# Serverless S3 File Downloader Using Lambda and API Gateway

## Contents:

1. Objective
2. Technology Stack
3. Architecture
4. Steps to create “Serverless S3 File Downloader using Lambda and API Gateway”
5. Output
6. Advantages
7. Disadvantages
8. Conclusion

## *Objective:*

The objective of this architecture is to build a serverless file download system where users can download files from an Amazon S3 bucket through an API Gateway endpoint triggered by an AWS (Amazon Web Services) Lambda function. This approach eliminates the need for provisioning and managing servers, offering a cost-effective and scalable solution.

## *Technology Stack:*

**AWS Lambda:** Serverless compute service for executing code in response to events. Handles the file download logic.

**AWS S3:** Simple Storage Service for storing and retrieving files. Lambda functions can interact with S3 to download files.

## Supported Storage Tiers for Direct Downloads:

* + Standard
  + Intelligent-Tiering
  + Infrequent Access (IA)
  + One Zone-Infrequent Access (One Zone-IA)
  + Glacier Instant Retrieval

## Limitations for Deep Archive and Flexible Retrieval Tiers:

1. Direct downloads not supported:
   * Unlike files in Standard and other accessible tiers, files stored in Deep Archive and Flexible Retrieval cannot be directly downloaded through the current architecture. Downloading them requires additional steps.
2. Retrieval process and considerations:

Deep Archive:

* + Downloading files from Deep Archive involves initiating a retrieval job through the AWS Console, API, or SDK. This job incurs an additional cost and takes time to complete (typically hours).
  + Once the job finishes, you can integrate a separate Lambda function or background process to download the file, adding complexity.

Flexible Retrieval:

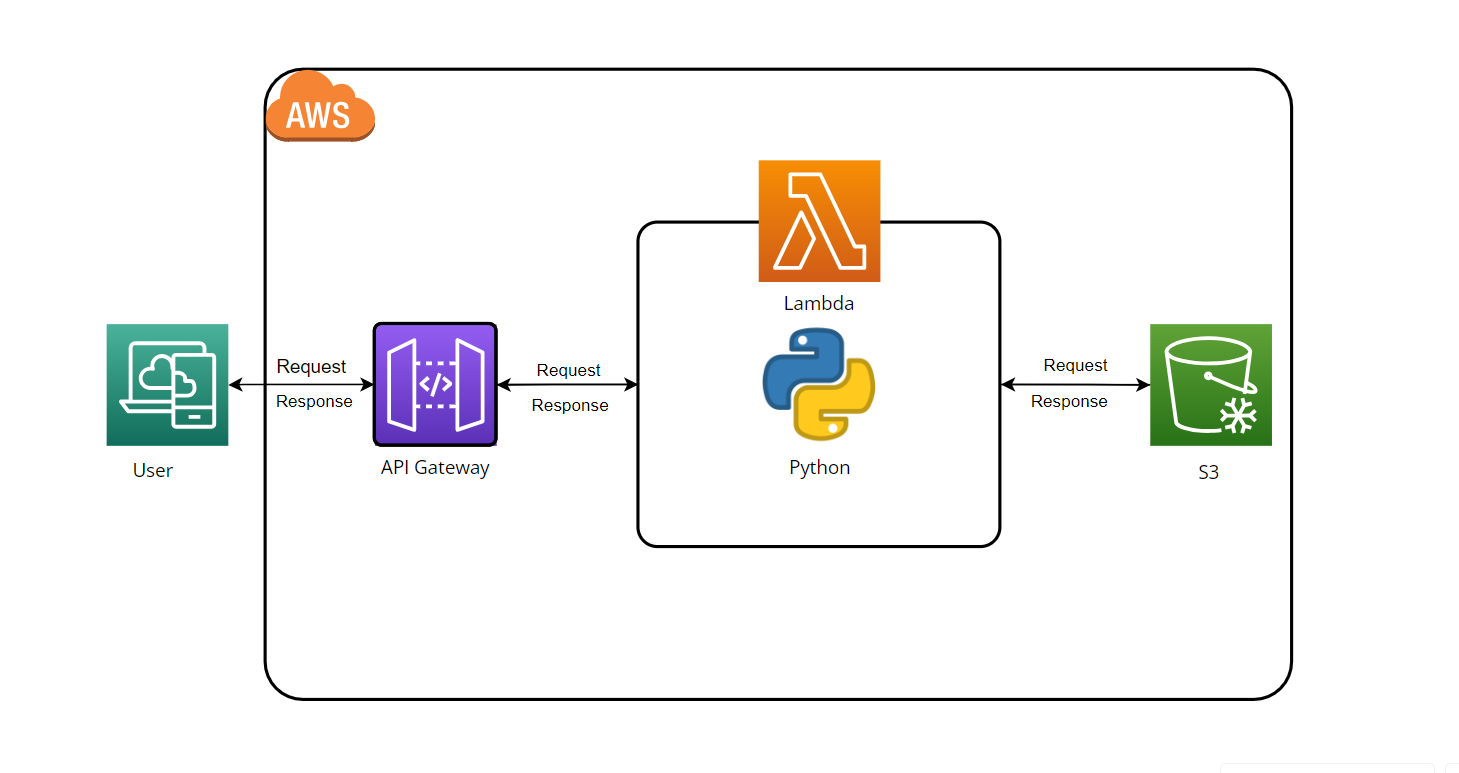
* + Offers two retrieval options: Standard (3-5 hours) and Bulk (5-12 hours). Both incur additional costs compared to Standard tier downloads.
  + These retrievals must be initiated outside of your current Lambda function flow through the same tools as Deep Archive.

1. Performance and cost implications:
   * Downloading from Deep Archive and Flexible Retrieval adds extra steps and potential delays compared to other tiers, impacting responsiveness.
   * The retrieval process incurs additional costs, which can become significant for frequent downloads or large files.
2. Increased complexity and management:
   * Integrating retrieval jobs and potentially additional Lambda functions adds complexity to your architecture and requires separate management and monitoring.

**AWS API Gateway:** Fully managed service for creating and managing RESTful APIs. Servers as the HTTP endpoint for clients to trigger Lambda functions.

**AWS IAM:** Identity and Access Management for securing and managing access to AWS services.

## *Architecture*:

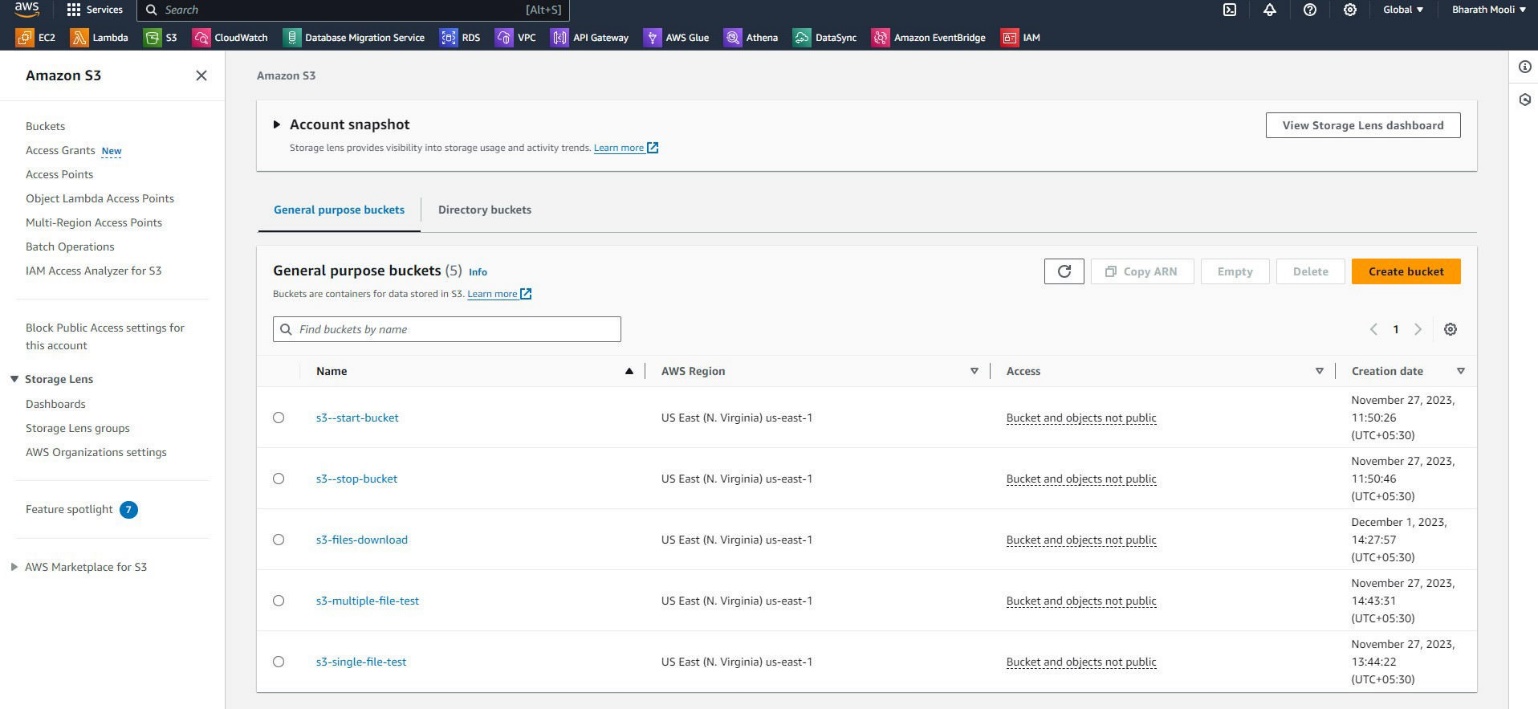


***Steps for creation of “Serverless S3 File Downloader using AWS Lambda and API Gateway.”***

**S3 BUCKET**

**Creating a Bucket:**

1. Click on the **Create bucket** button.
2. **Enter a unique bucket name** (must be lowercase and between 3-63 characters long, without special characters or underscores).
3. Choose a **region** where you want your bucket to reside. This will determine the physical location of your data.
4. (Optional) Configure any additional settings like **versioning** or **server-side encryption**.
5. Click **Create bucket**.

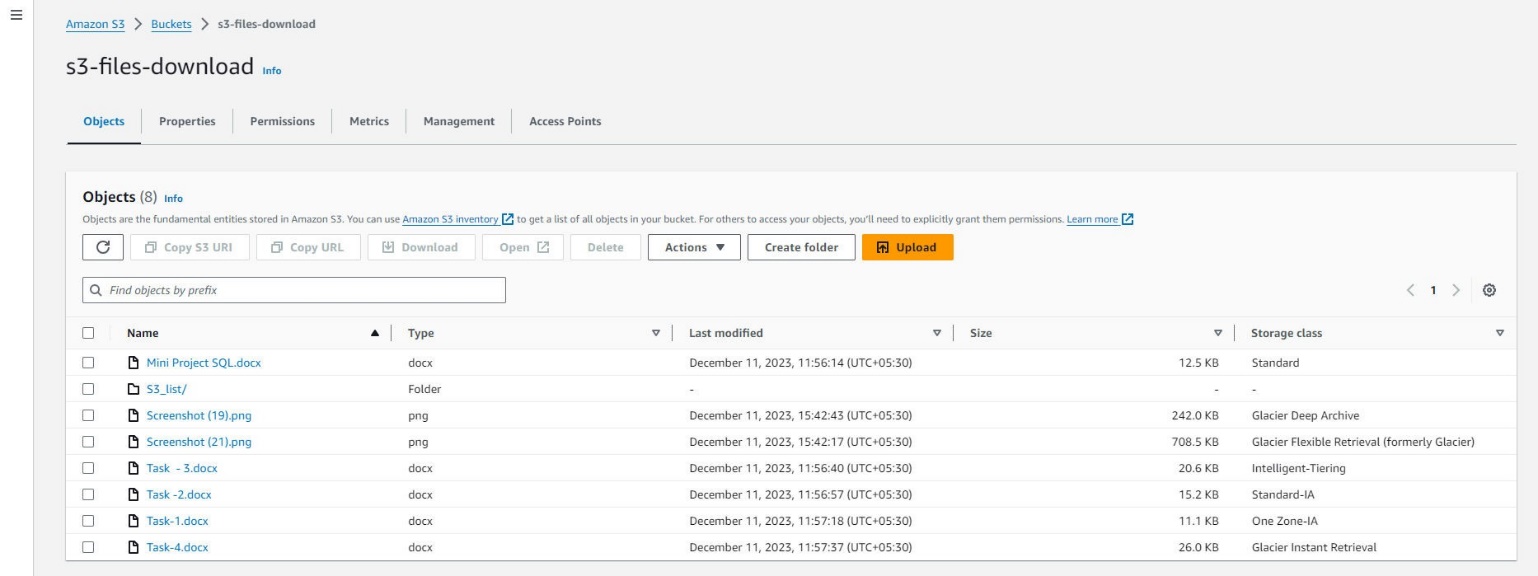


## Selecting the Bucket:

1. Once created, your new bucket will appear on the list.
2. Click on the **checkbox** next to your newly created bucket to select it.

## Uploading Files:

1. Click on the **Upload** button
2. You can either **drag and drop files** from your computer or click **choose files** to select them manually.
3. Click on **Upload**.

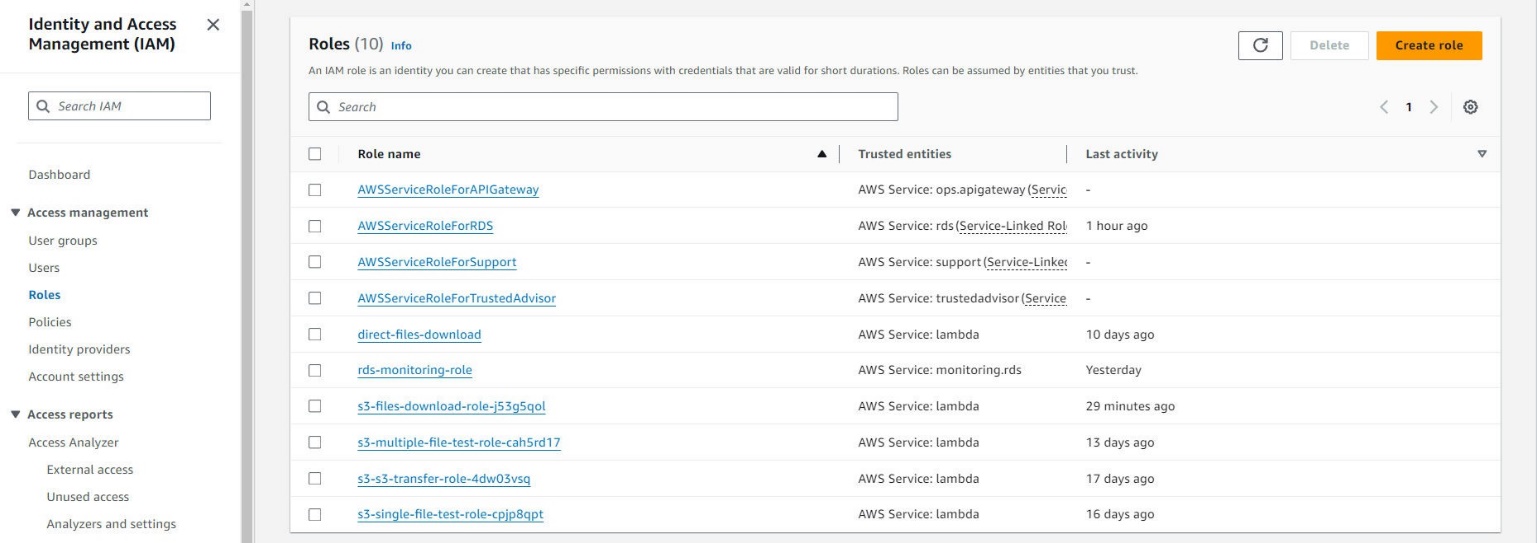


## IAM ROLE CREATION

**Creating the IAM Role:**

Navigate to the IAM console: Open the AWS Management Console and go to the Identity and Access Management (IAM) service.

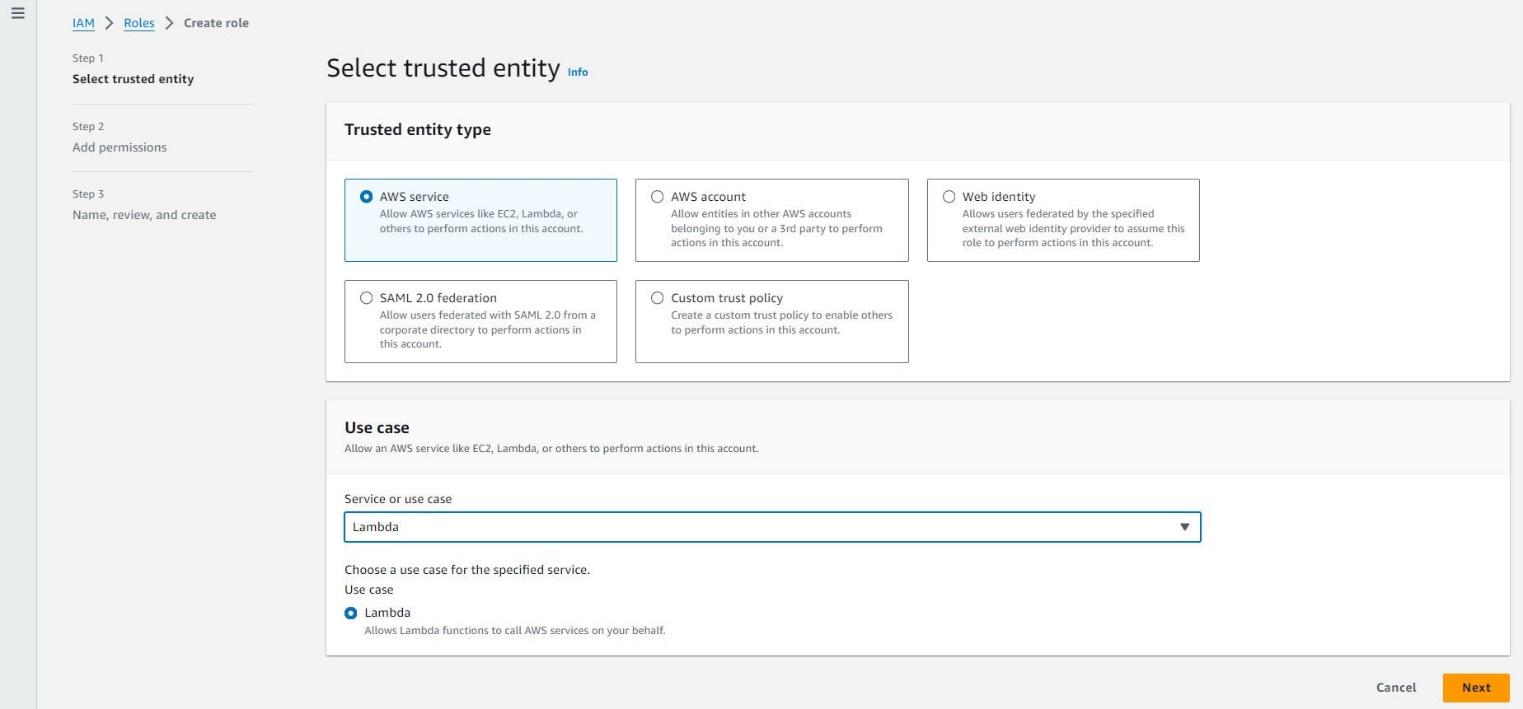
Role's tab: Click on the "Roles" tab in the navigation pane.



Create role: Click on the "Create role" button.

Trusted entity type: Select "AWS service" as the trusted entity type.

Use case or service: Choose "Lambda" from the service list.



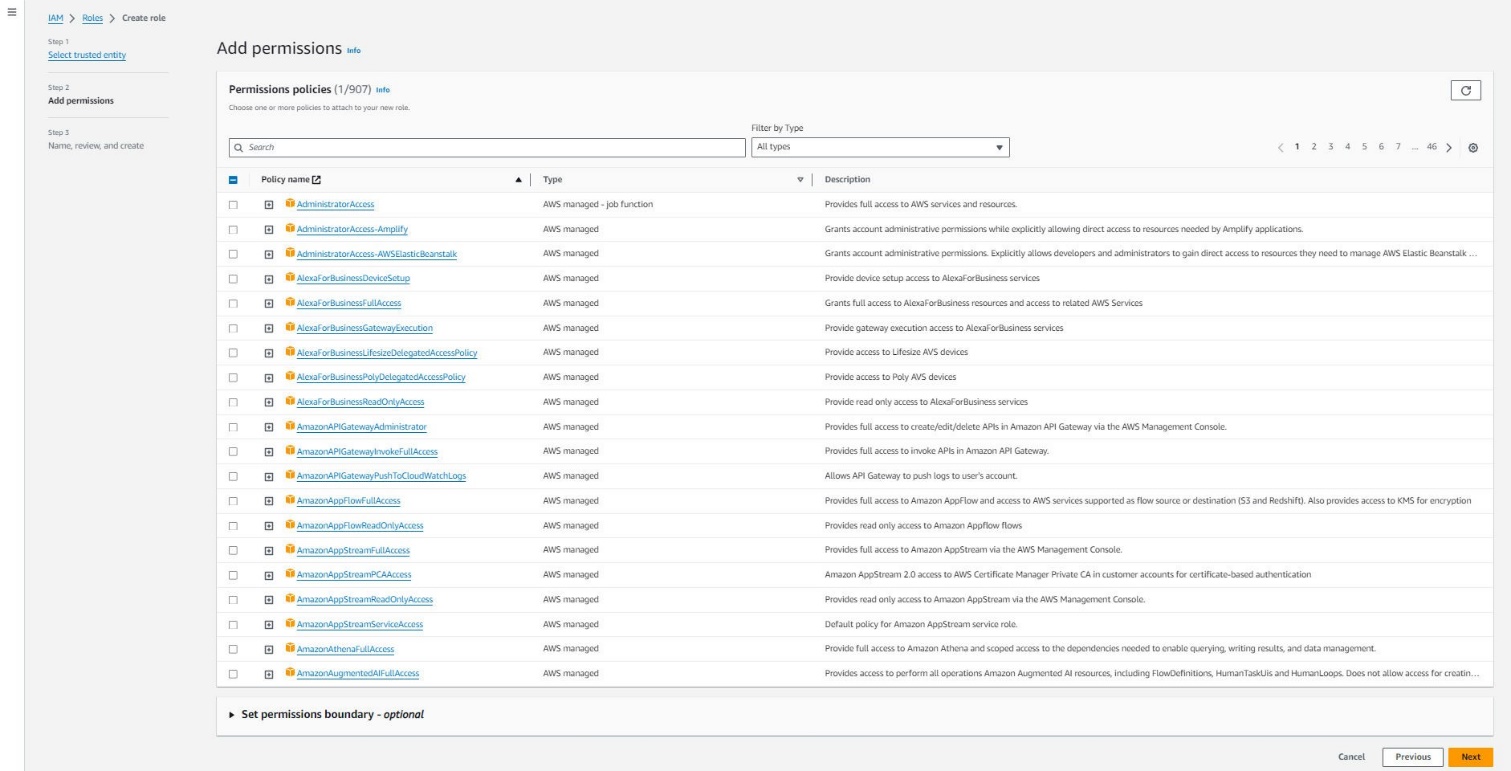
Add permissions: Click on the "Add permissions" tab. Choose permissions: Select the following IAM policies:

**AWSLambdaBasicExecutionRole:** Allows the Lambda function to run code.

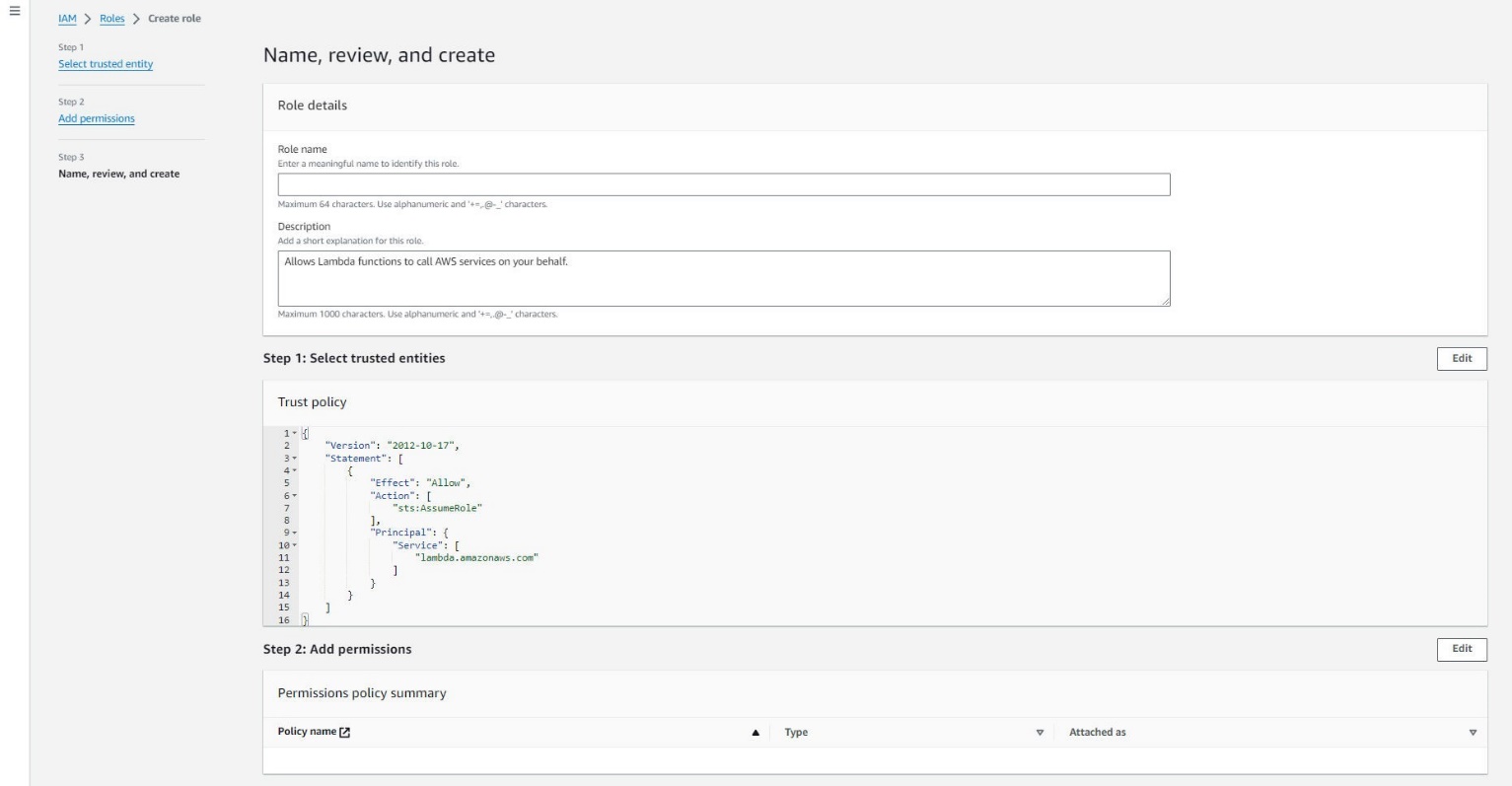
**AmazonS3FullAccess:** Grants full access to read and write files in your S3 bucket.

**AmazonAPIGatewayInvokeAccess:** Allows API Gateway to invoke your Lambda function.

Next: Click on "Next" to proceed.



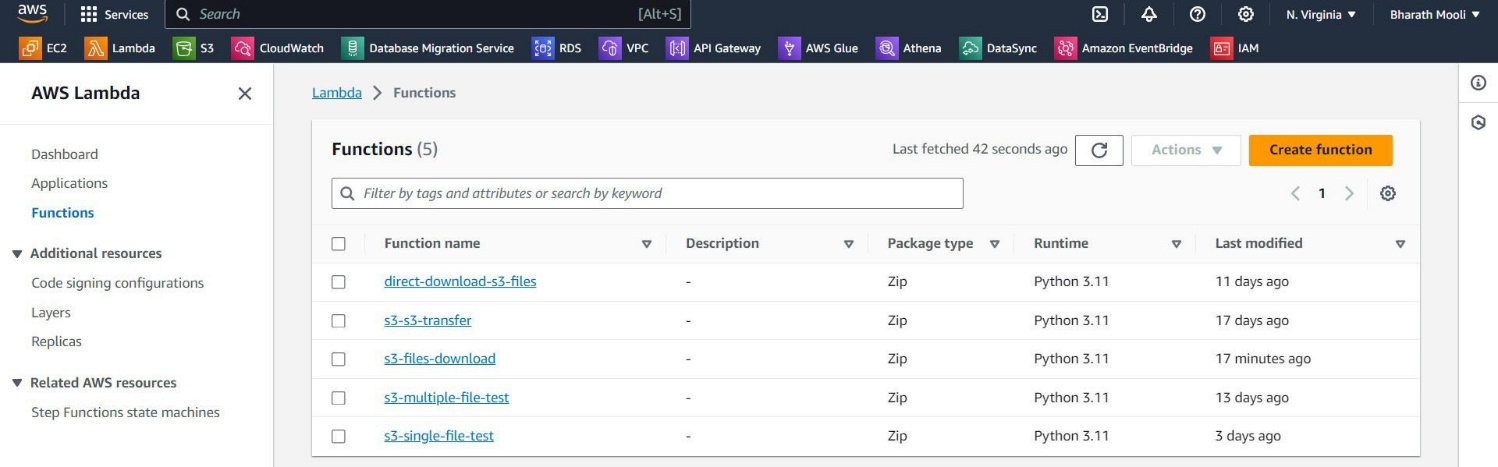
Role name and description: Provide a name for your IAM role and an optional description. Create role: Click on "Create role" to create the IAM role.



## LAMBDA FUNCTION

1. **Navigate to the Lambda service:**

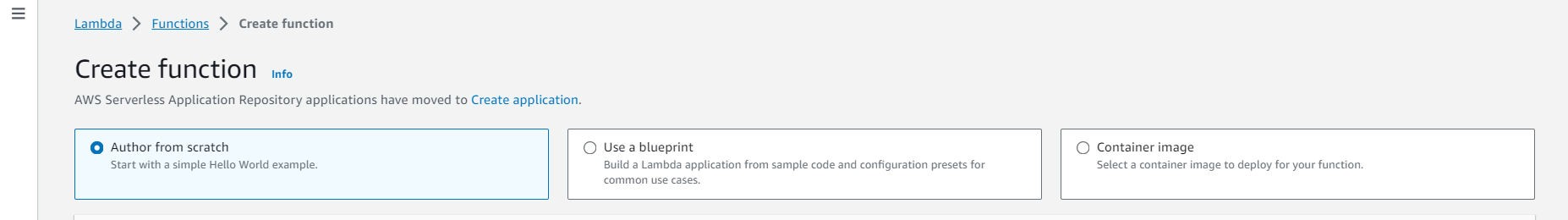
Open the AWS Management Console and go to the Lambda service.



## Create a new function:

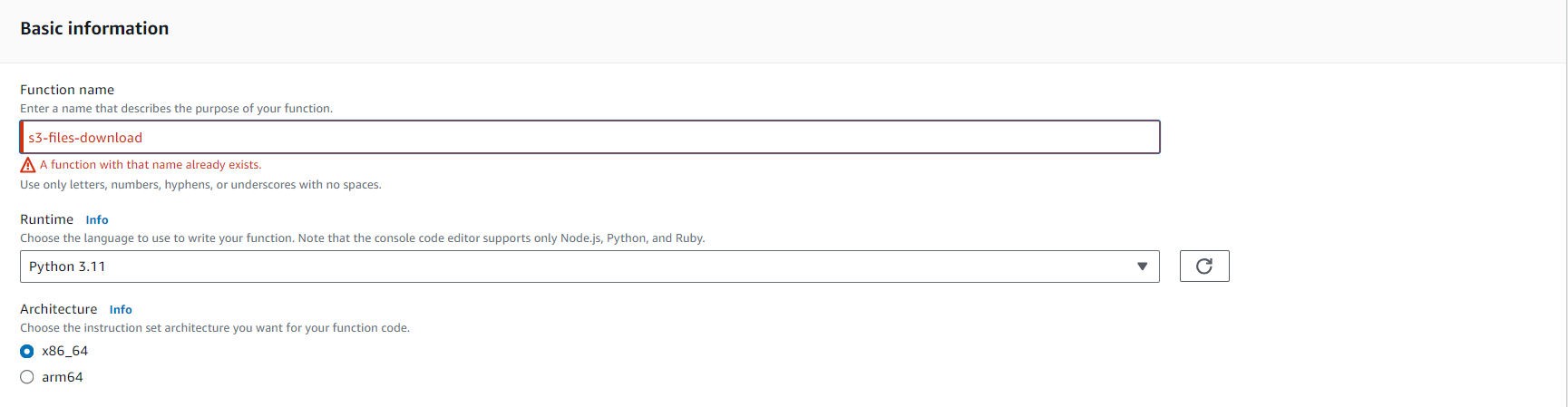
Click on the "Create function" button.

Select "Author from scratch" as the creation method.



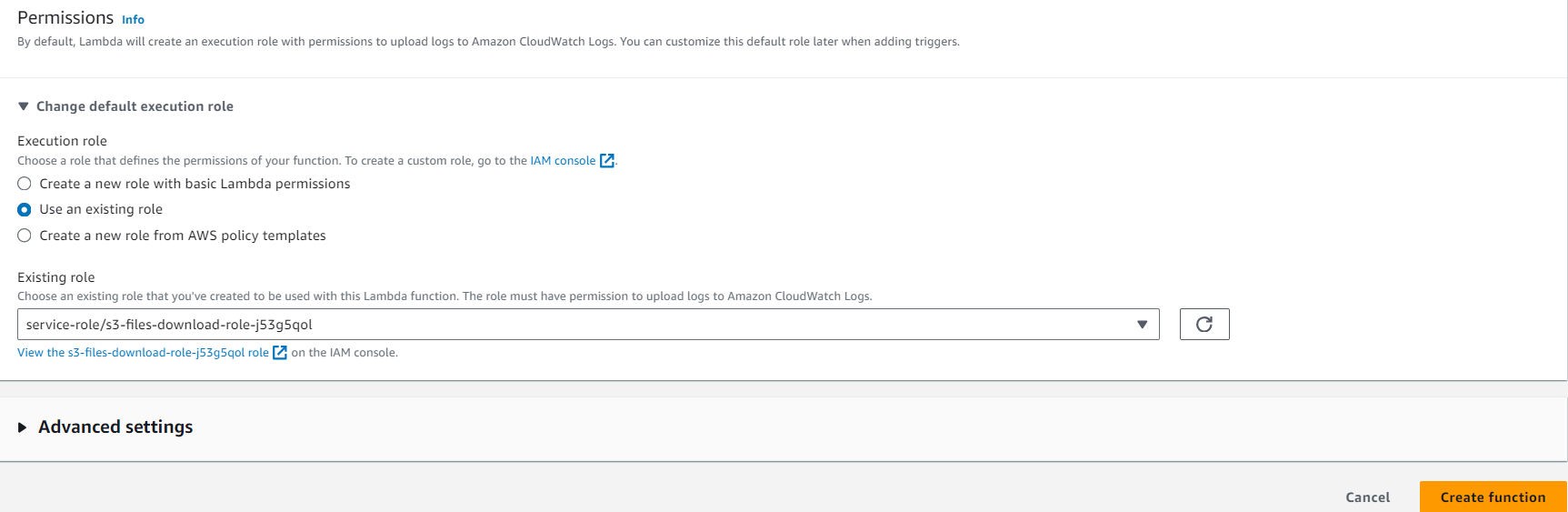
## Configure basic information:

Function name: Provide a descriptive name for your Lambda function. Runtime: Choose "Python 3.11" as the runtime environment for your function. Architecture: Select "x86\_64" as the architecture.



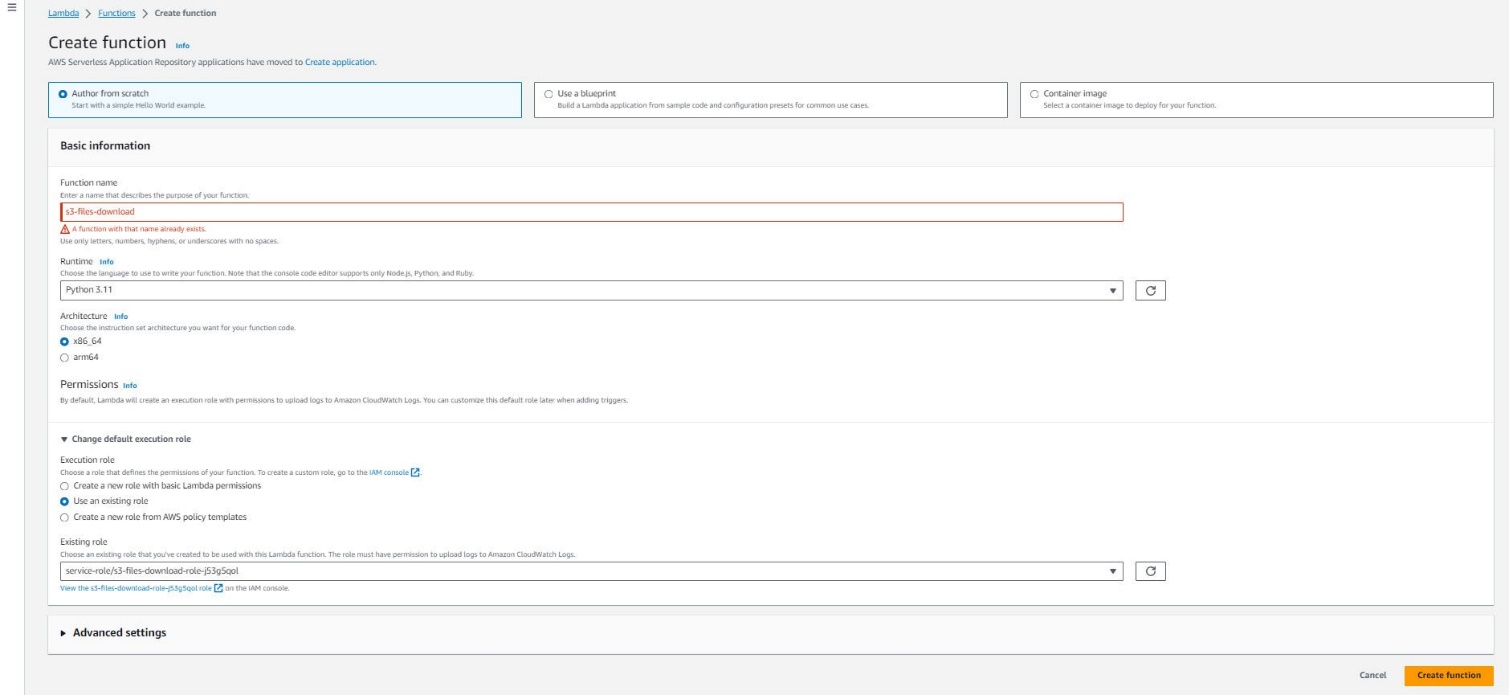
## Assign IAM role:

In the "Permissions" section, choose the IAM role you created earlier from the "Existing IAM roles" list. This role will grant your function the necessary permissions to access resources and perform its tasks.



## Create the function:

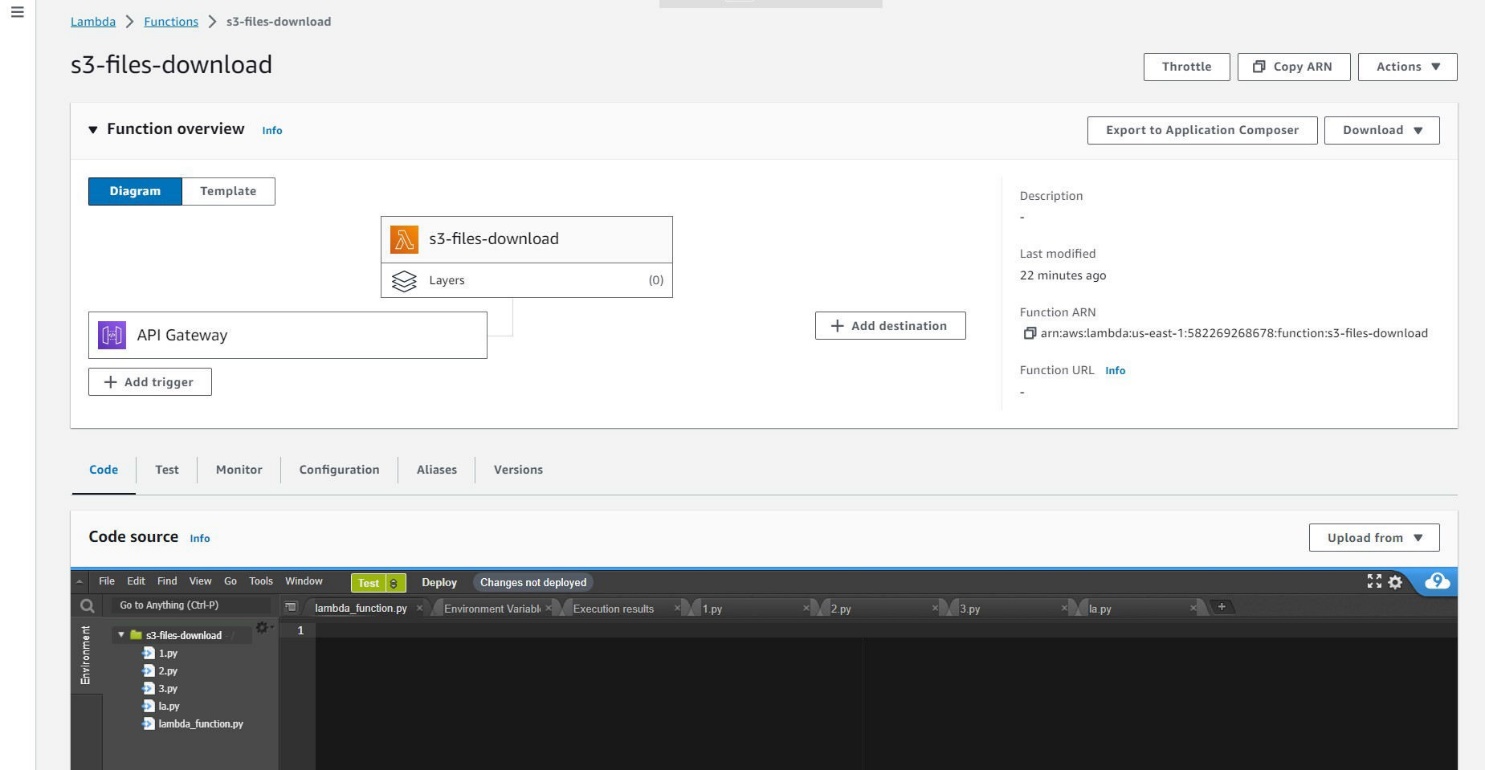
Click on "Create function" to create the Lambda function based on your chosen settings.



## 6. Copy and deploy code:

Once the function is created, you will be redirected to the code editor. Copy and paste the Python code you want your function to execute into the editor.

Click on "Deploy" to upload the code and deploy the function to AWS.



## Setting Environment Variables:

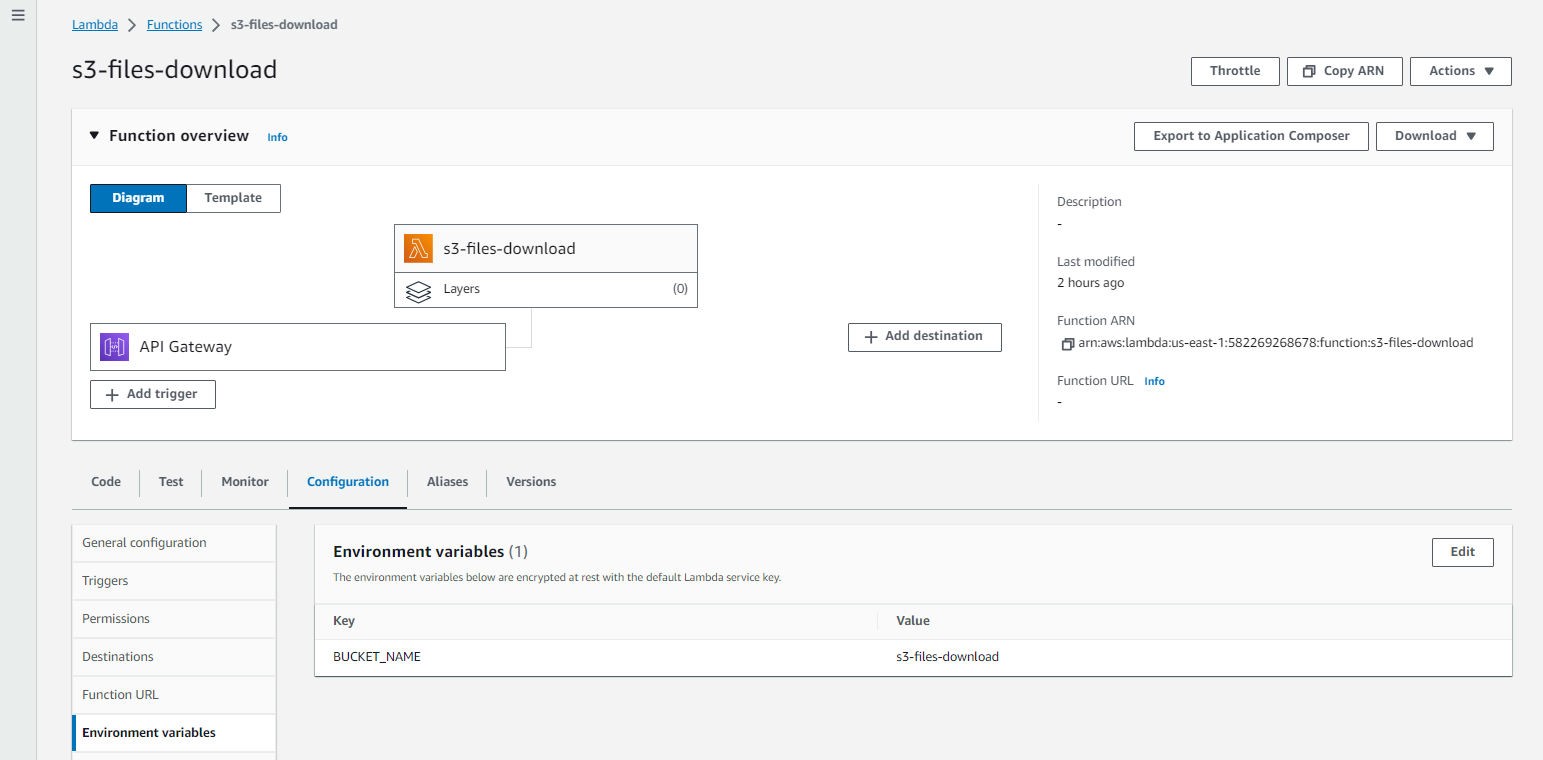
Navigate to the Lambda function: Go to the Lambda service in the console and select your function.

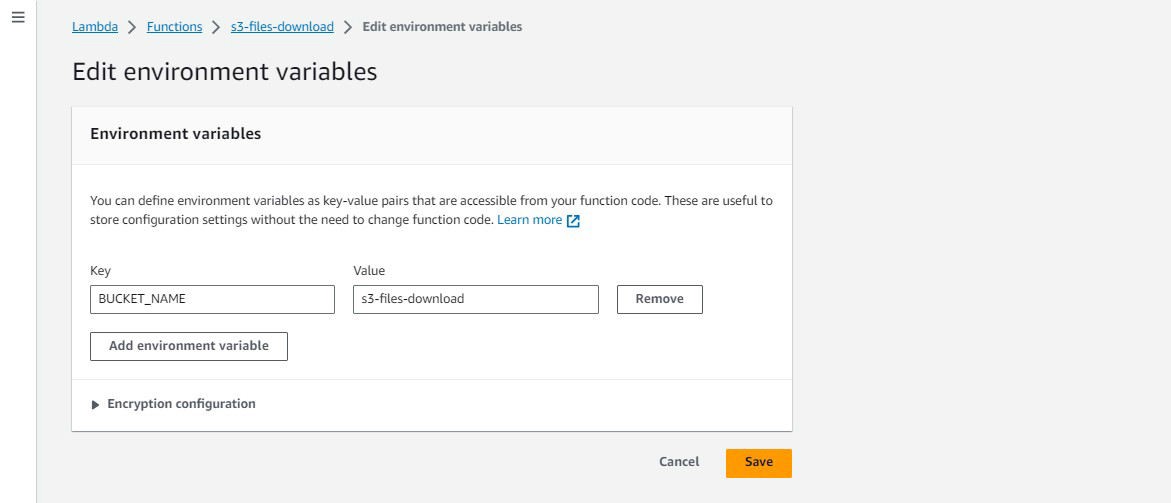
Configuration: Click on the "Configuration" tab.

Environment variables: Click on the "Environment variables" section. Add variable: Click on "Add environment variable".

Name: Enter "BUCKET\_NAME" as the name of the environment variable.

Value: Enter the name of your S3 bucket as the value of the environment variable. Save: Click on "Save" to update the environment variables.

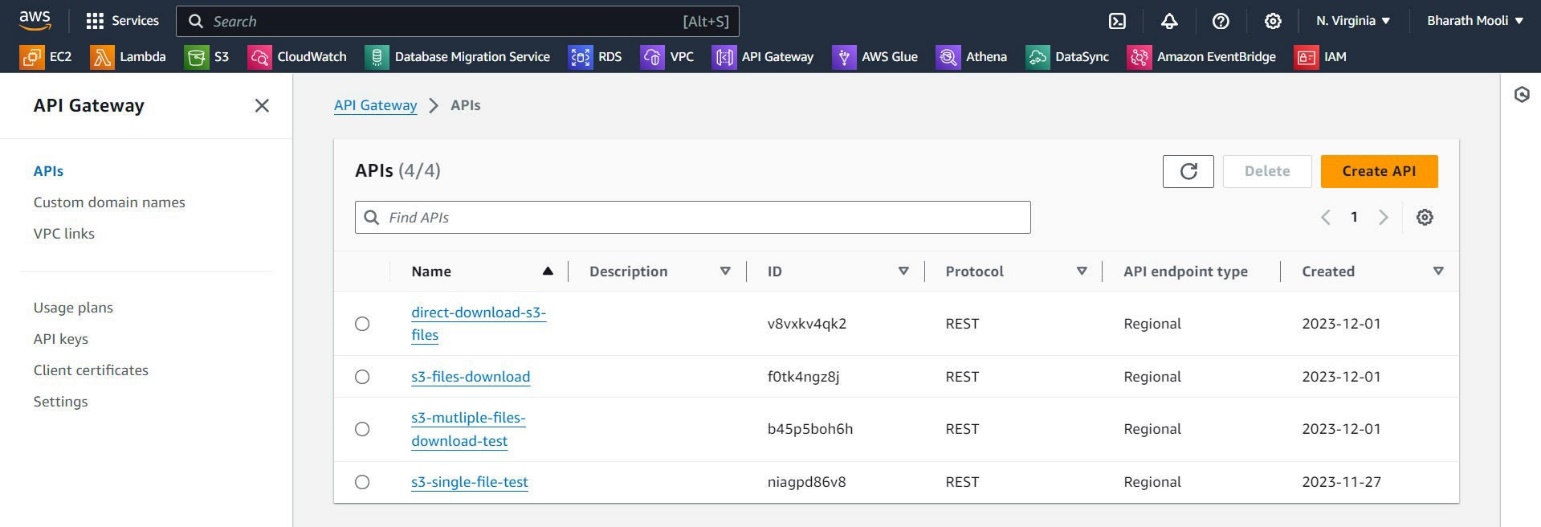




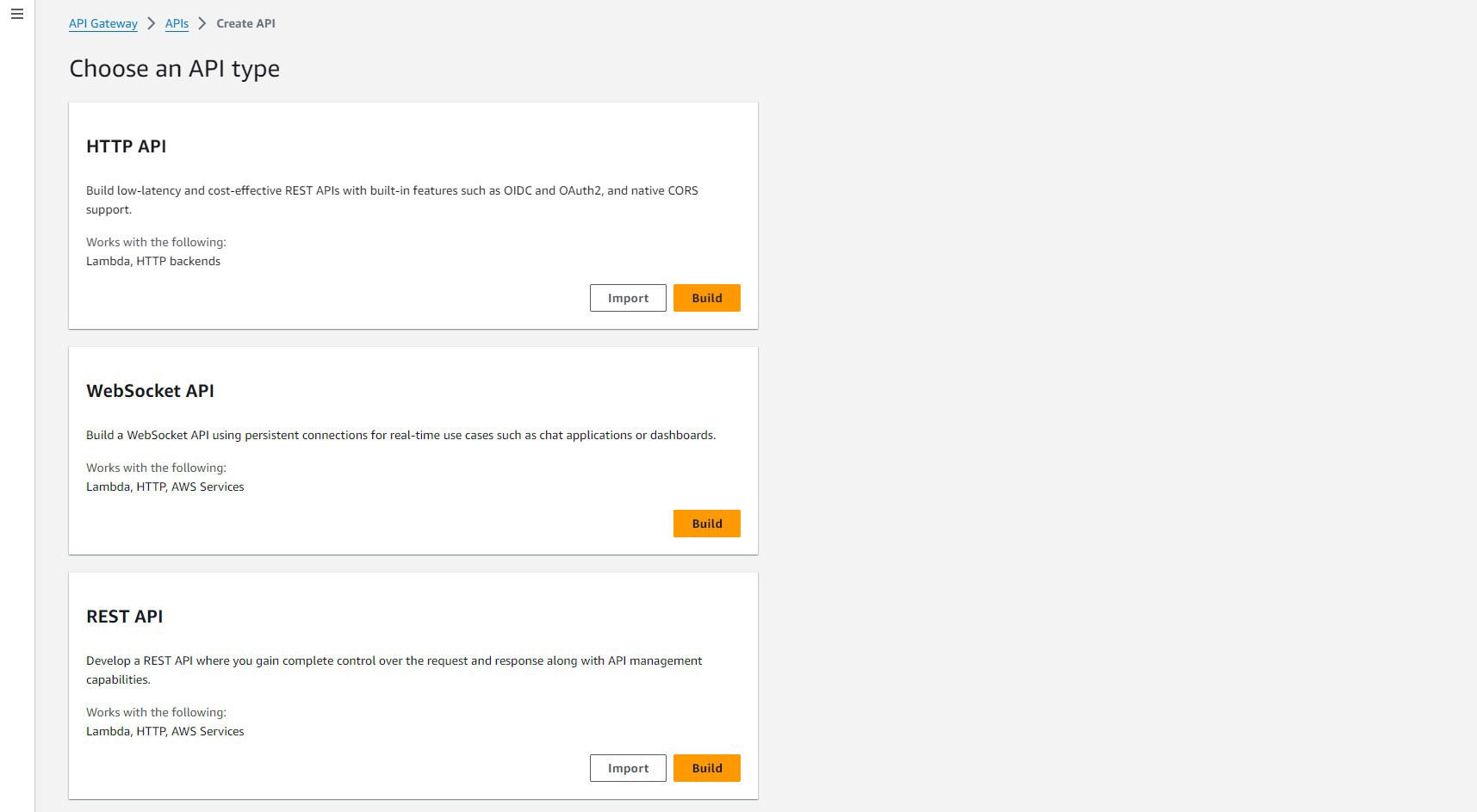
## API GATEWAY

1. **Creating the API:**

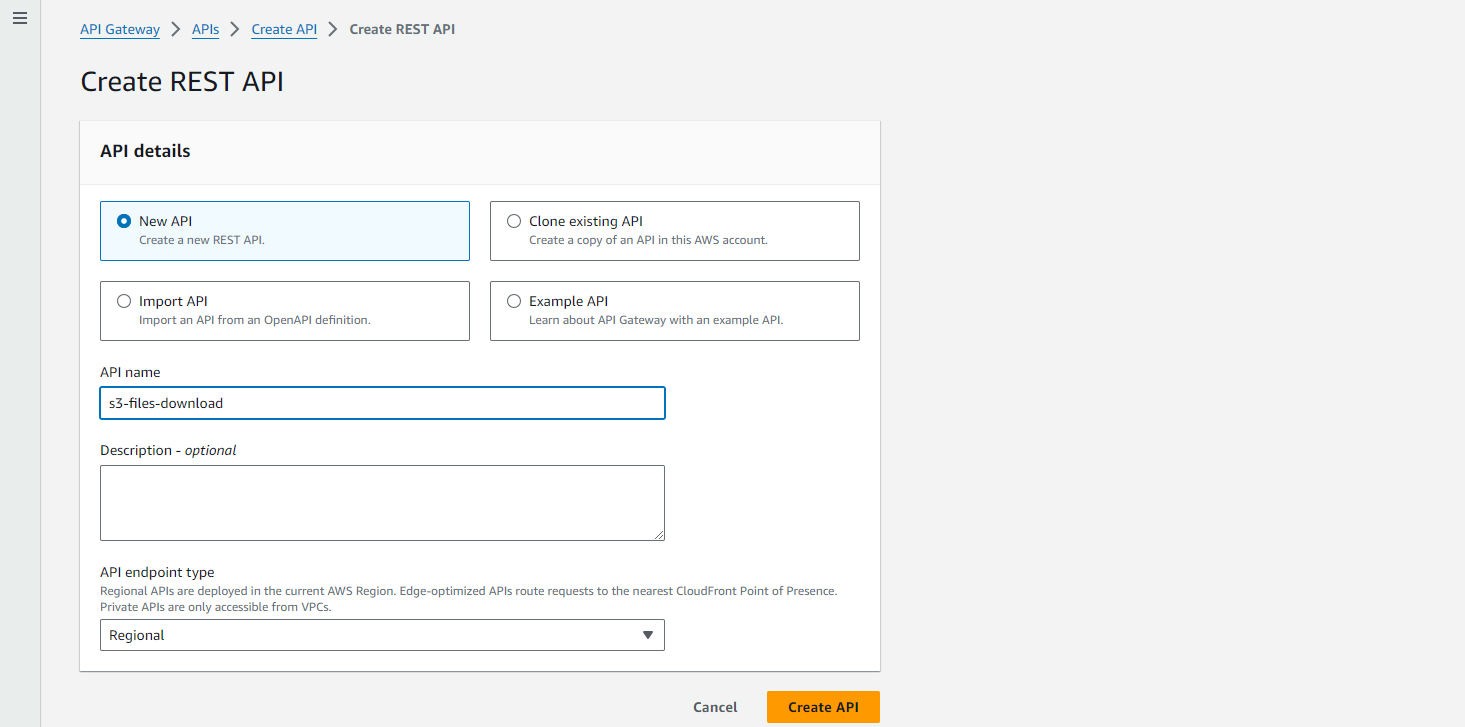
Navigate to API Gateway: Open the AWS Management Console and navigate to the API Gateway service.

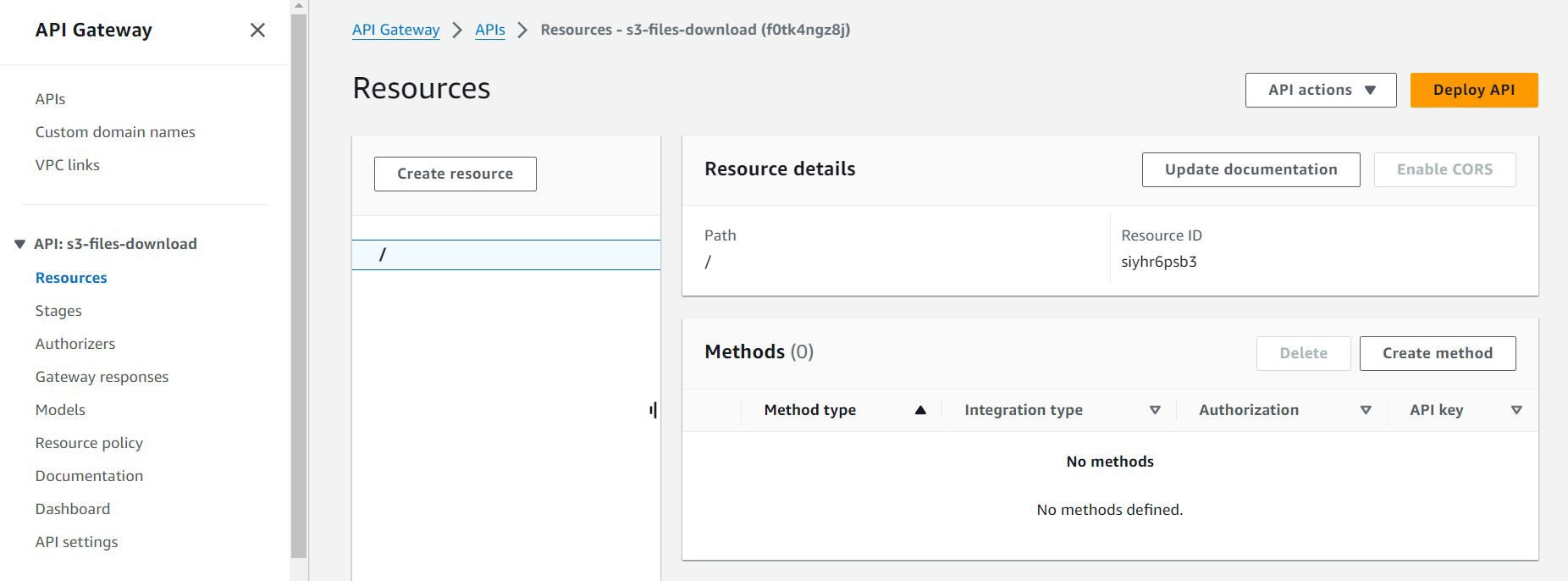


Choose REST API and Build: Select the "Create API" button and choose "REST API" as the type. Click "Build" to proceed.



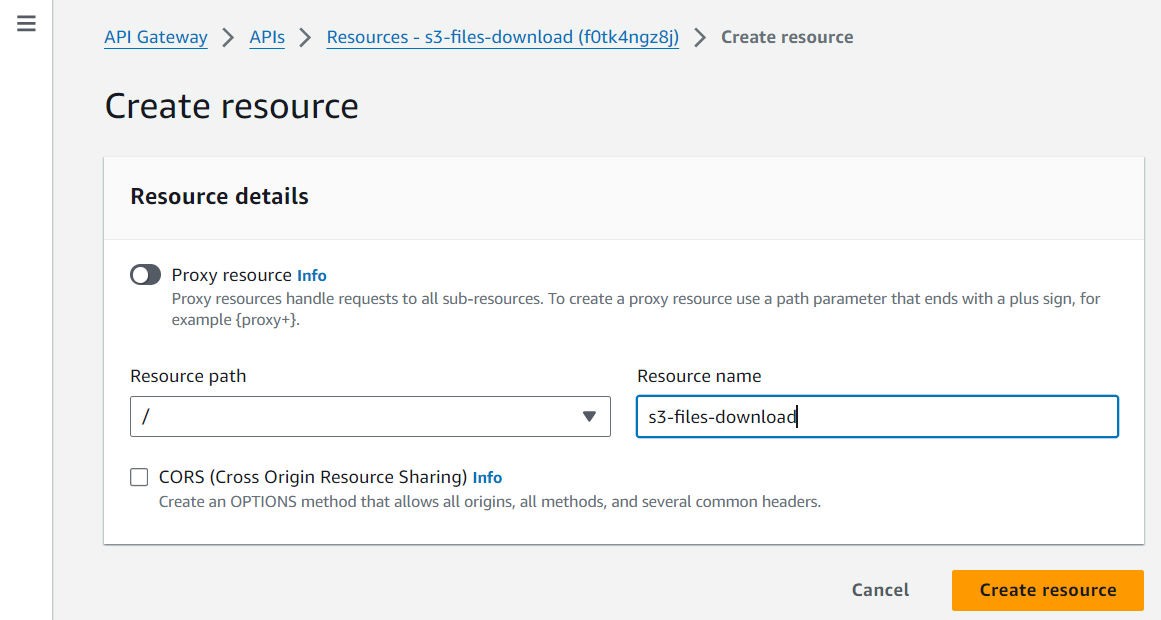
New API and Name: Enter a descriptive name for your API. Choose "REGIONAL" for the API endpoint type, indicating it will be accessible from any region within your AWS account. Click "Create API" to finalize.



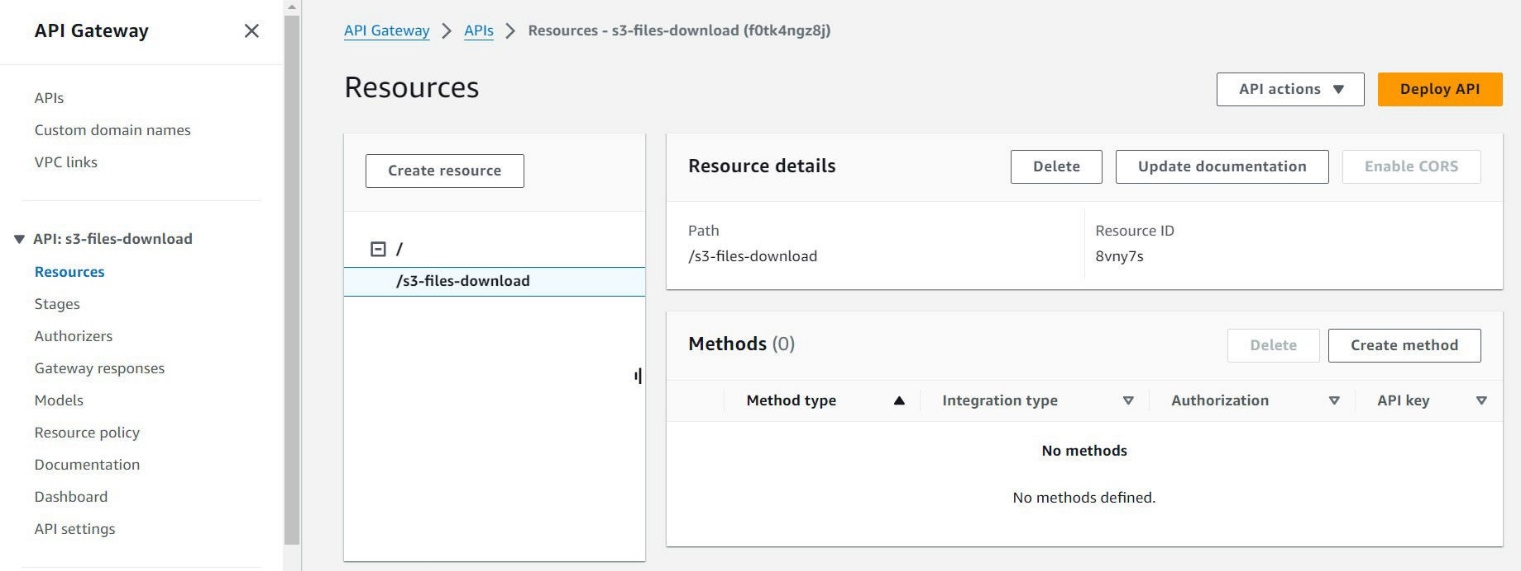


## Adding a Resource and Method:

Create Resource: Click the "Create Resource" button. Enter a path name for your resource, which will be part of the API URL.



Create Method: Click the "+" button next to the newly created resource and select "Add Method." Choose the "GET" method, as it will be used to access your binary content.

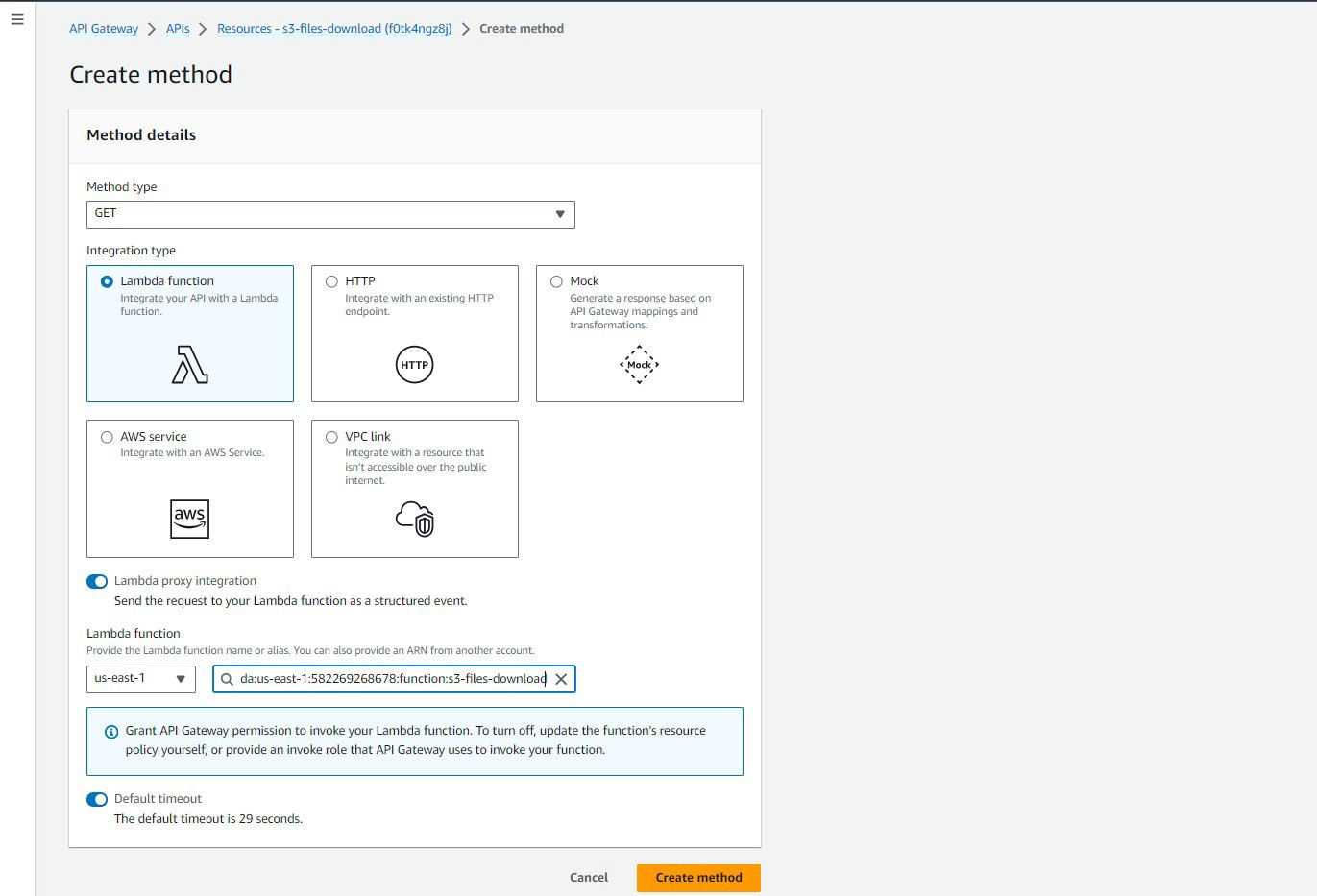


Lambda Integration: Select "Lambda Function" as the integration type. This means your API method will trigger a Lambda function to process the request.

Lambda Proxy Integration: Choose "Lambda Proxy integration" for optimal handling of requests and responses, especially with binary data.

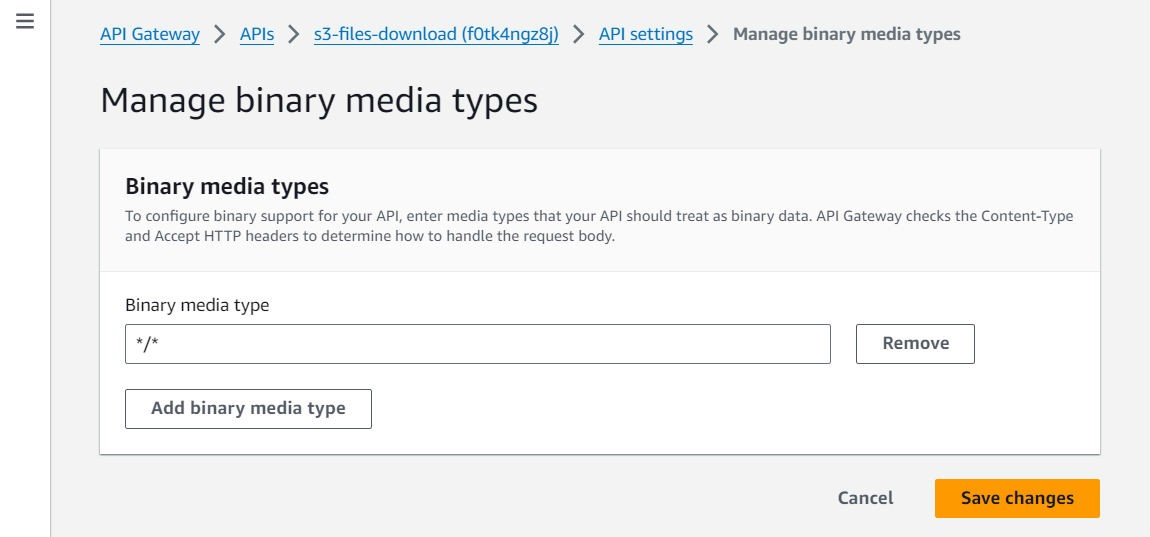
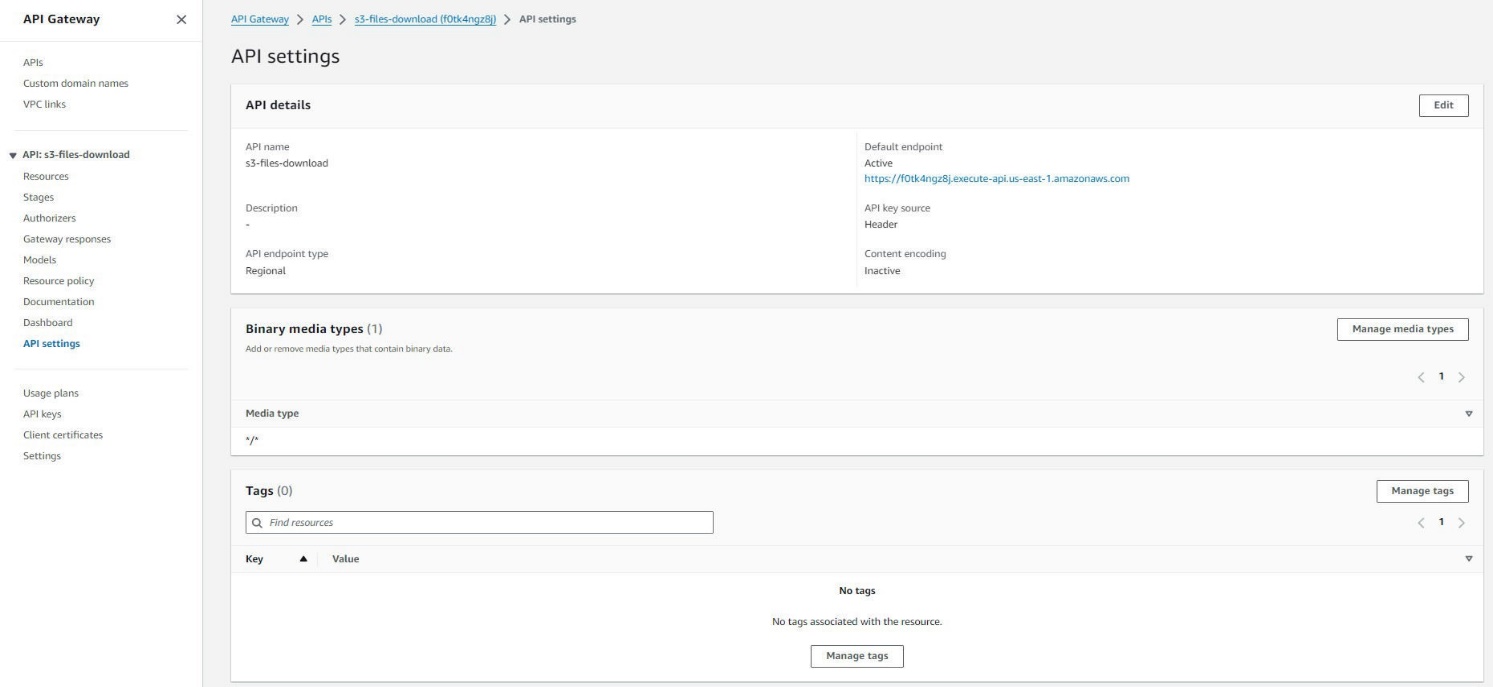
Select Lambda Function: Select the Lambda function where your code resides. This code will handle the logic for accessing and processing the binary media.

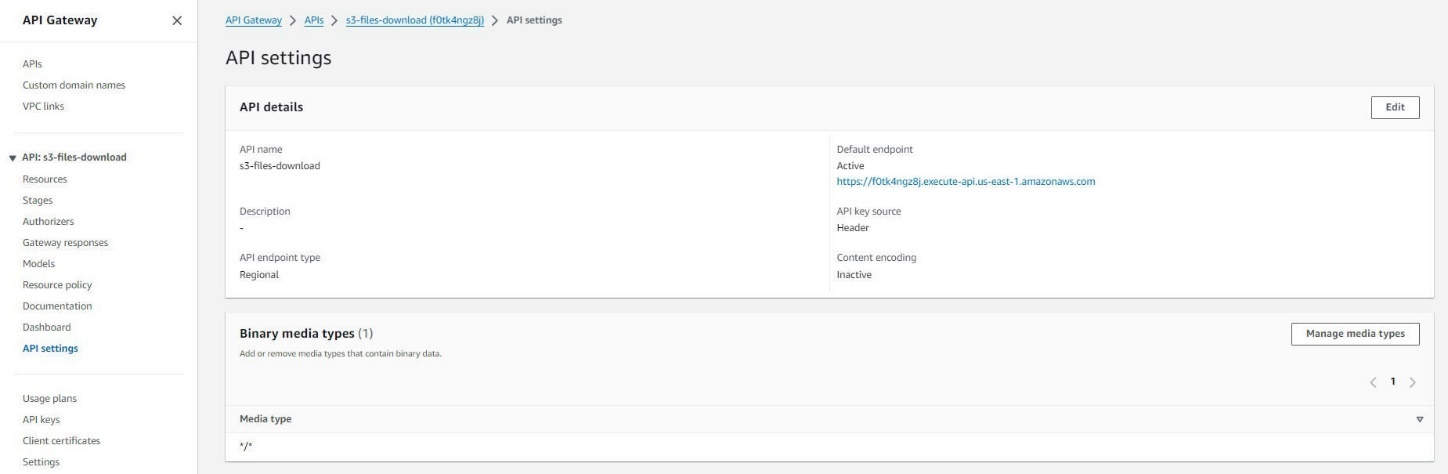
Create Method: Click "Create Method" to finalize the configuration of your API method.



## Enabling Binary Media Types:

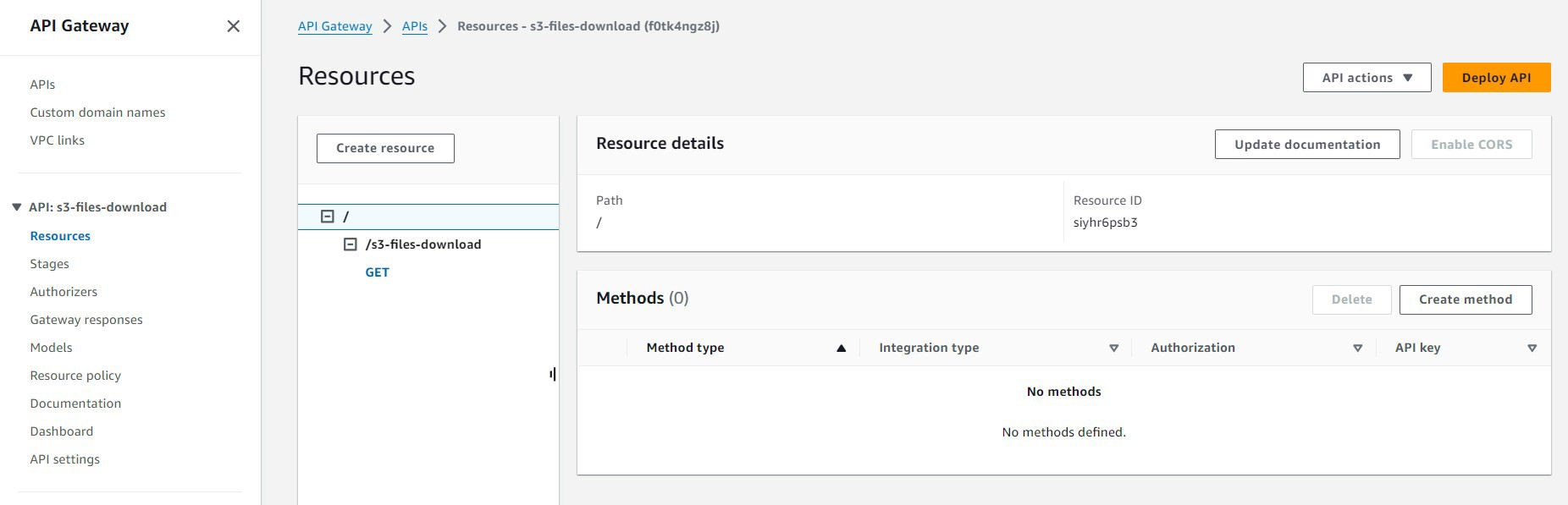
API Settings: Go to the "Settings" tab of your API.





## Deploying the API:

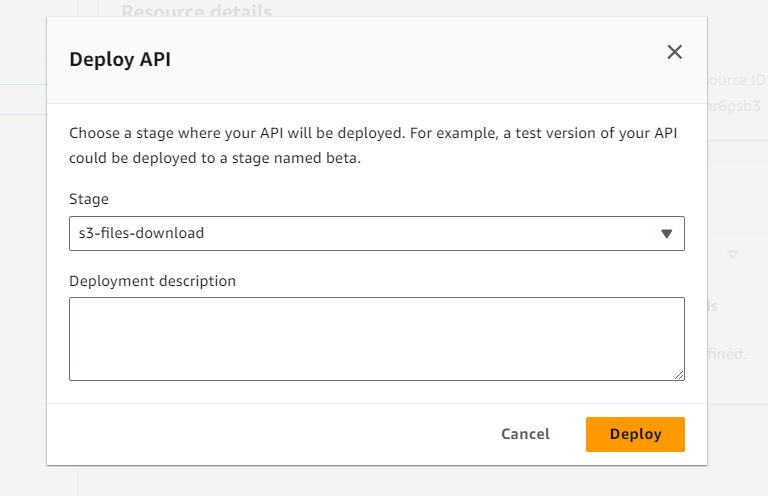
Resource Tab: Go back to the "Resources" tab of your API.



Deploy API: Click the "Actions" menu for your resource and select "Deploy API".

Stage and Stage Name: Choose a stage for deployment. Enter a descriptive name for the stage.

API URL Generation: Click "Deploy". This will create a deployment of your API and generate a URL for accessing it.

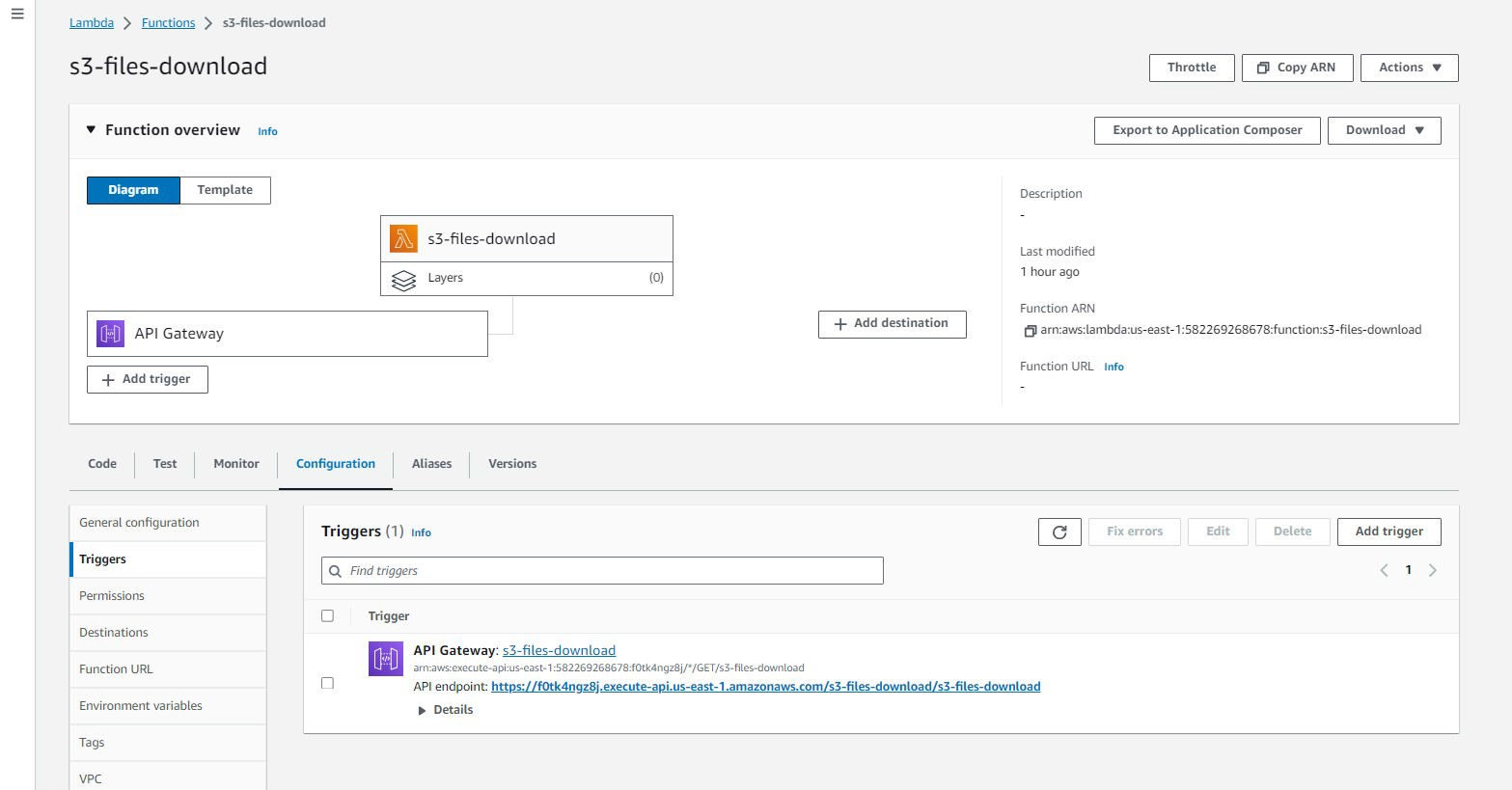


Navigate to the Lambda function: Go to the Lambda service in the console and select your function.

Configuration: Click on the "Configuration" tab.

Click on the "Triggers" section.

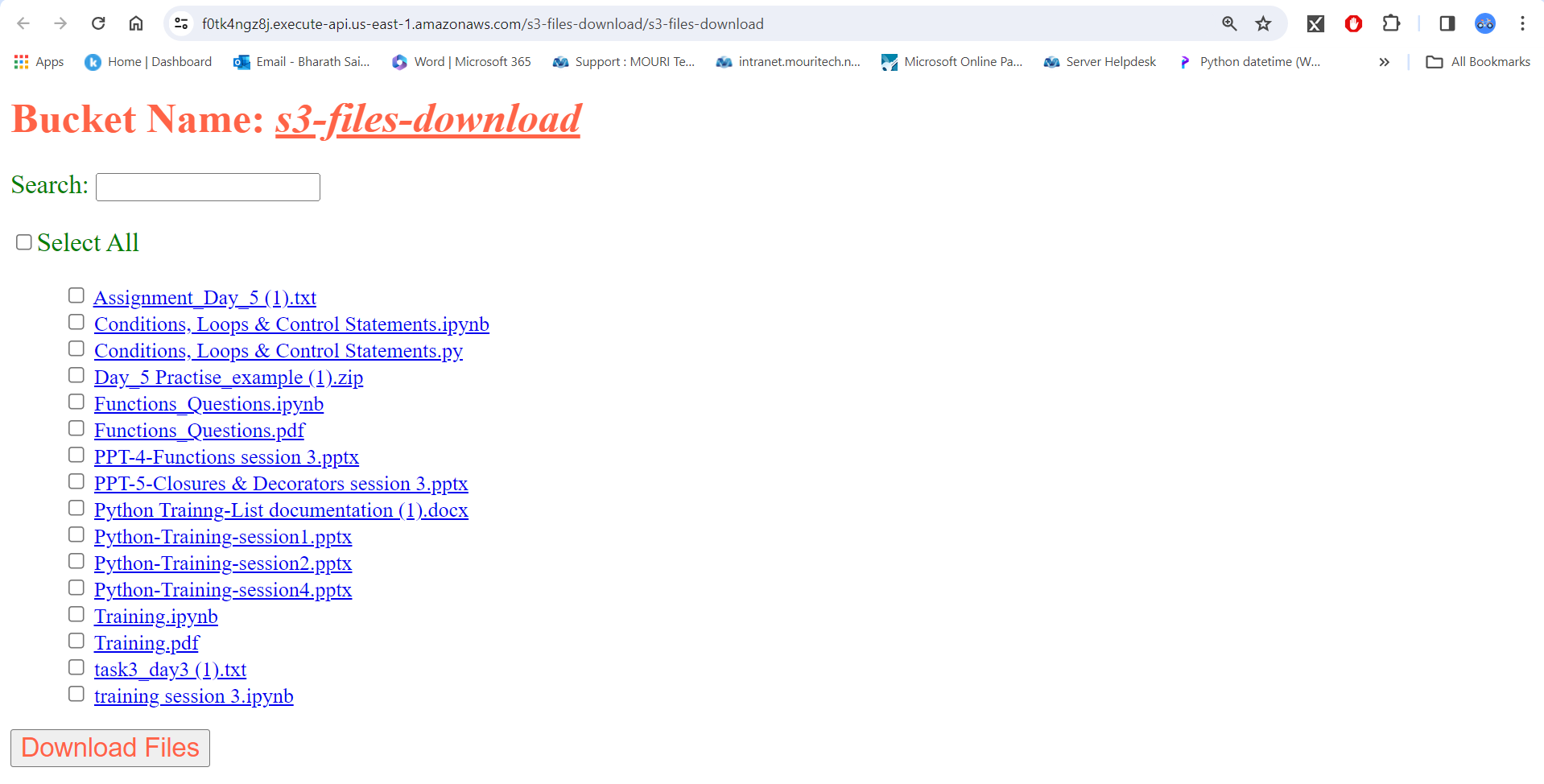
Copy the API endpoint.



## Instead of manually navigating the S3 console, the API offers a streamlined directory view of all files in the bucket. This allows us to easily identify and download the specific files we need, saving time and effort.

***Output:***

**Link:** [**https://f0tk4ngz8j.execute-api.us-east-1.amazonaws.com/s3-files-download/s3-files-download**](https://f0tk4ngz8j.execute-api.us-east-1.amazonaws.com/s3-files-download/s3-files-download)



## *Advantages*:

* Cost-efficiency: Serverless architecture eliminates costs associated with idle servers, as you only pay for Lambda execution time.
* Scalability: Handles traffic spikes automatically, without manual intervention.
* Simplified management: AWS manages infrastructure, freeing you to focus on application logic.
* Security: API Gateway and Lambda offer built-in security features, protecting sensitive files.
* Flexibility: Integrates with other AWS services for enhanced functionality.
* Faster deployments: Rapidly deploy updates without server provisioning delays.

## *Disadvantages*:

* Cold start latency: Lambda functions might experience delays for infrequent downloads.
* Limited execution time: Default 15-minute limit for Lambda functions might not suffice for exceptionally large files.
* Vendor lock-in: Reliance on AWS services can make migration to other platforms challenging.
* Cost considerations for high download volumes: Pay-per-execution model can become expensive for frequent, high-volume downloads.

## *Conclusion*:

A serverless file downloader using API Gateway and Lambda offers a cost effective, scalable, and minimal maintenance solution. While there are some limitations, careful design and consideration of these constraints lead to a robust and efficient system. This technology stack leverages AWS services to provide a fully managed and scalable architecture for file downloading.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* THE END \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*