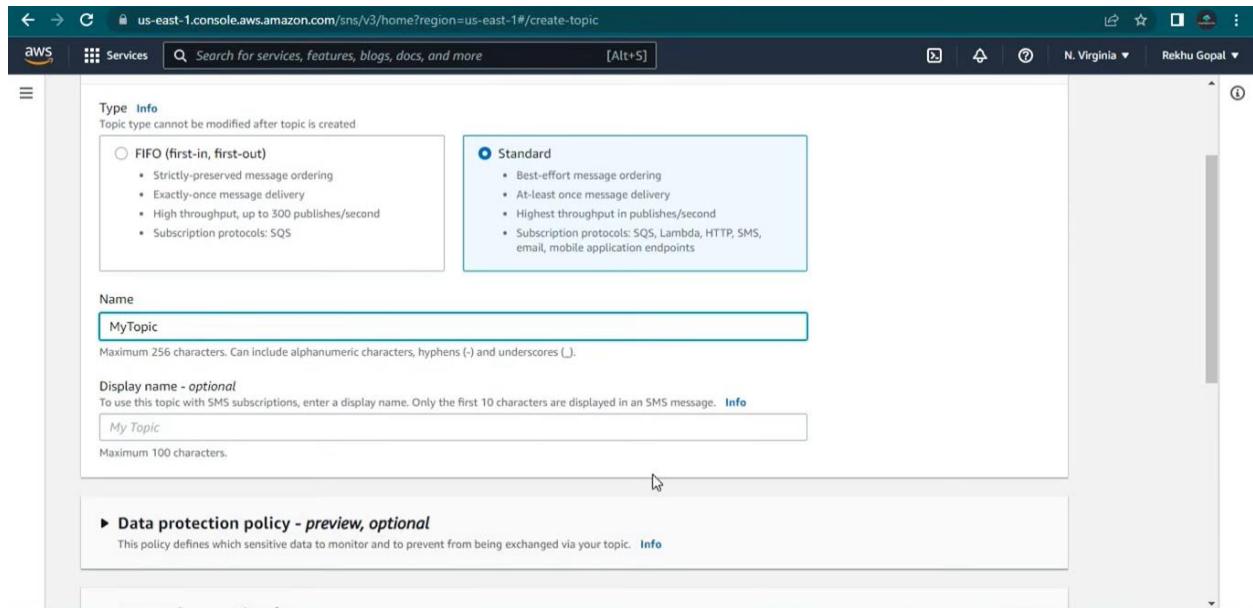


STEP 12: In Amazon SNS, click “Create topic”.



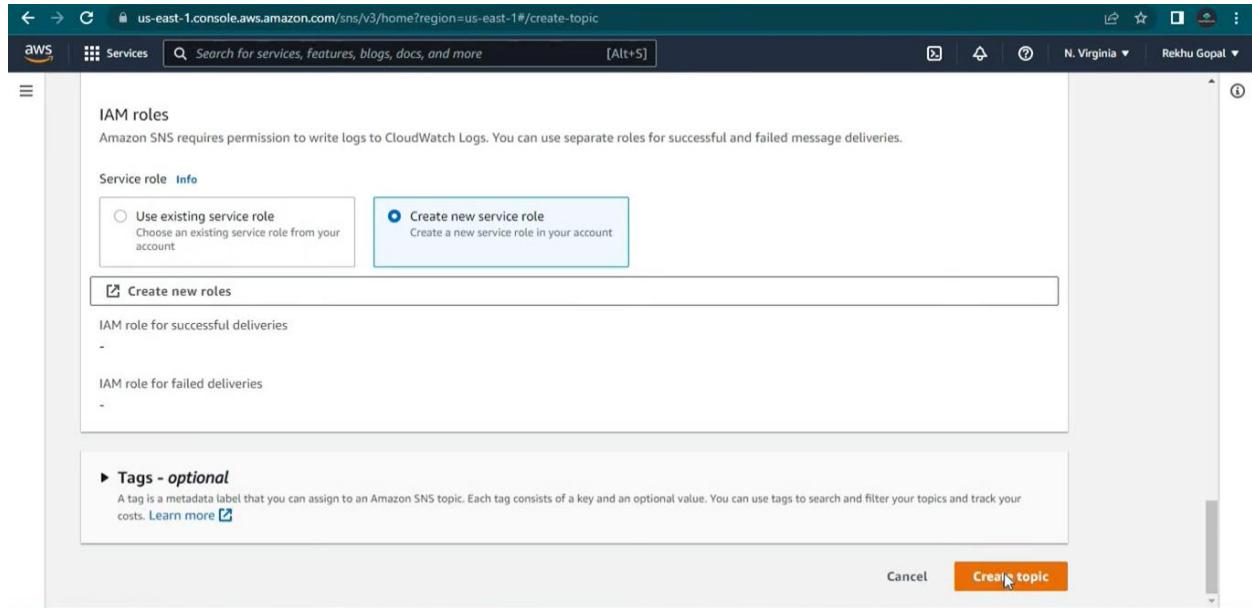
The screenshot shows the AWS SNS Topics page. On the left, there's a sidebar with options like Dashboard, Topics (which is selected and highlighted in orange), Subscriptions, and Mobile (Push notifications, Text messaging (SMS), Origination numbers). The main area is titled "Topics (0)" and contains a search bar and buttons for Edit, Delete, Publish message, and Create topic. Below the search bar, there's a table header with columns Name, Type, and ARN. A message says "No topics" and "To get started, create a topic." A prominent orange "Create topic" button is located at the bottom of the main content area.

STEP 13: Mention topic name.

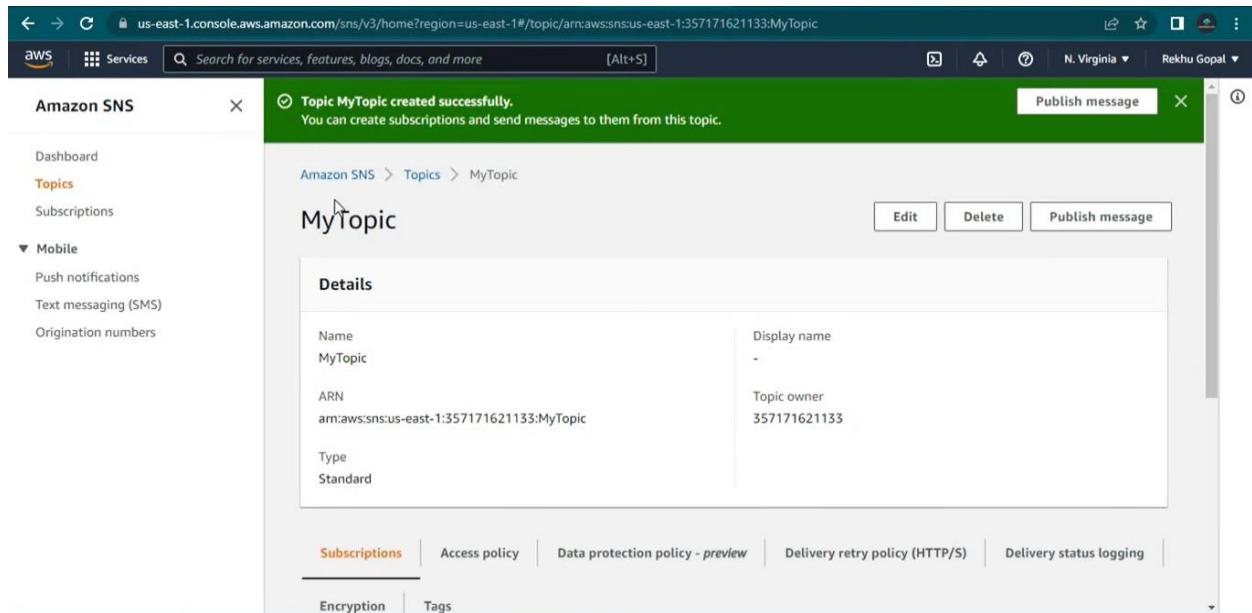


The screenshot shows the "Create topic" wizard. Step 1 is "Topic type". It has two options: "FIFO (first-in, first-out)" and "Standard". "Standard" is selected and highlighted with a blue border. Both options have associated bullet points. Below this, there's a "Name" field containing "MyTopic", a "Display name - optional" field containing "My Topic", and a "Data protection policy - preview, optional" section which is currently empty. A note states: "This policy defines which sensitive data to monitor and to prevent from being exchanged via your topic." A small "Info" link is also present.

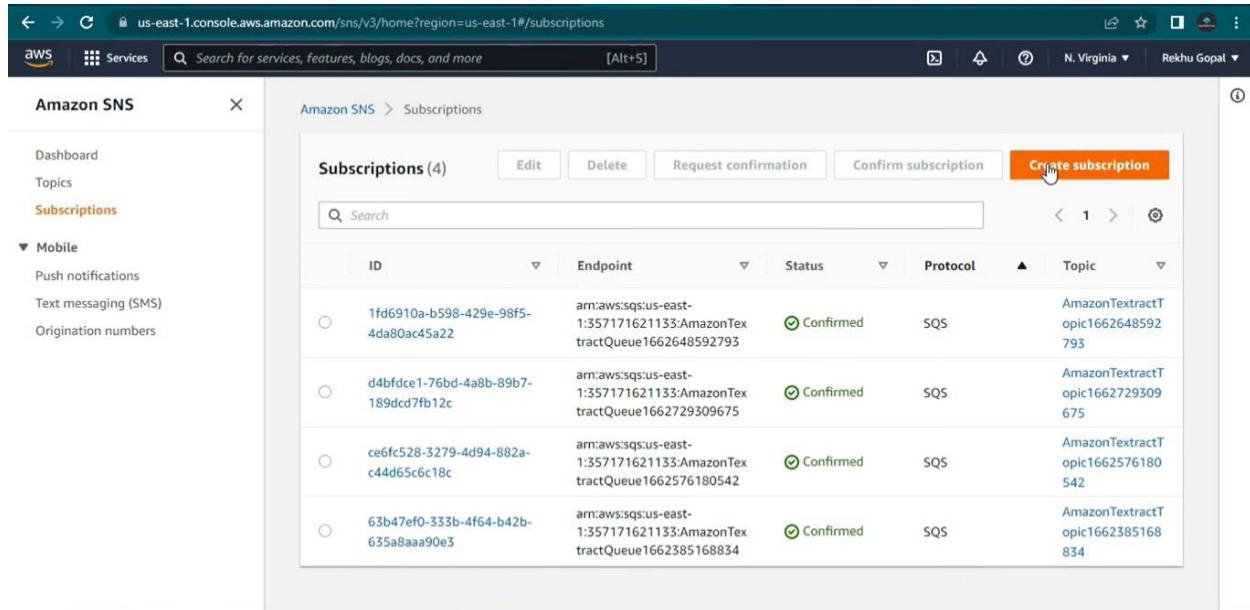
STEP 14: Choose IAM role and click “create topic”.



STEP 15: Topic created successfully.



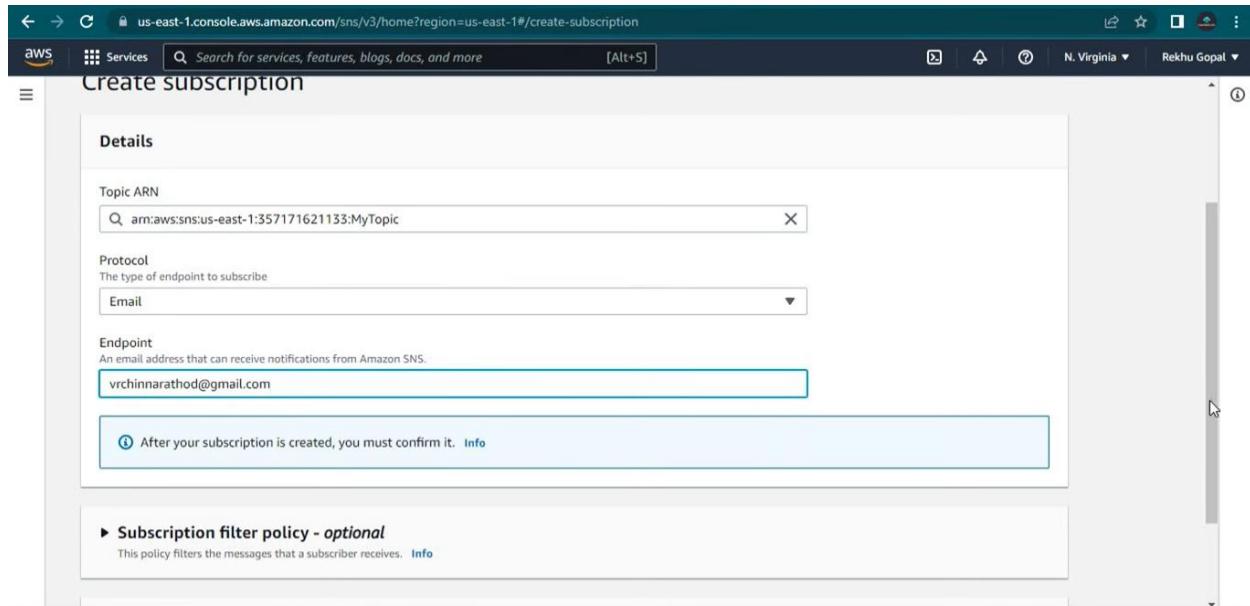
STEP 16: Next, click “Create subscription” to subscribe the topic.



The screenshot shows the AWS SNS Subscriptions page. On the left, there's a sidebar with links for Dashboard, Topics, and Subscriptions. Under Mobile, there are links for Push notifications, Text messaging (SMS), and Origination numbers. The main area is titled "Subscriptions (4)" and lists four entries:

ID	Endpoint	Status	Protocol	Topic
1fd6910a-b598-429e-98f5-4da80ac45a22	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662648592793	Confirmed	SQS	AmazonTextractTopic1662648592793
d4bfdce1-76bd-4a8b-89b7-189ddcd7fb12c	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662729309675	Confirmed	SQS	AmazonTextractTopic1662729309675
ce6fc528-3279-4d94-882a-c44d65c6c18c	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662576180542	Confirmed	SQS	AmazonTextractTopic1662576180542
63b47ef0-333b-4f64-b42b-635a8aaa90e3	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662385168834	Confirmed	SQS	AmazonTextractTopic1662385168834

STEP 17: Choose topic ARN, protocol and endpoint.



The screenshot shows the "Create subscription" wizard. In the "Details" step, the following fields are filled:

- Topic ARN:** arn:aws:sns:us-east-1:357171621133:MyTopic
- Protocol:** Email
- Endpoint:** vrchinmarathod@gmail.com

A note at the bottom says: "After your subscription is created, you must confirm it." Below that, there's a section for "Subscription filter policy - optional".

STEP 18: Subscription to topic created successfully.

The screenshot shows the AWS SNS console in the N. Virginia region. The left sidebar has 'Subscriptions' selected. The main area shows a green success message: 'Subscription to MyTopic created successfully. The ARN of the subscription is arn:aws:sns:us-east-1:357171621133:MyTopic:836bea37-b27f-4a9a-80ba-76d5d6beb415.' Below it, a table displays the subscription details:

Details	
ARN	arn:aws:sns:us-east-1:357171621133:MyTopic:836bea37-b27f-4a9a-80ba-76d5d6beb415
Endpoint	vrchinnarathod@gmail.com
Topic	MyTopic
Subscription Principal	
Status	Pending confirmation
Protocol	EMAIL

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

The screenshot shows the AWS SNS console in the N. Virginia region. The left sidebar has 'Subscriptions' selected. The main area shows a table of subscriptions:

ID	Endpoint	Status	Protocol	Topic
Pending confirmation	vrchinnarathod@gmail.com	Pending confirmation	EMAIL	MyTopic
1fd6910a-b598-429e-98f5-4da80ac45a22	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662648592793	Confirmed	SQS	AmazonTextractTopic1662648592793
d4bfdce1-76bd-4a8b-89b7-189cd7fb12c	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662729309675	Confirmed	SQS	AmazonTextractTopic1662729309675
ce6fc528-3279-4d94-882a-c44d65c6c18c	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662576180542	Confirmed	SQS	AmazonTextractTopic1662576180542
C74b47a60-777b-4f64-b47b-4763734c3a27-a	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662576180542			AmazonTextractTopic1662576180542

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

The screenshot shows a Gmail inbox with 8,537 messages. An email from "AWS Notifications <no-reply@sns.amazonaws.com>" is selected. The subject is "AWS Notification - Subscription Confirmation". The email body contains the following text:

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:357171621133:MyTopic

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#).

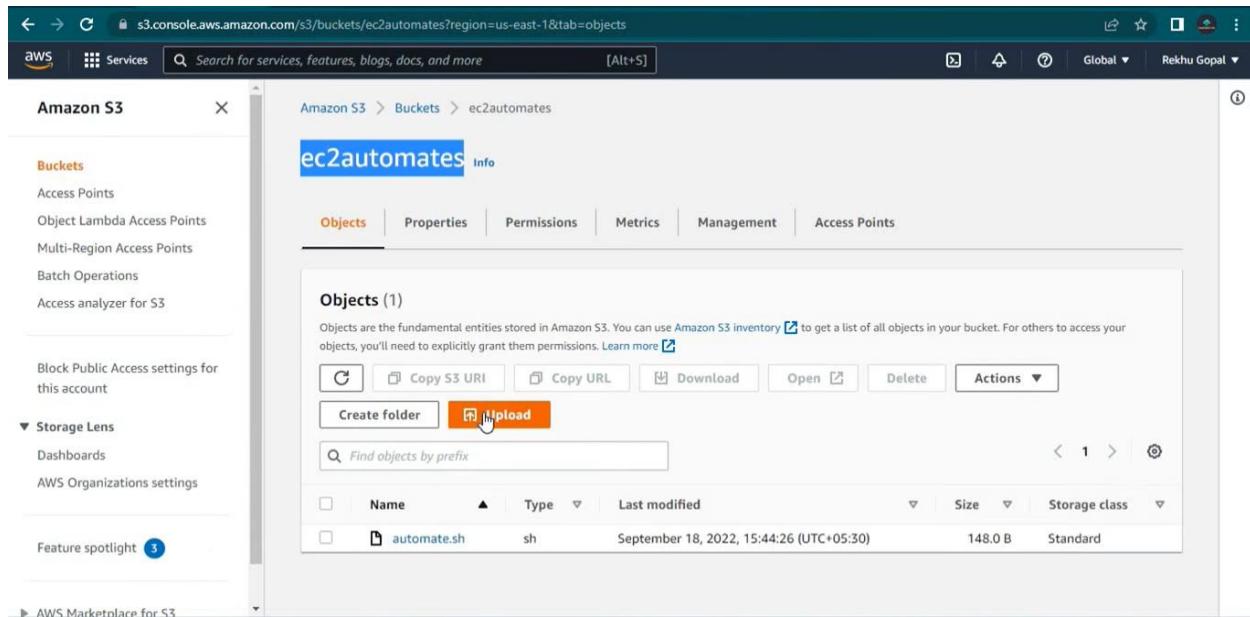
Below the email are "Reply" and "Forward" buttons.

STEP 21: Then status changed to confirmed state.

The screenshot shows the "Subscriptions" page in the Amazon SNS console. The left sidebar shows "Subscriptions" is selected. The main table lists five subscriptions:

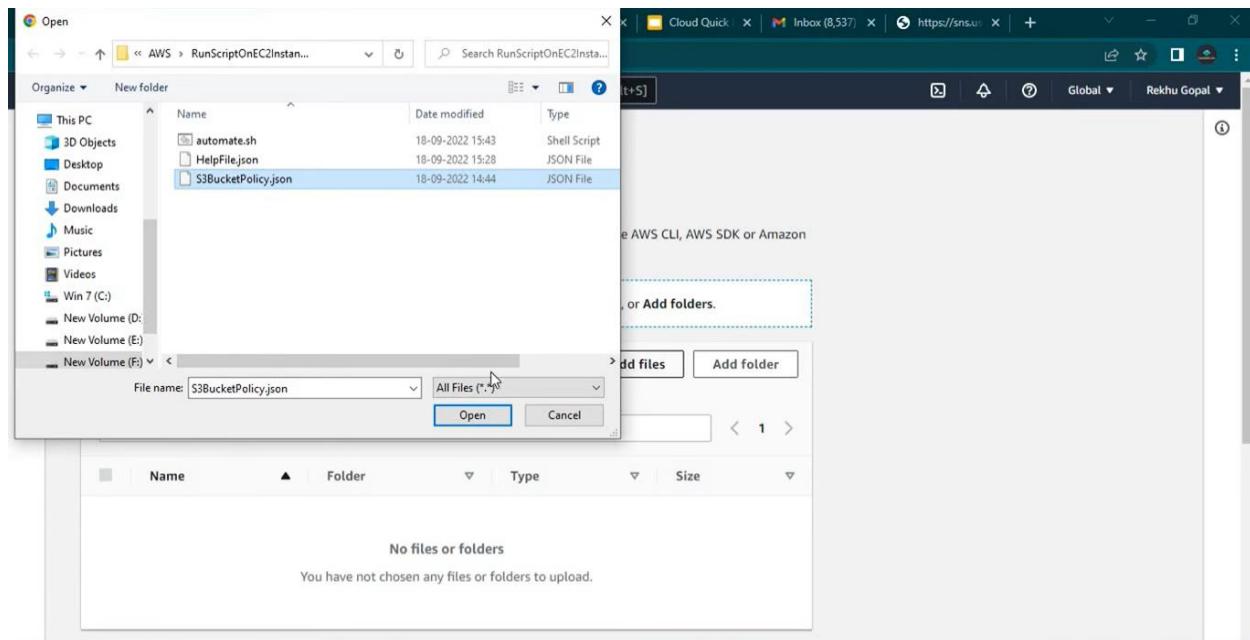
ID	Endpoint	Status	Protocol	Topic
836bea37-b27f-4a9a-80ba-76d5d6beb415	vrcchinnarathod@gmail.com	Confirmed	EMAIL	MyTopic
1fd6910a-b598-429e-98f5-4da80ac45a22	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662648592793	Confirmed	SQS	AmazonTextract Topic166264859 2793
d4bfdce1-76bd-4a8b-89b7-189dd7fb12c	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662729309675	Confirmed	SQS	AmazonTextract Topic166272930 9675
ce6fc528-3279-4d94-882a-c44d65c6c18c	arn:aws:sqs:us-east-1:357171621133:AmazonTextractQueue1662576180542	Confirmed	SQS	AmazonTextract Topic166257618 0542

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



The screenshot shows the AWS S3 console interface. On the left, the navigation pane includes 'Buckets' (selected), 'Access Points', 'Object Lambda Access Points', 'Multi-Region Access Points', 'Batch Operations', 'Access analyzer for S3', 'Block Public Access settings for this account', 'Storage Lens' (with 'Dashboards' and 'AWS Organizations settings'), and a 'Feature spotlight'. The main content area shows the 'ec2automates' bucket. The 'Objects' tab is selected, displaying one object: 'automate.sh' (Type: sh, Last modified: September 18, 2022, 15:44:26 (UTC+05:30), Size: 148.0 B, Storage class: Standard). Below the table are buttons for 'Actions' (Copy S3 URI, Copy URL, Download, Open, Delete, Upload, Create folder), a search bar ('Find objects by prefix'), and navigation controls ('< 1 >').

STEP 23: Choose one file from PC.



The screenshot shows a Windows File Explorer window with the path 'AWS > RunScriptOnEC2Insta...'. The left sidebar shows 'This PC' with various drives and volumes listed. The main area displays three files: 'automate.sh' (Shell Script), 'HelpFile.json' (JSON File), and 'S3BucketPolicy.json' (JSON File). A file selection dialog box is open over the File Explorer, prompting the user to 'Select files or folders to upload'. The 'File name:' dropdown is set to 'S3BucketPolicy.json', and the 'Open' button is highlighted. Below the dialog, a message states 'No files or folders' and 'You have not chosen any files or folders to upload.'

STEP 24: File uploaded successfully in S3 bucket.

The screenshot shows the AWS S3 console at the URL s3.console.aws.amazon.com/s3/upload/ec2automates?region=us-east-1. A green header bar indicates "Upload succeeded". Below it, the "Upload: status" section shows a summary table:

Destination	Succeeded	Failed
s3://ec2automates	✓ 1 file, 393.0 B (100.00%)	✗ 0 files, 0 B (0%)

Below the summary, there are tabs for "Files and folders" and "Configuration". Under "Files and folders", it says "Files and folders (1 Total, 393.0 B)" and includes a search bar and navigation arrows.

STEP 25: Notification received to email successfully.

The screenshot shows a Gmail inbox at the URL mail.google.com/mail/u/0/?tab=rm#inbox/FMfcgzGqQcnjTTgVzprqFhlGXVPNQTDw. The inbox list shows one email from "AWS Notifications <no-reply@sns.amazonaws.com>" with the subject "Processes completion Notification". The email content is:

The file is uploaded at S3 bucket path s3://ec2automates/S3BucketPolicy.json

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:357171621133:MyTopic:836bea37-b27f-4a9a-80ba-76d5d6beb415&Endpoint=vrchinna Rathod@gmail.com>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

DASHBOARD AND APPLICATIONS

The screenshot shows the AWS Lambda dashboard for the Asia Pacific (Mumbai) region. On the left, a sidebar menu includes options like Dashboard, Applications, Functions, Additional resources, and Related AWS resources. The main area displays 'Resources for Asia Pacific (Mumbai)' with key metrics: 54 Lambda function(s), 245.2 kB of code storage (0% of 75.0 GB), Full account concurrency at 1000, and Unreserved account concurrency at 1000. Below this, there's a section for Account-level metrics with three charts: Error count and success rate, which are currently empty due to permission issues.

The screenshot shows the AWS Lambda Applications page. The sidebar menu is identical to the dashboard. The main area shows a table titled 'Applications (0) info' with a search bar and a 'Create application' button. A message indicates that no applications are present yet, with a link to learn more about creating serverless applications using Lambda.

The screenshot shows the AWS Lambda console at <https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/create/application>. The left sidebar has 'AWS Lambda' selected under 'Applications'. The main area is titled 'Create a Lambda application' and describes it as a combination of Lambda functions, triggers, and other resources. A red box highlights an error message: '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/*'. Below the error, there are two tabs: 'CI/CD pipeline application' (selected) and 'Serverless application'. Under 'From scratch', it says 'Create a repository and pipeline to use with your own application.' and lists 'Runtime: Node.js 18.x' and 'Uses: Lambda'. At the bottom, there's a link to 'Serverless API backend'.

The screenshot shows the same AWS Lambda console page with the 'Serverless API backend' template selected. The template description is 'A RESTful web API that uses DynamoDB to manage state.' It lists 'Runtime: Node.js 18.x' and 'Uses: API Gateway, DynamoDB, Lambda'. Below it are two more templates: 'File processing' (using Amazon S3 to trigger Lambda for file processing like thumbnails and transcodes) and 'Scheduled job' (using CloudWatch Events for regular scheduled triggers). Both also list 'Runtime: Node.js 18.x' and 'Uses: Lambda'. The sidebar remains the same with 'AWS Lambda' selected under 'Applications'.

The screenshot shows the AWS Lambda console interface. On the left, a sidebar menu includes options like Dashboard, Applications (which is selected), Functions, Additional resources, and Related AWS resources. The main content area displays three examples of Lambda functions:

- Notifications processing**: Runtime: Node.js 18.x, Uses: CloudWatch Events, Lambda.
- Queue processing**: Runtime: Node.js 18.x, Uses: Lambda, SNS.
- Queue processing**: Runtime: Node.js 18.x, Uses: Lambda, SQS.

Each example provides a brief description of its purpose and the services it interacts with.

The screenshot shows the configuration page for creating a new Lambda application. The sidebar remains the same as the previous screen. The main area is titled "Application details" and contains the following fields:

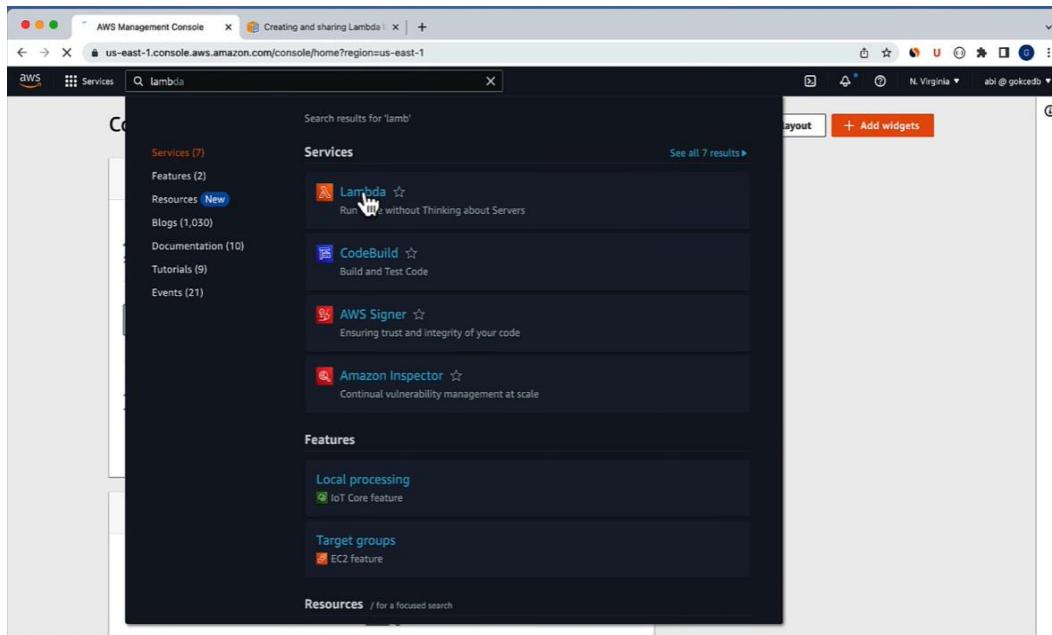
- Name**: sampleapp (with a note: "Use only lowercase letters, numbers, or hyphens. The maximum length is 20 characters.")
- Language**: Runtime: Node.js 18.x
- Template format**: Info
 AWS SAM (YAML)
 AWS CDK (TypeScript)
- Source control**: This section is currently empty.

The screenshot shows the AWS Lambda 'Create Application' configuration page. On the left, a sidebar menu includes 'Dashboard', 'Applications' (which is selected), 'Functions', 'Additional resources' (with 'Code signing configurations', 'Layers', and 'Replicas' listed), and 'Related AWS resources' (with 'Step Functions state machines'). The main content area is titled 'Source control'. It displays two options: 'CodeCommit' (selected) and 'CodeStar Connections'. Under 'CodeCommit', there is a description: 'Create a repository in your AWS account. Manage SSH keys and HTTP credentials for users in the IAM console.' A 'Repository name' input field contains 'sampleapp', with a note below stating 'The maximum length is 100 characters.' Below this is a 'Permissions' section with a 'Info' link.

This screenshot shows the continuation of the 'Create Application' configuration page. The 'Source control' section is identical to the previous screenshot. In the 'Permissions' section, there is a note: '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' with a link. A checkbox labeled 'Create roles and permissions boundary' is checked. At the bottom right, there are 'Cancel', 'Previous', and 'Create' buttons, where 'Create' is highlighted in orange.

LAYERS

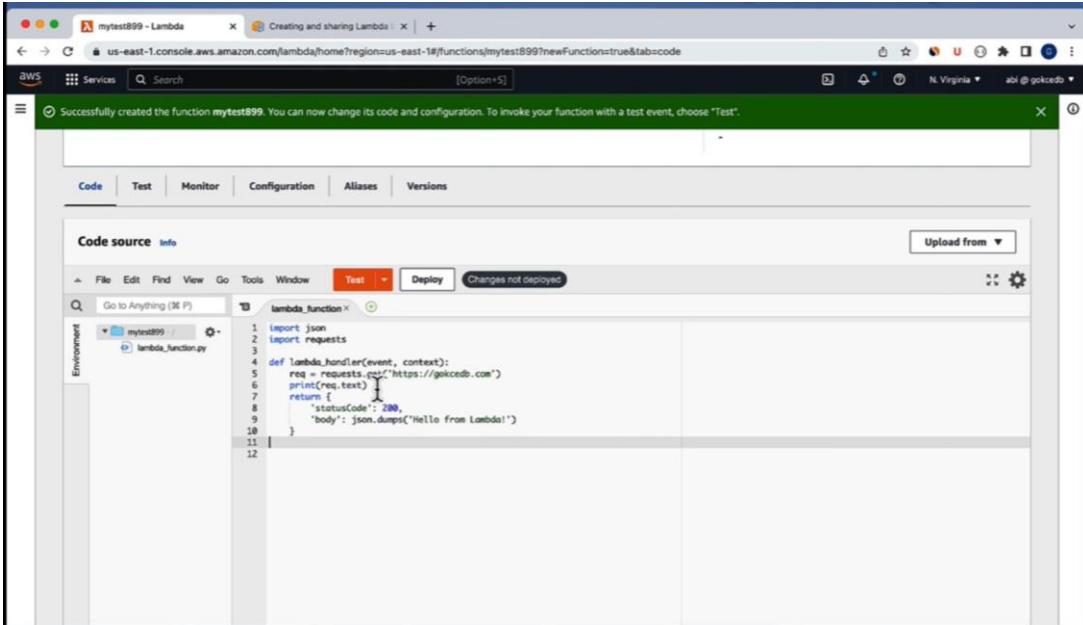
STEP 1: Search Lambda



STEP 2: Create A Function

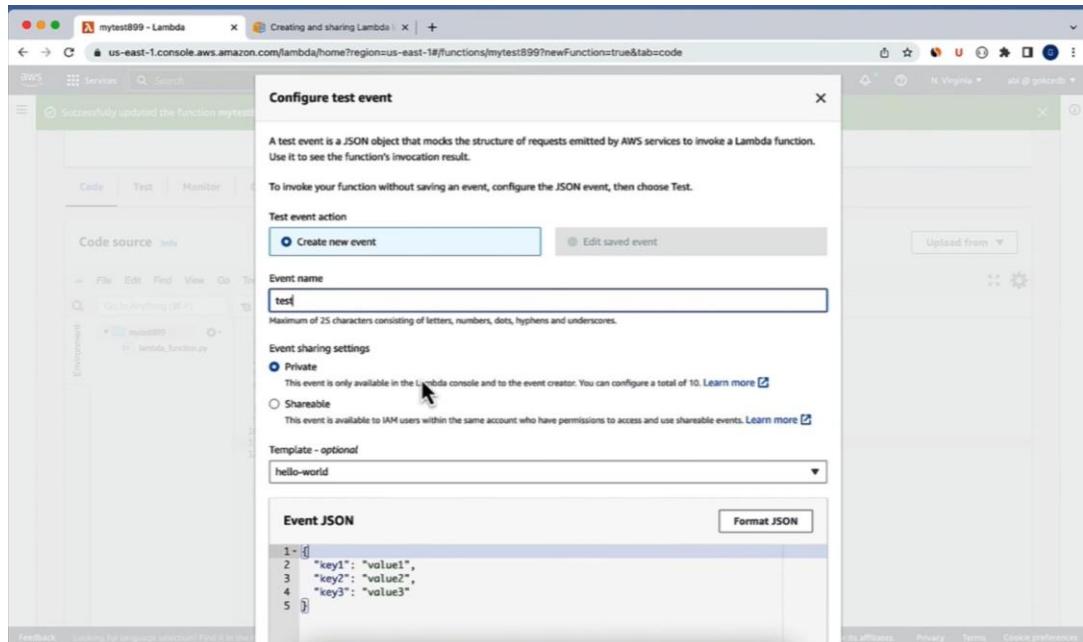
A screenshot of the 'Create function - Lambda' wizard. The page is titled 'Basic information'. It includes fields for 'Function name' (set to 'mytest899'), 'Runtime' (set to 'Python 3.9'), 'Architecture' (set to 'x86_64'), and 'Permissions' (info about execution role). At the bottom right are 'Cancel' and 'Create function' buttons.

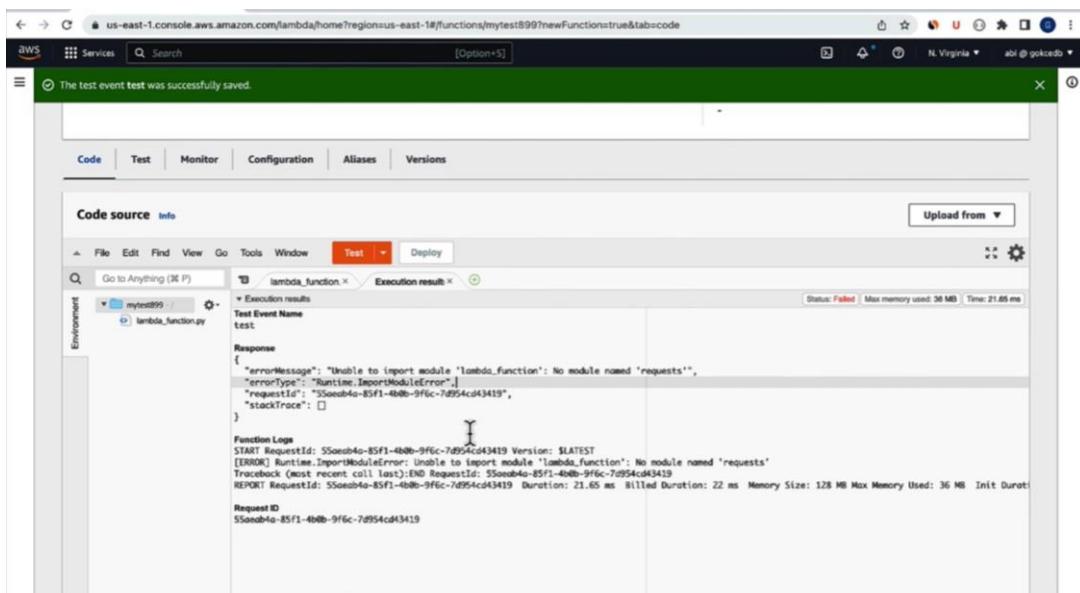
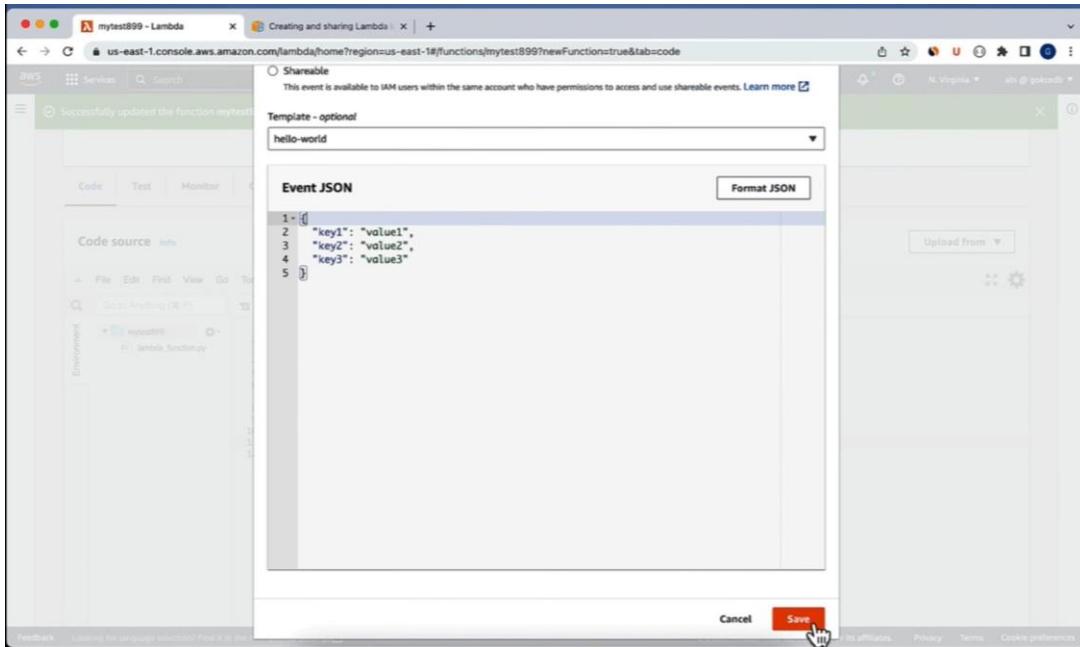
STEP 3: Deploy and Test The Code



The screenshot shows the AWS Lambda console interface. A success message at the top states: "Successfully created the function mytest899. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below this, the "Code source" tab is selected in the navigation bar. The main area displays the code for "lambda_function.py":

```
1 import json
2 import requests
3
4 def lambda_handler(event, context):
5     req = requests.get('https://gokcedb.com')
6     print(req.text)
7     return {
8         'statusCode': 200,
9         'body': json.dumps('Hello from Lambda!')
10    }
11
12
```





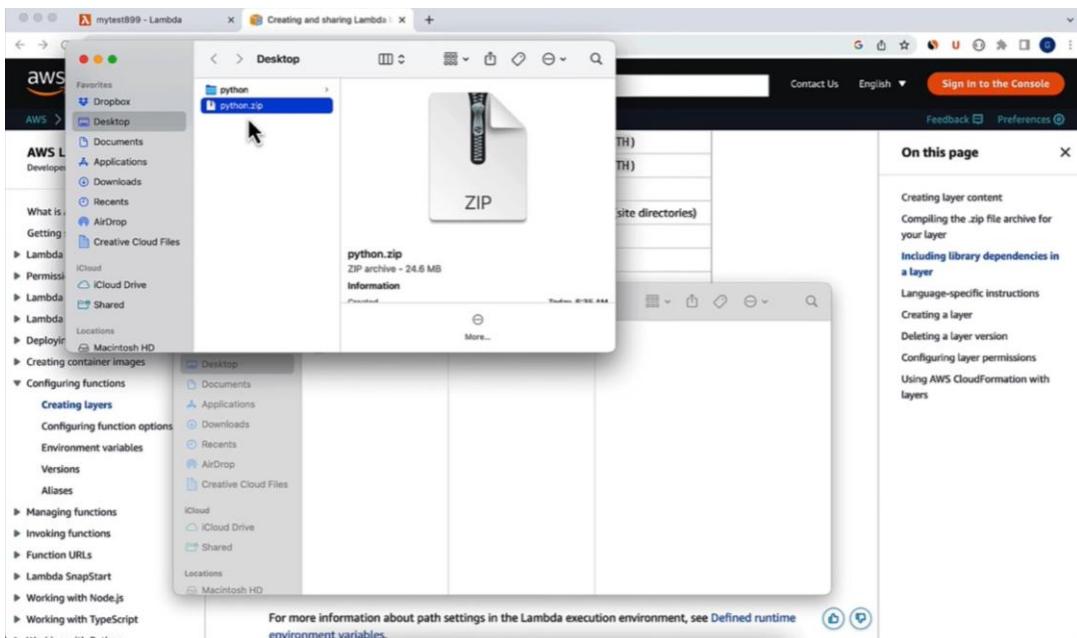
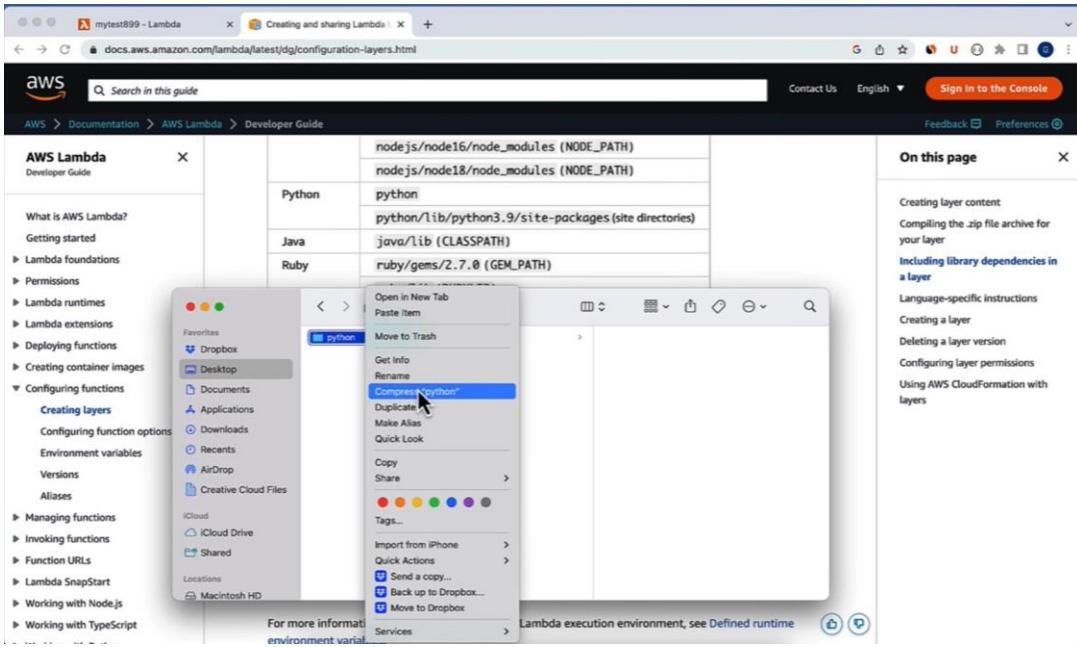
STEP 4: Compressing Package Dependencies into A Zip File

The screenshot shows the AWS Lambda Developer Guide page. On the left, there's a sidebar with navigation links like 'What is AWS Lambda?', 'Getting started', and 'Configuring functions'. The main content area has a table titled 'Layer paths for each Lambda runtime'.

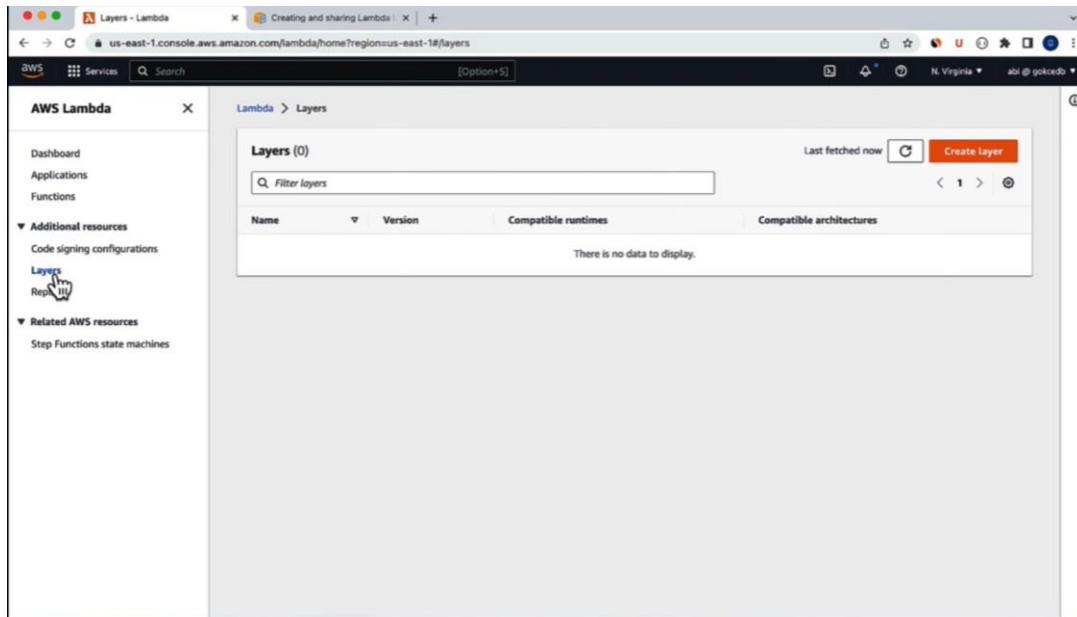
Runtime	Path
Node.js	nodejs/node_modules nodejs/node14/node_modules (NODE_PATH) nodejs/node16/node_modules (NODE_PATH) nodejs/node18/node_modules (NODE_PATH)
Python	python python/lib/python3.9/site-packages (site directories)
Java	java/lib (CLASSPATH)
Ruby	ruby/gems/2.7.0 (GEM_PATH) ruby/lib (RUBYLIB)
All runtimes	bin (PATH) lib (LD_LIBRARY_PATH)

Below the table, a note says: "The following examples show how you can structure the folders in your layer .zip archive." There are two small icons at the bottom right of the table.

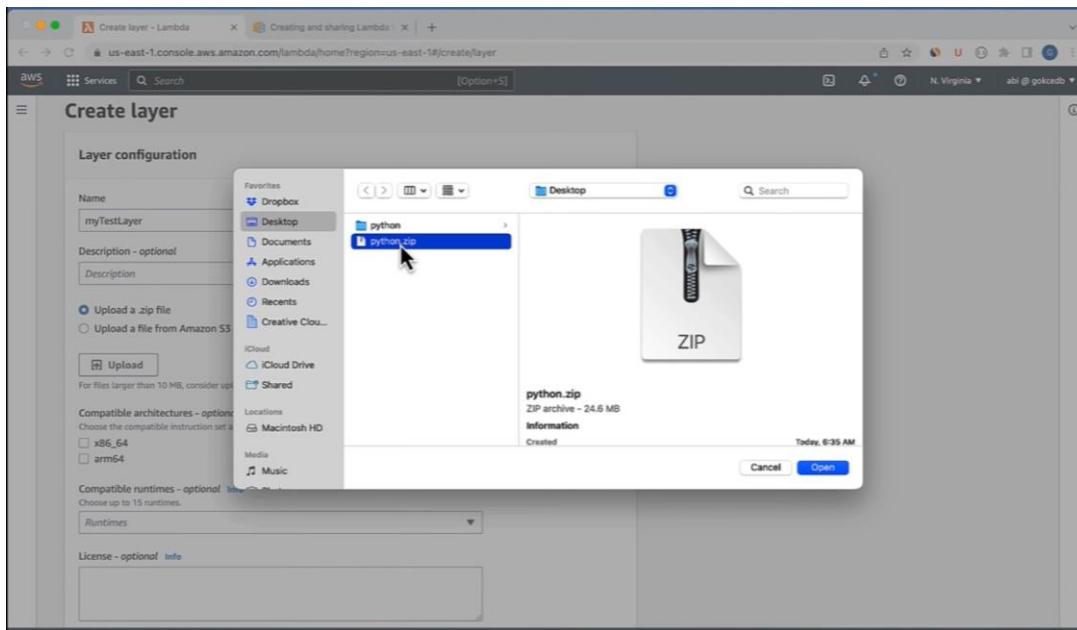
This screenshot shows a Mac OS X Finder window. The path in the address bar is 'site-packages'. Inside the window, there's a single folder named 'python'. The sidebar on the left shows standard OS X locations like 'Desktop', 'Documents', and 'Downloads'. At the bottom of the window, it says 'For more information about path settings in the Lambda execution environment, see Defined runtime environment variables.'

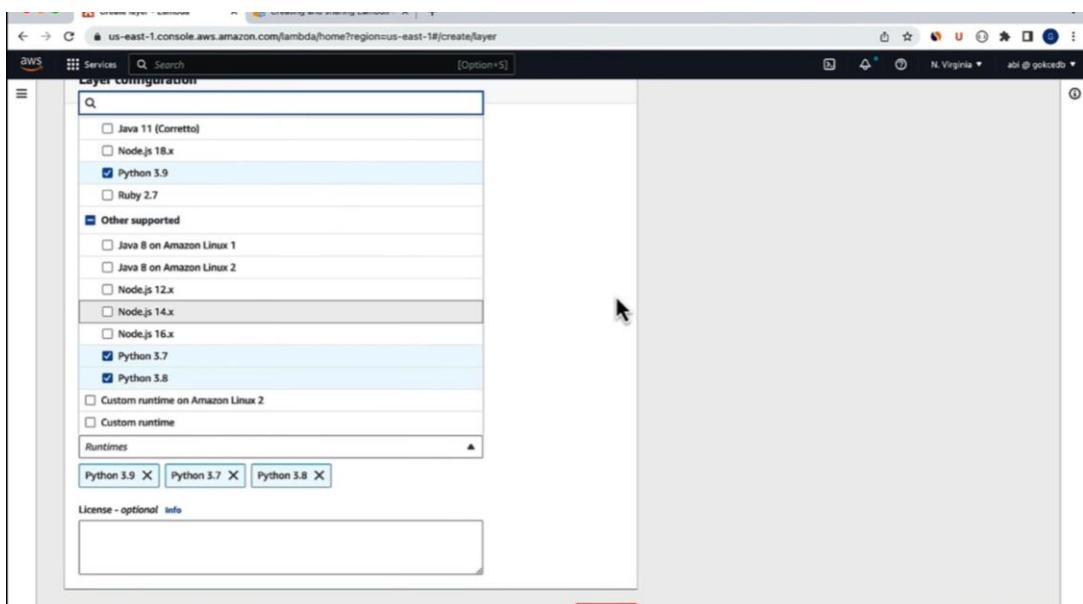
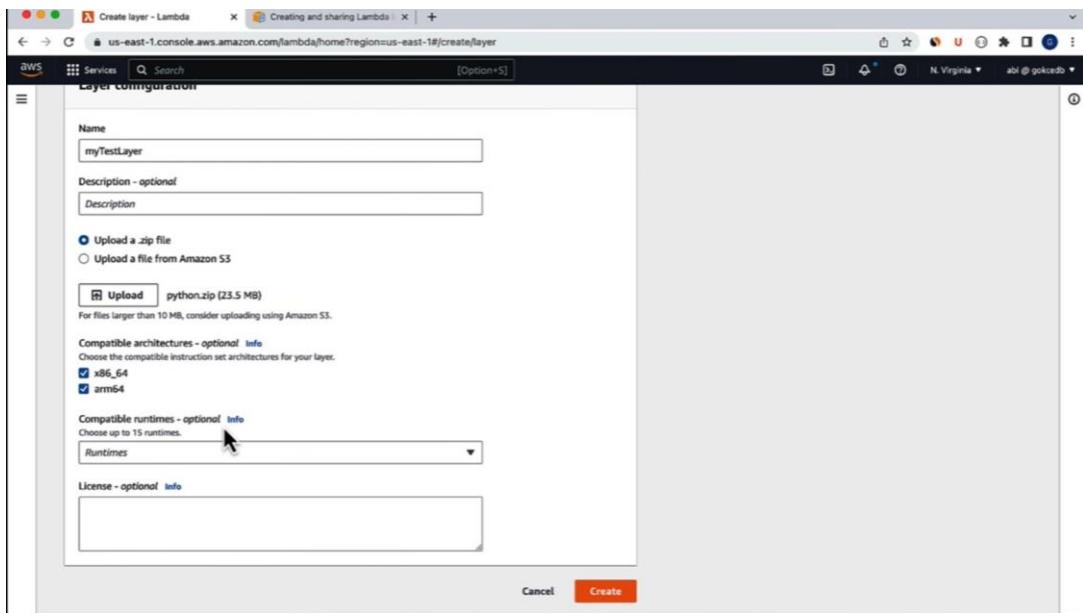


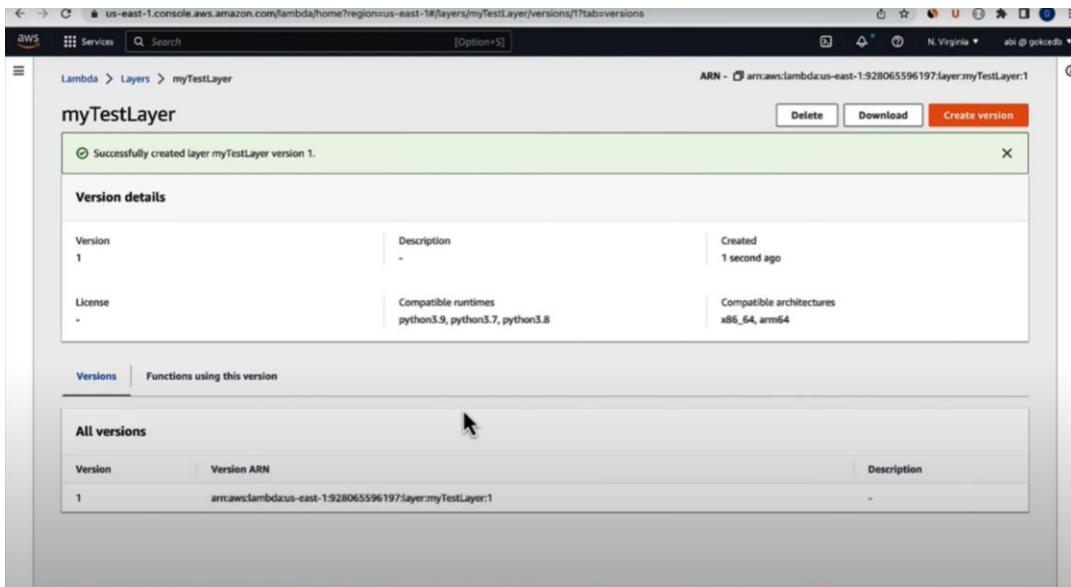
STEP 5: In Layers, Create A New Layer



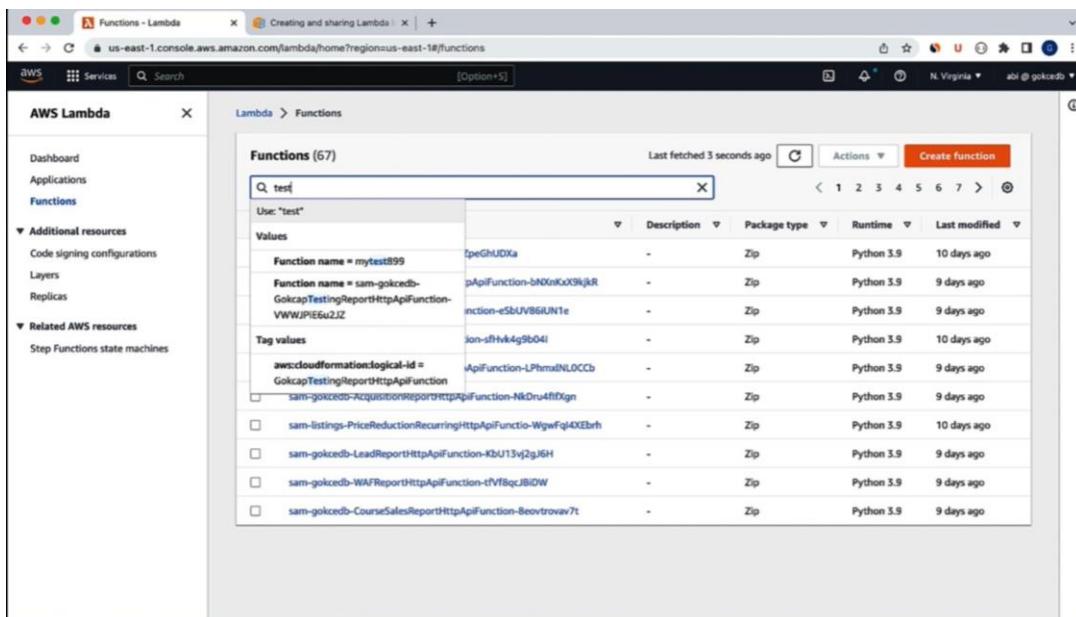
STEP 6: Upload the Zip File And Create A Layer







STEP 7: In the Created Function, Add A Layer



The screenshot shows the AWS Lambda function configuration interface. At the top, there's a search bar and navigation links for services and regions. Below the header, the main content area is divided into sections:

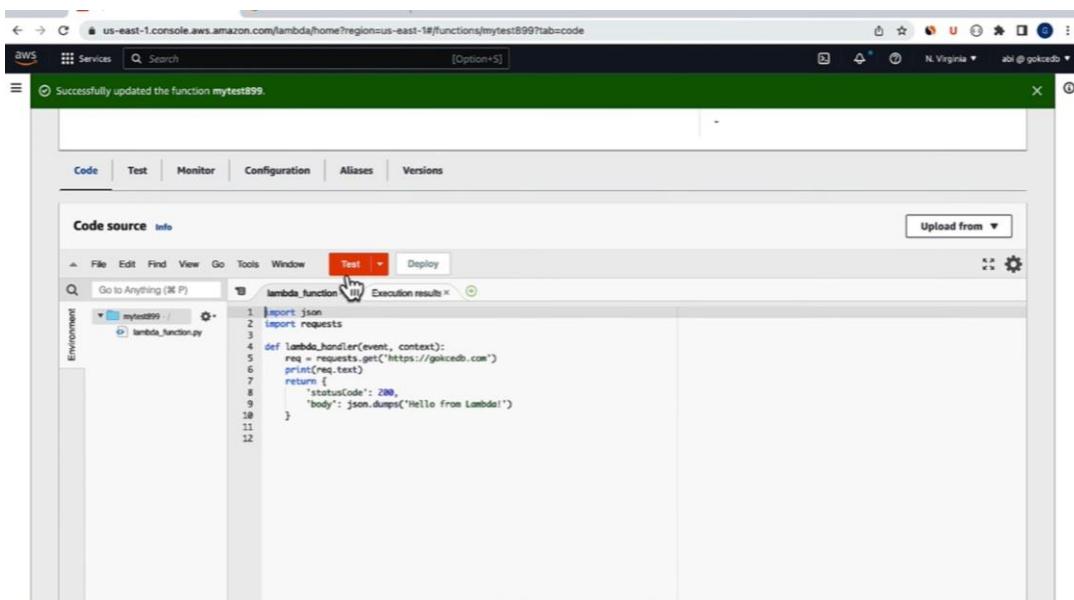
- Code properties**: Shows package size (296.0 byte), SHA256 hash (G6sgrpmPkJYAmzemLP+bxp4/QxEyq9jv+gQfB8j=), and last modified date (January 18, 2023 at 06:32 AM EST).
- Runtime settings**: Shows runtime (Python 3.9) and handler (lambda_function.lambda_handler). There's an "Edit" button for this section.
- Layers**: Shows a table with columns: Merge order, Name, Layer version, Compatible runtimes, Compatible architectures, and Version ARN. A message indicates "There is no data to display."

The screenshot shows the "Add layer" dialog box. It has two main sections:

- Function runtime settings**: Shows runtime (Python 3.9) and 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. 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**: A dropdown menu showing the value "1".

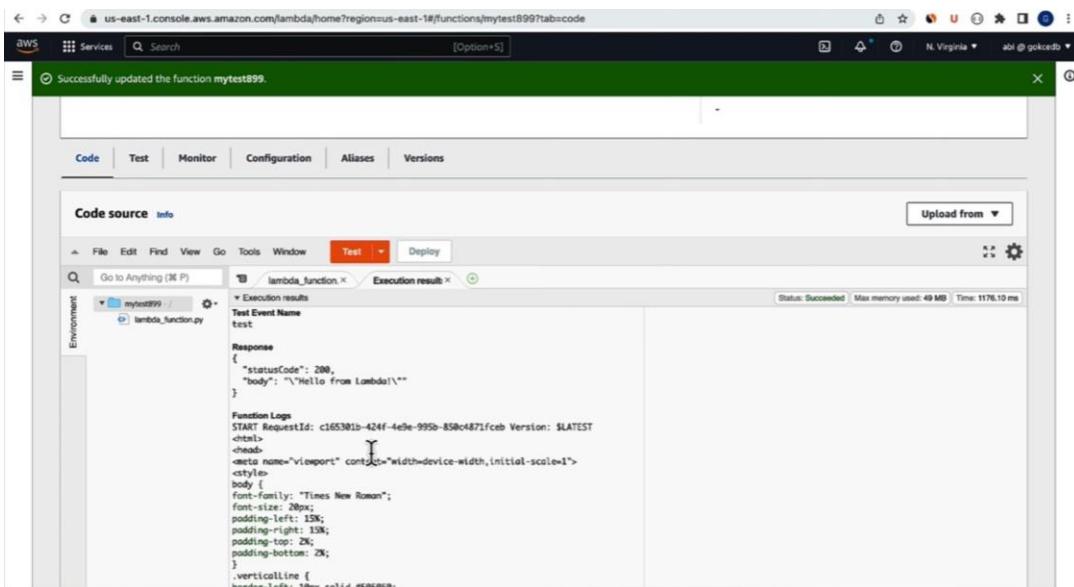
At the bottom right, there are "Cancel" and "Add" buttons, with the "Add" button being highlighted.

STEP 8: Test the Code



The screenshot shows the AWS Lambda function editor for the function 'mytest899'. The 'Code source' tab is selected. The code in 'lambda_function.py' is:

```
1 import json
2 import requests
3
4 def lambda_handler(event, context):
5     req = requests.get('https://gokcedb.com')
6     print(req.text)
7     return {
8         'statusCode': 200,
9         'body': json.dumps('Hello from Lambda!')
10    }
11
12
```



The screenshot shows the AWS Lambda function editor for the function 'mytest899'. The 'Test' tab is selected. The execution results show:

Execution results

Execution Name: test

Response:

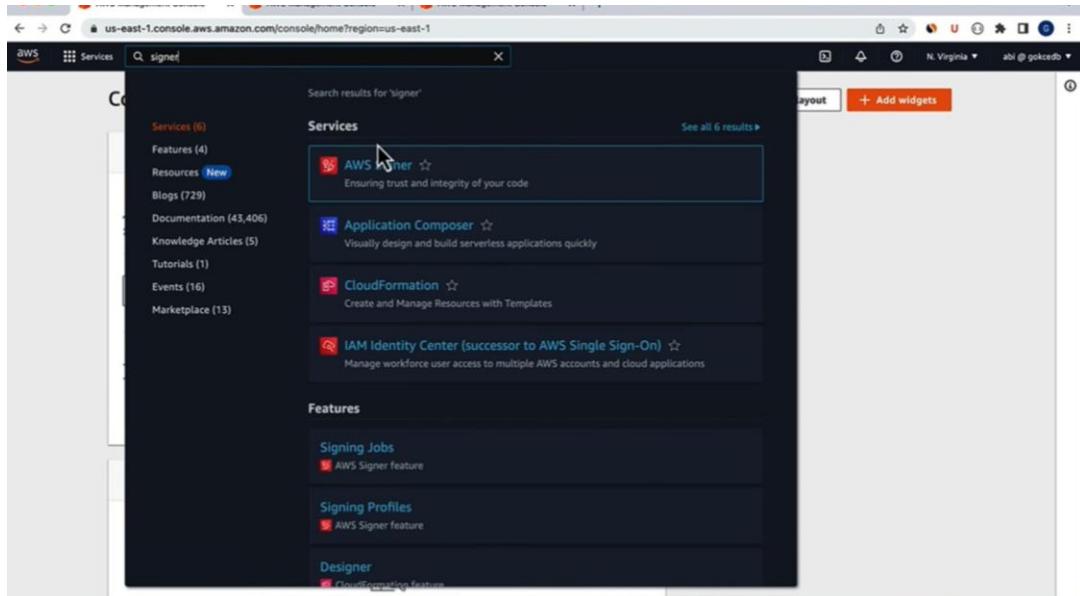
```
{ "statusCode": 200, "body": "\"Hello from Lambda!\""} 
```

Function Logs:

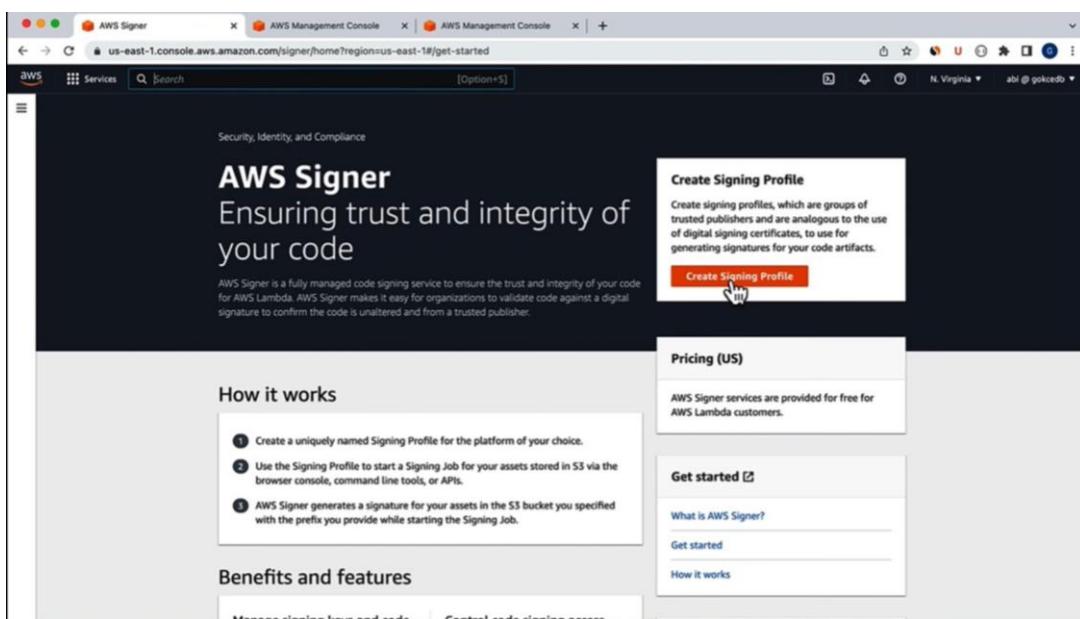
```
START RequestId: c165301b-424f-4e9e-995b-858c4871fcab Version: $LATEST
<html>
<head>
<meta name="viewport" content="width=device-width,initial-scale=1">
<style>
<body>
<font-family: "Times New Roman";>
<font-size: 20px;>
<padding-left: 15px;>
<padding-right: 15px;>
<padding-top: 20px;>
<padding-bottom: 20px;>
</font>
<.verticalline {>
<border-left: 10px solid #000000;>
</style>
</head>
<body>
```

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

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.
 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.
 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
<input type="text" value="Q Enter key"/>	<input type="text" value="Q Enter value"/>
<input type="button" value="Remove tag"/>	
<input type="button" value="Add tag"/>	

You can add 49 more tag(s).

us-east-1.console.aws.amazon.com/signer/home?region=us-east-1#/signing-profiles/myTestSigningProfile

AWS Signer > Signing Profiles > myTestSigningProfile

myTestSigningProfile

Start Signing Job

Success!
Successfully created Profile with ARN: arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile.

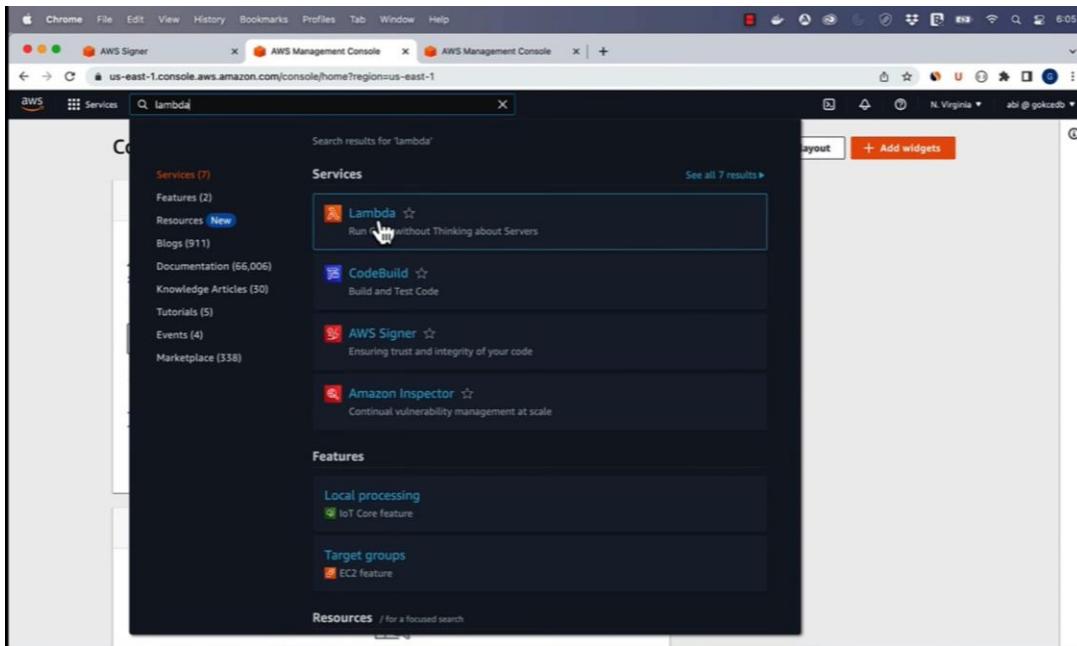
Signing Profile details

Profile name	Profile ARN
myTestSigningProfile	arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile
Profile version	Versioned Profile ARN
zwSURRGX9o	arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile/zwSURRGX9o
Platform	Signature validity period
AWS Lambda	135 months
Profile status	
Active	

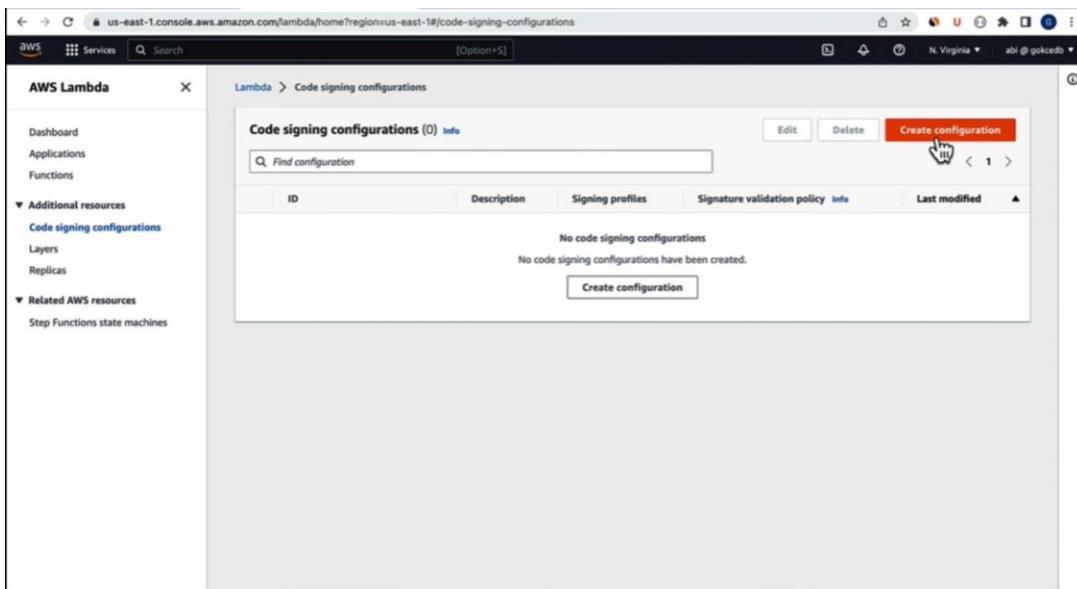
Tags

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

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.

The maximum length is 256 characters.

Signing profile [Info](#)
Choose or enter a signing profile version ARN. To create a signing profile, use AWS Signer [Create](#).
Signing profile version ARN

[myTestSigningProfile](#)
arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile/zw5URR6X3o

[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.

[Create configuration](#)

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/code-signing-configuration

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.

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 [Create](#).
Signing profile version ARN

[C](#)

[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.

[Create configuration](#)

The screenshot shows the AWS Lambda Code signing configurations page. A green success message at the top states: "The code signing configuration was successfully created: myTestCodeSigningConfig." Below this, the configuration details are listed:

- Description: myTestCodeSigningConfig
- Last modified: 35 seconds ago
- Configuration ID: csc-0c37b8858832c8c8a
- Configuration ARN: arn:aws:lambda:us-east-1:928065596197:code-signing-config:csc-0c37b8858832c8c8a
- Signing profiles: arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile/zwSURRE6X9o

The "Policies" section shows "Signature validation policy: Enforce".

The "Functions using this configuration" section is empty, displaying "There is no data to display."

STEP 5: Compressing the Code File Into A Zip File

The screenshot shows the same AWS Lambda Code signing configurations page as before. A file browser window is overlaid on the page, showing a file named "lambda_function.py" in the "Desktop" folder. The file content is displayed in the main pane:

```
import json
def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

The "Information" pane below the file preview shows:

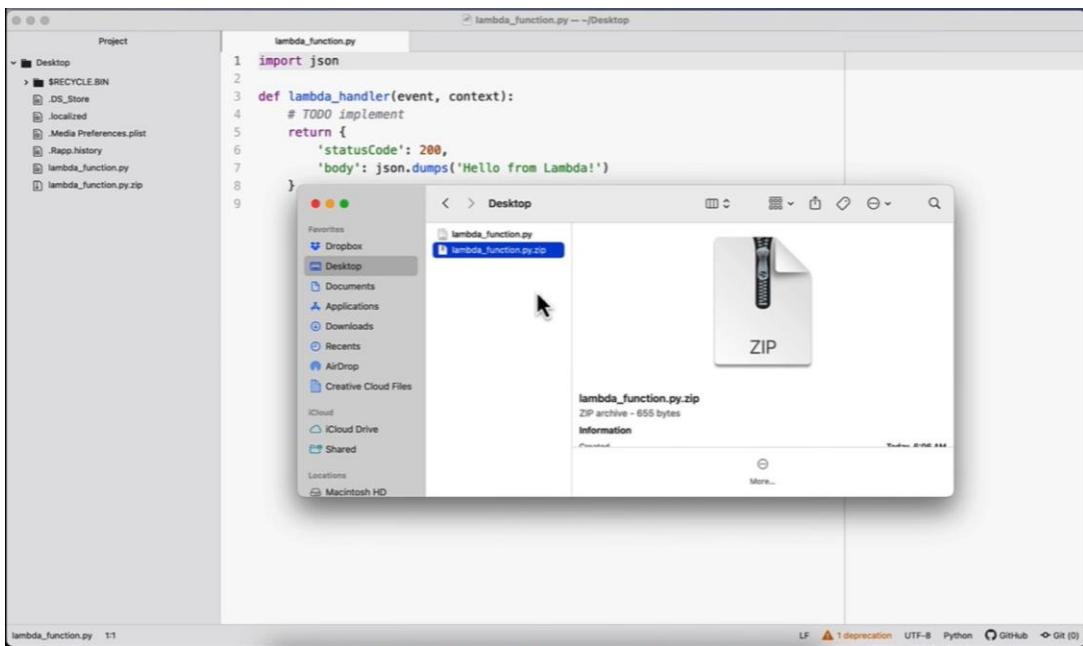
- lambda_function.py
- Python Script - 165 bytes
- Information

The "Details" sidebar on the left remains the same as in the first screenshot.

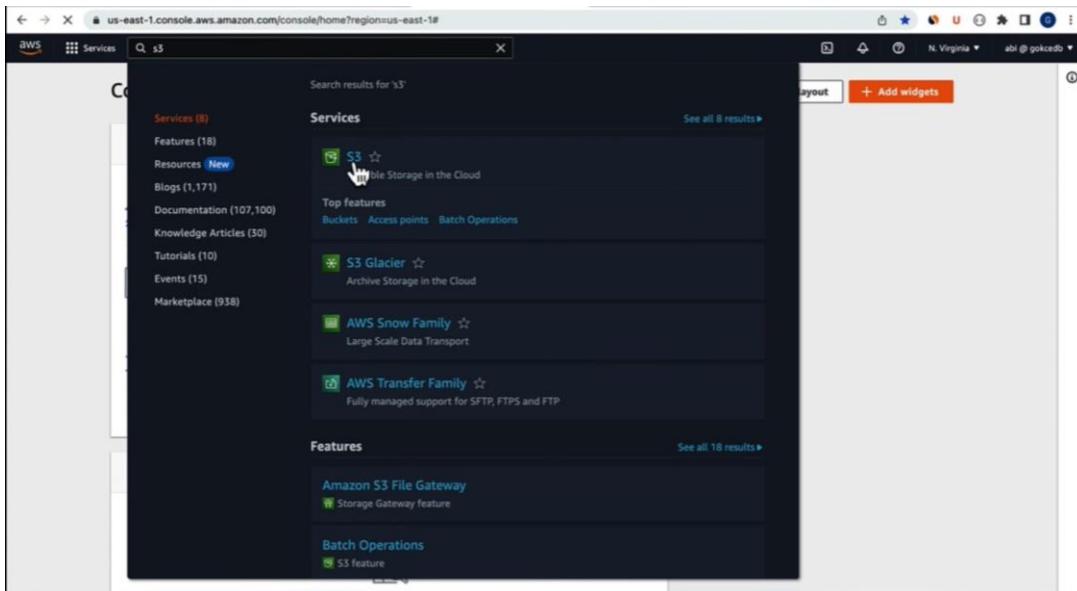
A screenshot of a code editor window titled "lambda_function.py -- ~/Desktop". The left sidebar shows a project structure with files like "\$RECYCLE.BIN", ".DS_Store", "Localized", "Media Preferences.plist", ".Rapp.history", and "lambda_function.py". The main editor area contains the following Python code:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

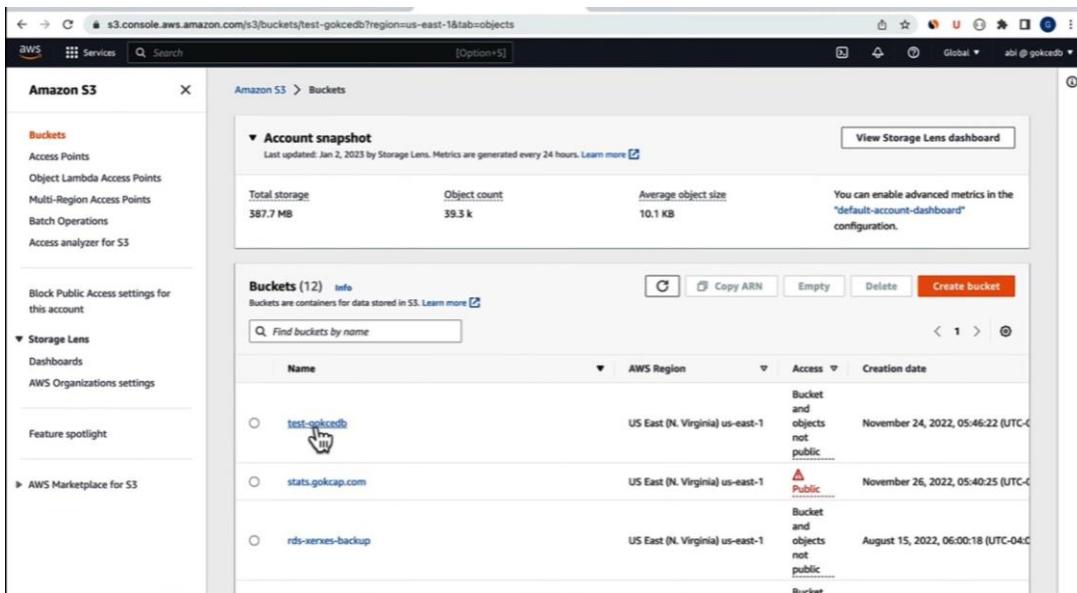
The status bar at the bottom indicates "lambda_function.py 1:1" and "LF 1 deprecation UTF-8 Python GitHub Git (0)".



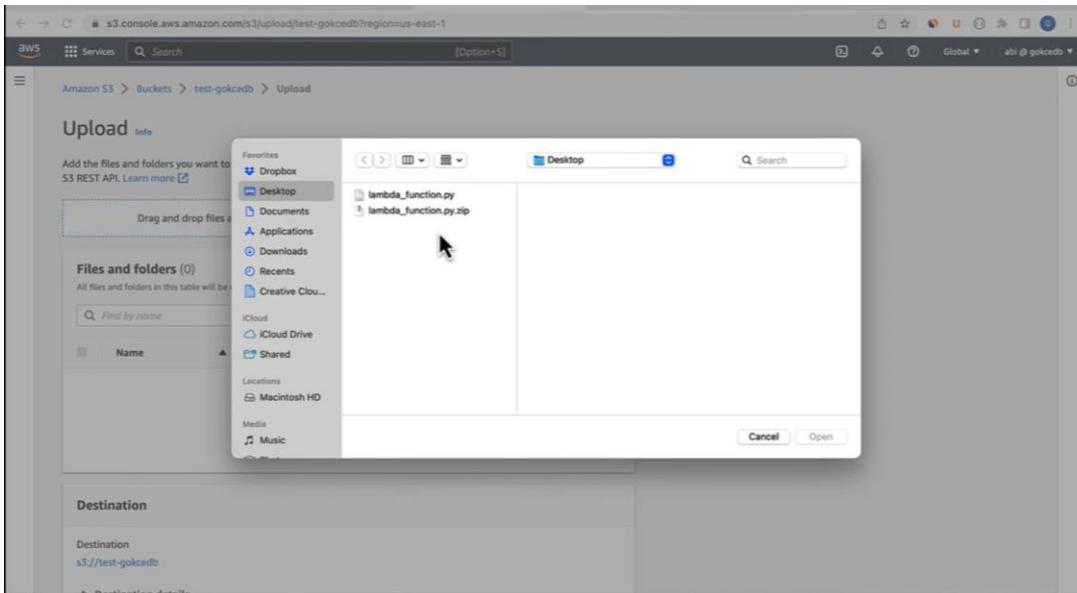
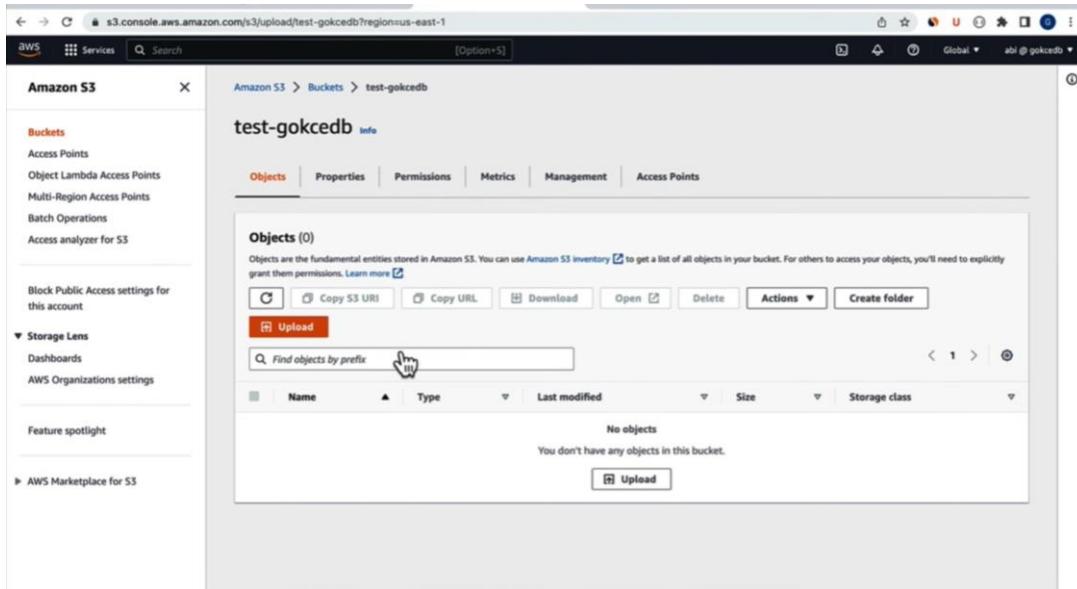
STEP 6: Navigate to S3

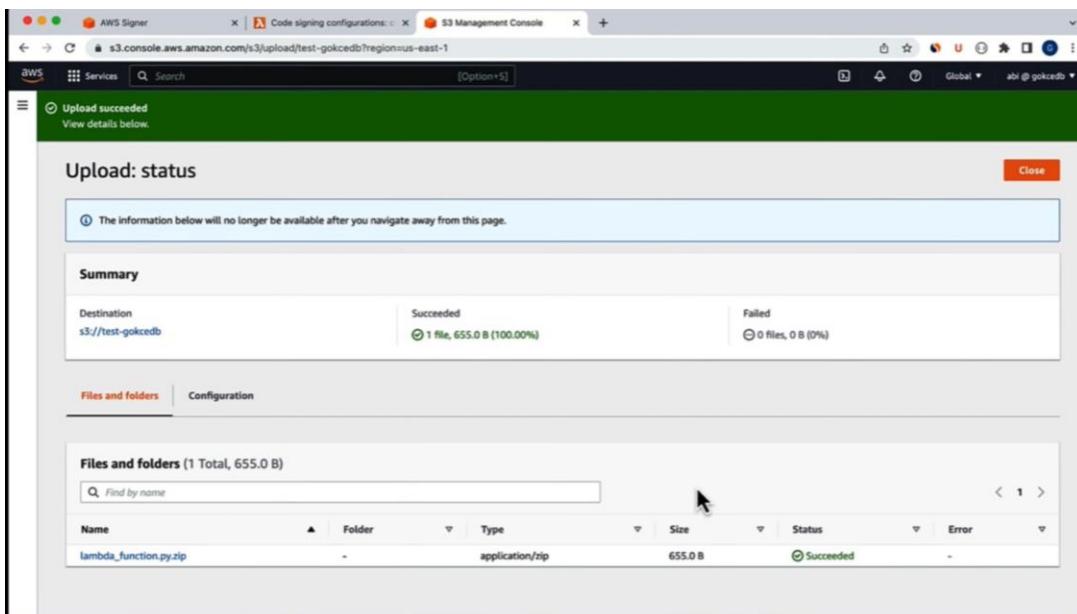
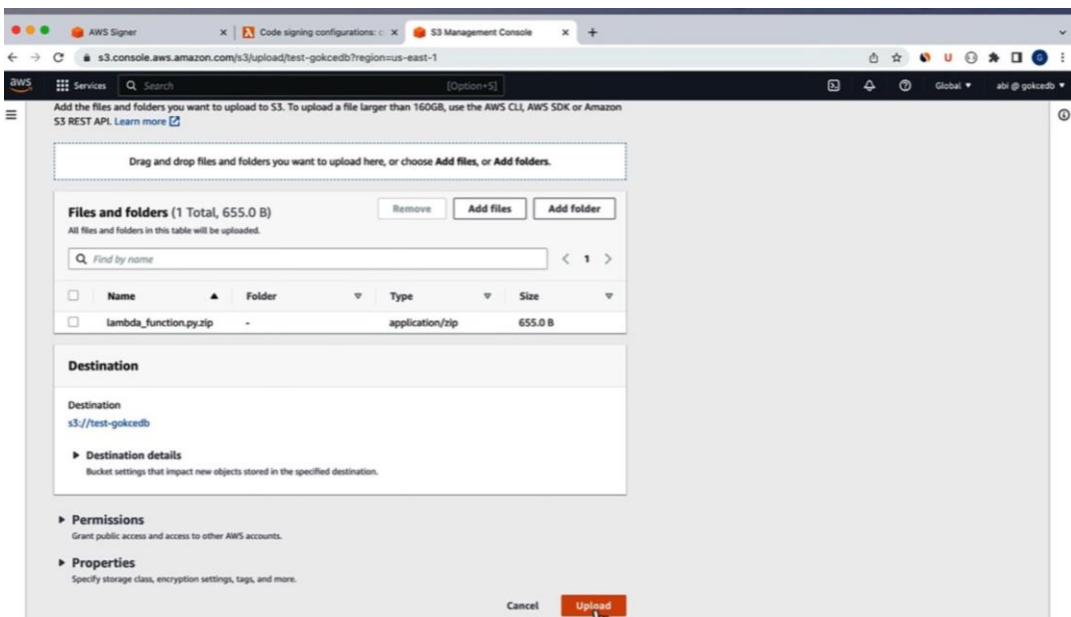


STEP 7: Select an Already Created Bucket



STEP 8: Upload the Zip File





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

The screenshot shows the AWS Signer console with the URL us-east-1.console.aws.amazon.com/signer/home?region=us-east-1#/signing-profiles/myTestSigningProfile. A green success banner at the top states: "Successfully created Profile with ARN: arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile". Below the banner, the profile name is listed as "myTestSigningProfile". The "Start Signing Job" button is highlighted with a cursor icon. Other buttons include "Cancel Profile" and "Revoke Profile". The "Signing Profile details" section shows the following information:

Profile name	myTestSigningProfile	Profile ARN	arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile
Profile version	zwSURRGX9o	Versioned Profile ARN	arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile/zwSURRGX9o
Platform	AWS Lambda	Signature validity period	135 months
Profile status	Active		

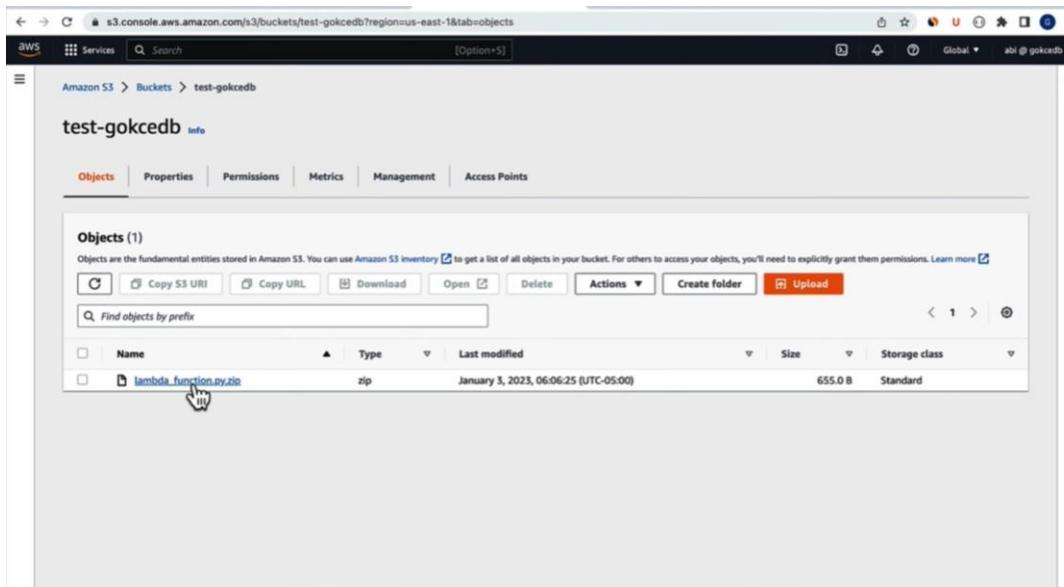
The "Tags" section shows a table with one row: "Tag key" and "Tag value". The message "No tags associated with this resource." is displayed.

The screenshot shows the AWS Signer console with the URL us-east-1.console.aws.amazon.com/signer/home?region=us-east-1#/signing-jobs/start/AWSLambda-SHA384-ECDSA/myTestSigningProfile. A green success banner at the top states: "Successfully created Profile with ARN: arn:aws:signer:us-east-1:928065596197:/signing-profiles/myTestSigningProfile". Below the banner, the path is "AWS Signer > Signing Jobs > Start Signing Job". The main title is "Start Signing Job". A note says: "A Signing Job is an asynchronous process to generate a signature for your code. Starting a Job through the console is only supported for AWS Lambda Platform." The "Signing Job details" section contains the following fields:

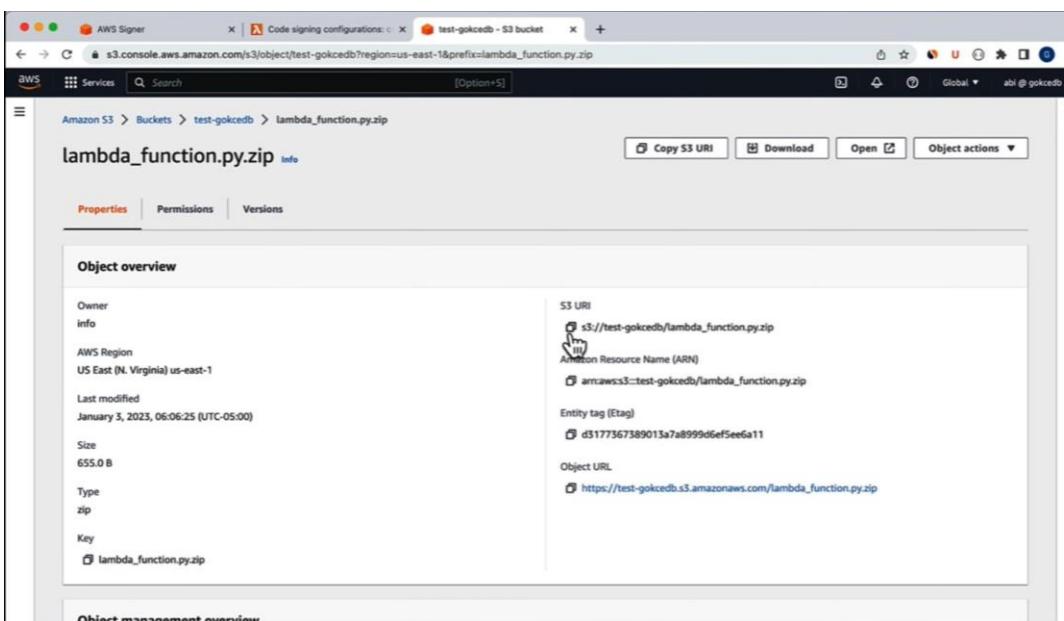
- Signing Profile to use:** Choose the Signing Profile to perform code signing. The dropdown menu shows "myTestSigningProfile".
- Code asset source location:** Specify the S3 location of the code asset to be signed. Only ZIP-formatted assets are accepted.
 - S3Uri:** Input field: "s3://bucket/prefix", dropdown: "Choose a version", buttons: "View" and "Browse S3".
 - Only buckets in current region are displayed. Cross region signing is not supported.
- Signature destination path with prefix:** Specify the S3 path to upload the signature. The signature object name will be the provided prefix appended with the Signing Job ID.
 - S3Uri:** Input field: "s3://bucket/prefix", dropdown: "Choose a version", buttons: "View" and "Browse S3".
 - Only buckets in current region are displayed. Cross region signing is not supported.

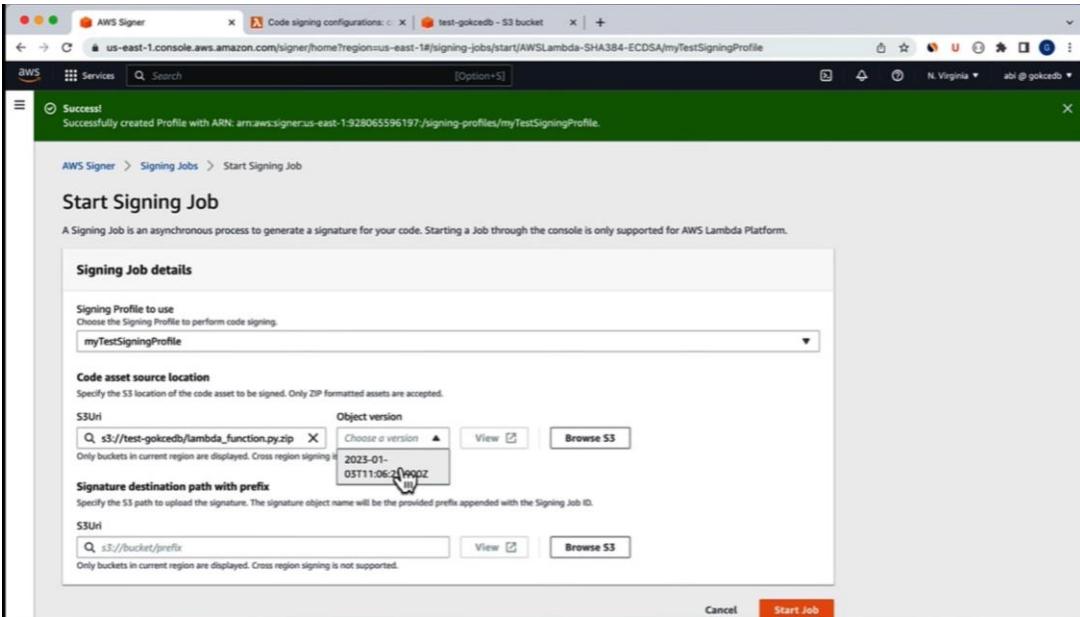
At the bottom right are "Cancel" and "Start Job" buttons.

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

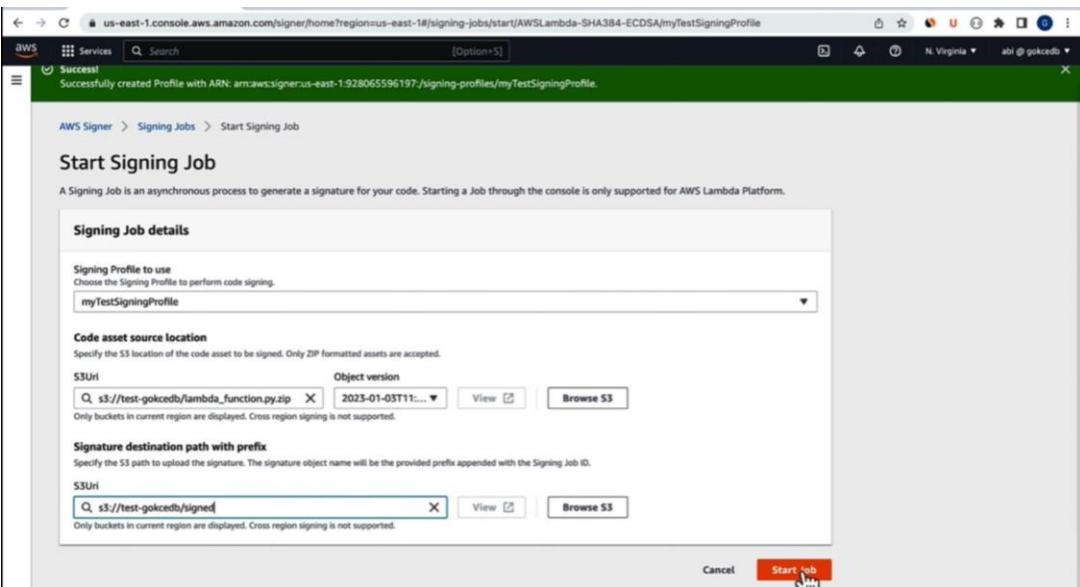


STEP 11: Copy the S3 Url And Paste It In The Signing Job Source Location





STEP 12: Enter the Destination Path



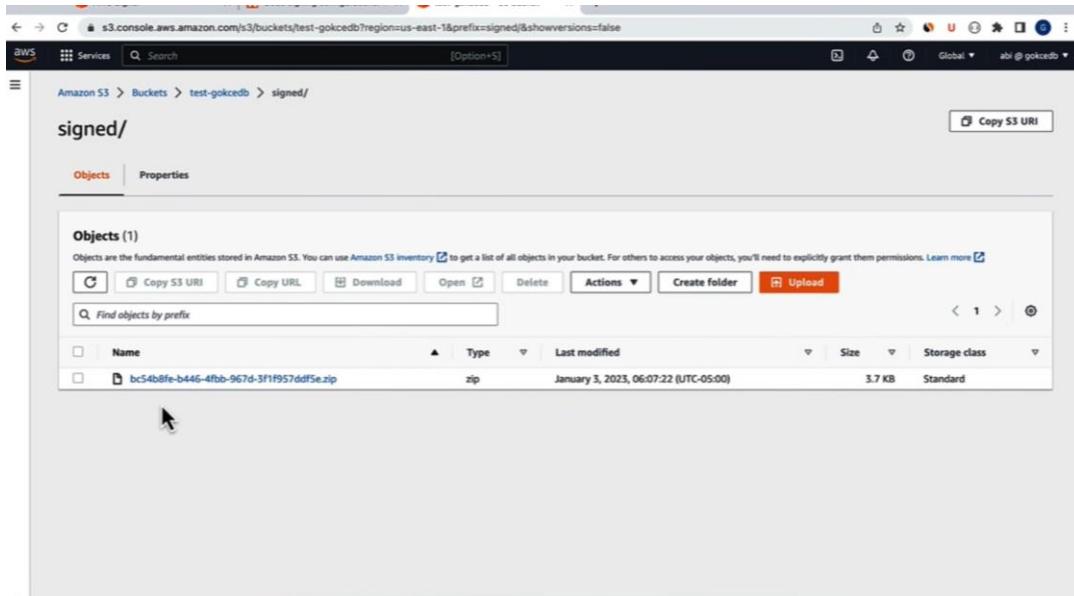
The screenshot shows the AWS Signer web interface. At the top, a green banner indicates a 'Success!' message: 'Successfully started Job with ID: bc54b8fe-b446-4fb9-967d-3f1f957dd5e.' Below this, the 'AWS Signer > Signing Jobs' page is displayed. A table titled 'Signing Jobs (2)' lists two entries:

Job ID	Profile name	Platform	Signing Job status	Signature status
bc54b8fe-b446-4fb9-967d-3f1f957dd5e	myTestSigningProfile	AWS Lambda	Succeeded	Active
ea021ac1-78d1-4240-b7ad-9113dd5f9c69	testSignerProfile	AWS Lambda	Succeeded	Active

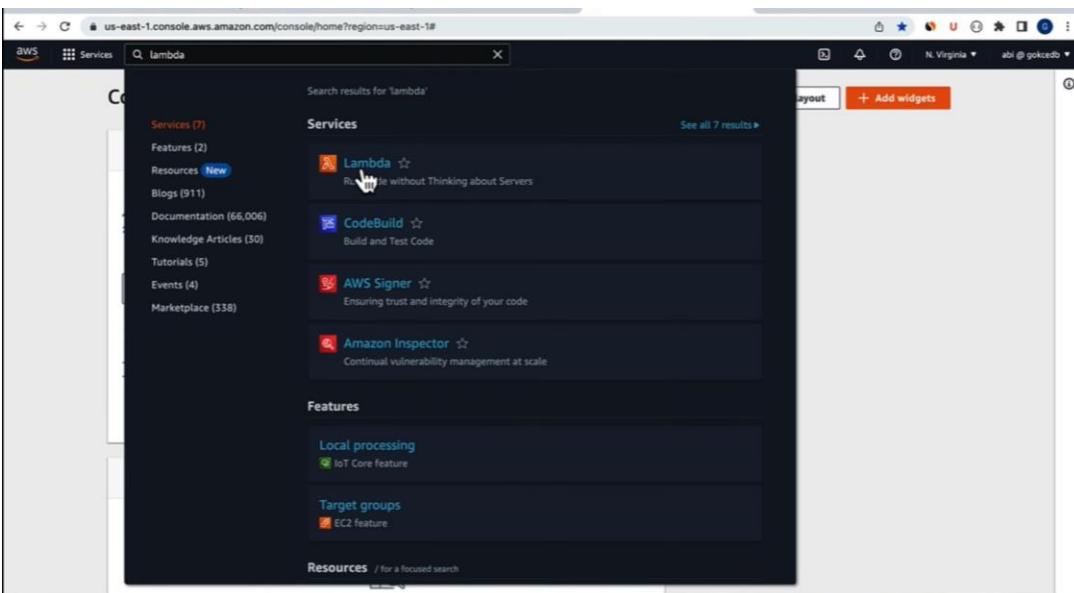
The screenshot shows the AWS S3 web interface. The browser tab is 'Code signing configurations: c | test-gokcedb - S3 bucket'. The main view is 'Amazon S3 > Buckets > test-gokcedb'. The 'Objects (2)' section displays two items:

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	-	-	-

A hand cursor is hovering over the 'signed/' folder entry.



STEP 13: Create A Function in Lambda



Functions (62)						
	Function name	Description	Package type	Runtime	Last modified	
<input type="checkbox"/>	sam-listings-BuyHttpApiFunction-UyOZpeGHDXa	-	Zip	Python 3.9	2 days ago	
<input type="checkbox"/>	sam-gokcedb-ParseGPSCoordinatesHttpApiFunction-bhNXnKxX9kjR	-	Zip	Python 3.9	6 days ago	
<input type="checkbox"/>	sam-gokcedb-InsertPropertyHttpApiFunction-e5bUVB6iUN1e	-	Zip	Python 3.9	6 days ago	
<input type="checkbox"/>	sam-listings-ListingsStatsHttpApiFunction-sfHvk4g9b04I	-	Zip	Python 3.9	2 days ago	
<input type="checkbox"/>	sam-gokcedb-MonthlySalesReportHttpApiFunction-LPhmxNL0CCb	-	Zip	Python 3.9	6 days ago	
<input type="checkbox"/>	sam-gokcedb-AcquisitionReportHttpApiFunction-NkDru4ffDgn	-	Zip	Python 3.9	6 days ago	
<input type="checkbox"/>	sam-listings-PriceReductionRecurringHttpApiFunction-WgwFql4XEBrh	-	Zip	Python 3.9	2 days ago	
<input type="checkbox"/>	sam-gokcedb-LeadReportHttpApiFunction-KbU13v2gJ6H	-	Zip	Python 3.9	6 days ago	
<input type="checkbox"/>	sam-gokcedb-WAFReportHttpApiFunction-tfVfRqcJBIDW	-	Zip	Python 3.9	6 days ago	
<input type="checkbox"/>	sam-gokcedb-CourseSalesReportHttpApiFunction-Beovtrovav7t	-	Zip	Python 3.9	6 days ago	

configuration presets for common use cases.

Basic information

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

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.

▲

Latest supported

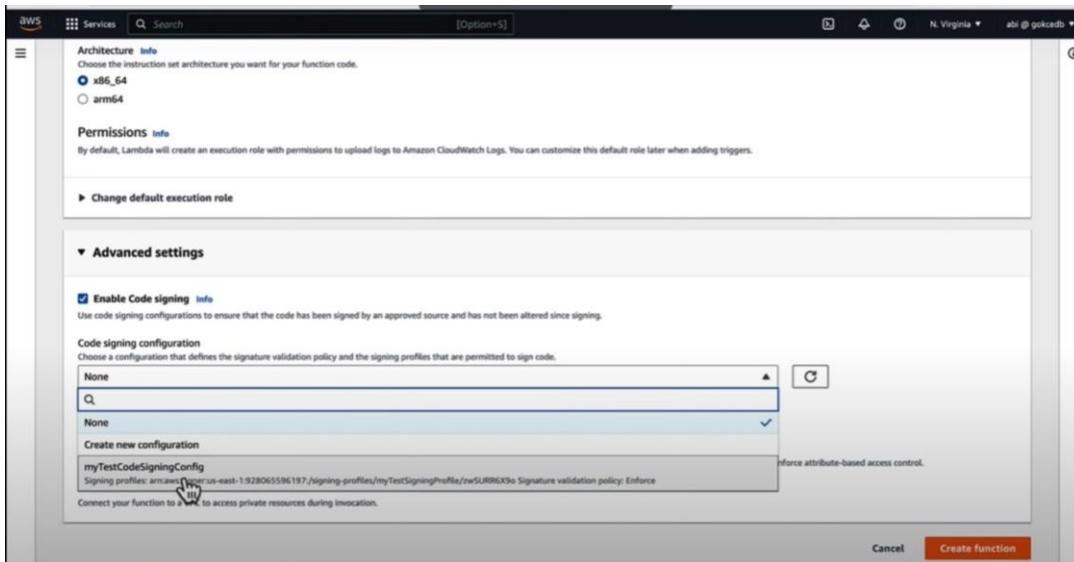
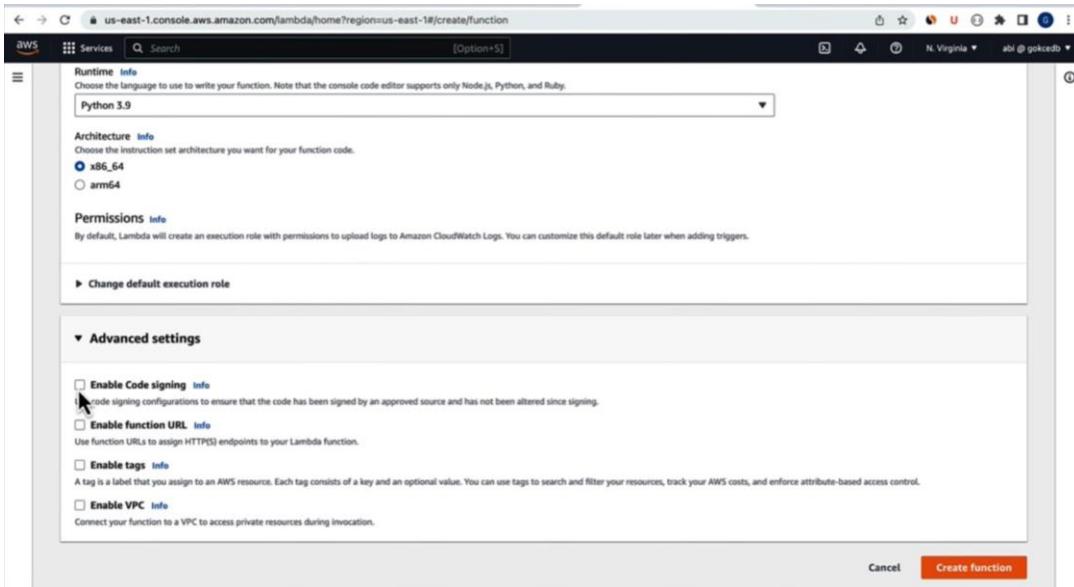
- .NET 6 (.NET/PowerShell)
- .NET Core 3.1 (.NET/PowerShell)
- Go 1.x
- Java 11 (Corretto)
- Node.js 18.x
- Python 3.9 (selected)
- Ruby 2.7

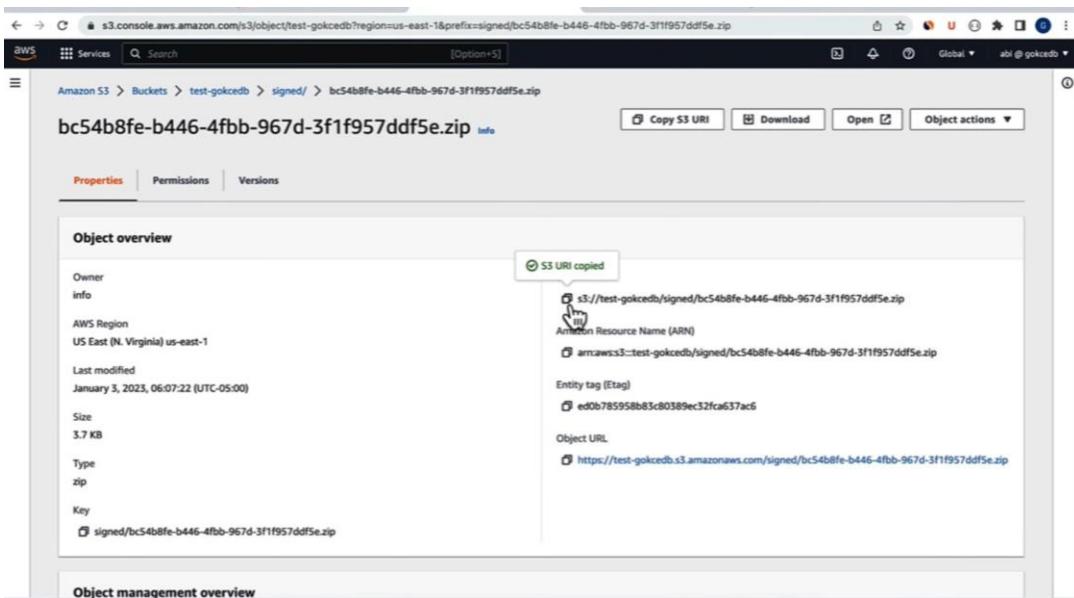
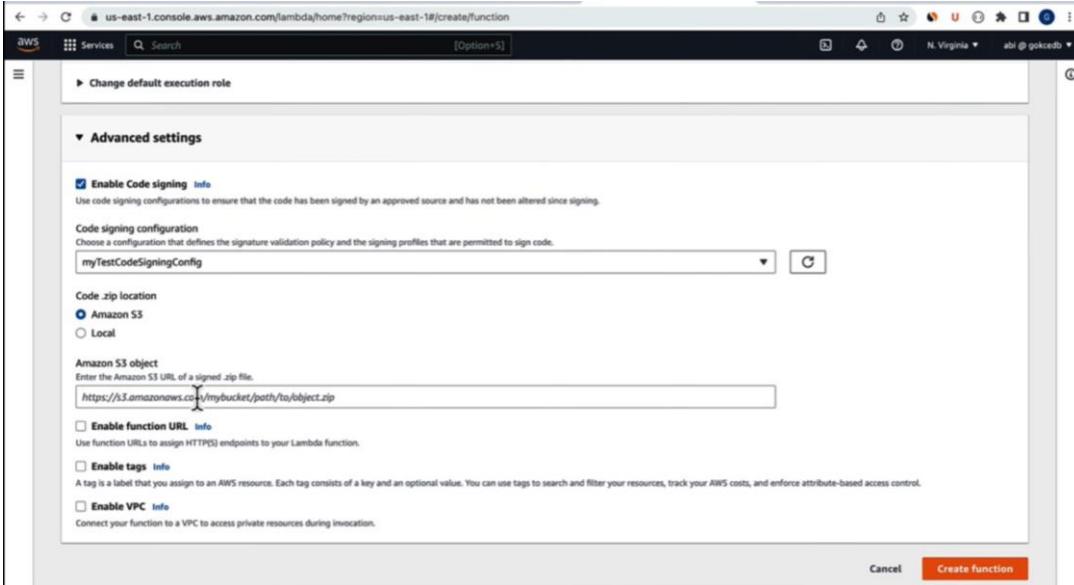
Other supported

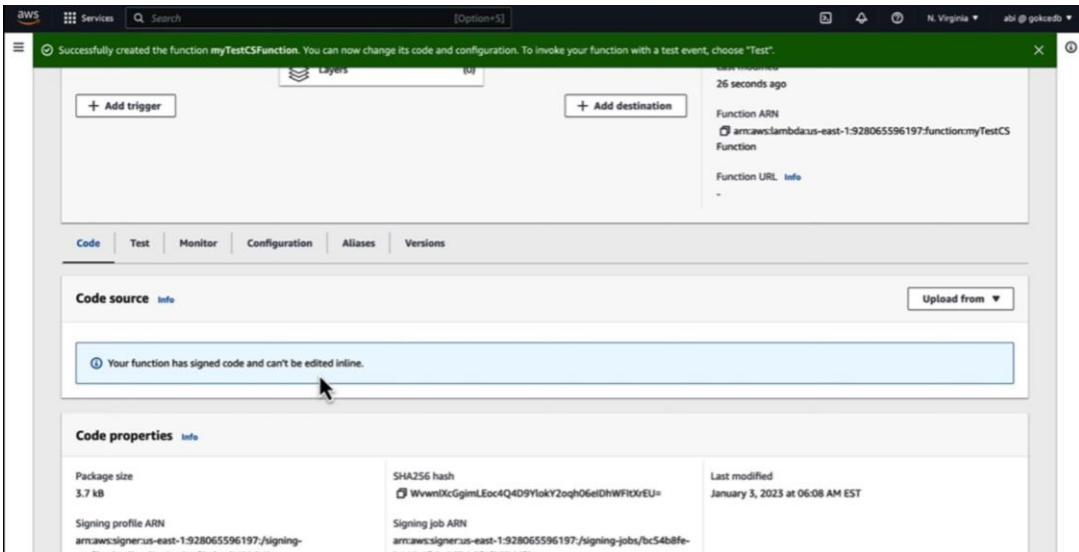
- Java 8 on Amazon Linux 1
- Java 8 on Amazon Linux 2
- Node.js 12.x

Cancel Create function

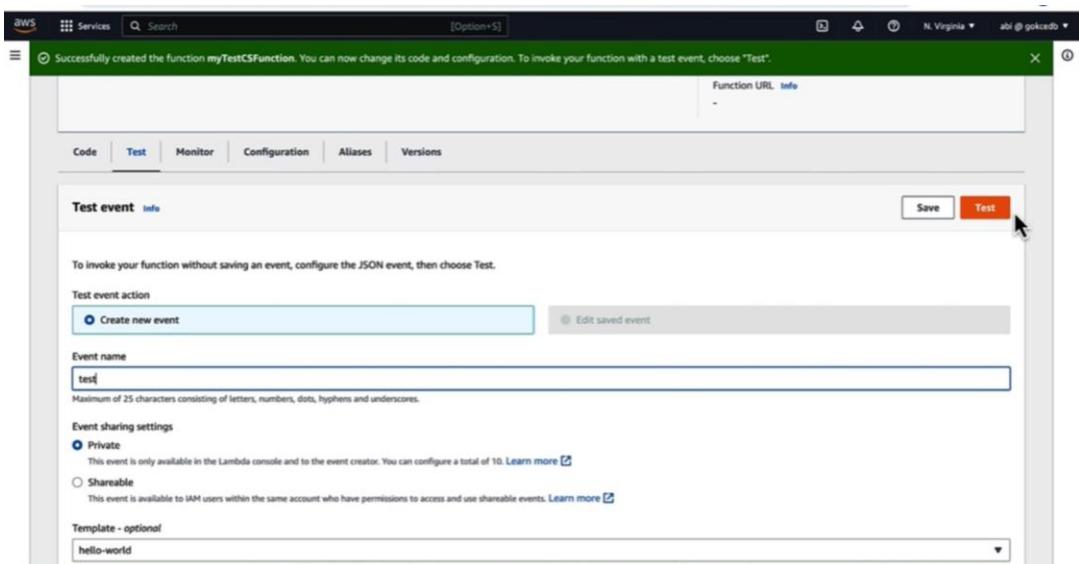
STEP 14: Enable Code Signing







STEP 15: Test the File



The screenshot shows the AWS Lambda console interface. At the top, a message states: "Successfully created the function myTestCSFunction. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below this, there are tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Test tab is selected. A modal window is open, titled "Execution result: succeeded (logs)". It contains a collapsible "Details" section with the sub-section "Logs". The log output is as follows:

```
{  
  "statusCode": 200,  
  "body": "Hello from Lambda!"  
}
```

Below the logs, there is a "Summary" section with the following data:

Code SHA-256	Request ID
WvwnIXcGgimLEoc4Q4D9YlokY2oqh06elDhWFtXrEU=	28675750-4736-42c8-9a5b-083a3c73842d
Init duration	Duration
123.37 ms	1.79 ms
Billed duration	Resources configured
2 ms	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

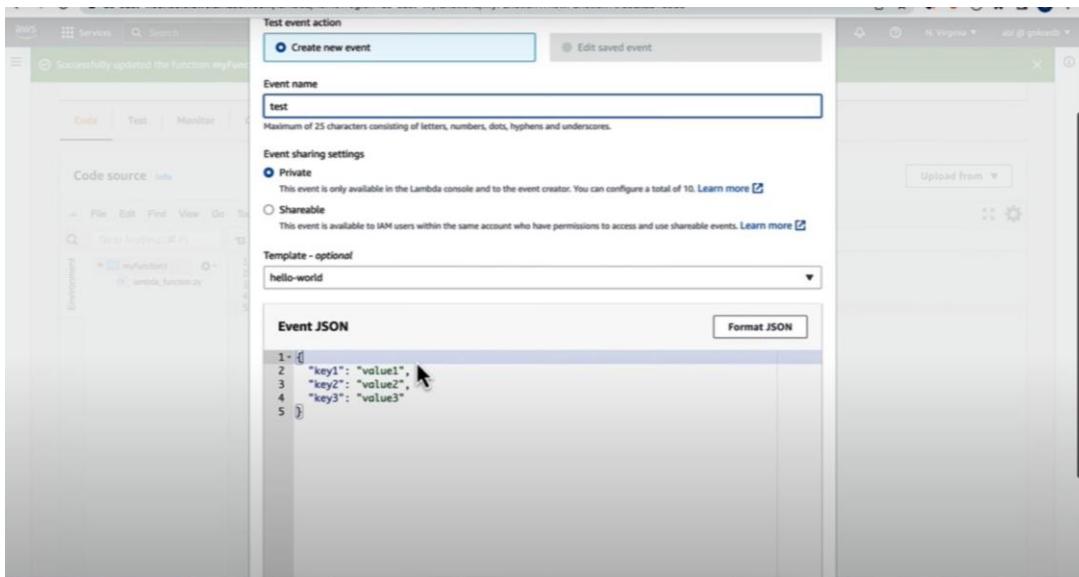
The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with navigation links like Dashboard, Applications, Functions, Additional resources, and Related AWS resources. The main area is titled 'Lambda > Functions' and displays a table of 55 functions. The columns include 'Function name', 'Description', 'Package type', 'Runtime', and 'Last modified'. A red box highlights the 'Create function' button at the top right of the table header.

Function name	Description	Package type	Runtime	Last modified
sam-listings-BuyHttpApiFunction-UyOZpeGHUDxa	-	Zip	Python 3.9	4 days ago
sam-gokcedb-ParseGPSCoordinatesHttpApiFunction-bNDXnOxX9kjR	-	Zip	Python 3.9	19 hours ago
sam-gokcedb-InsertPropertyHttpApiFunction-e5bUV8GiUN1e	-	Zip	Python 3.9	19 hours ago
sam-listings-ListingsStatusHttpApiFunction-shHvk4g9b04l	-	Zip	Python 3.9	4 days ago
sam-gokcedb-MonthlySalesReportHttpApiFunction-LPhmsxtNLOCCb	-	Zip	Python 3.9	19 hours ago
sam-gokcedb-AcquisitionReportHttpApiFunction-NkDru4ffXgn	-	Zip	Python 3.9	19 hours ago
sam-gokcedb-LeadReportHttpApiFunction-K3U13v2gJ6H	-	Zip	Python 3.9	19 hours ago
sam-gokcedb-WAFReportHttpApiFunction-tfVFBqJBDW	-	Zip	Python 3.9	19 hours ago
sam-gokcedb-CourseSalesReportHttpApiFunction-Beovtrovav7t	-	Zip	Python 3.9	19 hours ago
sam-gokcedb-RunQueryHttpApiFunction-NCwdccnE3Ap	-	Zip	Python 3.9	19 hours ago

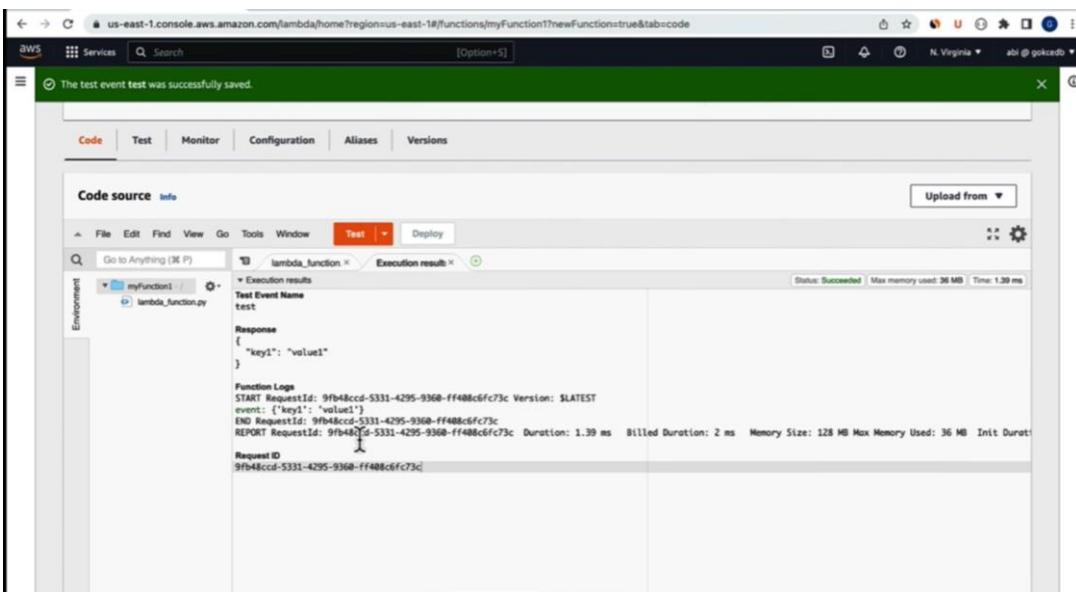
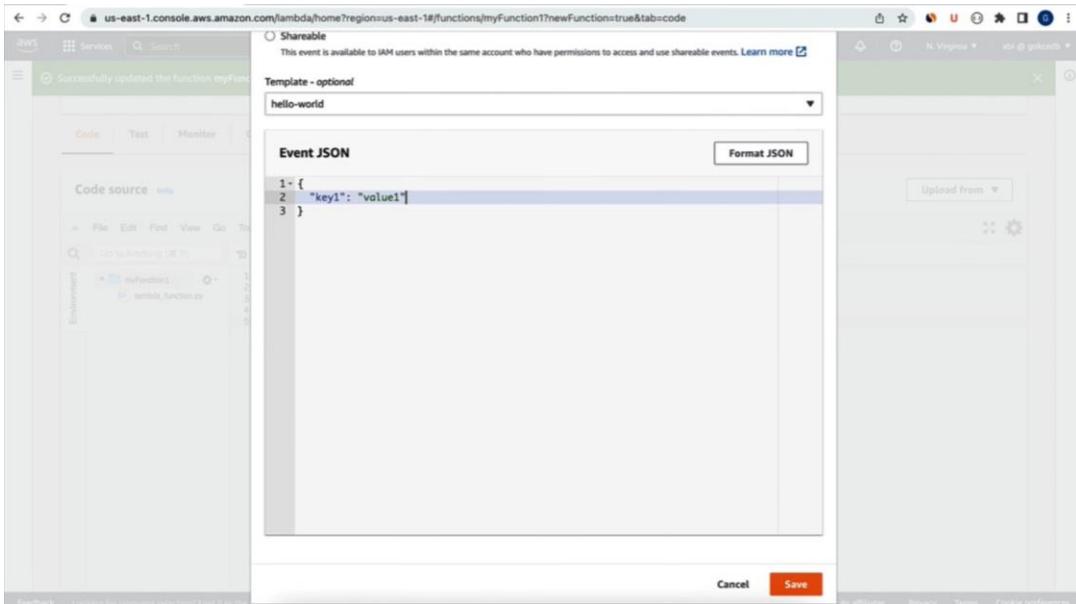
This screenshot shows the 'Basic information' step of the Lambda function creation wizard. It includes fields for 'Function name' (set to 'myFunction1'), 'Runtime' (set to 'Python 3.9'), 'Architecture' (set to 'x86_64'), and 'Permissions' (which is currently collapsed). At the bottom right, there are 'Cancel' and 'Create function' buttons, with the latter being highlighted by a red box.

The screenshot shows the AWS Lambda console interface. At the top, the URL is us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/myFunction1?newFunction=true&tab=code. The browser title bar says "Updating the function myFunction1". The main area has tabs for "Code", "Test", "Monitor", "Configuration", "Aliases", and "Versions". The "Code" tab is selected. On the left, there's a sidebar with "Environment" and a file tree showing "myFunction1" with "lambda_function.py" selected. The main pane displays the Python code:

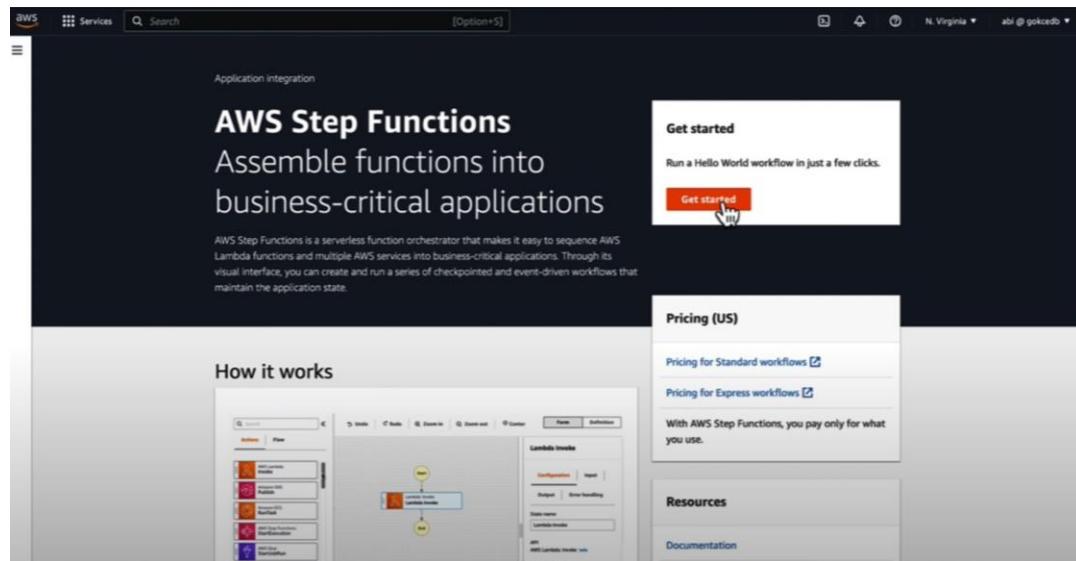
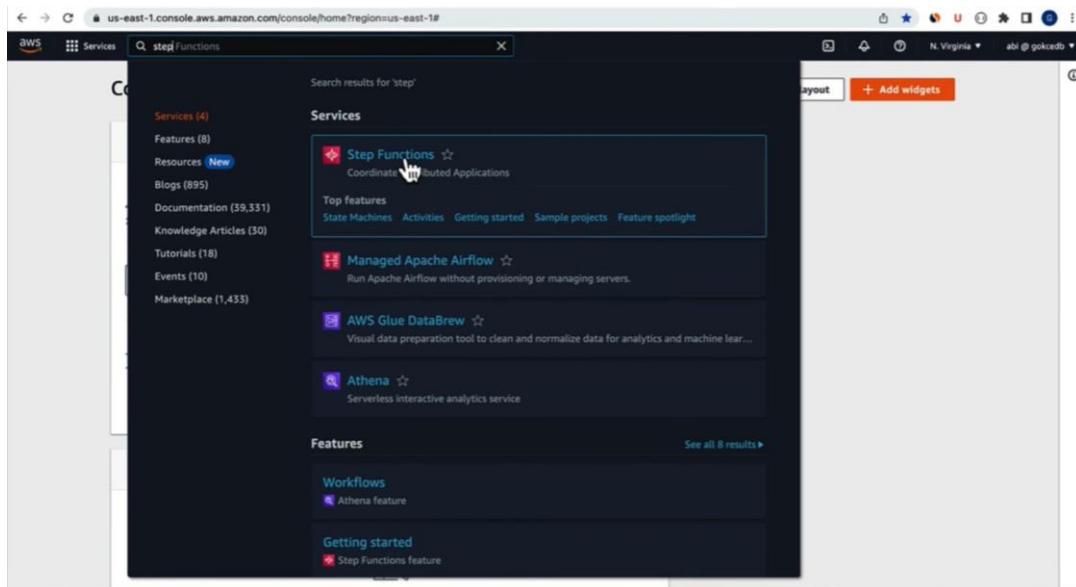
```
1 import json
2
3 def lambda_handler(event, context):
4     print("event:", event)
5     return event
```



STEP 2: Give A Single Key



STEP 3: Search Step Functions



STEP 4: Go to State Machines

```
1+ {
2+   "Comment": "A Hello World example demonstrating various state types
3+   of the Amazon States Language",
4+   "StartAt": "Pass",
5+   "States": {
6+     "Pass": {
7+       "Comment": "A Pass state passes its input to its output, without
8+       performing work. Pass states are useful when constructing and
9+       debugging state machines.",
10+      "Type": "Pass",
11+      "Next": "Hello World example?"
12+    },
13+    "Hello World example?": {
14+      "Comment": "A Choice state adds branching logic to a state
15+      machine. Choice rules can implement 16 different comparison operators,
16+      and can be combined using And, Or, and Not",
17+      "Type": "Choice",
18+      "Choices": [
19+        {
20+          "Variable": "$.IsHelloWorldExample",
21+          "BooleanEquals": true,
22+          "Next": "Yes"
23+        },
24+        {
25+          "Variable": "$.IsHelloWorldExample"
26+        }
27+      ]
28+    }
29+  }
30+}
```

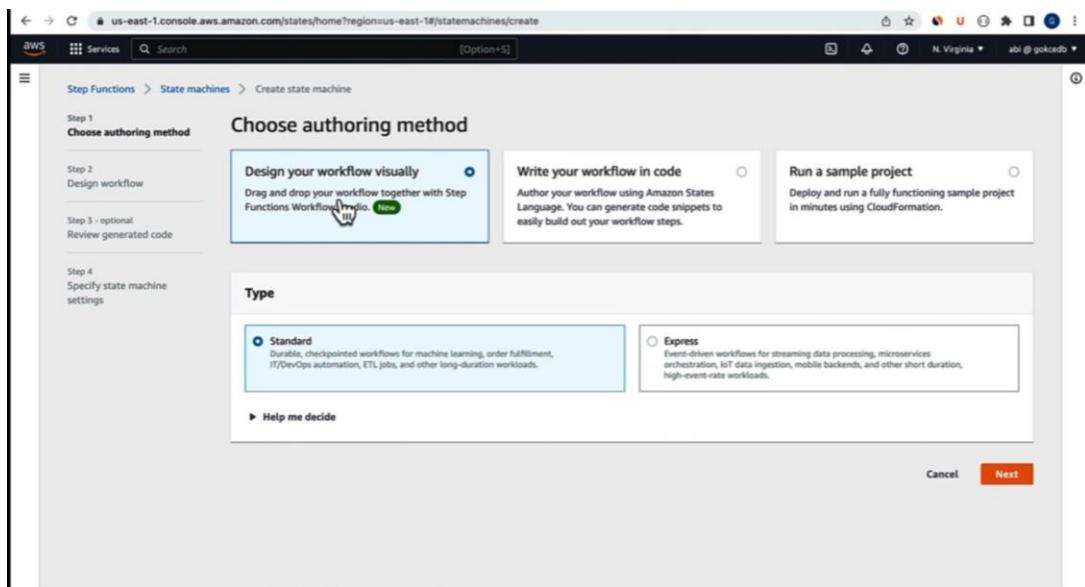
The state machine diagram shows:

- Start** → **Pass**
- Pass** → **Hello World example?**
- Hello World example?** branches to:
 - Yes** → **Wait 1 sec** → **Hello** and **World** (parallel states) → **End**
 - No** → **End**

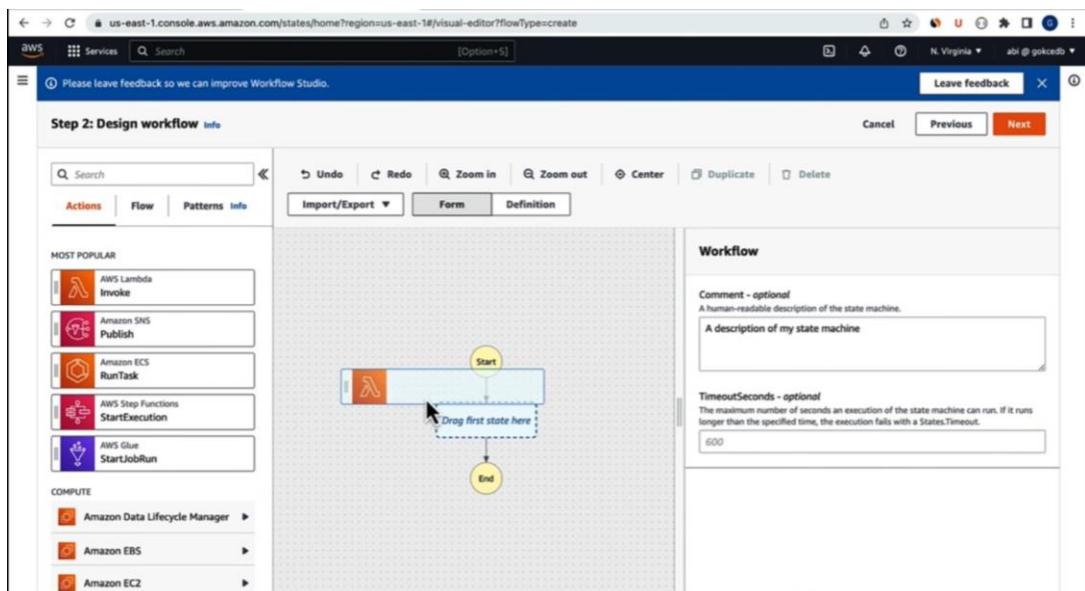
Create A State Machine

Name	Type	Creation date	Status	Logs	Total	Running	Succeeded	Failed	Timed out	Aborted
No state machines										

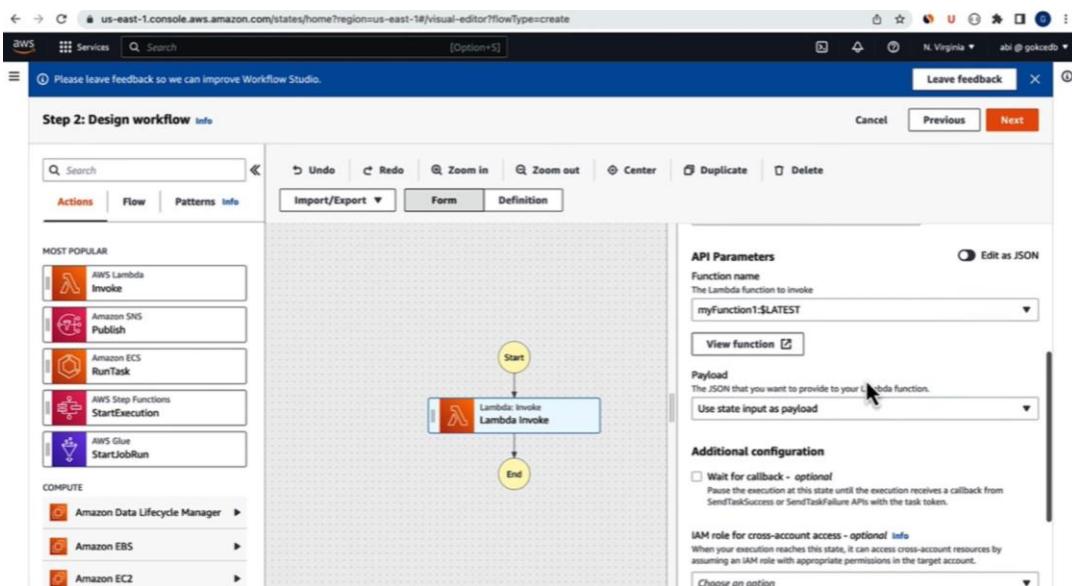
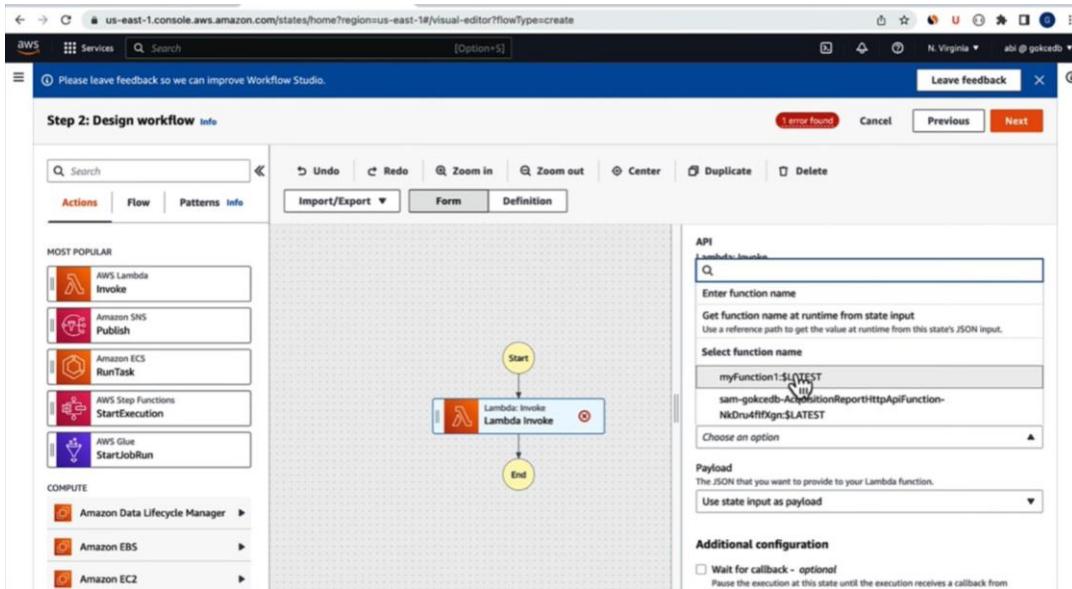
Create state machine



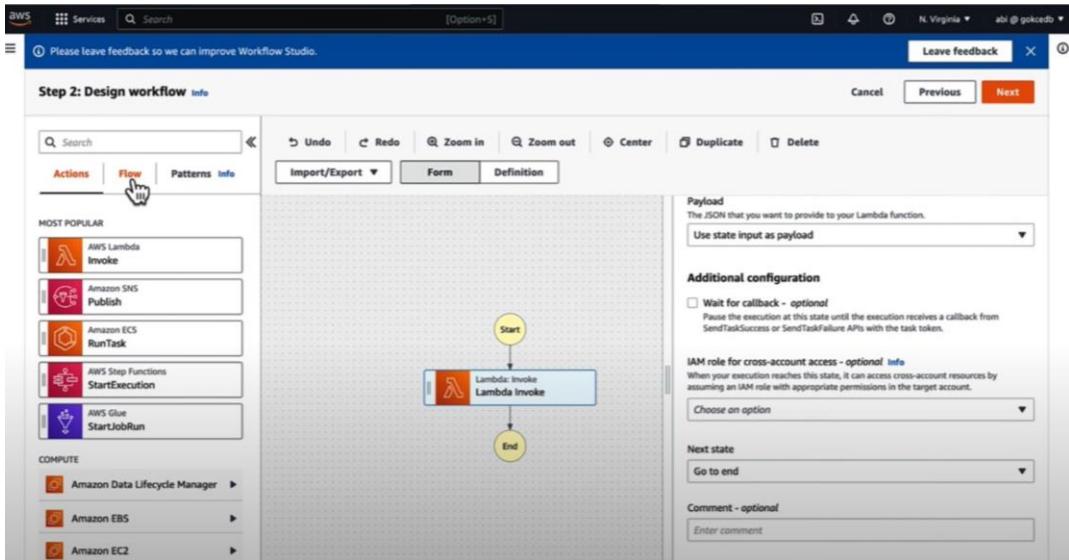
STEP 5: Drag and Drop The Lambda



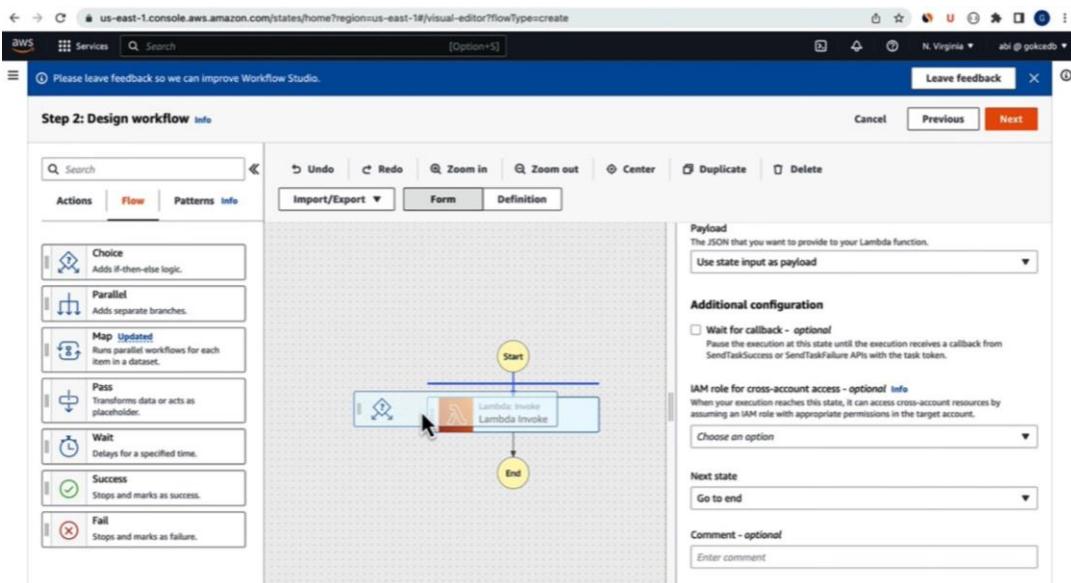
STEP 6: Select the Function Name



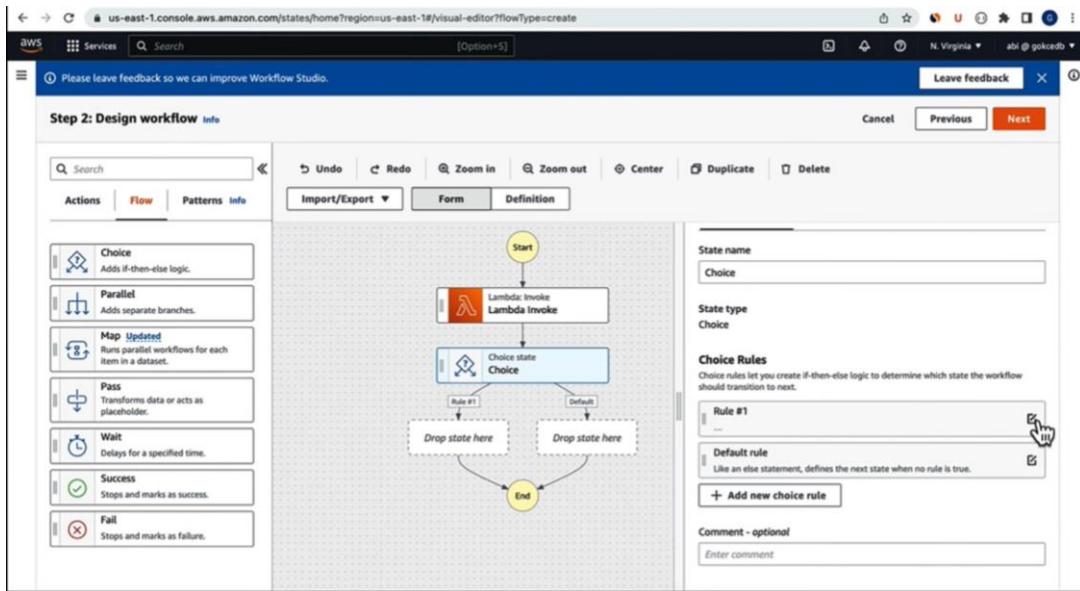
STEP 7: Go to Flow



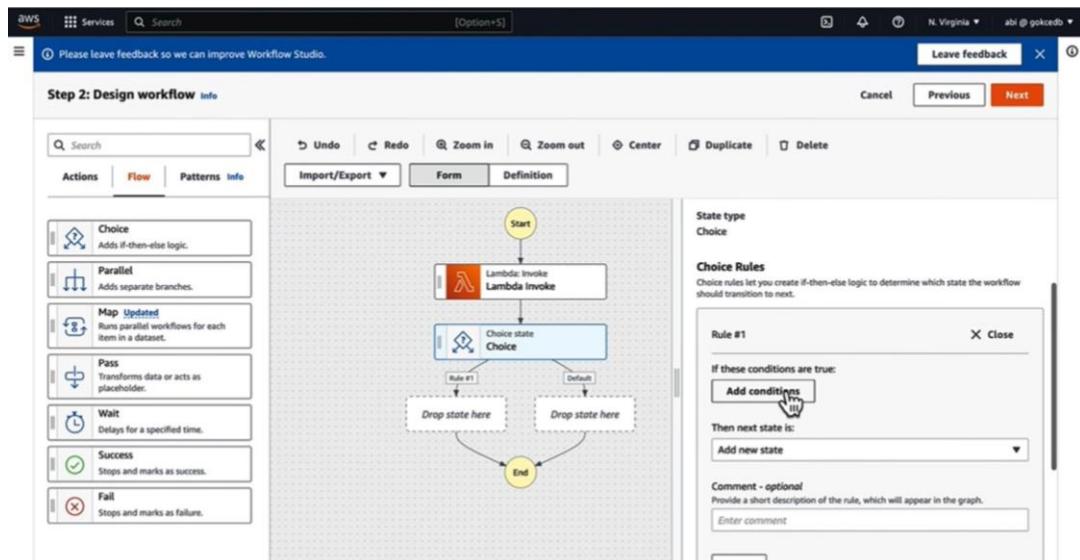
STEP 8: Drag and Drop The Choice



STEP 9: Go to The Choice Rules



STEP 10: Add Conditions



Step 2: Design workflow [Info](#)

Conditions for rule #1

Choice rules contain conditional statements, which are used to evaluate one or more node values (called variables) in your state's JSON input. Learn more [? \[x\]](#)

Simple
Evaluates a single conditional statement.

Not	Variable	Operator	Value
NOT	\$key1	matches string	true

Must use JsonPath.

Success
Stops and marks as success.

Fail
Stops and marks as failure.

Add new state

Comment - optional
Provide a short description of this rule, which will appear in the graph.

Cancel Save conditions

STEP 11: Drag and Drop The Fail

Please leave feedback so we can improve Workflow Studio.

Step 2: Design workflow [Info](#)

Actions **Flow** Patterns Info

Import/Export Form Definition

Start **Lambda Invoke Lambda Invoke** Choice state **Choice** End

should transition to next.

Rule #1 [Close](#)

If these conditions are true:

```
not($key1 == "true")
```

[Edit conditions](#)

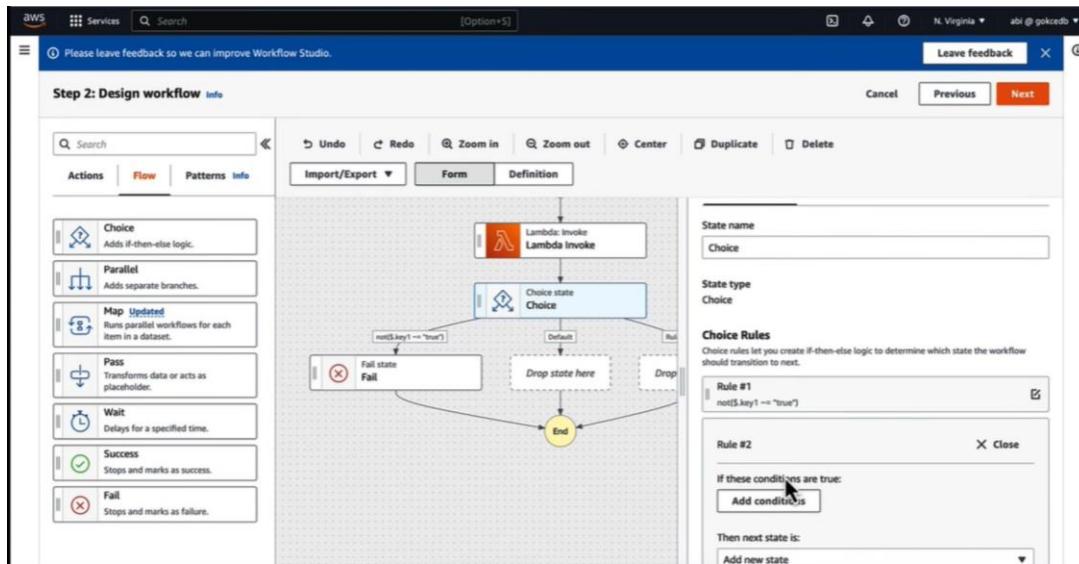
Then next state is:

Add new state

Comment - optional
Provide a short description of the rule, which will appear in the graph.

Default rule

STEP 12: Add Conditions

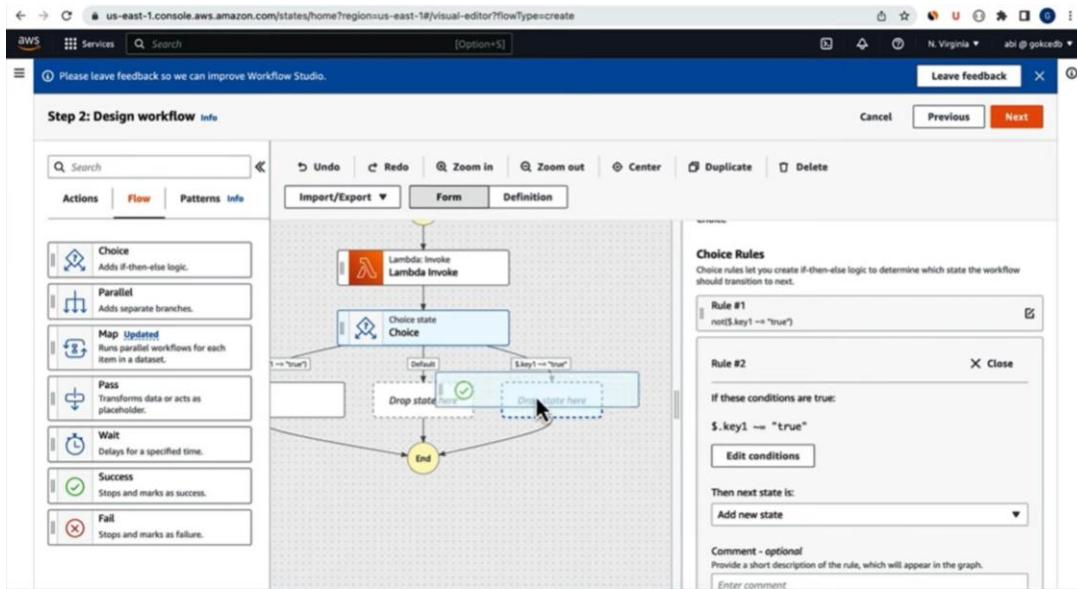


This screenshot shows the AWS Step Functions Management console. It's a modal window titled 'Conditions for rule #2'. The instructions say: 'Choice rules contain conditional statements, which are used to evaluate one or more node values (called variables) in your state's JSON input. Learn more [?]' Below this, there's a 'Simple' section with the sub-instruction 'Evaluates a single conditional statement.' A table is provided for defining the condition:

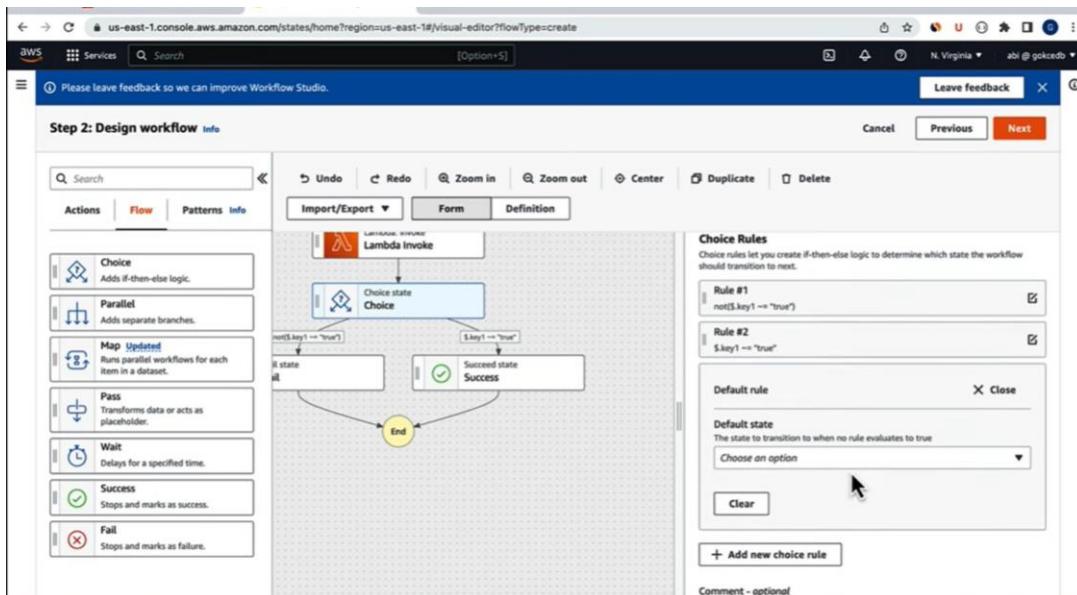
Not	Variable	Operator	Value
<input checked="" type="checkbox"/>	\$key1	matches string	true

Below the table, it says 'Must use JsonPath.' At the bottom right of the modal, there are 'Cancel' and 'Save conditions' buttons, with a hand cursor hovering over 'Save conditions'. The background of the modal shows a preview of the workflow graph with a 'Success' and 'Fail' state option.

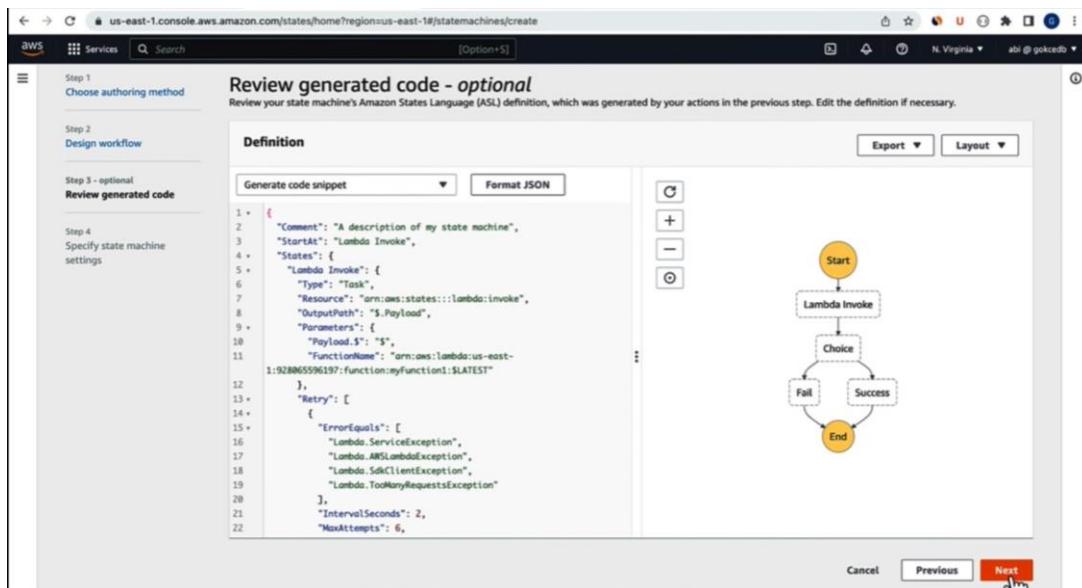
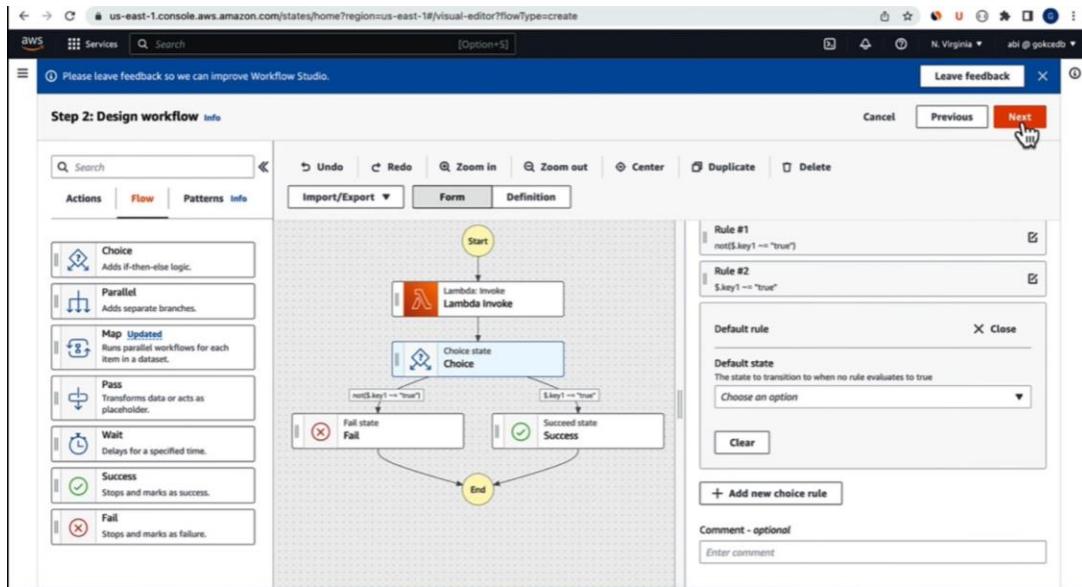
STEP 13: Drag and Drop The Success



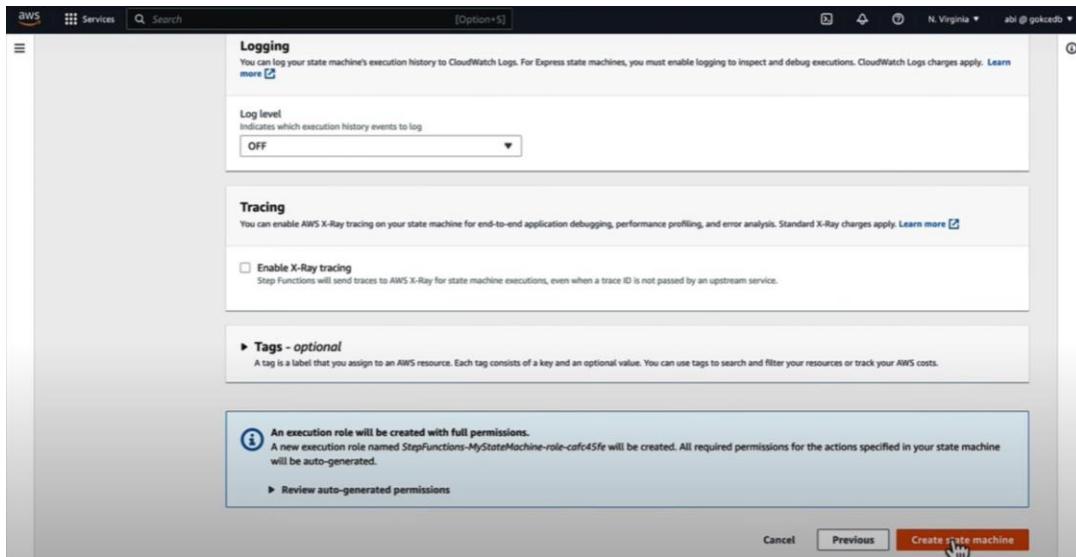
STEP 14: Close Default



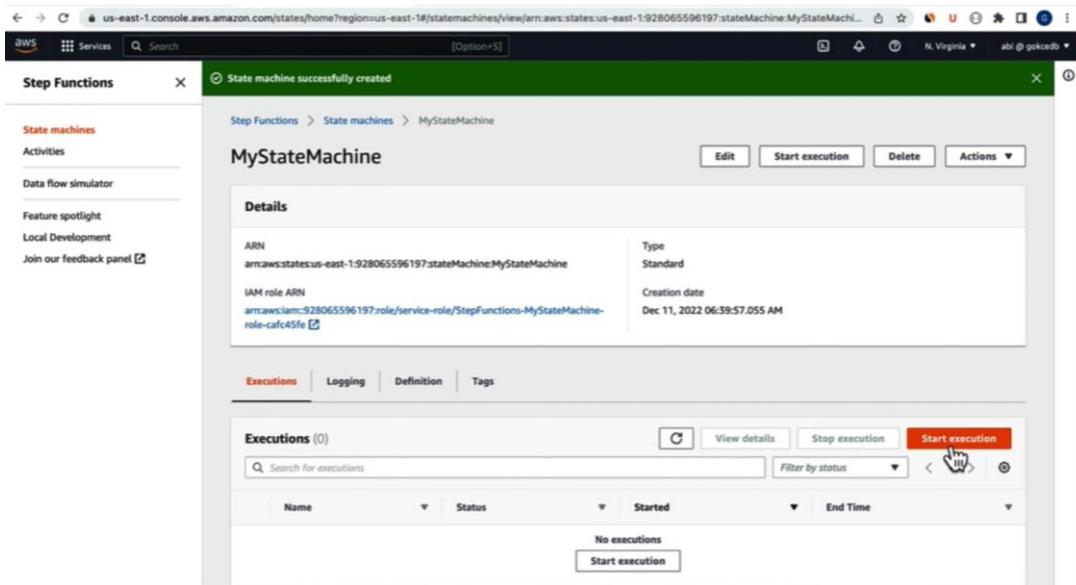
STEP 15: Click Next



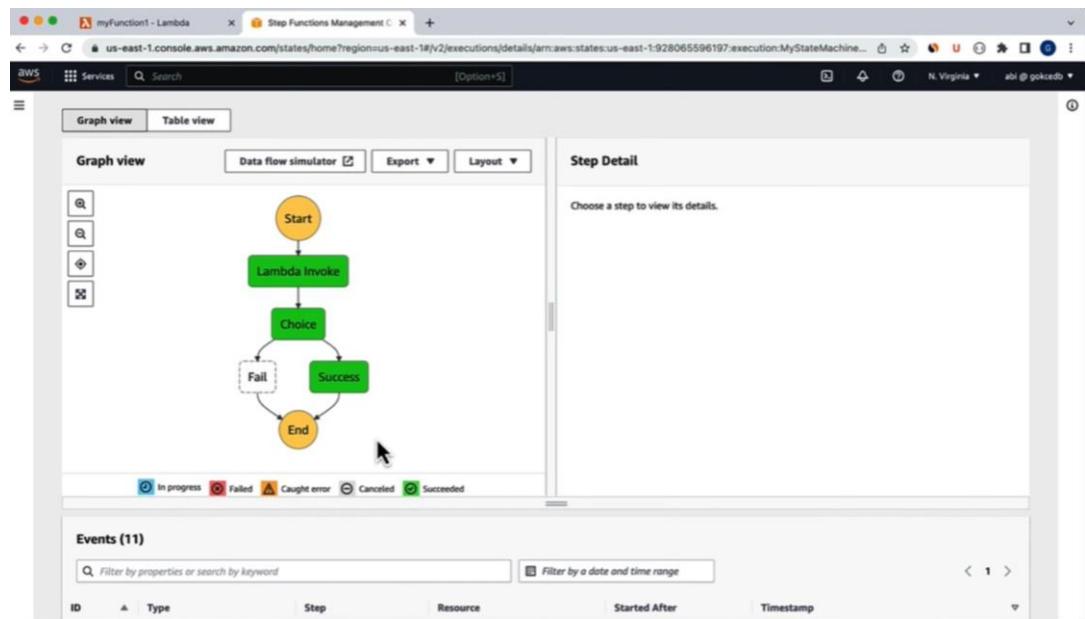
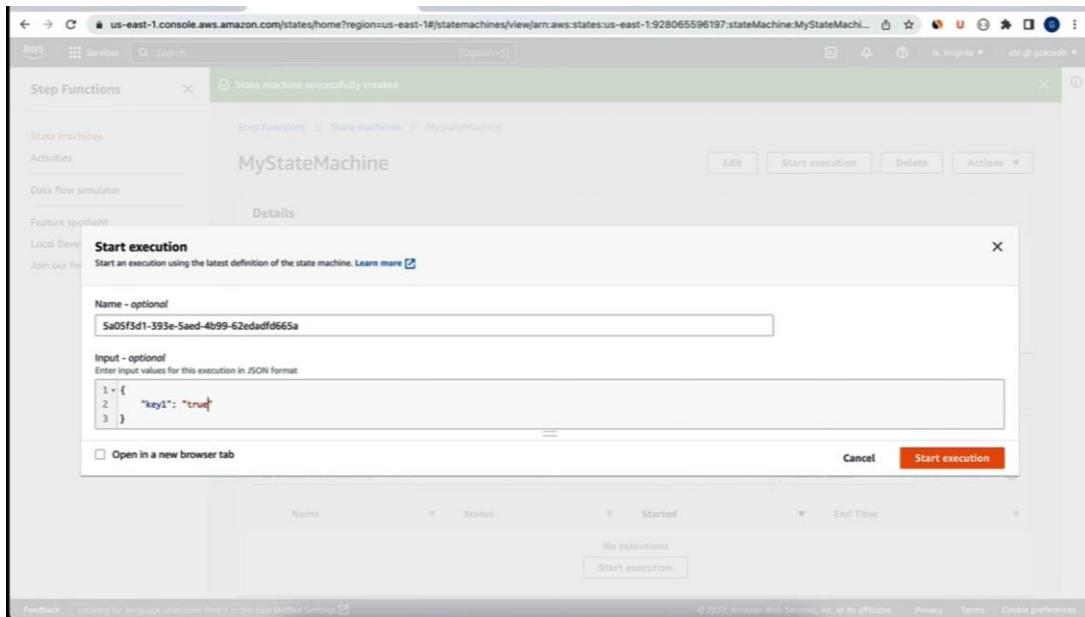
STEP 16: Create State Machine



STEP 17: Start Execution



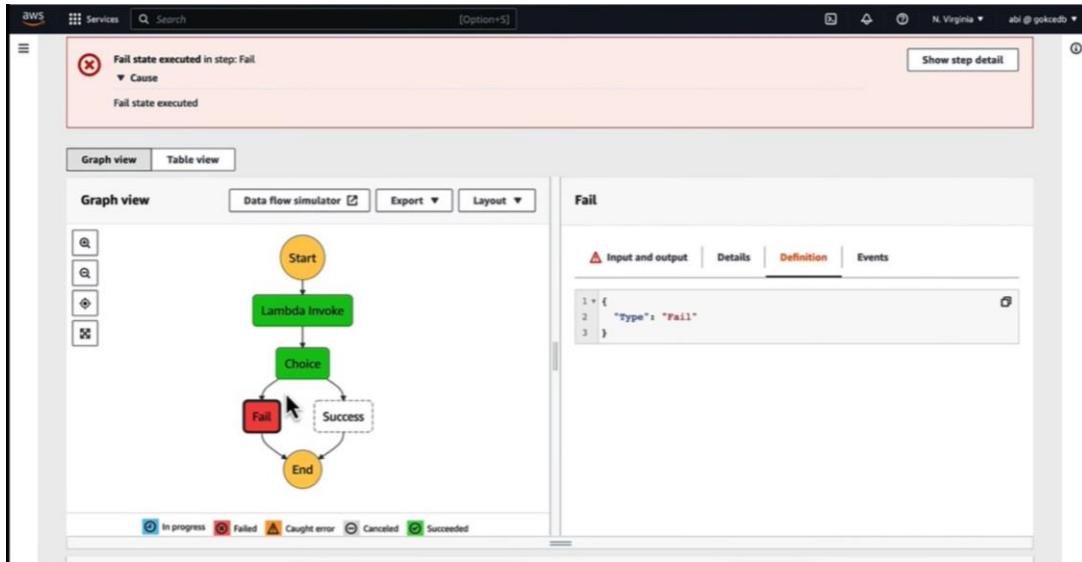
STEP 18: Give the Key Value True And Check



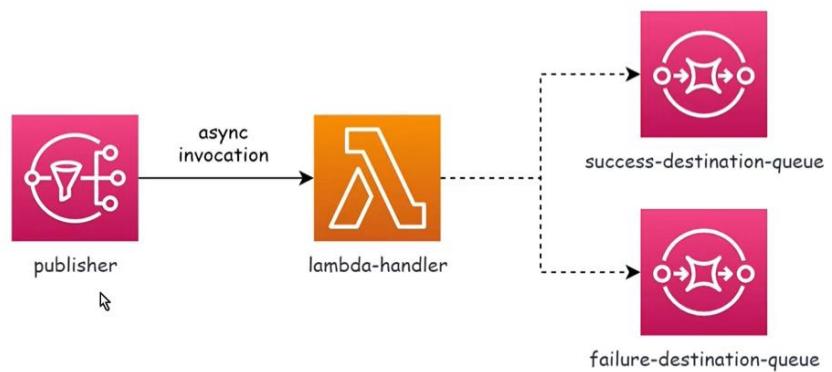
The screenshot shows the AWS Lambda execution details page. The execution status is Succeeded. The execution type is Standard. The execution ARN is arn:aws:states:us-east-1:928065596197:execution:MyStateMachine:5a05f3d1-393e-5aed-4b99-62edadfd665a. The start time is Dec 11, 2022, 06:40:07.965 (EST) and the end time is Dec 11, 2022, 06:40:08.233 (EST). The duration is 00:00:00.268. The state transitions section shows a single transition labeled 'S'.

STEP 19: Give the Key Value False

The screenshot shows the 'Start execution' dialog box. The name field contains 'bd64cc4-40d4-4643-b24a-d9a757ae7bc7'. The input field contains the JSON value '{ "key1": "false" }'. The 'Start execution' button is visible at the bottom right.



ADD DESTINATION IN AWS LAMBDA WORKFLOW DIAGRAM



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

The screenshot shows the AWS SNS service interface. In the top navigation bar, 'Amazon SNS' is selected. Below it, the 'Topics' section is highlighted. On the left sidebar, 'Topics' is also selected. The main content area displays a single topic named 'publisher'. The 'Details' section shows the following information:

Name	Display name
publisher	-

Other details shown include:

- ARN: arn:aws:sns:us-east-1:755314965794:publisher
- Type: Standard
- Topic owner: 755314965794

Buttons for 'Edit', 'Delete', and 'Publish message' are visible at the top right of the topic card.

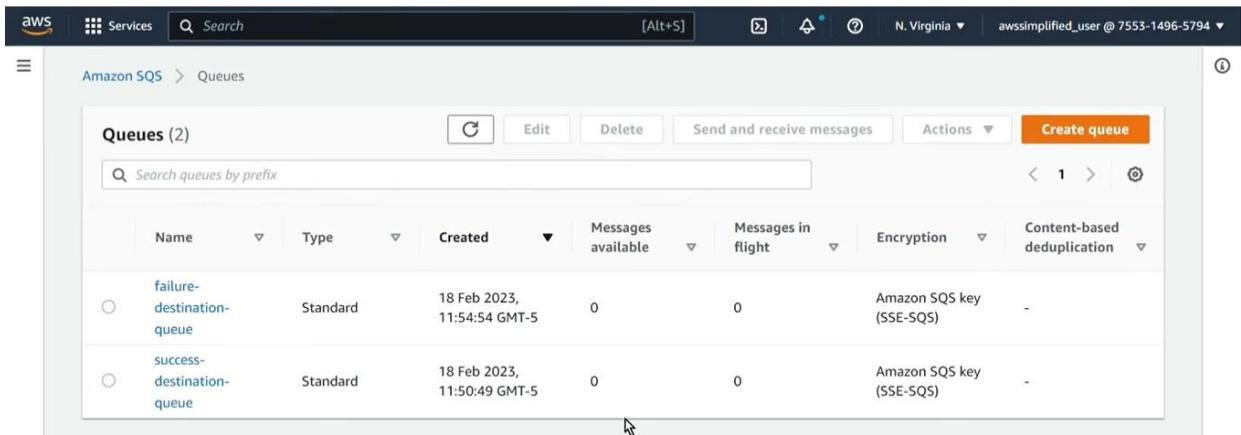
STEP 2: Change “numRetries” to 0.

The screenshot shows the configuration page for the 'publisher' topic. The left sidebar shows 'Topics' is selected. The main content area displays the topic's ARN and type. Below this, the 'Delivery retry policy (HTTP/S)' tab is selected. The policy configuration is as follows:

```
{  
  "http": {  
    "defaultHealthyRetryPolicy": {  
      "minDelayTarget": 20,  
      "maxDelayTarget": 20,  
      "numRetries": 0  
    }  
  }  
}
```

The 'Delivery retry policy (HTTP/S)' tab is highlighted in blue, indicating it is active. Other tabs like 'Subscriptions', 'Access policy', 'Data protection policy', and 'Delivery status' are also visible.

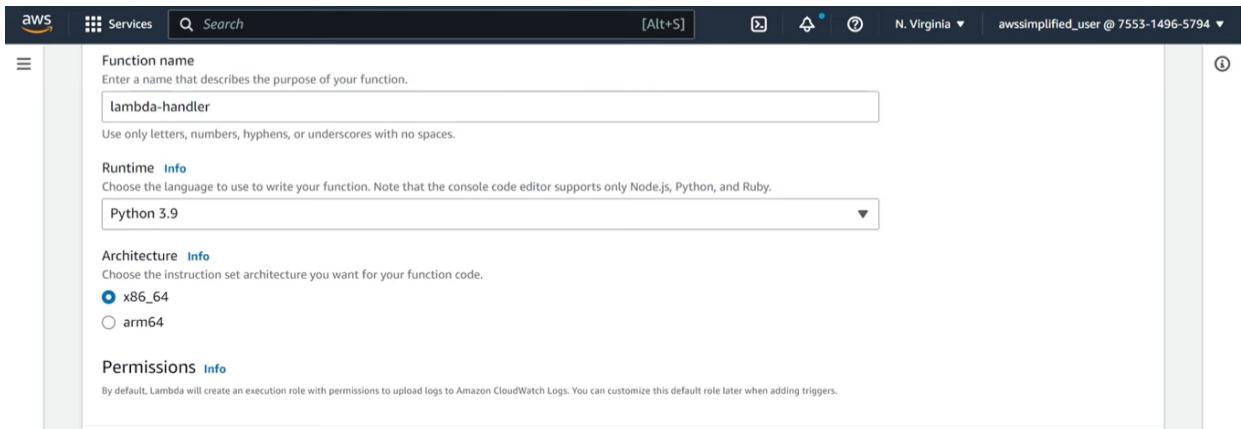
STEP 3: Open SQS service, create two queues namely “success-destination-queue” and “failure-destination-queue”.



The screenshot shows the AWS SQS service interface. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, a search bar, and account information ('N. Virginia' and 'awssimplified_user @ 7553-1496-5794'). Below the navigation is a breadcrumb trail: 'Amazon SQS > Queues'. The main area is titled 'Queues (2)' with a 'Create queue' button. A table lists two queues:

Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication
failure-destination-queue	Standard	18 Feb 2023, 11:54:54 GMT-5	0	0	Amazon SQS key (SSE-SQS)	-
success-destination-queue	Standard	18 Feb 2023, 11:50:49 GMT-5	0	0	Amazon SQS key (SSE-SQS)	-

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



The screenshot shows the AWS Lambda service interface. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, a search bar, and account information ('N. Virginia' and 'awssimplified_user @ 7553-1496-5794'). The main area is titled 'Function name' with a note: 'Enter a name that describes the purpose of your function.' A text input field contains 'lambda-handler'. Below it is a note: 'Use only letters, numbers, hyphens, or underscores with no spaces.' Under 'Runtime', a dropdown menu shows 'Python 3.9'. Under 'Architecture', a note says 'Choose the instruction set architecture you want for your function code.' Two options are shown: 'x86_64' (selected) and 'arm64'. Under 'Permissions', a note says 'By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.'

STEP 5: Lambda function created successfully.

The screenshot shows the AWS Lambda console. At the top, a green banner says "Successfully created the function lambda-handler. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below the banner, the function name "lambda-handler" is displayed. On the left, there's a sidebar with "Lambda" and "Functions". On the right, there are buttons for "Throttle", "Copy ARN", and "Actions". Under "Function overview", there's a box for "lambda-handler" with a "Layers" section showing "(0)". Buttons for "+ Add trigger" and "+ Add destination" are visible. A "Description" field is empty, and a "Last modified" field shows "2 seconds ago".

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

The screenshot shows the "Add trigger" dialog in the AWS Lambda console. The title is "Add trigger". Under "Trigger configuration", it says "Trigger type: SNS". A dropdown menu shows "SNS" selected. Below it, "aws messaging notifications pub-sub push" are listed. A "SNS topic" section asks "Select the SNS topic to subscribe to." An input field contains "arn:aws:sns:us-east-1:755314965794:publisher" with a clear button "X" and a copy button "C". A note at the bottom says "Lambda will add the necessary permissions for AWS SNS to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model." At the bottom right, there are "Cancel" and "Add" buttons, with "Add" being highlighted.

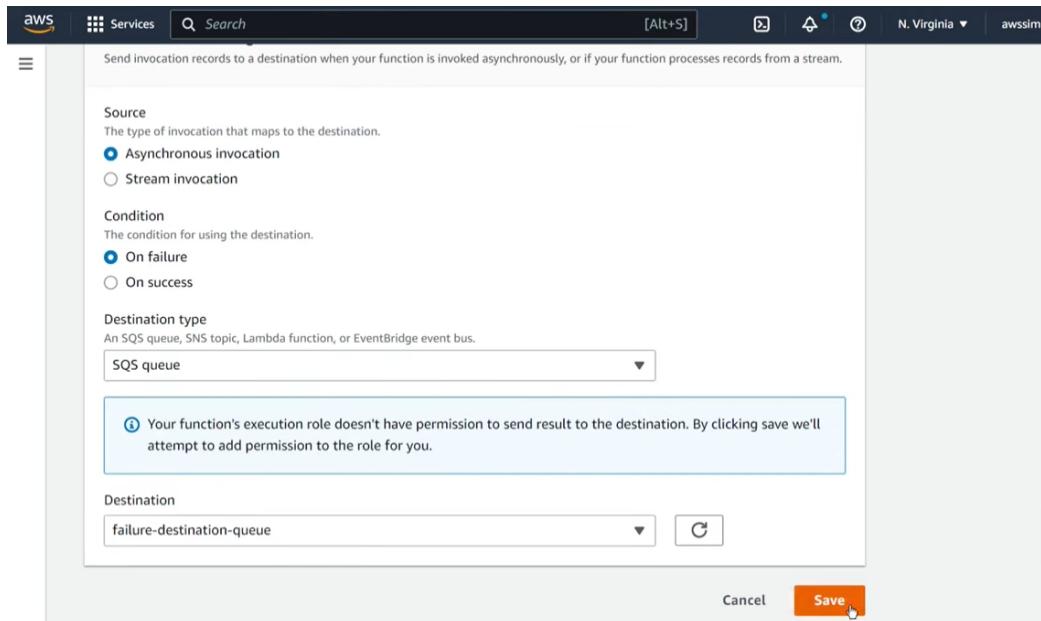
STEP 7: Trigger added successfully.

The screenshot shows the AWS Lambda Functions console. In the left navigation bar, 'Lambda' is selected. Under 'lambda-handler', there is a green success message: 'The trigger publisher was successfully added to function lambda-handler. The function is now receiving events from the trigger.' Below this, the 'Function overview' section is expanded, showing the function name 'lambda-handler' and its configuration. It lists an SNS trigger and a 'Layers' section. On the right, detailed information is provided: Description (empty), Last modified (39 seconds ago), Function ARN (arn:aws:lambda:us-east-1:755314965794:function:lambda-handler), and a thumbnail image of a person's head.

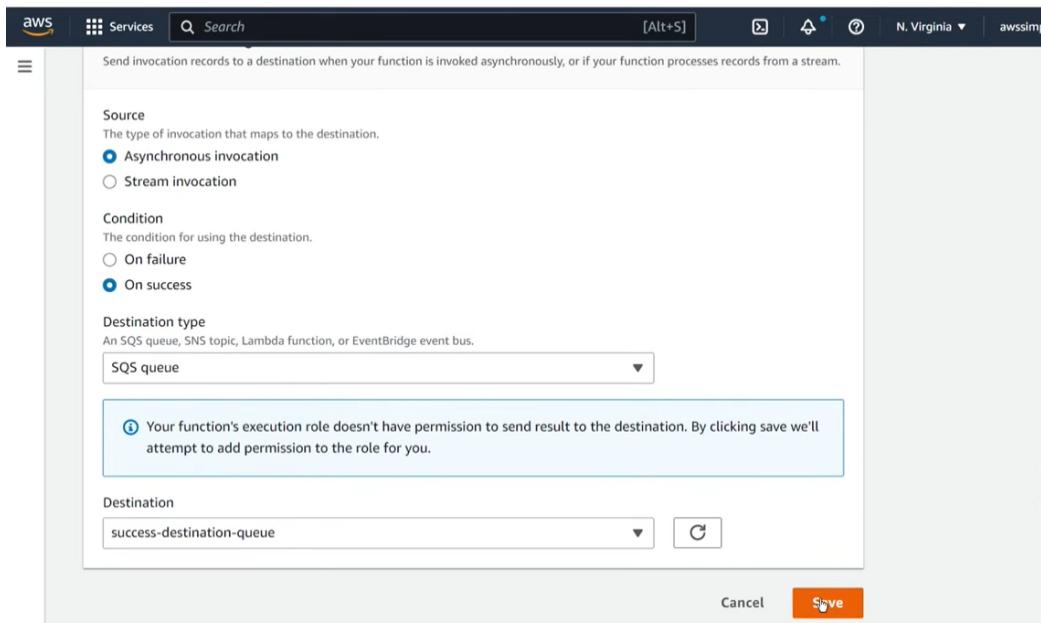
STEP 8: Click “Add destination”.

This screenshot is identical to the previous one, showing the AWS Lambda Functions console with the same function details and success message. However, the 'Add destination' button in the 'Triggers' section is now highlighted with a cursor, indicating the next step in the process.

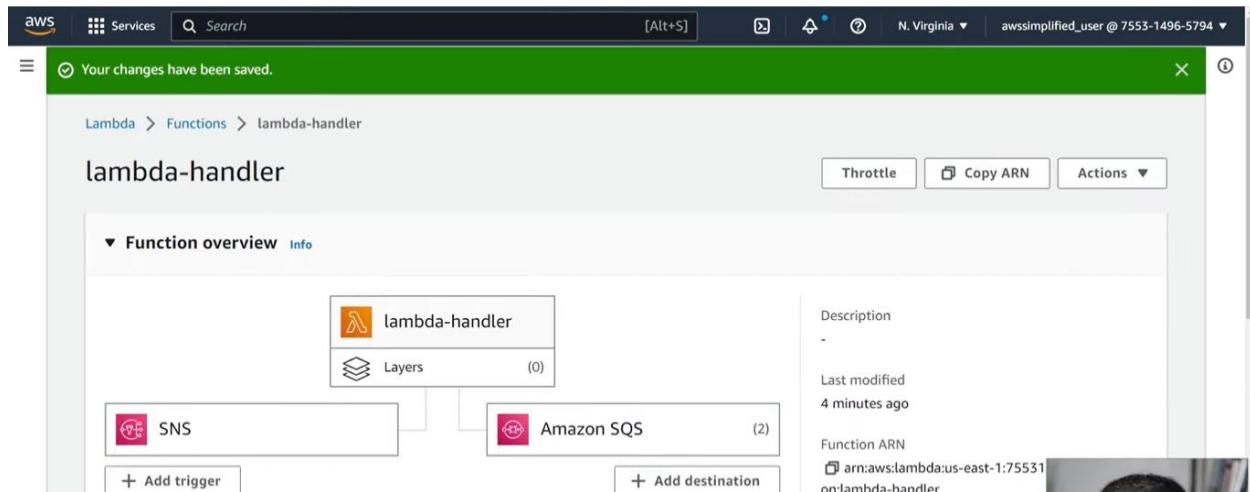
STEP 9: Choose destination type as “SQS queue” and destination as “failure-destination-queue”.



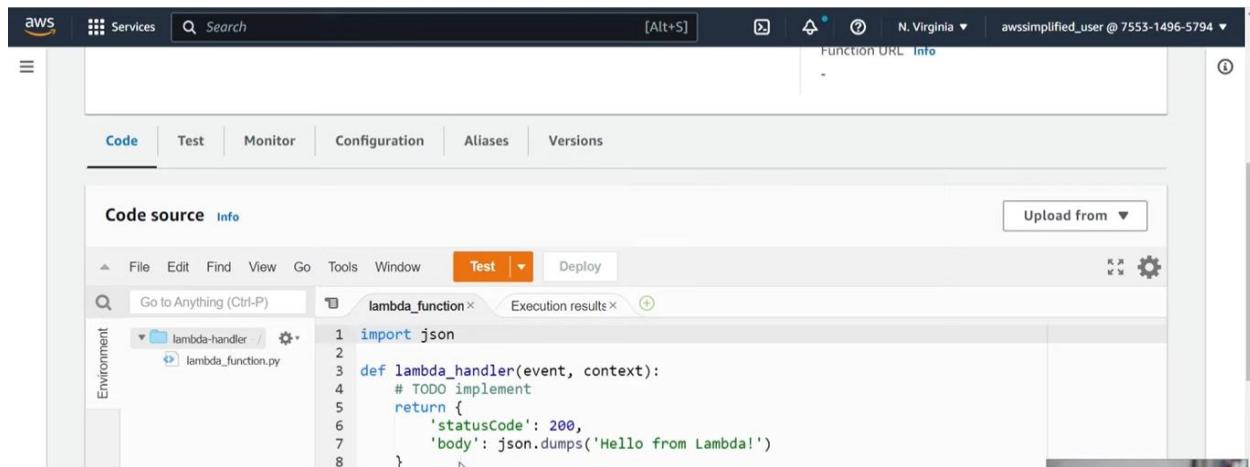
STEP 10: In another destination, Choose destination type as “SQS queue” and destination as “success-destination-queue”.



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”.

The screenshot shows the AWS SNS Topics page. On the left, there's a sidebar with 'Amazon SNS' and sections for 'Dashboard', 'Topics' (which is selected and highlighted in blue), and 'Subscriptions'. Below that is a 'Mobile' section with 'Push notifications', 'Text messaging (SMS)', and 'Origination numbers'. The main content area shows a topic named 'publisher'. The 'Details' section contains fields: 'Name' (publisher), 'Display name' (empty), 'ARN' (arn:aws:sns:us-east-1:755314965794:publisher), and 'Topic owner' (755314965794). At the top right of this section are 'Edit', 'Delete', and 'Publish message' buttons. The 'Publish message' button is highlighted with a mouse cursor.

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

The screenshot shows the 'Publish message' dialog box. At the top, it says 'Maximum 100 printable ASCII characters'. Below that is a 'Time to Live (TTL) - optional' section with an info link and a text input field. The main area is titled 'Message body'. It has a 'Message structure' section with two options: 'Identical payload for all delivery protocols.' (selected, with a note about sending the same payload to all endpoints) and 'Custom payload for each delivery protocol.' (with a note about sending different payloads based on delivery protocol). Below this is a text input field labeled 'Message body to send to the endpoint' containing the value '1 successBody'.

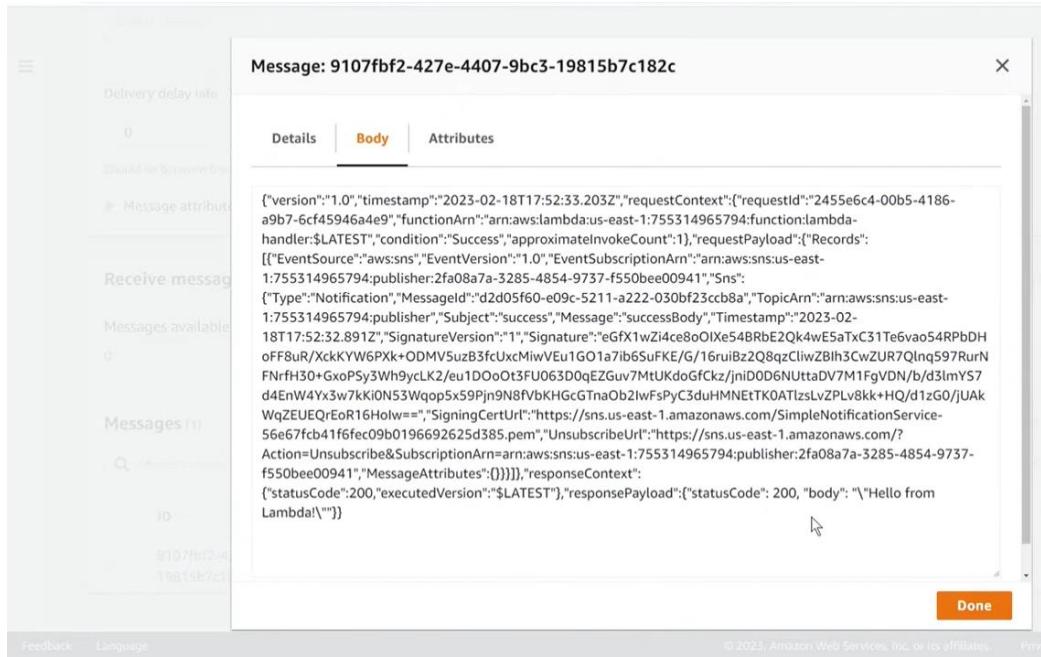
STEP 15: Message published successfully.

The screenshot shows the AWS SNS console in the N. Virginia region. The left sidebar shows 'Amazon SNS' with 'Topics' selected. The main area displays a green success message: 'Message published to topic publisher successfully.' with details: Message "ID": d2d05f60-e09c-5211-a222-030bf23ccb8a and Request "ID": 961ab944-4af7-54c7-a8e5-e52f5857fd5c. Below this, the 'publisher' topic details are shown: Name: publisher, Display name: -, ARN: arn:aws:sns:us-east-1:755314965794:publisher, Topic owner: 755314965794. Buttons for 'Edit', 'Delete', and 'Publish message' are visible.

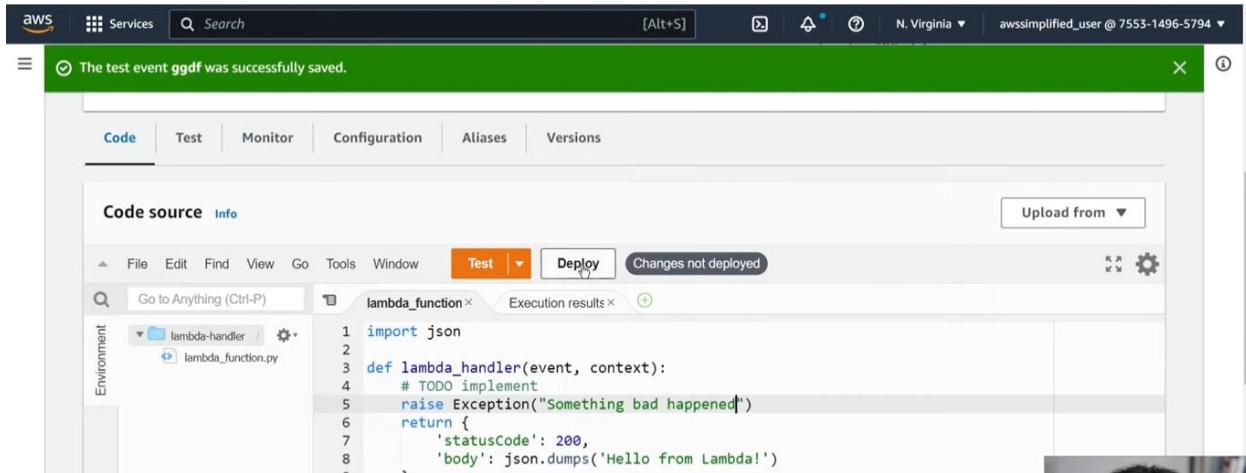
STEP 16: Check the success queue, one message added to the queue.

The screenshot shows the AWS SQS console in the N. Virginia region. The left sidebar shows 'Amazon SQS' with 'Queues' selected. The main area displays the 'Queues (2)' list: failure-destination-queue (Standard, 18 Feb 2023, 11:54:54 GMT-5, 0 messages available, 0 messages in flight, Amazon SQS key (SSE-SQS), - content-based deduplication) and success-destination-queue (Standard, 18 Feb 2023, 11:50:49 GMT-5, 1 message available, 0 messages in flight, Amazon SQS key (SSE-SQS), - content-based deduplication). A cursor points to the 'success-destination-queue' row.

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.

The screenshot shows the AWS SNS Publisher topic configuration page. At the top, there is a search bar with the word "failure". Below it, a "Time to Live (TTL) - optional" field is present with an info link. The "Message body" section contains a "Message structure" dropdown set to "Identical payload for all delivery protocols". A note states: "The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol." Another option, "Custom payload for each delivery protocol.", is also available. The "Message body to send to the endpoint" field contains the JSON payload: `1 { "reason": "failure"}`.

STEP 20: Message published successfully.

The screenshot shows the AWS SNS Topics page. The left sidebar includes "Amazon SNS", "Dashboard", "Topics" (which is selected), "Subscriptions", and "Mobile" sections with "Push notifications", "Text messaging (SMS)", and "Origination numbers". The main content area shows a green success message: "Message published to topic publisher successfully. Message ID: 172f456f-5627-574a-abd7-1904451f2ab5 Request ID: 70066826-8e78-582f-b310-344d9d4085d0". Below this, the "publisher" topic details are shown, including its ARN: arn:aws:sns:us-east-1:755314965794:publisher. There are "Edit", "Delete", and "Publish message" buttons.

STEP 21: Now one message added to failure queue.

The screenshot shows the AWS SQS console with the following details:

Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication
failure-destination-queue	Standard	18 Feb 2023, 11:54:54 GMT-5	1	0	Amazon SQS key (SSE-SQS)	-
success-destination-queue	Standard	18 Feb 2023, 11:50:49 GMT-5	1	0	Amazon SQS key (SSE-SQS)	-

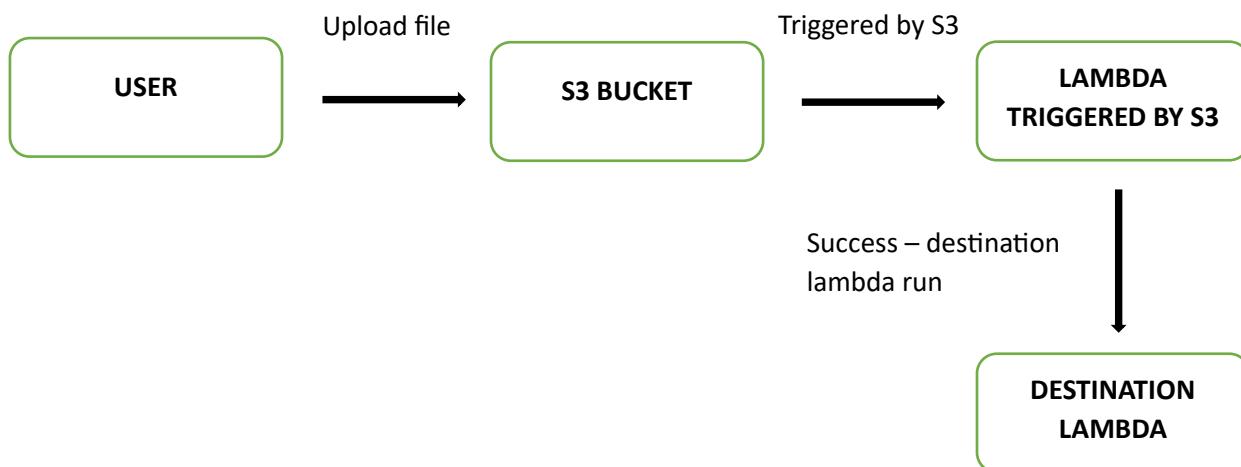
STEP 22: Open and explore failure queue to see the message.

The screenshot shows the AWS SQS message details view for the 'failure-destination-queue'. The message ID is 81c51032-e959-45d3-8c32-8d18f64031bf. The 'Body' tab displays the following JSON payload:

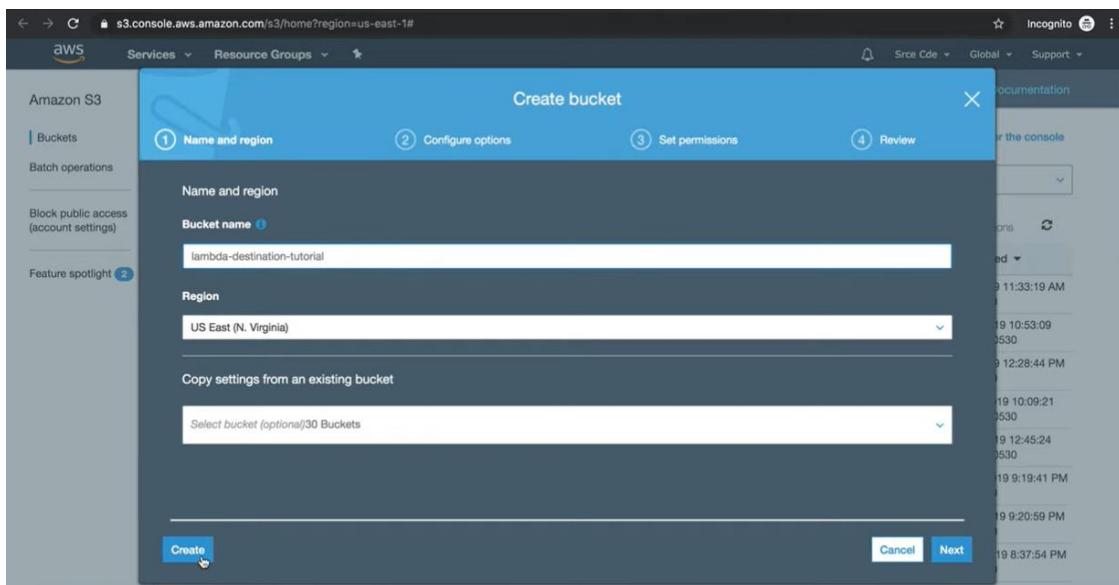
```
{"version": "1.0", "timestamp": "2023-02-18T17:58:11.577Z", "requestContext": {" requestId": "f648eb6e-1879-4403-a2a9-962f378853bf", "functionArn": "arn:aws:lambda:us-east-1:755314965794:function:lambda-handler:$LATEST", "condition": "RetriesExhausted", "approximateInvokeCount": 3}, "requestPayload": [{"Records": [
```

ADD DESTINATION IN AMAZON LAMBDA (WITH S3)

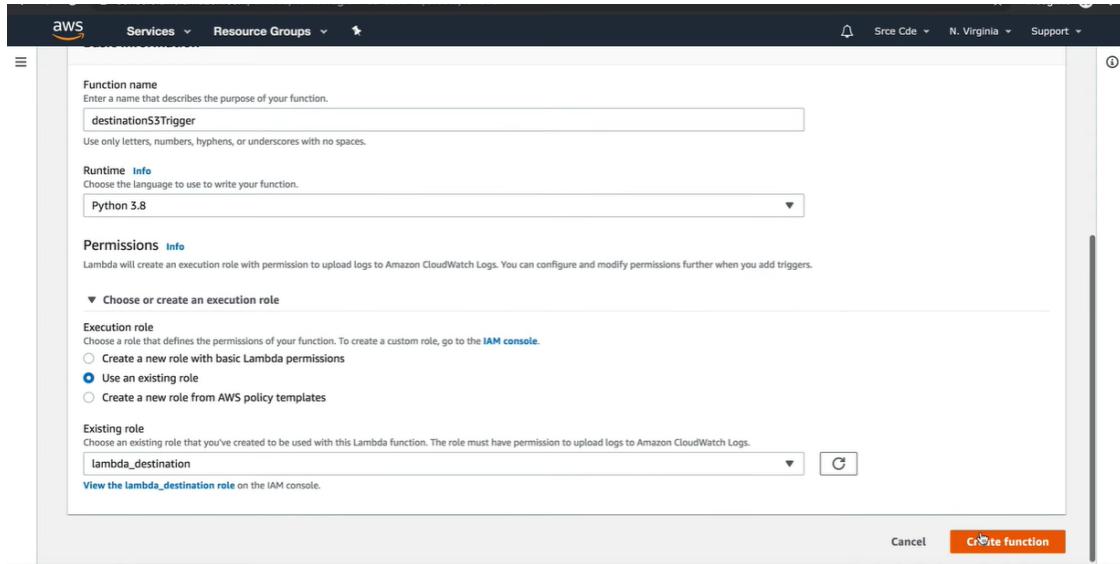
WORKFLOW DIAGRAM



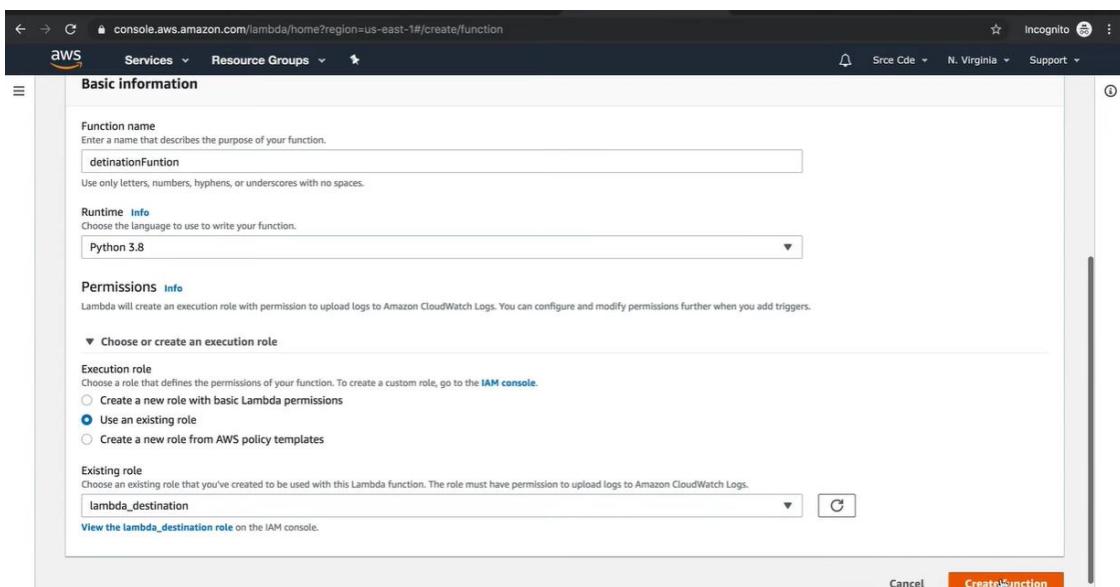
STEP 1: Create a bucket.



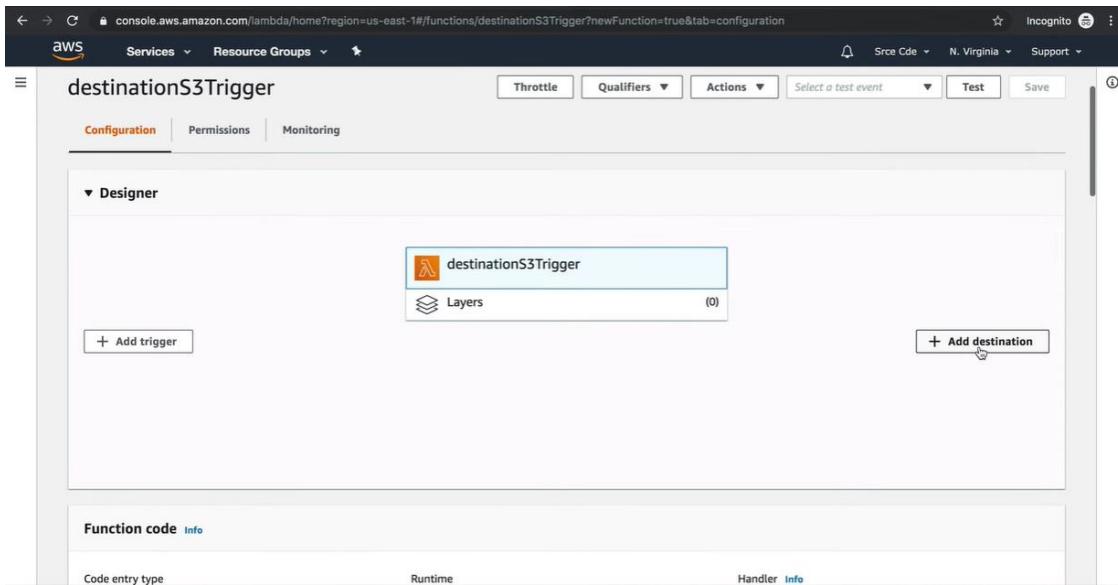
STEP 2: Create a lambda function for triggering.



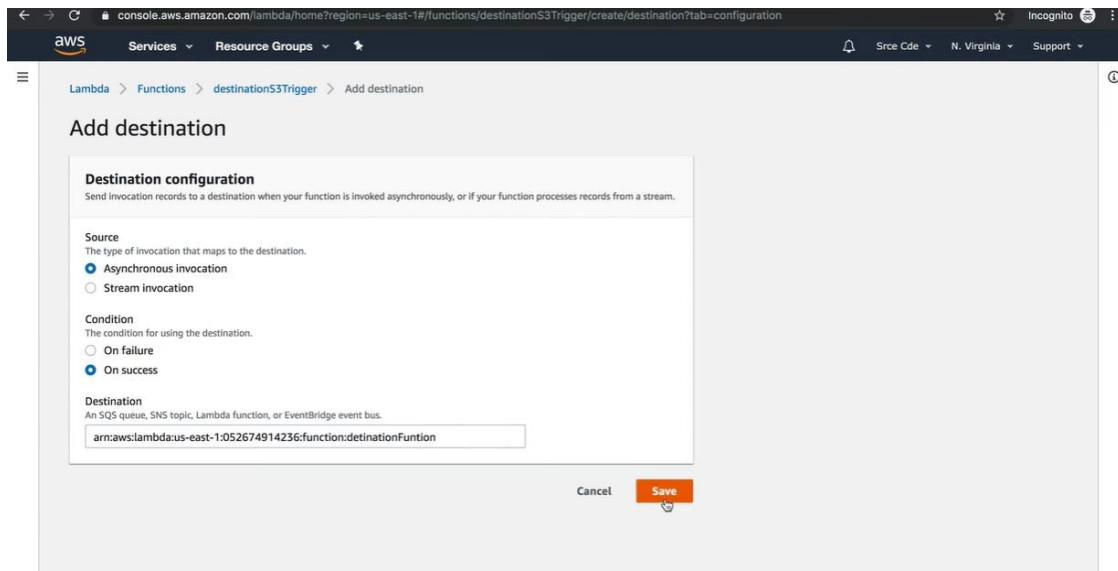
STEP 3: Create another lambda function for destination.



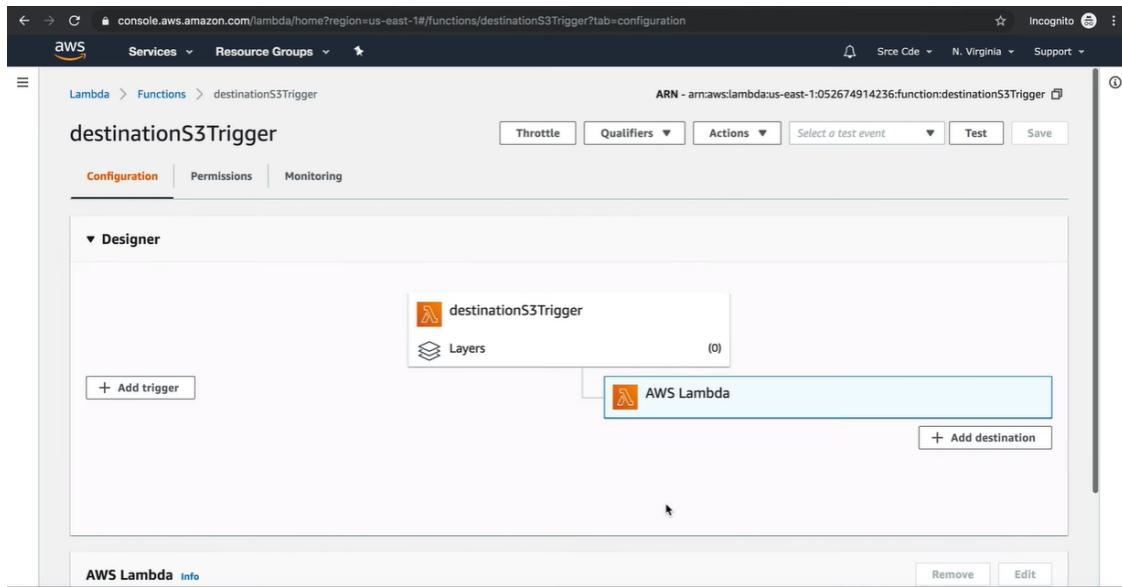
STEP 4: Click “Add destination”.



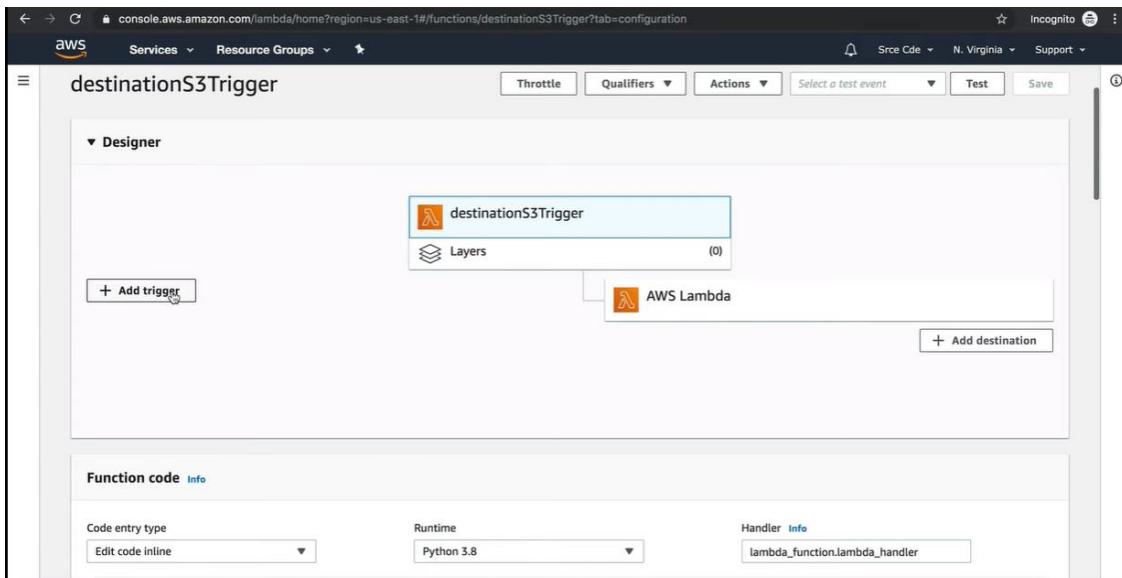
STEP 5: Add ARN for destination.



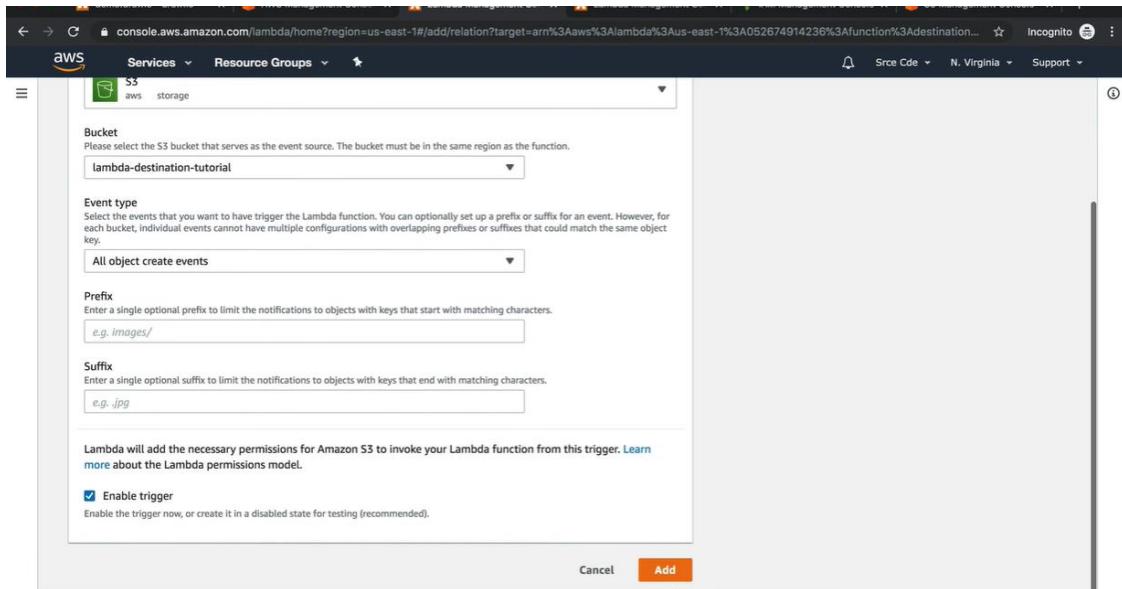
STEP 6: Destination added successfully.



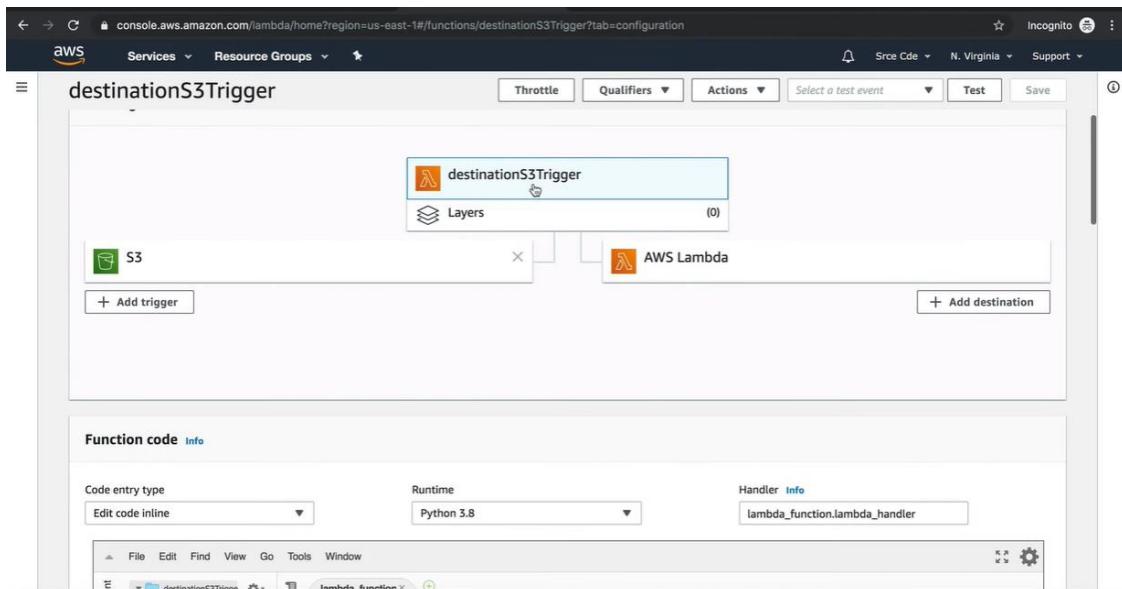
STEP 7: Click “Add trigger”.



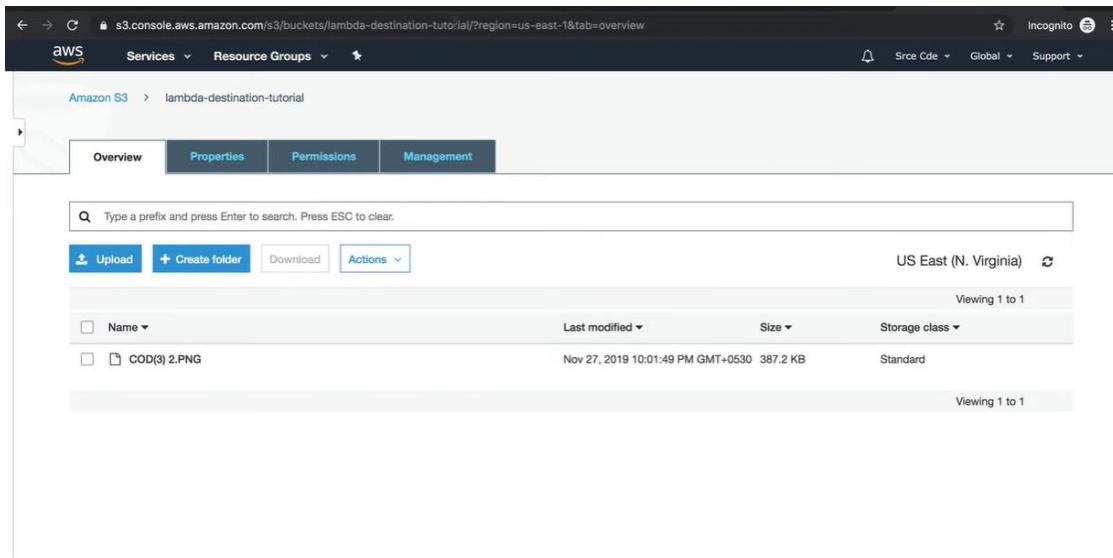
STEP 8: Choose S3 and bucket and click “Add”.



STEP 9: Trigger added successfully.

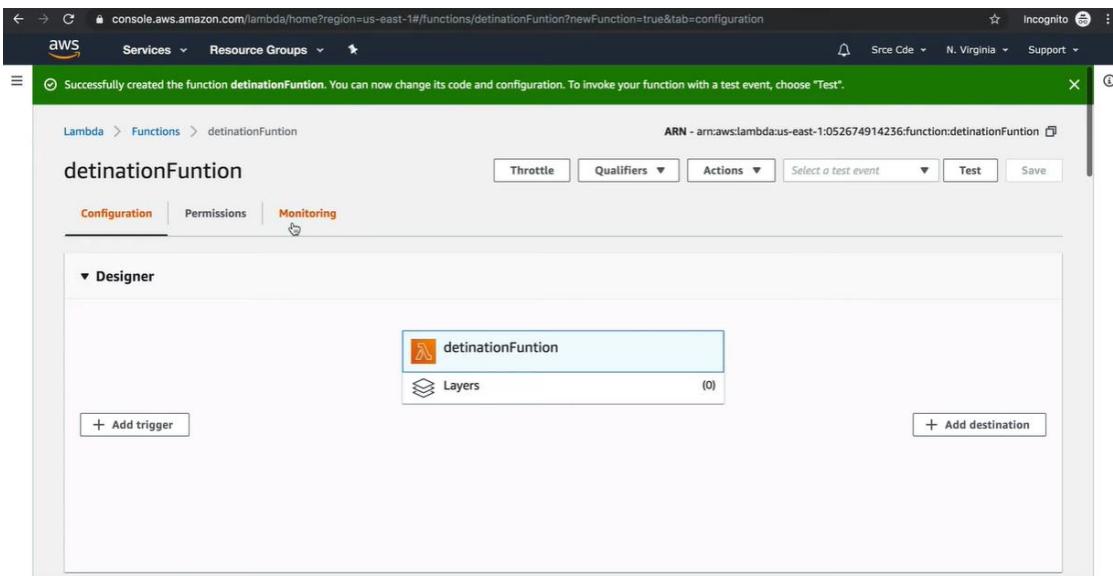


STEP 10: Upload one file in S3 bucket.



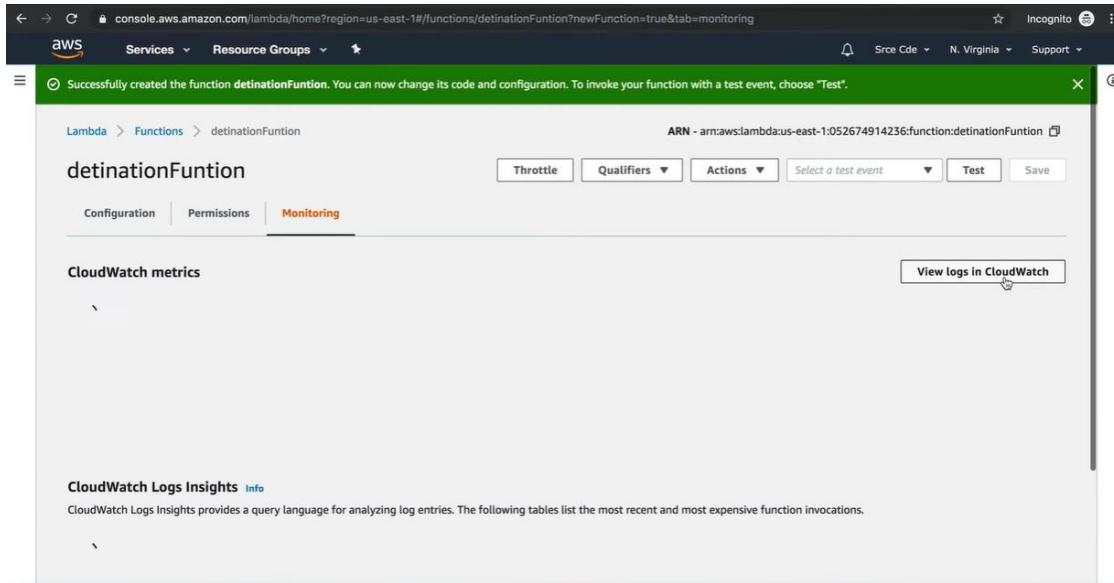
The screenshot shows the AWS S3 console interface. At the top, the URL is s3.console.aws.amazon.com/s3/buckets/lambda-destination-tutorial/?region=us-east-1&tab=overview. The navigation bar includes 'AWS Services' and 'Resource Groups'. Below the navigation, it says 'Amazon S3 > lambda-destination-tutorial'. The main area has tabs for 'Overview', 'Properties', 'Permissions', and 'Management', with 'Overview' selected. A search bar at the top says 'Type a prefix and press Enter to search. Press ESC to clear.' Below the search bar are buttons for 'Upload' (highlighted in blue), '+ Create folder', 'Download', and 'Actions'. To the right, it shows 'US East (N. Virginia)' and 'Viewing 1 to 1'. The list contains one item: 'COD(3) 2.PNG' with a size of '387.2 KB' and a 'Last modified' date of 'Nov 27, 2019 10:01:49 PM GMT+0530'. The storage class is 'Standard'.

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



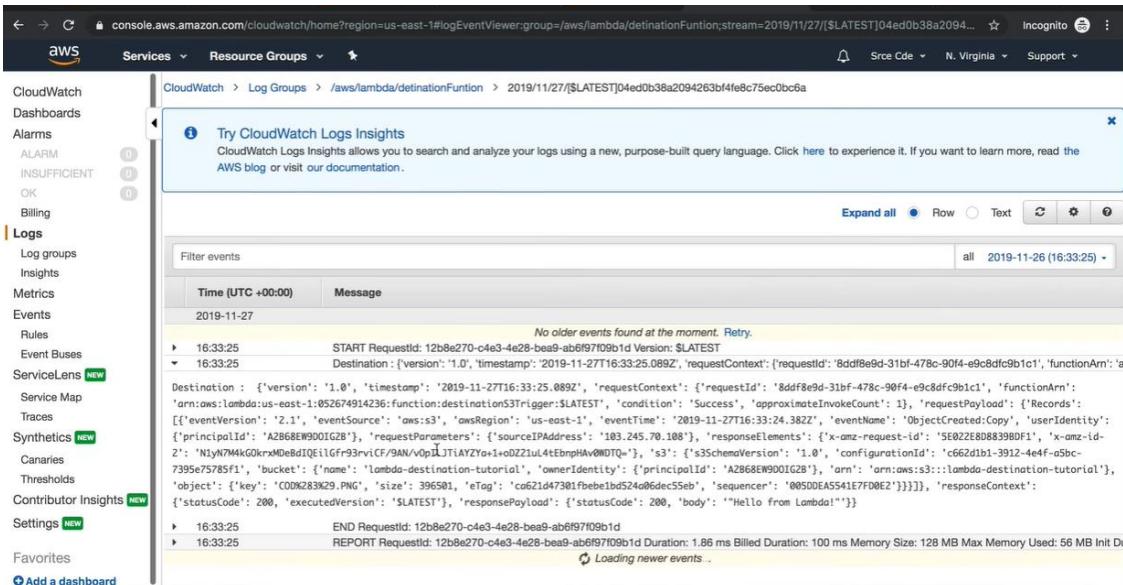
The screenshot shows the AWS Lambda console interface. The URL is console.aws.amazon.com/lambda/home?region=us-east-1#/functions/destinationFunction?newFunction=True&tab=configuration. The navigation bar includes 'AWS Services' and 'Resource Groups'. A green success message at the top says 'Successfully created the function destinationFunction. You can now change its code and configuration. To invoke your function with a test event, choose "Test".'. Below the message, it says 'Lambda > Functions > destinationFunction' and shows the ARN 'arn:aws:lambda:us-east-1:052674914236:function:destinationFunction'. The main area shows the 'destinationFunction' configuration page with tabs for 'Configuration' (selected), 'Permissions', and 'Monitoring'. Under the 'Designer' section, there is a box labeled 'destinationFunction' with a 'Layers' section below it containing '(0)'. Buttons for '+ Add trigger' and '+ Add destination' are visible.

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



The screenshot shows the AWS Lambda console. In the top navigation bar, 'Services' is selected. Under 'Resource Groups', there is a dropdown menu. The main content area shows a Lambda function named 'destinationFunction'. At the top right, there are tabs for 'Throttle', 'Qualifiers', 'Actions', 'Select a test event', 'Test', and 'Save'. Below these tabs, there are three tabs: 'Configuration', 'Permissions', and 'Monitoring', with 'Monitoring' being the active tab. A large button labeled 'View logs in CloudWatch' is located at the bottom right of the monitoring section. The URL in the browser is 'console.aws.amazon.com/lambda/home?region=us-east-1#/functions/destinationFunction?newFunction=true&tab=monitoring'.

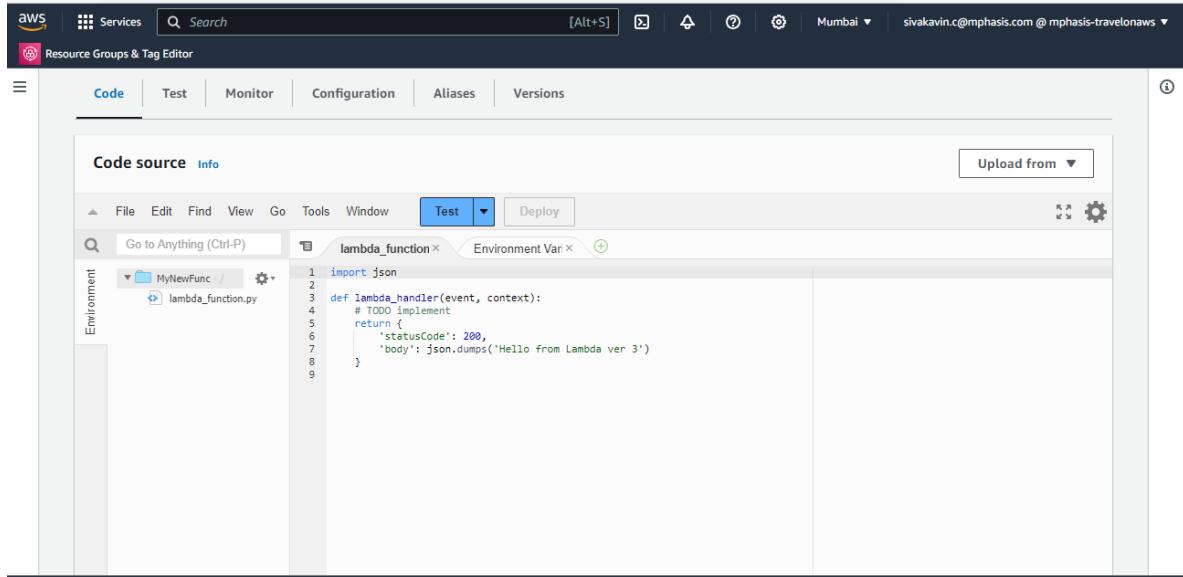
STEP 13: Here destination run successfully.



The screenshot shows the AWS CloudWatch Log Event Viewer. The left sidebar has a 'Logs' section with various sub-options like 'Log groups', 'Metrics', 'Events', 'Rules', etc. The main content area shows a log group for the 'destinationFunction' Lambda function. It displays log entries from November 27, 2019. One entry is expanded, showing detailed log data. The expanded log entry includes fields such as 'Time (UTC +00:00)', 'Message', and a large JSON payload representing the Lambda function's execution context and logs. The URL in the browser is 'console.aws.amazon.com/cloudwatch/home?region=us-east-1#logEventViewer:group=/aws/lambda/destinationFunction;stream=2019/11/27[\$LATEST]04ed0b38a2094263bf4fe8c75ec0bc6a'.

CODE - TEST - MONITOR - CONFIGURATION - ALIASES – VERSIONS

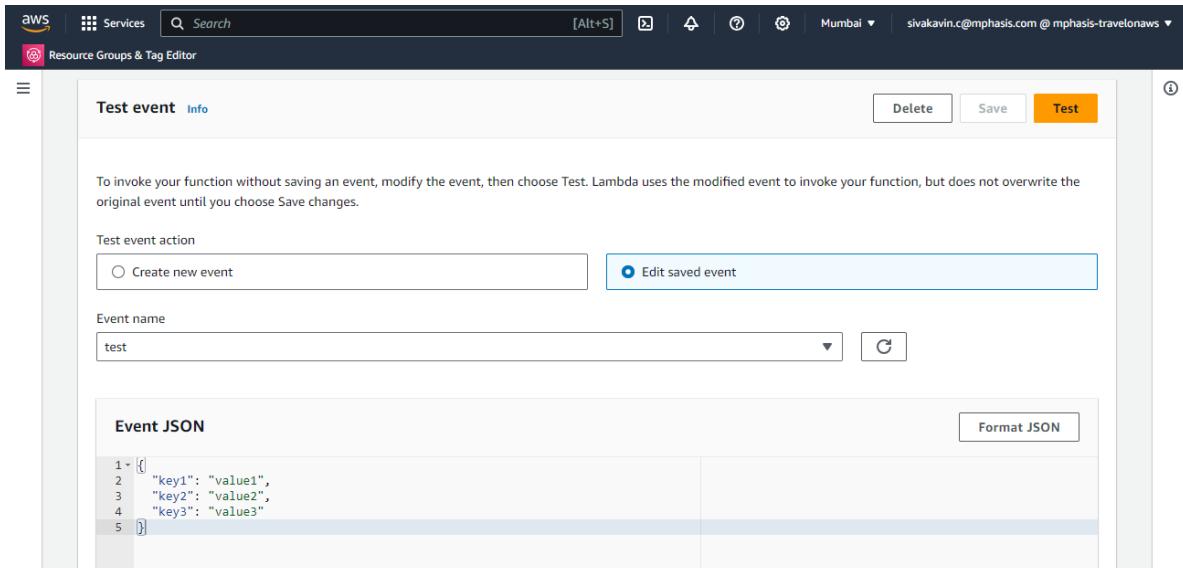
CODE: In coding part, write a code and deploy it.



The screenshot shows the AWS Lambda Code source interface. The top navigation bar includes 'Services', 'Search', 'Mumbai', and the user 'sivakavin.c@mphasis.com @ mphasis-travelonaws'. Below the navigation is a tab bar with 'Code' (selected), 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The main area is titled 'Code source' with tabs for 'Info' and 'Code'. A toolbar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test' (selected), and 'Deploy' is visible. On the left, there's a sidebar for 'Environment' and a search bar 'Go to Anything (Ctrl-P)'. The central code editor shows a Python file named 'lambda_function.py' with the following code:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda ver 3')
8     }
9
```

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

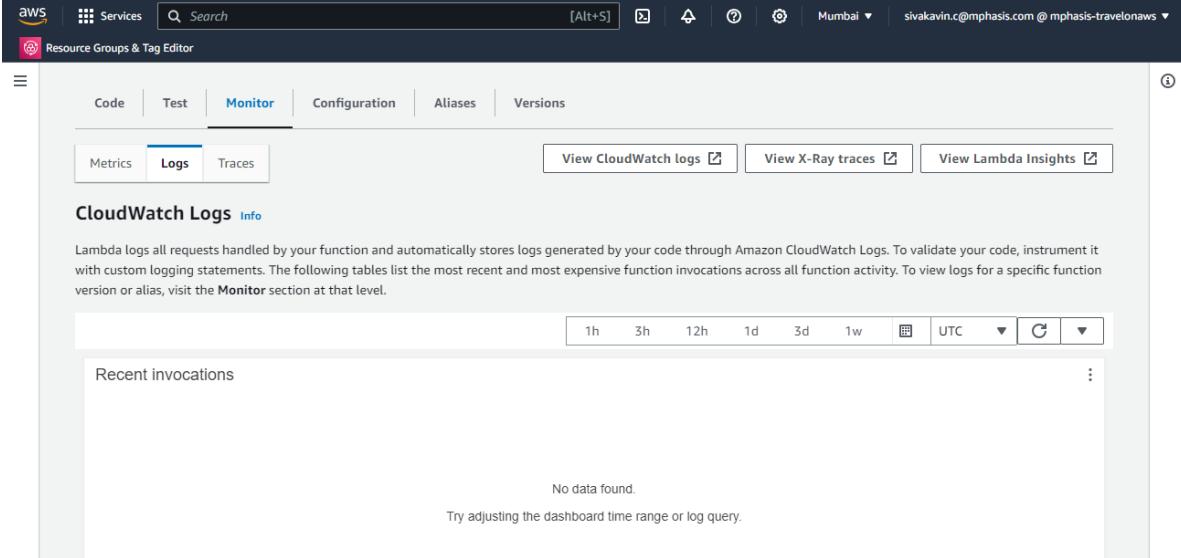


The screenshot shows the AWS Lambda Test event interface. The top navigation bar includes 'Services', 'Search', 'Mumbai', and the user 'sivakavin.c@mphasis.com @ mphasis-travelonaws'. Below the navigation is a tab bar with 'Delete', 'Save', and 'Test' (selected). The main area is titled 'Test event' with tabs for 'Info' and 'Event JSON'. A note says: 'To invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but does not overwrite the original event until you choose Save changes.' Below this, there are two options: 'Create new event' (radio button) and 'Edit saved event' (radio button, selected). An 'Event name' field contains 'test'. At the bottom is an 'Event JSON' section with a code editor containing the following JSON:

```
1 [{}]
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 ]
```

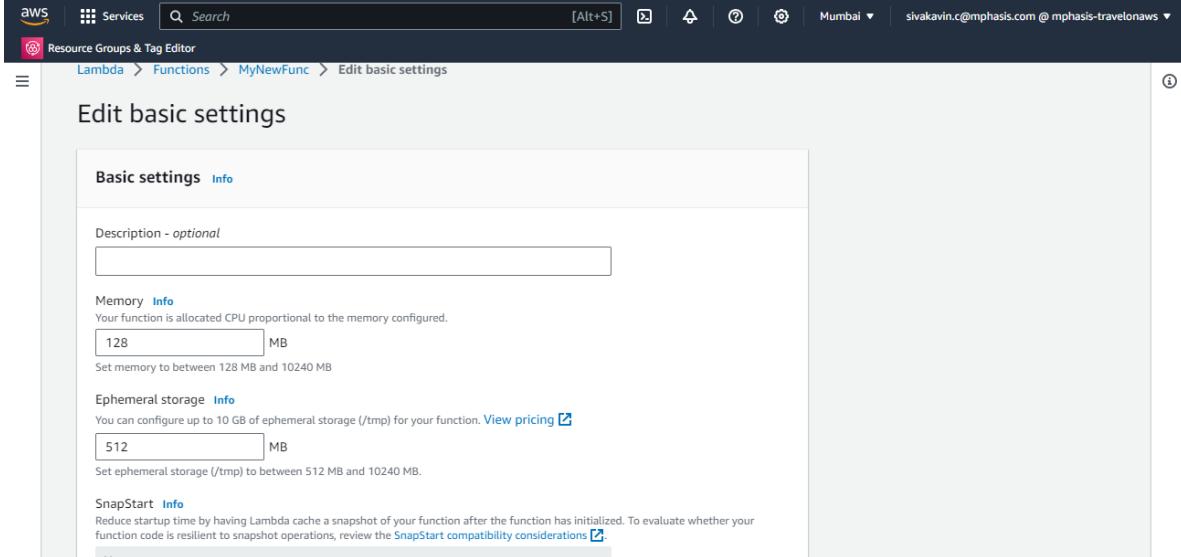
A 'Format JSON' button is located in the top right corner of the JSON editor.

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



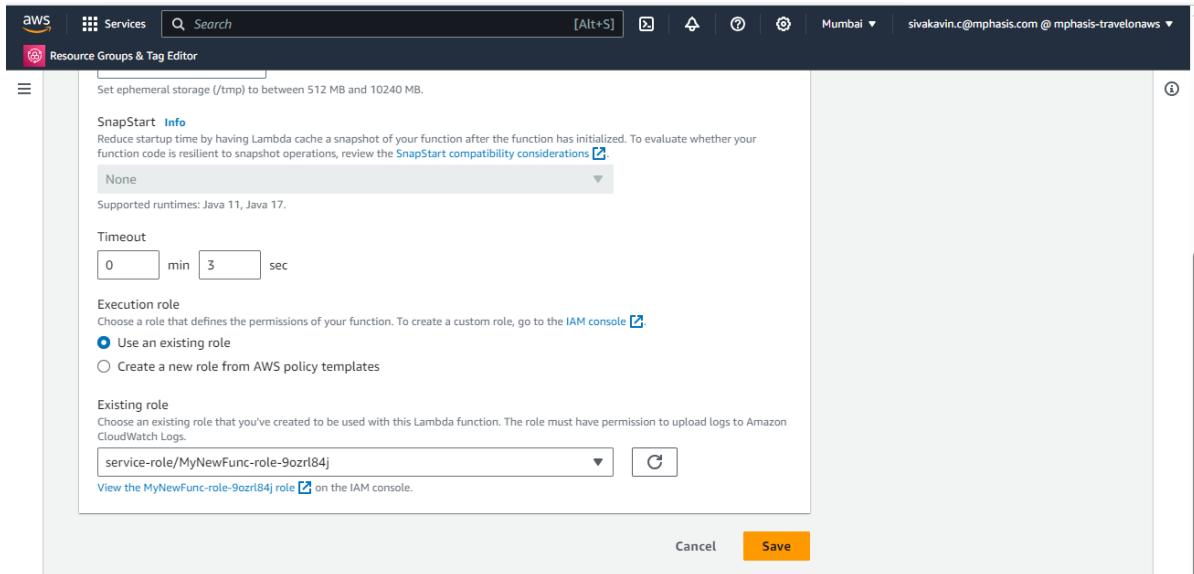
The screenshot shows the AWS Lambda Monitor interface. At the top, there are tabs for Code, Test, Monitor (which is selected), Configuration, Aliases, and Versions. Below the tabs are three buttons: View CloudWatch logs, View X-Ray traces, and View Lambda Insights. The main section is titled "CloudWatch Logs" with an "Info" link. It contains a message stating that Lambda logs all requests handled by your function and automatically stores logs generated by your code through Amazon CloudWatch Logs. It encourages instrumenting code with custom logging statements. A table is shown with columns for Recent invocations, but it displays "No data found." with a note to adjust the dashboard time range or log query. Time range controls at the bottom allow selecting 1h, 3h, 12h, 1d, 3d, 1w, UTC, and a date picker.

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

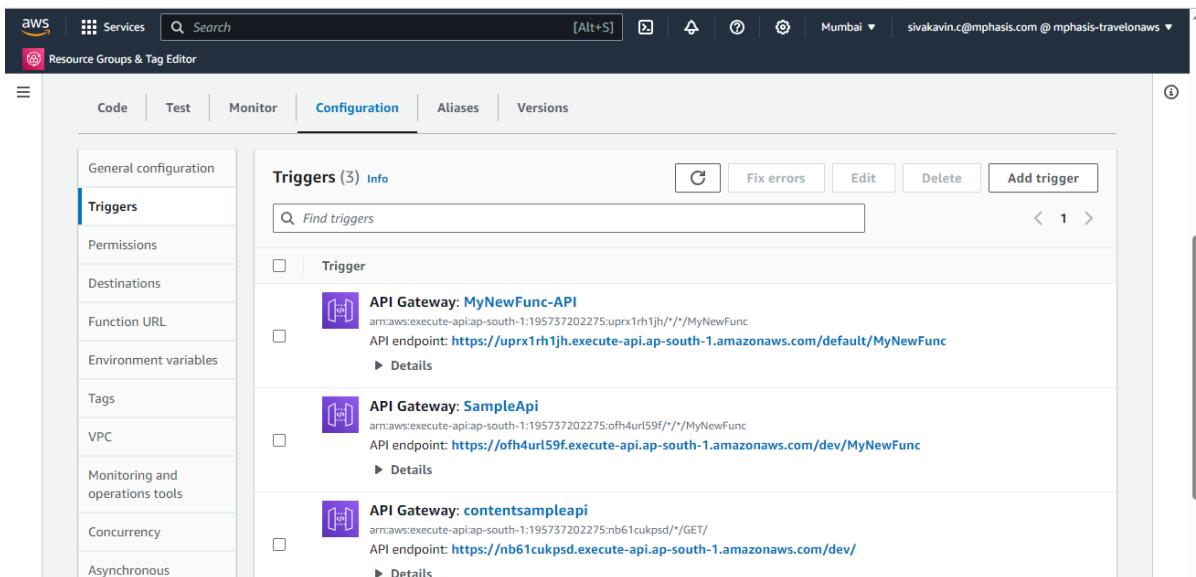


The screenshot shows the "Edit basic settings" page for a Lambda function named "MyNewFunc". The top navigation bar includes the AWS logo, Services, a search bar, and account information for Mumbai and sivakavin.c@mphasis.com @ mphasis-travelonaws. The page title is "Edit basic settings". Under "Basic settings", there is a "Description - optional" input field. The "Memory" section shows a value of 128 MB, with a note that memory must be between 128 MB and 10240 MB. The "Ephemeral storage" section shows a value of 512 MB, with a note that storage must be between 512 MB and 10240 MB. The "SnapStart" section has a dropdown set to "None".

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.

Execution role

Role name
MyNewFunc-role-9ozrl84j

Resource summary

⚠ User: arn:aws:iam::195737202275:user/sivakavin.c@mphasis.com is not authorized to perform: iam:GetRole on resource: role MyNewFunc-role-9ozrl84j because no identity-based policy allows the iam:GetRole action

Resource-based policy statements (3) Info

Statement ID	Principal	PrincipalOrgID	Conditions	Action
lambda-a0ab4a00-b7e0-4819-9df2-d752d2ccfcfb	apigateway.amazonaws.com	-	ArnLike	iam:GetRole

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

Function URL Info

Code Test Monitor Configuration Aliases Versions

Destinations Info

Source	Stream	Condition	Destination
No destinations No destinations are configured.			

Add destination

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

Monitoring and operations tools

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

The screenshot shows the AWS Lambda function configuration interface. The left sidebar lists various configuration sections: General configuration, Triggers, Permissions, Destinations, Function URL (which is selected and highlighted in blue), Environment variables, Tags, VPC, and Monitoring and operations tools. The main content area is titled 'Function URL Info' and displays the message 'No Function URL'. It includes a sub-section 'No Function URL is configured.' and a 'Create function URL' button. At the top of the page, there is a navigation bar with the AWS logo, a 'Services' dropdown, a search bar, and account information for 'sivakavin.c@mphasis.com @ mphasis-travelonaws'.

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.

The screenshot shows the AWS Lambda function configuration interface. The left sidebar lists various configuration sections: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables (which is selected and highlighted in blue), Tags, VPC, Monitoring and operations tools, and Concurrency. The main content area is titled 'Environment variables' and displays the message 'No environment variables'. It includes a sub-section 'No environment variables associated with this function.' and an 'Edit' button. At the top of the page, there is a navigation bar with the AWS logo, a 'Services' dropdown, a search bar, and account information for 'sivakavin.c@mphasis.com @ mphasis-travelonaws'.

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

The screenshot shows the AWS Lambda 'Tags' configuration page. The left sidebar lists options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, **Tags**, VPC, Monitoring and operations tools, and Concurrency. The 'Configuration' tab is selected. The main area displays a table with 'Key' and 'Value' columns, which is currently empty. A 'Manage tags' button is located at the top right of the table area. Below the table, a message states 'No tags associated with this function.' and another 'Manage tags' button.

In VPC, you can add VPC if its needed.

The screenshot shows the AWS Lambda 'VPC' configuration page. The left sidebar lists options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, **VPC**, Monitoring and operations tools, and Concurrency. The 'Configuration' tab is selected. The main area displays a table with 'Info' and 'Edit' buttons. The 'Info' section contains the message 'No VPC configuration' and 'This function isn't connected to a VPC.' An 'Edit' button is located at the bottom right of the table area.

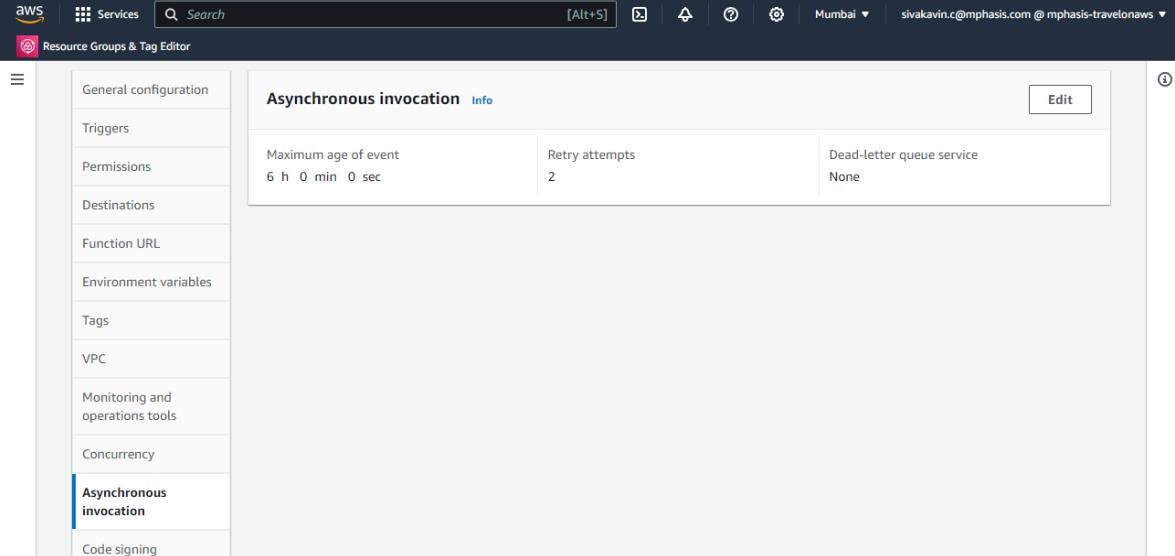
Here, we can see the monitoring details.

The screenshot shows the AWS Lambda Configuration page. The left sidebar lists various configuration tabs: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools (which is selected), and Concurrency. The main panel displays the 'Monitoring and operations tools' section. It includes a summary table with two rows: 'Logs and metrics (default)' (Enabled, Active tracing) and 'Enhanced monitoring' (Not enabled). Below this is an 'Extensions' section with a note about integrating tools via the Extensions page.

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.

The screenshot shows the AWS Lambda Configuration page with the 'Concurrency' tab selected in the sidebar. The main panel displays the 'Concurrency' section, which shows 'Function concurrency' set to 'Use unreserved account concurrency' with a value of '1000'. Below this is a 'Provisioned concurrency configurations' section. It contains a note about using provisioned concurrency for scaling, a 'Find configuration' search bar, and a table for managing provisioned concurrency configurations. The table has columns for Qualifier, Type, Provisioned concurrency, Status, and Details. A button labeled 'Add configuration' is visible at the bottom of the table.

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

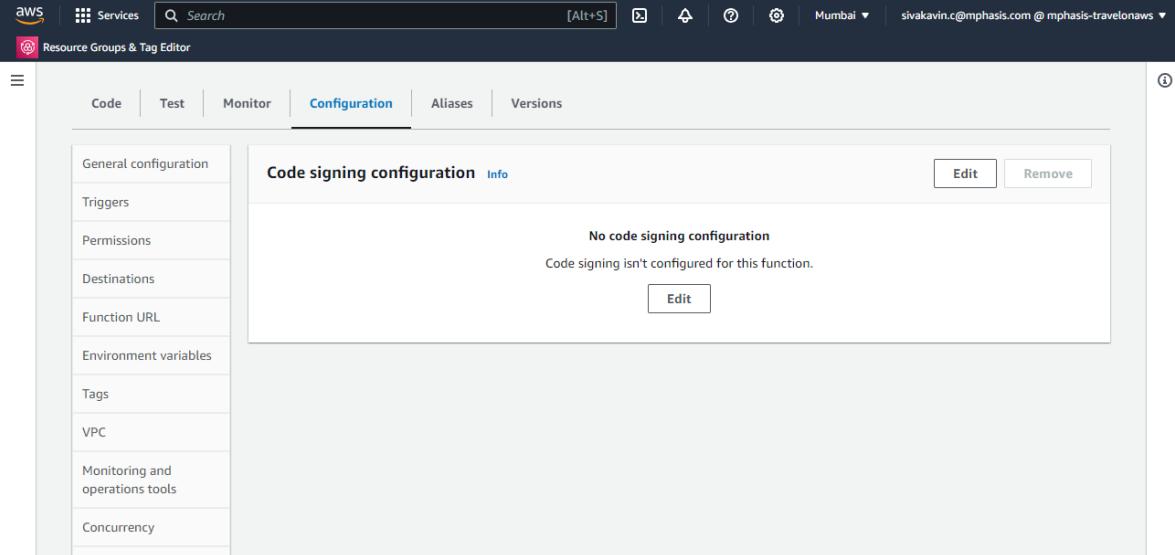


The screenshot shows the AWS Lambda function configuration interface. On the left, a sidebar lists various configuration tabs: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, **Asynchronous invocation** (which is selected and highlighted in blue), and Code signing. The main content area is titled "Asynchronous invocation" and displays the following settings:

Maximum age of event	Retry attempts	Dead-letter queue service
6 h 0 min 0 sec	2	None

An "Edit" button is located in the top right corner of this section.

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

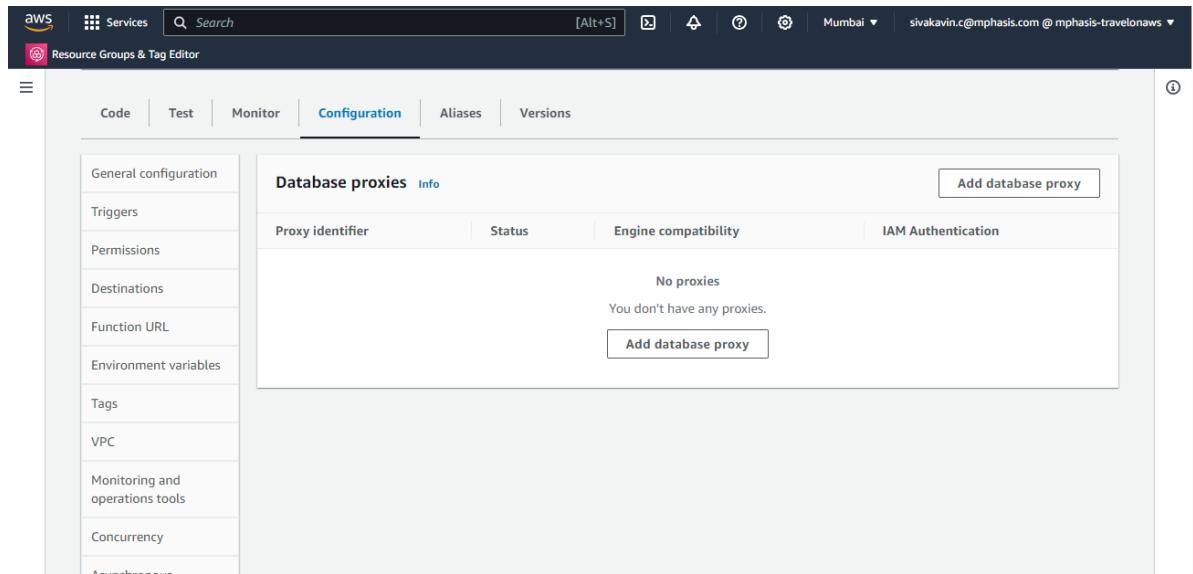


The screenshot shows the AWS Lambda function configuration interface with the "Configuration" tab selected. On the left, a sidebar lists various tabs: Code (which is selected and highlighted in blue), Test, Monitor, Configuration (selected), Aliases, and Versions. The main content area is titled "Code signing configuration" and displays the following message:

No code signing configuration
Code signing isn't configured for this function.

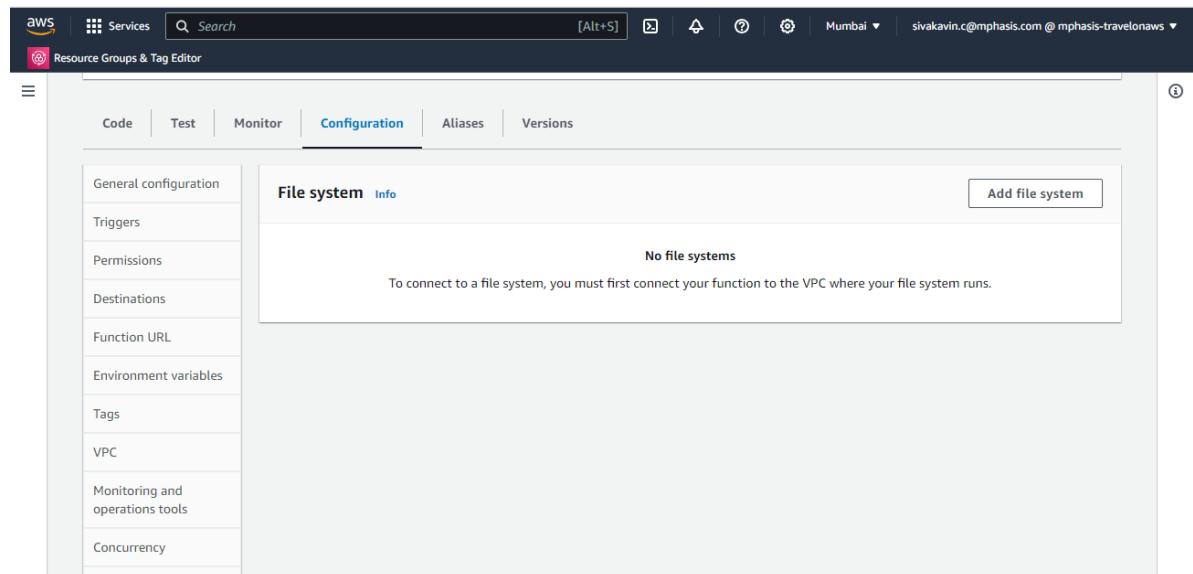
An "Edit" button is located in the center of this message area.

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.



The screenshot shows the AWS Lambda Configuration page. The left sidebar lists options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, and Concurrency. The main panel is titled "Database proxies" and contains tabs for "Info", "Proxy identifier", "Status", "Engine compatibility", and "IAM Authentication". A message states "No proxies" and "You don't have any proxies." with a "Add database proxy" button.

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



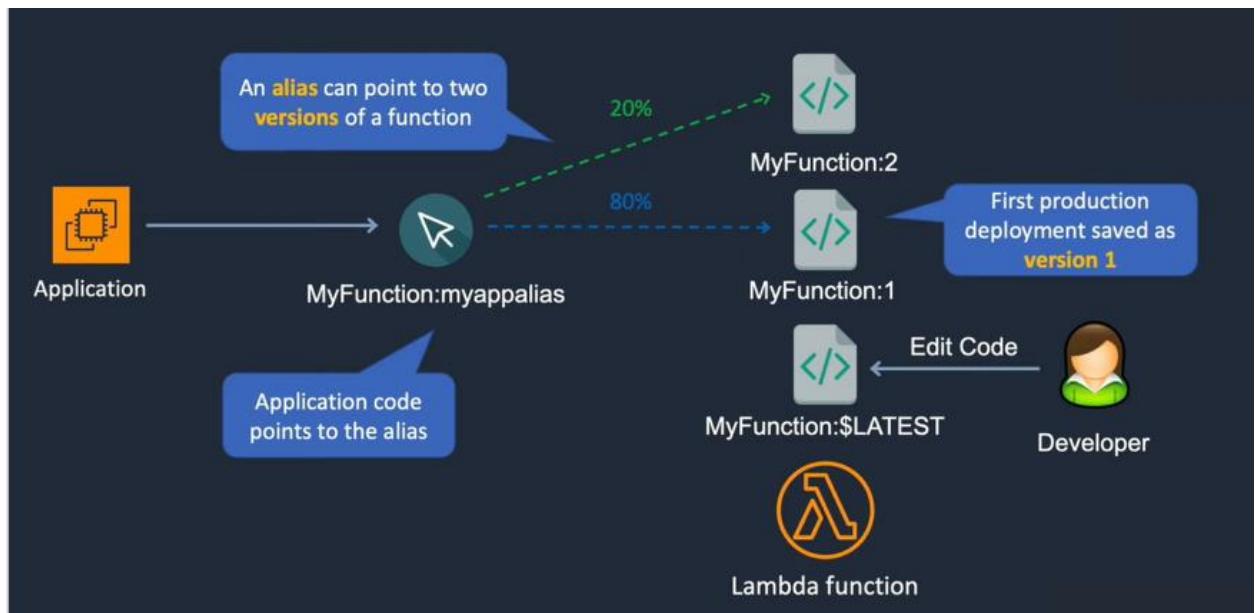
The screenshot shows the AWS Lambda Configuration page. The left sidebar lists options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, and Concurrency. The main panel is titled "File system" and contains tabs for "Info", "Add file system", and "No file systems". A message states "To connect to a file system, you must first connect your function to the VPC where your file system runs."

Here, you can create state machine.

The screenshot shows the AWS Lambda configuration interface. The top navigation bar includes the AWS logo, Services, a search bar, and account information for sivakavin.c@mphasis.com @ mphasis-travelonaws. Below the navigation is a tab bar with Code, Test, Monitor, Configuration (which is selected), Aliases, and Versions. On the left, a sidebar lists various configuration options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, and Asynchronous. The main content area is titled "State machines (0)" and contains a sub-header "Step Functions state machines which orchestrate this function." It features a search bar labeled "Search state machines" and a table header with columns Name, Type, and Created. A message "No state machines" is displayed, followed by a sub-message "Create a new state machine to orchestrate and apply error handling to your Lambda function." A prominent "Create state machine" button is located at the bottom of this section.

VERSIONS

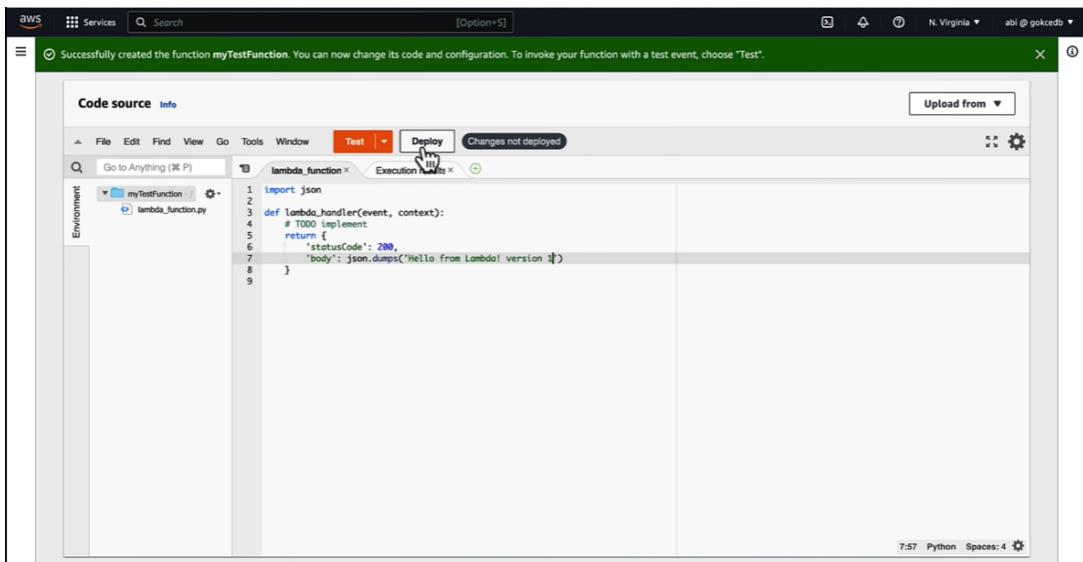
- You can have multiple versions of your function.
- You can use versions to manage the deployment of AWS lambda functions.
- You work on \$LATEST which is mutable.
- To publish a lambda function, create a version.
- Versions are immutable.
- Each function has its own ARN.



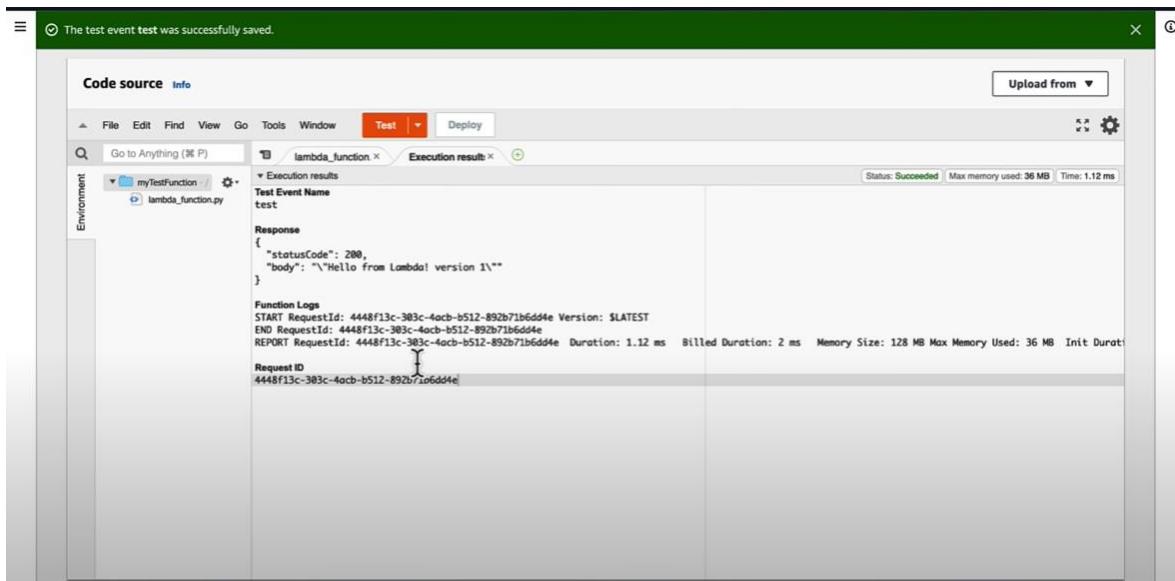
ALIAS

- 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

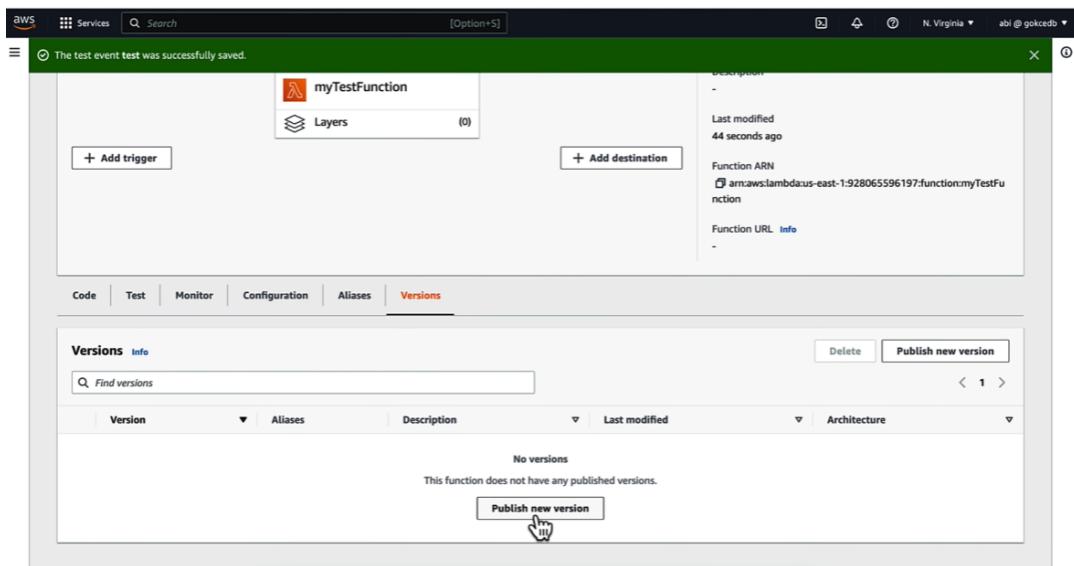
STEP 1: CREATE A FUNCTION (myTestFunction).



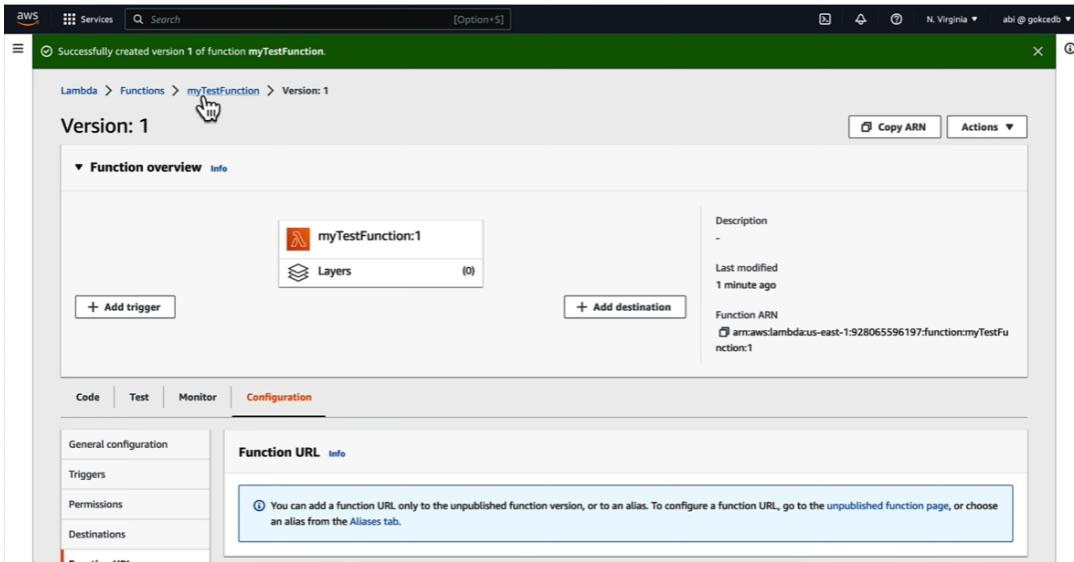
STEP 2: DEPLOY AND TEST THE CODE

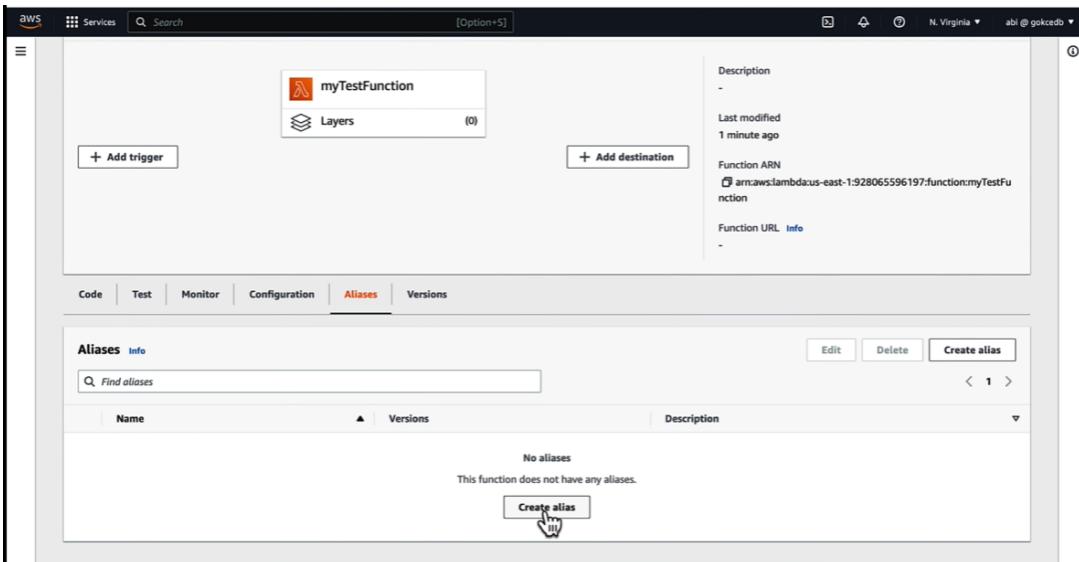


STEP 3: CREATE A NEW VERSION

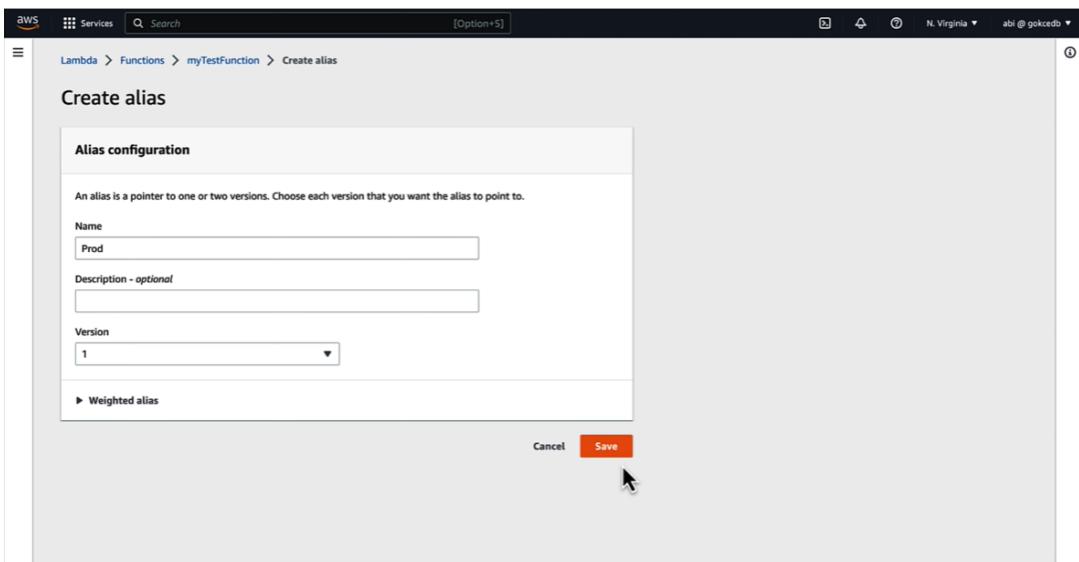


STEP 4: AFTER CREATING VERSION 1, CREATE ALIAS

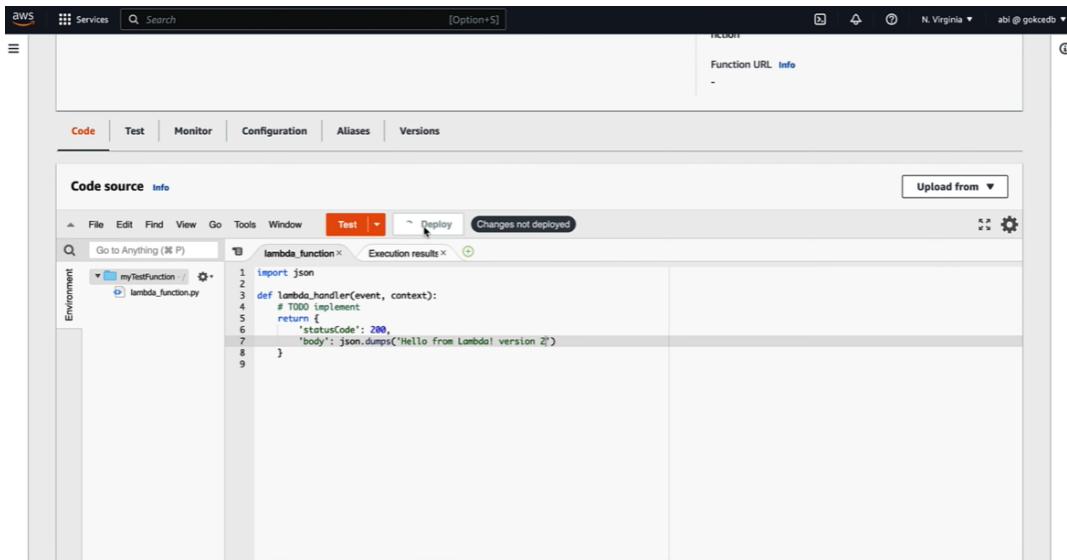




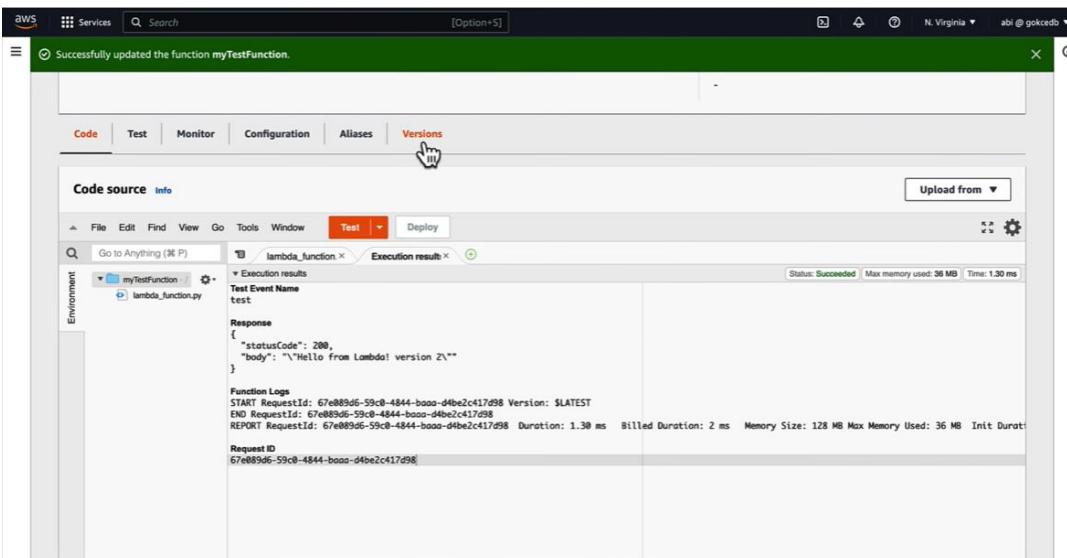
STEP 5: ENTER THE CONFIGURATIONS AND SAVE

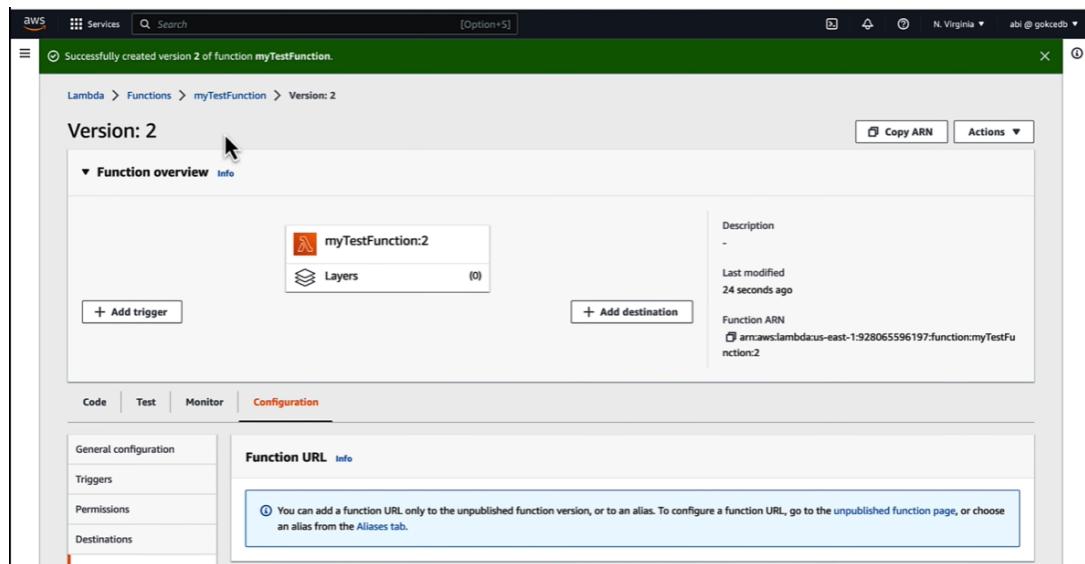
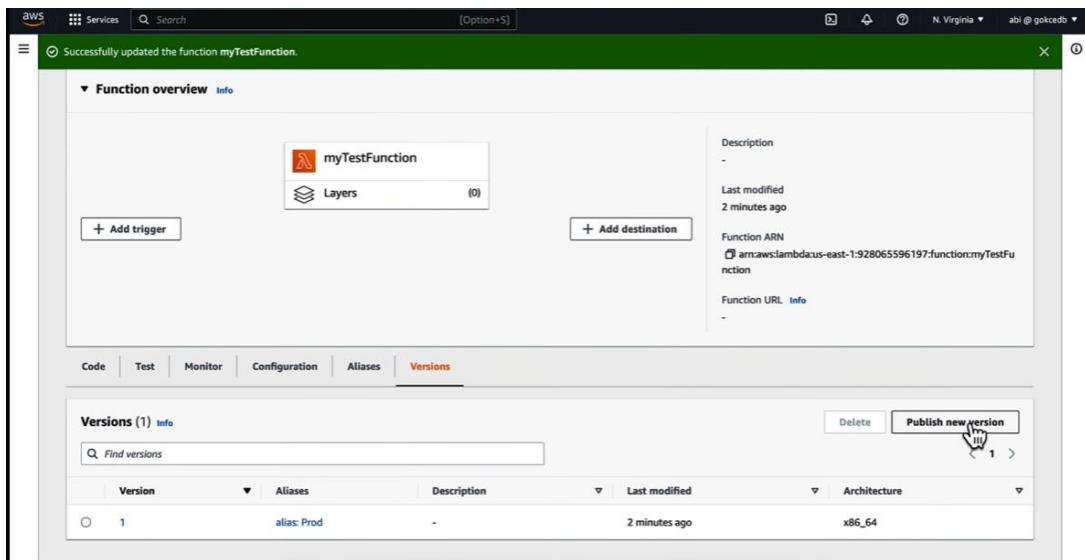


STEP 6: DEPLOY AND TEST THE CODE (VERSION 2)

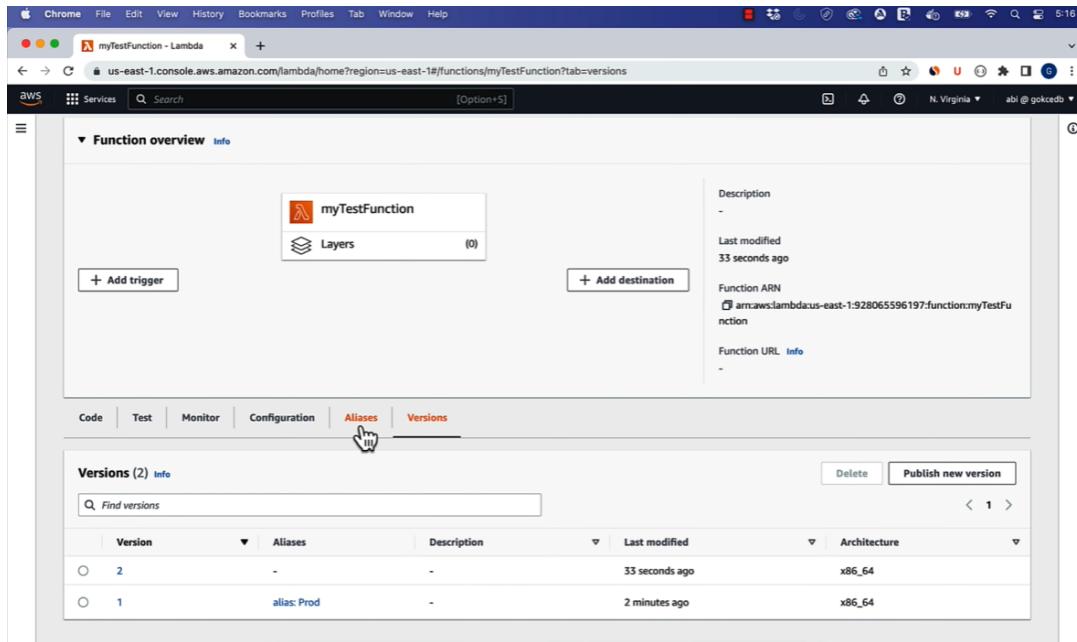


STEP 7: PUBLISH NEW VERSION



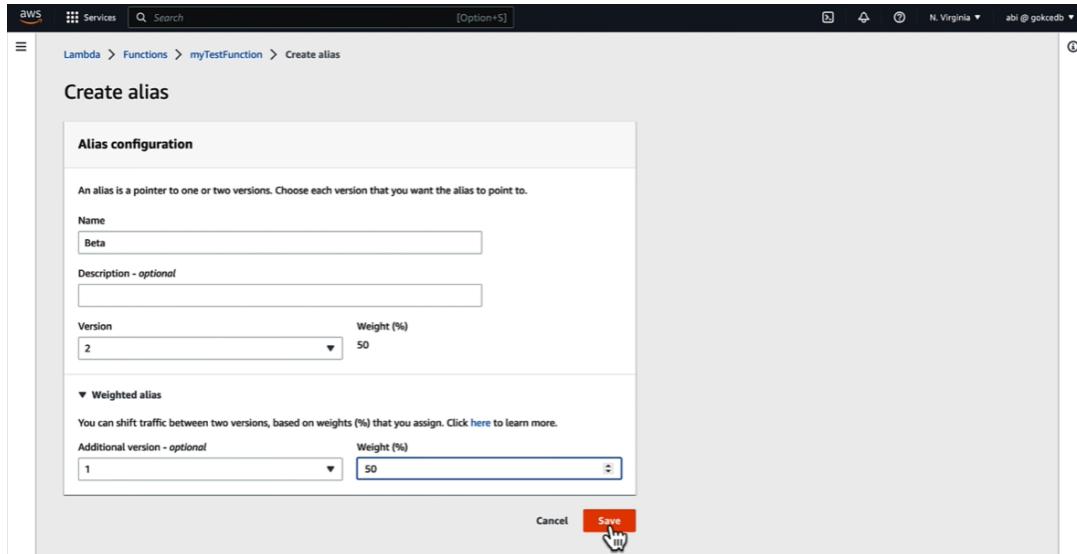


STEP 8: CREATE ALIAS FOR VERSION 2



The screenshot shows the AWS Lambda Function Overview page for a function named "myTestFunction". The "Aliases" tab is selected. There are two versions listed:

Version	Aliases	Description	Last modified	Architecture
2	-	-	33 seconds ago	x86_64
1	alias: Prod	-	2 minutes ago	x86_64



The screenshot shows the "Create alias" configuration dialog. In the "Alias configuration" section, a new alias named "Beta" is being created, pointing to Version 2 with a weight of 50%. In the "Weighted alias" section, Version 1 is listed with a weight of 50%.

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name: Beta

Description - optional:

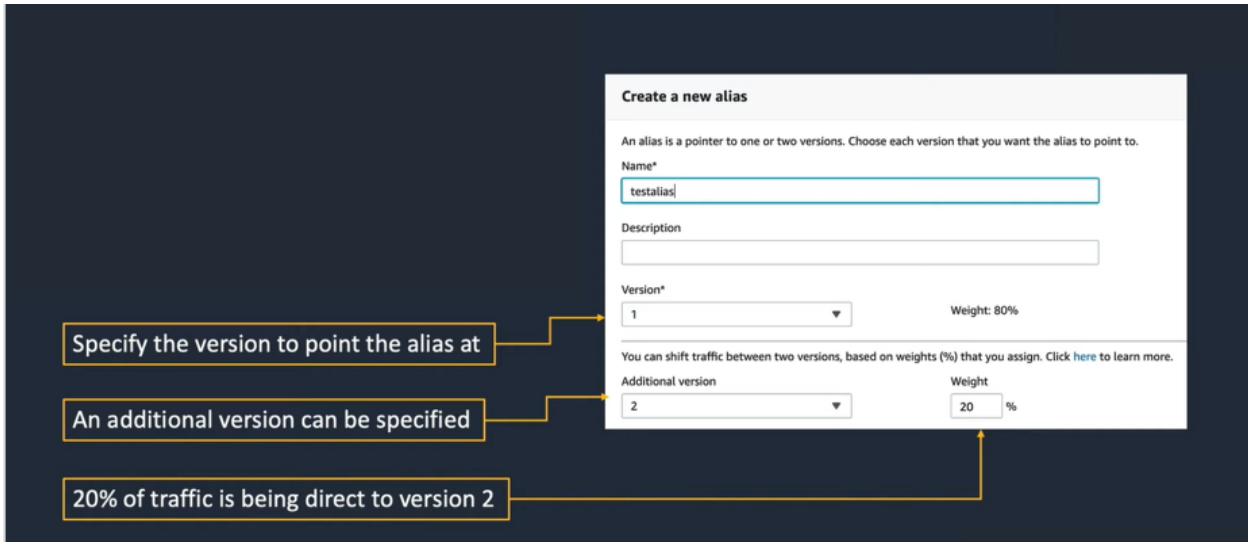
Version: 2 Weight (%): 50

Weighted alias

You can shift traffic between two versions, based on weights (%) that you assign. Click [here](#) to learn more.

Additional version - optional: 1 Weight (%): 50

Cancel **Save**



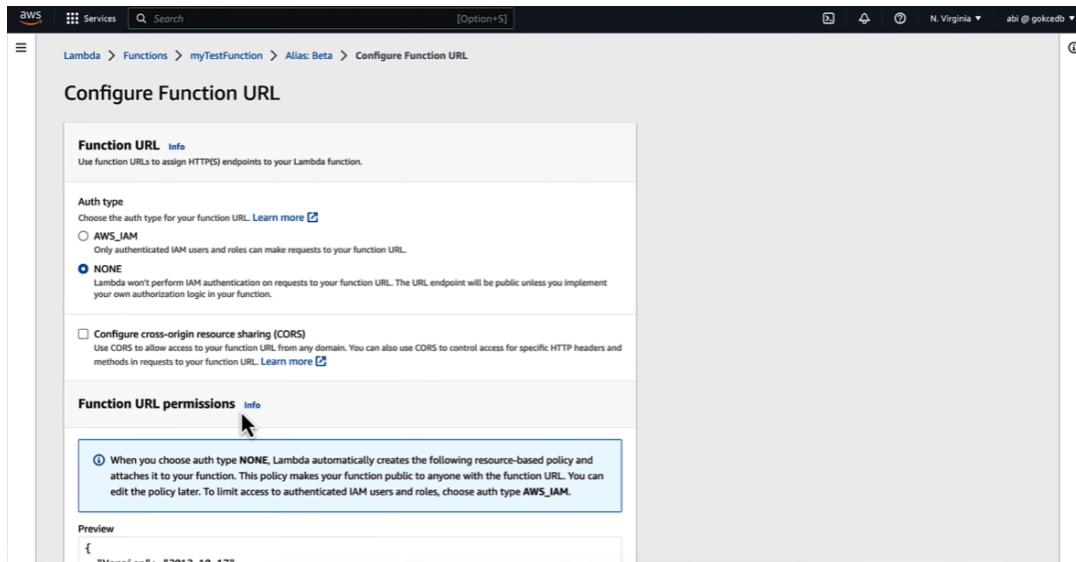
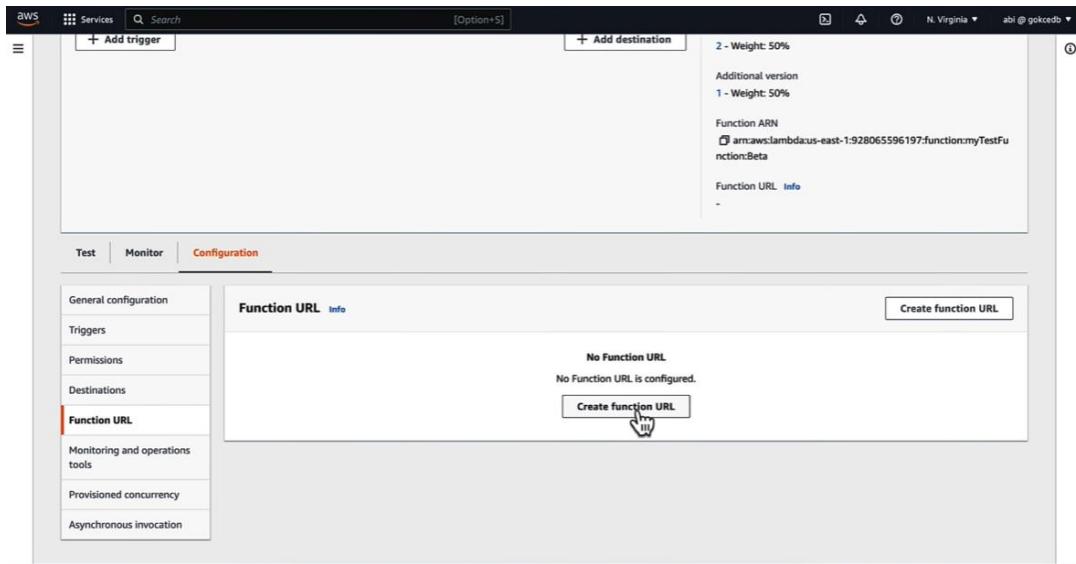
STEP 9: TWO ALIAS CREATED

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

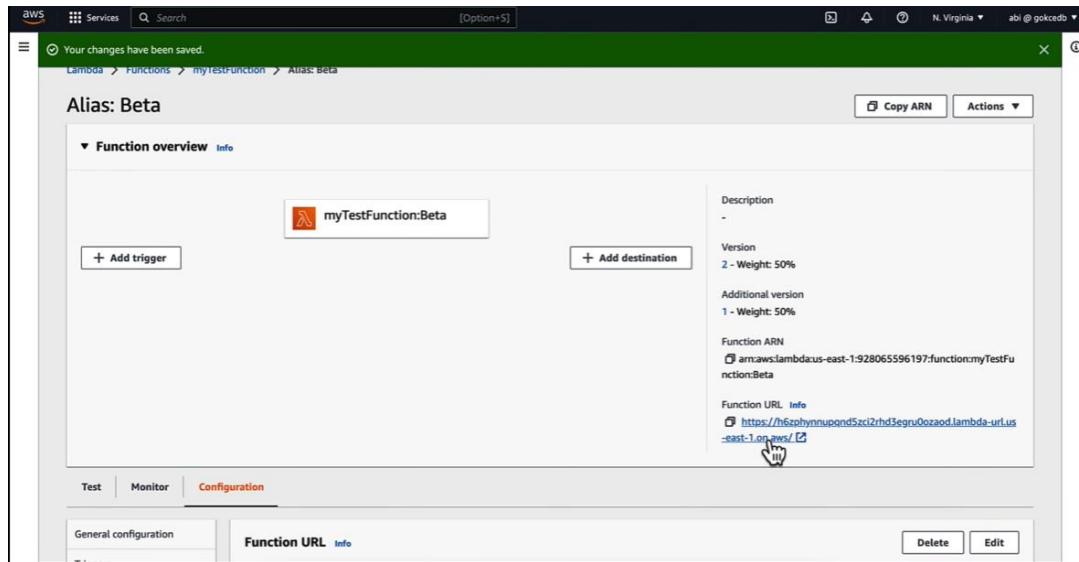
Aliases (2) Info

Name	Versions	Description
Beta	version: 2 (weight=50%) version: 1 (weight=50%)	-
Prod	version: 1 (weight=100%)	-

STEP 10: CREATE FUNCTION URL FOR BETA



STEP 11: CLICK THE FUNCTION URL AND CHECK THE OUTPUT



STEP 12: REFRESH THE PAGE

