

# Creating a Serverless API

The screenshot shows the 'Create function' page in the AWS Lambda console. At the top, there are three tabs: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below these is the 'Basic information' section with the following fields: 'Function name' (set to 'todo\_function'), 'Runtime' (set to 'Python 3.12'), and 'Architecture' (set to 'x86\_64'). The 'Permissions' section is expanded, showing 'Change default execution role' with options to 'Create a new role with basic Lambda permissions', 'Use an existing role' (selected), or 'Create a new role from AWS policy templates'. The 'Existing role' dropdown is set to 'LabRole'.

## Step1: Configure the lambda function

This screenshot shows the 'Advanced settings' section of the 'Create function' page. It displays the 'Execution role' section with the 'Existing role' dropdown set to 'LabRole'. At the bottom right, there are 'Cancel' and 'Create function' buttons.

## Step 2: Existing role, LabRole selected

The screenshot shows the 'todo\_function' overview page in the AWS Lambda console. A green banner at the top states: 'Successfully created the function todo\_function. You can now change its code and configuration. To invoke your function with a test event, choose "Test".' The page has tabs for 'Diagram' and 'Template'. The 'Diagram' tab is active, showing a visual representation of the function with a 'Layers' section. On the right, there is a 'Description' section with fields for 'Function ARN' (arn:aws:lambda:us-east-1:471112841584:function:todo\_function) and 'Function URL'. At the bottom, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'.

## Step 3: Successfully created the lambda function

DynamoDB > Tables > Create table

## Create table

### Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

**Sort key - optional**  
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

1 to 255 characters and case sensitive.

### Table settings

☒ **Default settings**  
The fastest way to create your table. You can modify these settings now or after your table has been created.

☐ **Customize settings**  
Use these advanced features to make DynamoDB work better for your needs.

Step 4: Configured the dynamoDB, adding the table name and partition key

The todos-db table was created successfully.

DynamoDB > Tables

Tables (1) Info

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size	Table class
<input type="checkbox"/>	todos-db	Active	id (S)	-	0	Off	Provisioned (S)	Provisioned (S)	0 bytes	Standard

Step 5: New database table created successfully

API Gateway > APIs

APIs (1/1)

<input type="radio"/>	Name	Description	ID	Protocol	API endpoint type	Created
<input type="radio"/>	serverless-demo		u64n6c33c	REST	Regional	2024-02-22

Step 6: Go to api gateway and click on create api to create the rest apis

API Gateway > APIs > Create API

## Choose an API type

### HTTP API

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:  
Lambda, HTTP backends

### WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:  
Lambda, HTTP, AWS Services

### REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:  
Lambda, HTTP, AWS Services

Step 7: Choose the rest api and click on build

API Gateway > APIs > Create API > Create REST API

## Create REST API

### API details

☒ **New API**  
Create a new REST API.

☐ **Clone existing API**  
Create a copy of an API in this AWS account.

☐ **Import API**  
Import an API from an OpenAPI definition.

☐ **Example API**  
Learn about API Gateway with an example API.

API name

todo-api

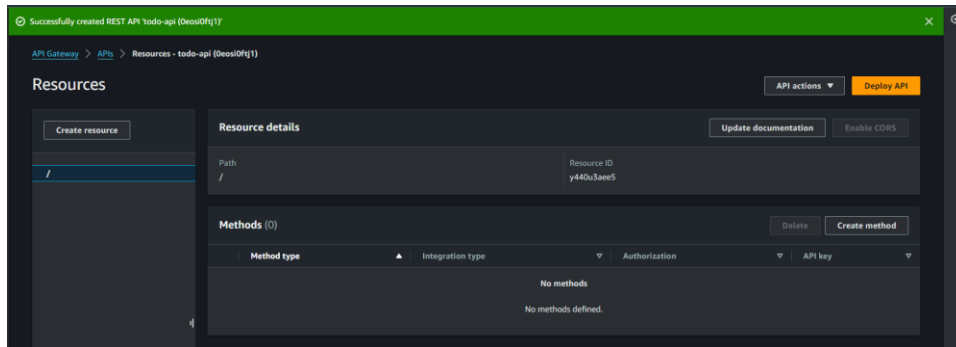
Description - optional

API endpoint type

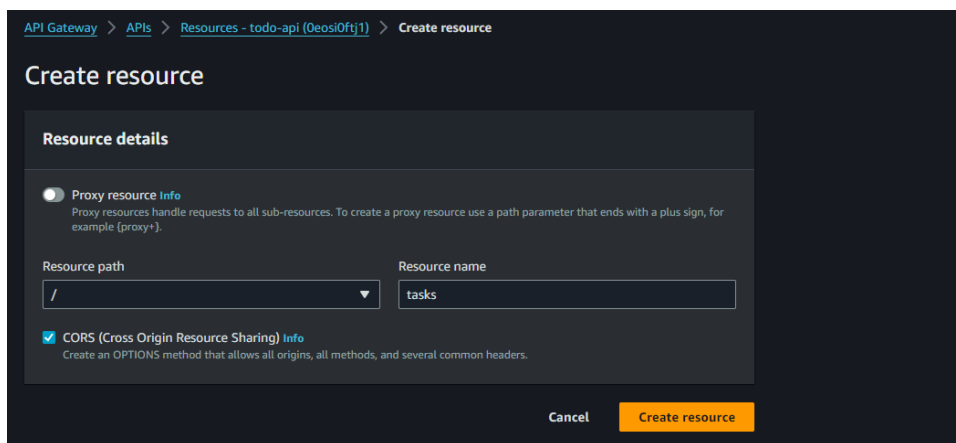
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.

Regional

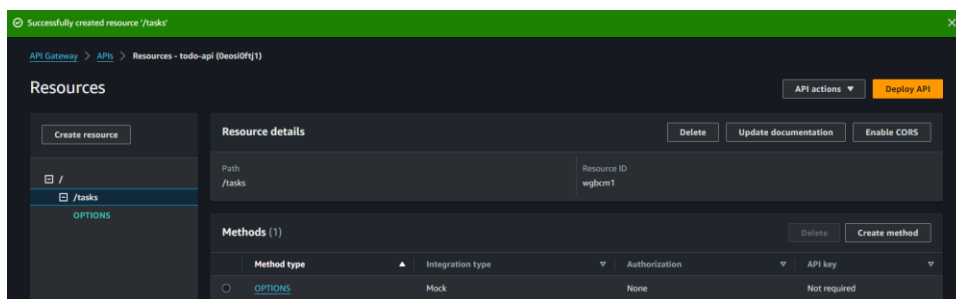
Step 8: Add the rest api name and click on create api



Step 9: Successfully created the api gateway



Step 10: Add the resource tasks and enable the cors and click on the create resource



Step 11: Resources successfully created and clicked on the create method to create different methods

## Create method

### Method details

#### Method type

GET

#### Integration type



##### Lambda function

Integrate your API with a Lambda function.



##### HTTP

Integrate with an existing HTTP endpoint.



##### Mock

Generate a response based on API Gateway mappings and transformations.



##### AWS service

Integrate with an AWS Service.



##### VPC link

Integrate with a resource that isn't accessible over the public internet.



##### Lambda proxy integration

Send the request to your Lambda function as a structured event.

#### Lambda function

Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1



arn:aws:lambda:us-east-1:471112841584:function:todo



Grant API Gateway permission to invoke your Lambda function. To turn off, update the function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.



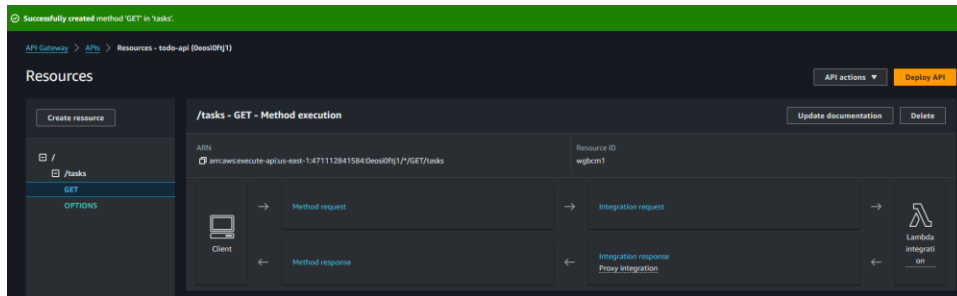
##### Default timeout

The default timeout is 29 seconds.

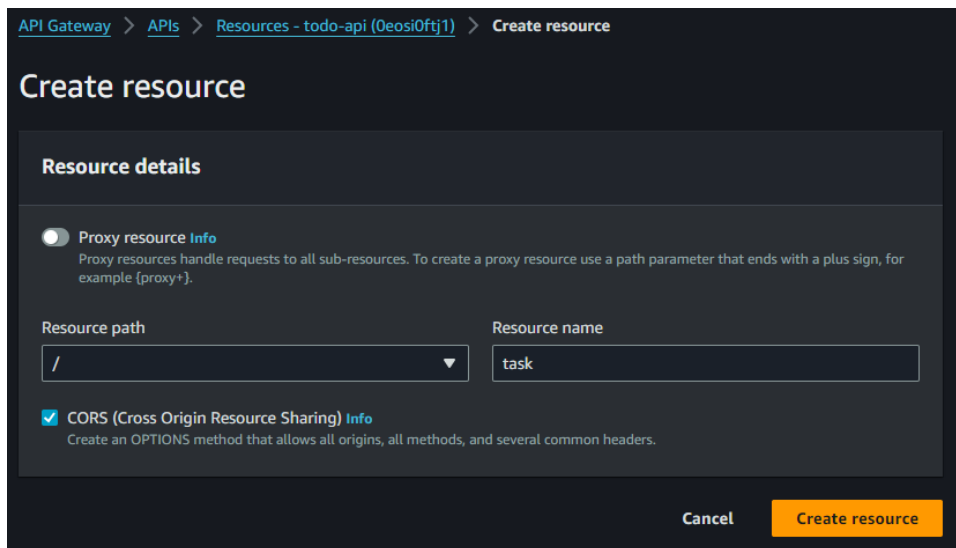
Cancel

Create method

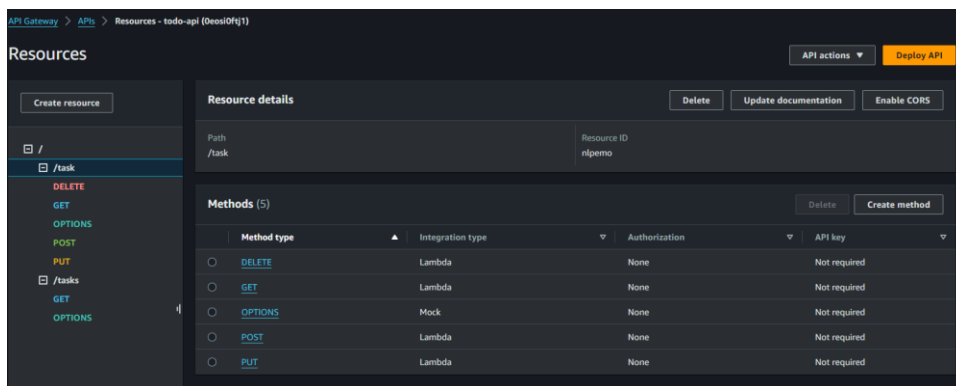
Step 12: Creating the get method enabling the lambda proxy integration and choosing the lambda function and click on the create method



Step 13: Get method created successfully.



Step 14: Similarly, add the task resource



Step 15: Added all methods for the task resource and click on the **Deploy Api** to deploy all rest apis

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Stage

\*New stage\*

Stage name

v1

A new stage will be created with the default settings. Edit your stage settings on the **Stage** page.

Deployment description

Cancel

Deploy

Step 16: Add the stage name and click on deploy

Successfully created deployment for todo-api. This deployment is active for v1.

API Gateway > APIs > todo-api [Dee0d0f1]

Stages

Stage actions Create stage

v1

Stage details info

Stage name

v1

Cache cluster info

Inactive

Default method-level caching

Inactive

Rate info

-

Burst info

-

Web ACL

-

Client certificate

-

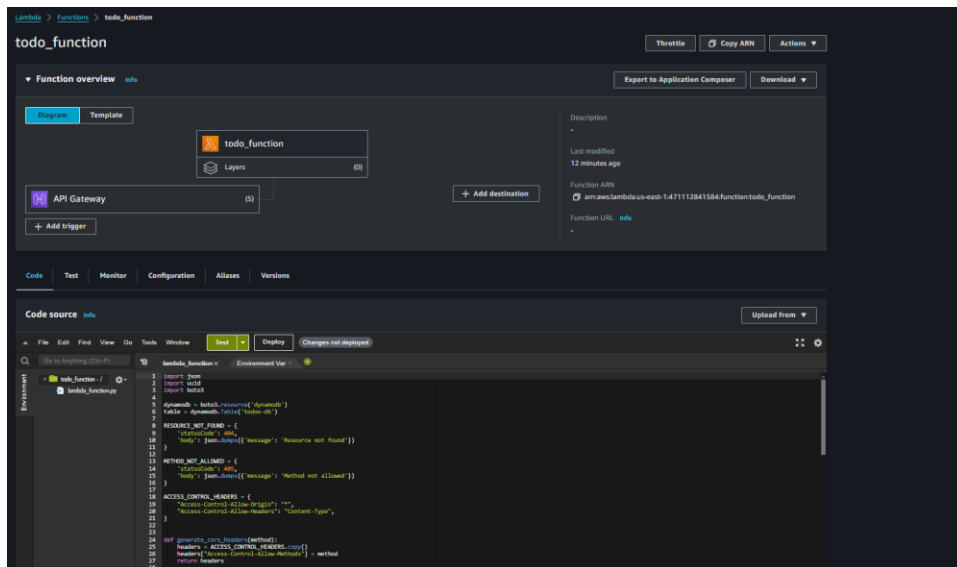
Copied

https://Dee0d0f1.execute-api.us-east-1.amazonaws.com/v1

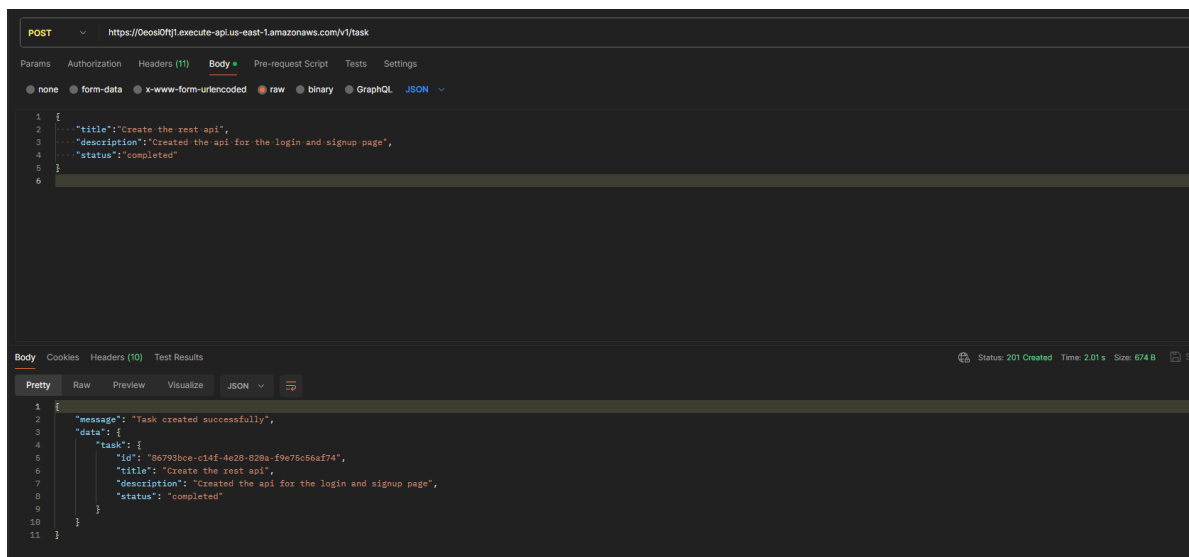
Active deployment

p3g519 on February 24, 2024, 18:55 (UTC+05:45)

Step 17: Api Deployed

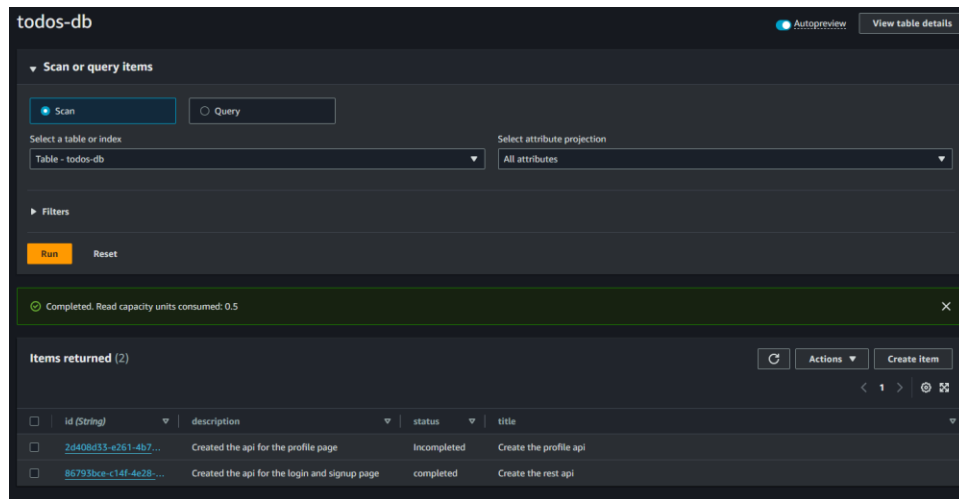


Step 18: Add the code of the rest apis in the lambda function and click on deploy

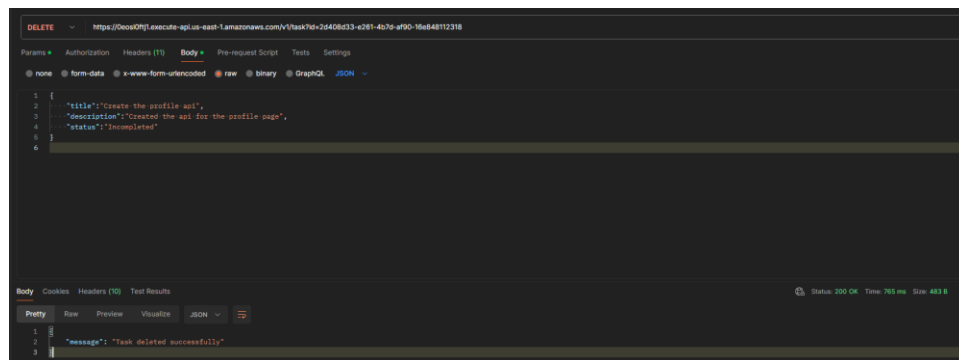


Post Request

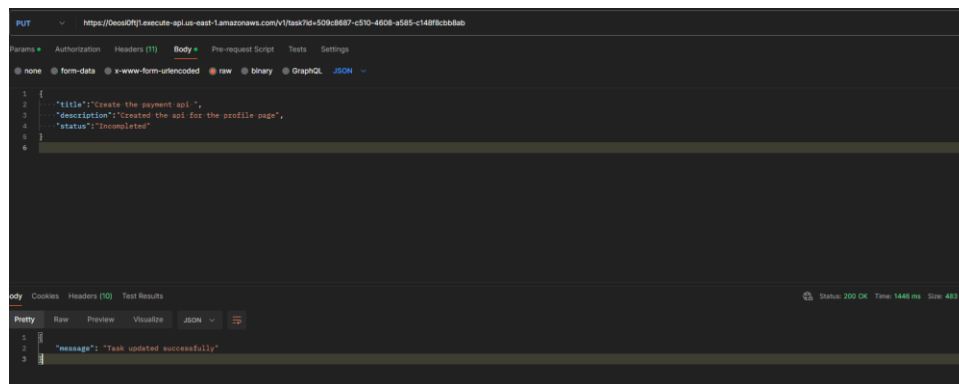




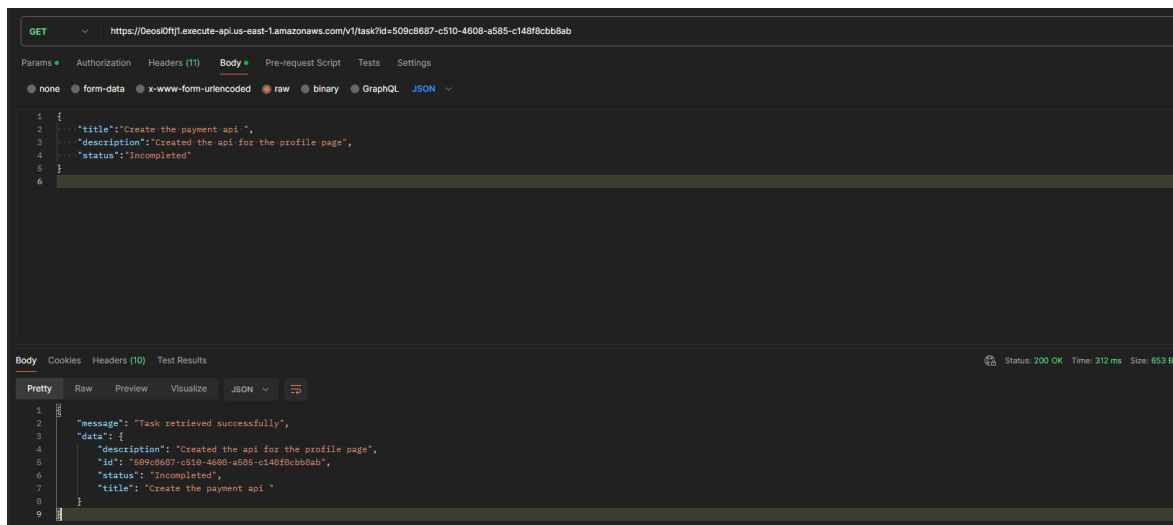
All items in the table



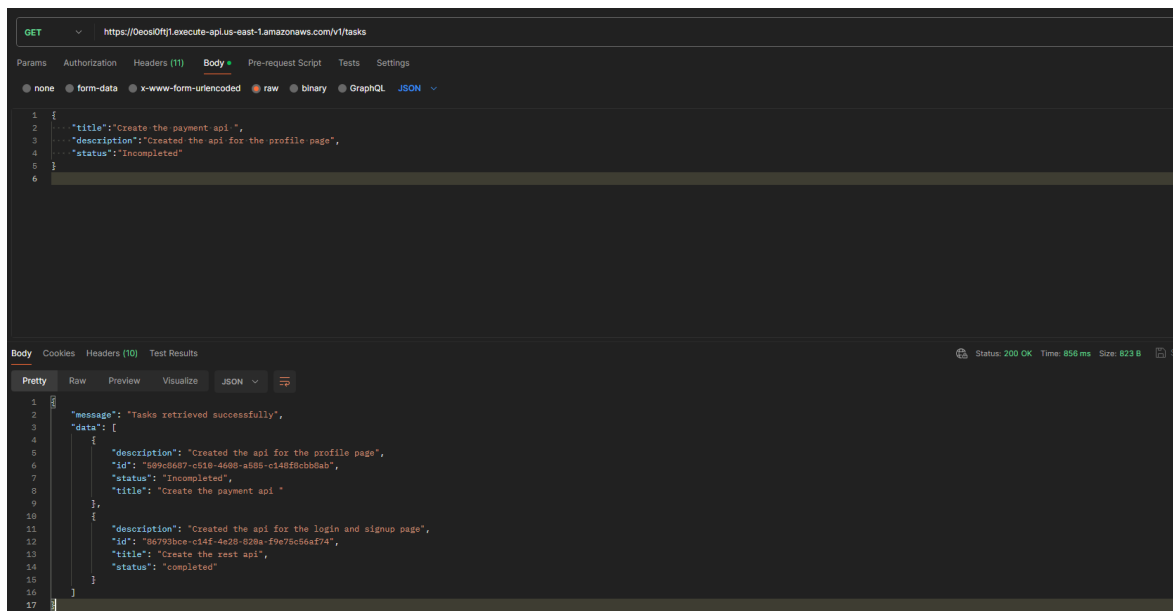
Delete method



Put method



## Get Method



## Step 19: Testing all the apis in the postman

# Building a Serverless Web Application

[Amazon S3](#) > [Buckets](#) > [Create bucket](#)

## Create bucket [Info](#)

Buckets are containers for data stored in S3.

### General configuration

AWS Region

US East (N. Virginia) us-east-1 ▼

Bucket type [Info](#)

☒ **General purpose**  
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

☐ **Directory - New**  
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name [Info](#)

frontend-website-for-todo-app

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#) ↗

Copy settings from existing bucket - *optional*  
Only the bucket settings in the following configuration are copied.

**Choose bucket**

Format: s3://bucket/prefix

### Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☐ **ACLs disabled (recommended)**  
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

☒ **ACLs enabled**  
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Step 1: add the bucket name and enable the ACLs


### Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block *all* public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☐ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**  
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ **Block public access to buckets and objects granted through *new* public bucket or access point policies**  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.



**Turning off block all public access might result in this bucket and the objects within becoming public**  
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Step 2: Enable the public access and click on create bucket

Successfully created bucket "frontend-website-for-todo-app".  
To upload files and folders, or to configure additional bucket settings, choose [View details](#).

Amazon S3 > Buckets

**Account snapshot** [View Storage Lens dashboard](#)  
Last updated: Feb 23, 2024 by Storage Lens. Metrics are generated every 24 hours. Metrics don't include directory buckets. [Learn more](#)

Metric	Value
Total storage	12.8 KB
Object count	12
Average object size	1.1 KB

You can enable advanced metrics in the "default-account-dashboard" configuration.

**General purpose buckets** | Directory buckets

**General purpose buckets (1)** [info](#)

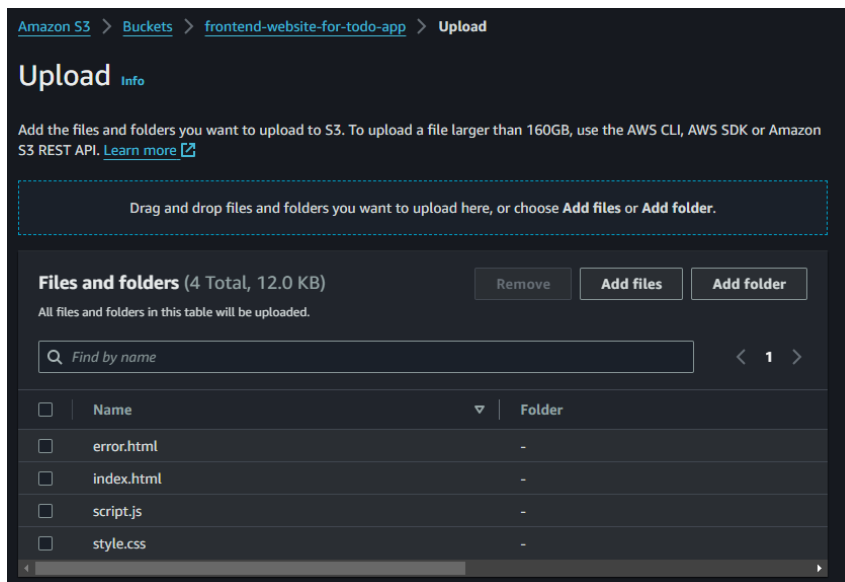
Buckets are containers for data stored in S3.

Find buckets by name

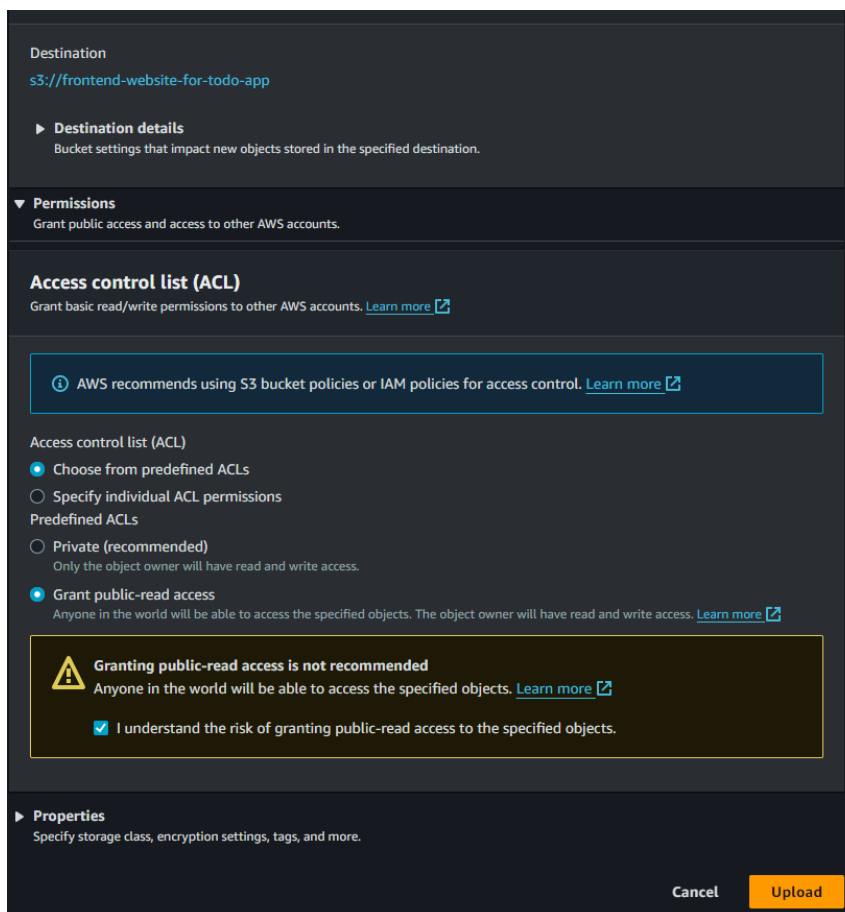
Buttons: Copy ARN, Empty, Delete, **Create bucket**

Name	AWS Region	Access	Creation date
frontend-website-for-todo-app	US East (N. Virginia) us-east-1	Objects can be public	February 24, 2024, 20:55:15 (UTC-05:45)

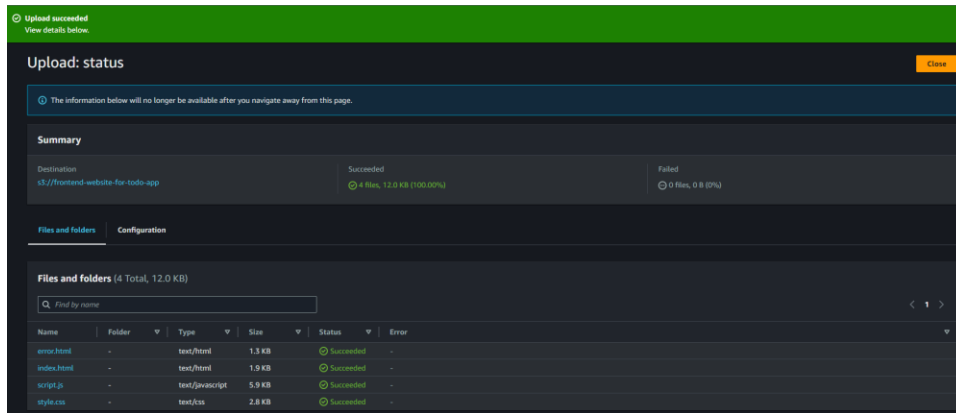
Step 3: Bucket created successfully



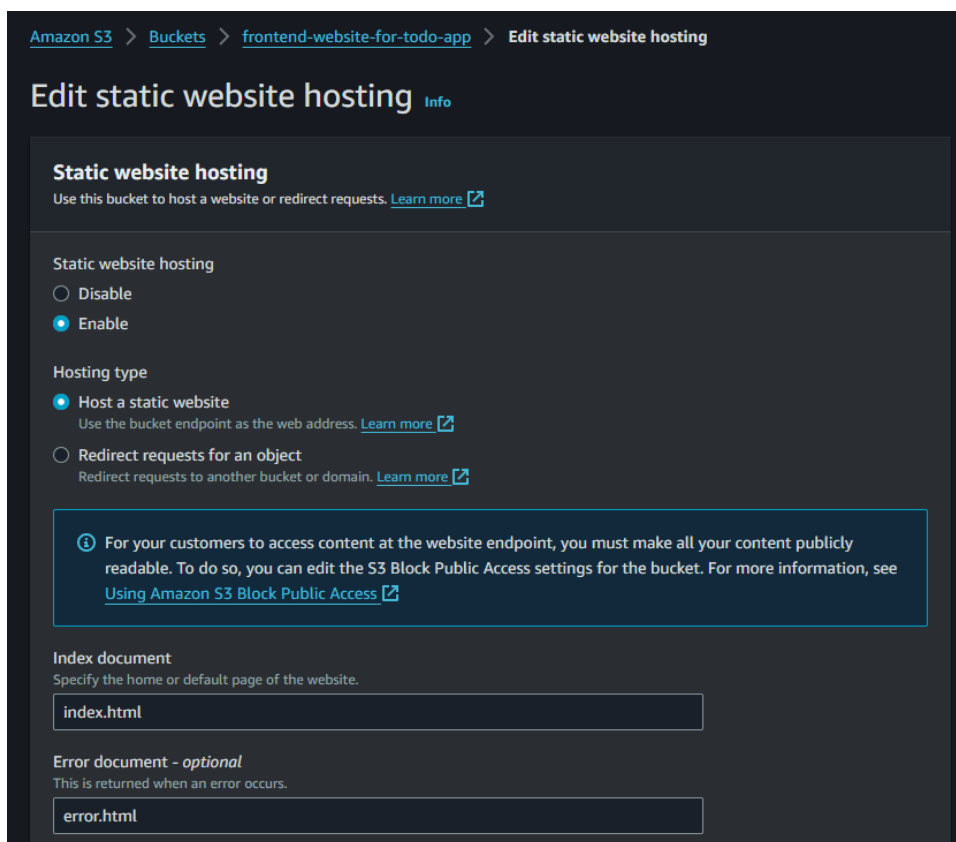
## Step 4: Adding the frontend files



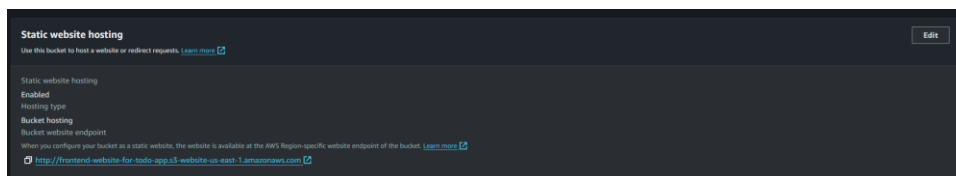
## Step 5: Granting the public access and clicking on the upload button to upload the files



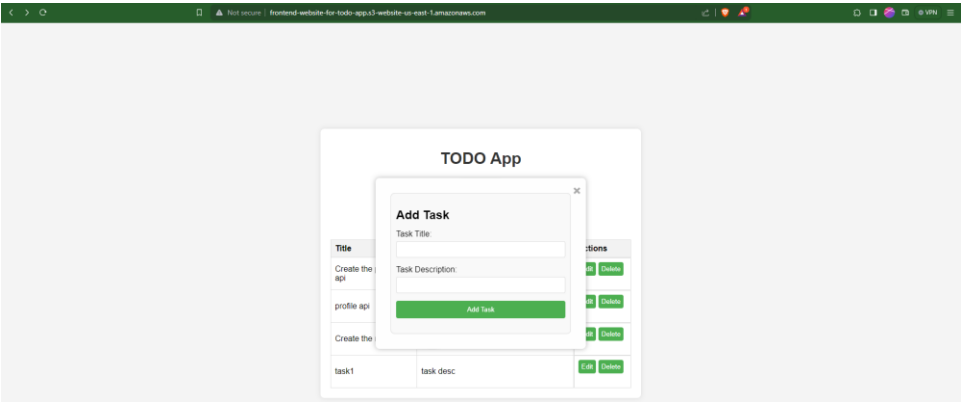
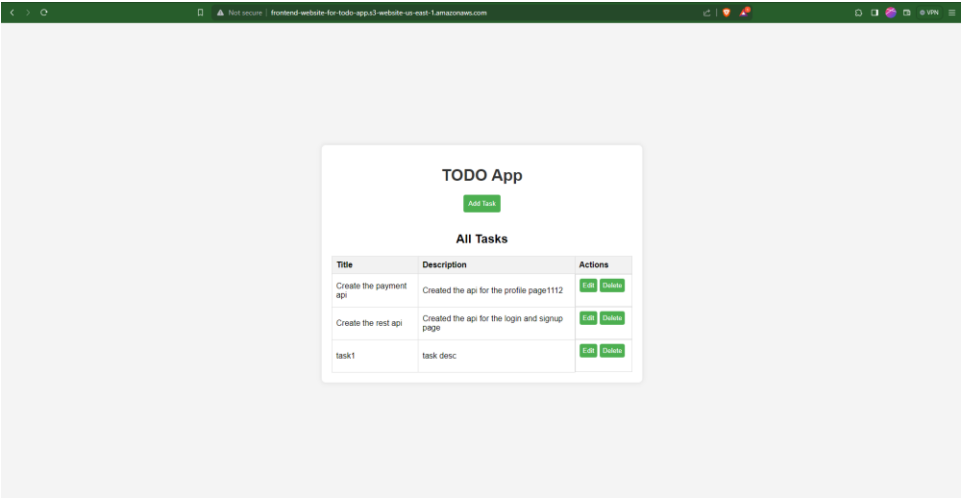
Successfully uploaded

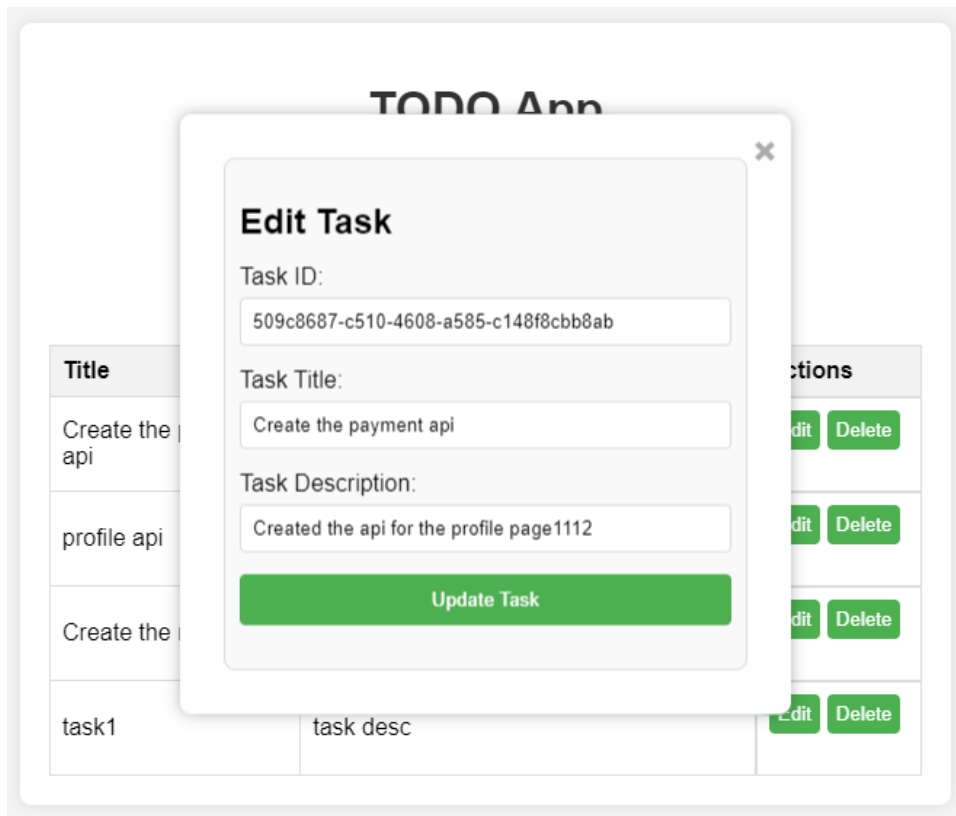


Step 6: Make the static site by enabling the hosting option



Site available for public access





Above all ui are static hosted website images

## Serverless Data Processing Pipeline

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.  
serverless\_pipeline\_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.  
Python 3.12

**Architecture** info  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**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

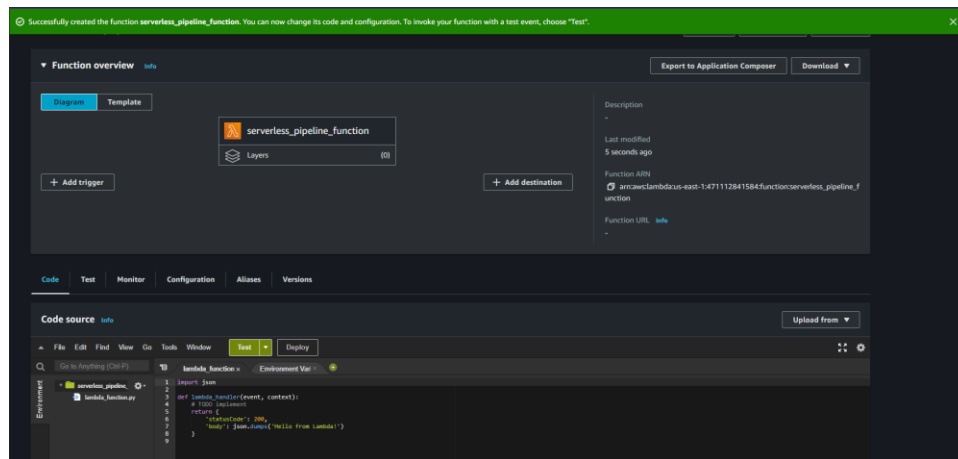
**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.  
LabRole

**Advanced settings**

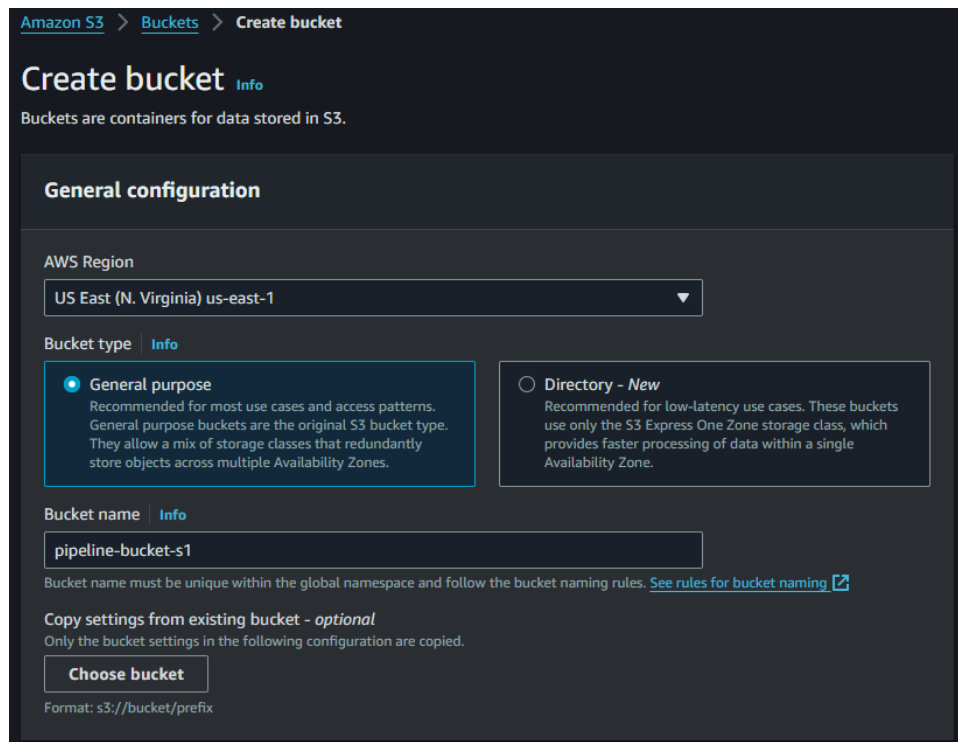
Cancel Create Function

Step 1: Create the lambda function

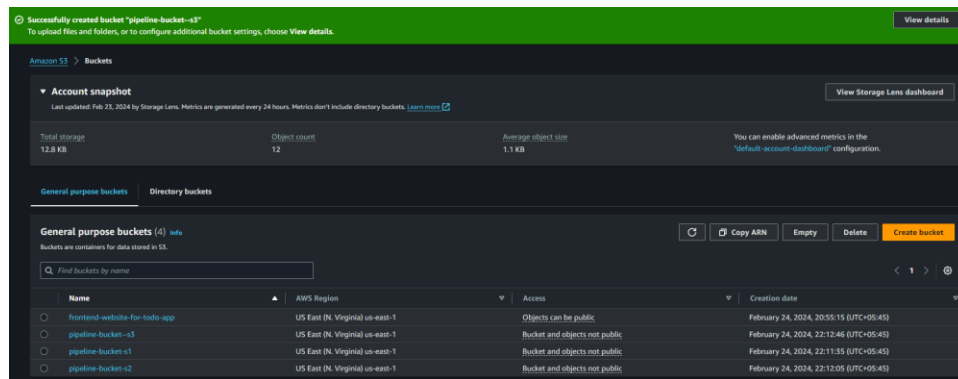




Lambda function created



Step 2: Create the first s3 bucket



### Step 3: Added two more buckets

## Add trigger

### Trigger configuration [Info](#)

S3  
aws asynchronous storage

**Bucket**  
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.

[×](#) [↺](#)

Bucket region: us-east-1

**Event types**  
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

[PUT](#) [×](#) [POST](#) [×](#) [COPY](#) [×](#)

**Prefix - optional**  
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.

**Suffix - optional**  
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.

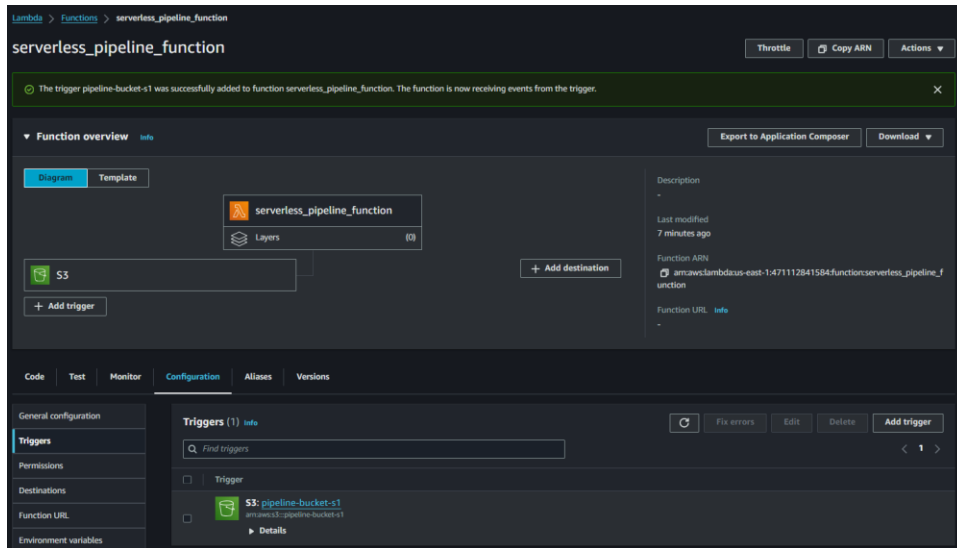
**Recursive Invocation**  
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)

☒ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

[Cancel](#) [Add](#)

### Step 4: Add one bucket as trigger for the lambda function



Trigger added successfully



Step 5: Added lambda function code and deployed

Amazon S3 > Buckets > pipeline-bucket-s1 > Upload

## Upload [Info](#)

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. [Learn more](#)

Drag and drop files and folders you want to upload here, or choose [Add files](#) or [Add folder](#).

**Files and folders** (1 Total, 35.0 B) [Remove](#) [Add files](#) [Add folder](#)

All files and folders in this table will be uploaded.

< 1 >

<input type="checkbox"/>	Name	Folder
<input type="checkbox"/>	text_file.txt	-

**Destination** [Info](#)

Destination  
s3://pipeline-bucket-s1

► **Destination details**  
Bucket settings that impact new objects stored in the specified destination.

► **Permissions**  
Grant public access and access to other AWS accounts.

► **Properties**  
Specify storage class, encryption settings, tags, and more.

[Cancel](#) [Upload](#)

Step 6: Added the text file to first s3 bucket that triggers the lambda function

Upload succeeded  
View details below.

### Upload: status [Close](#)

The information below will no longer be available after you navigate away from this page.

**Summary**

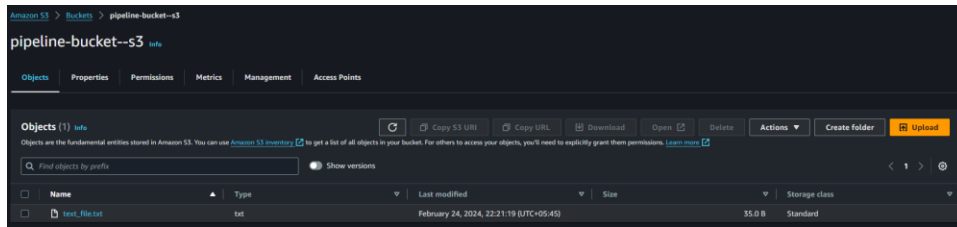
Destination s3://pipeline-bucket-s1	Succeeded 1 file, 35.0 B (100.00%)	Failed 0 files, 0 B (0%)
--	---------------------------------------	-----------------------------

**Files and folders** (1 Total, 35.0 B)

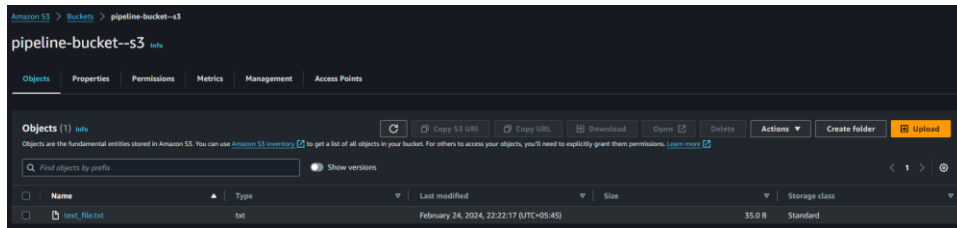
< 1 >

Name	Folder	Type	Size	Status	Error
text_file.txt	-	text/plain	35.0 B	Succeeded	-

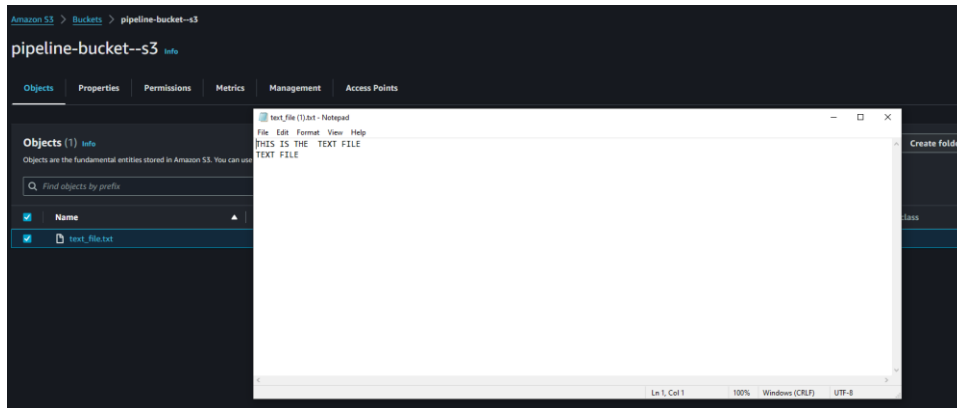
File uploaded successfully



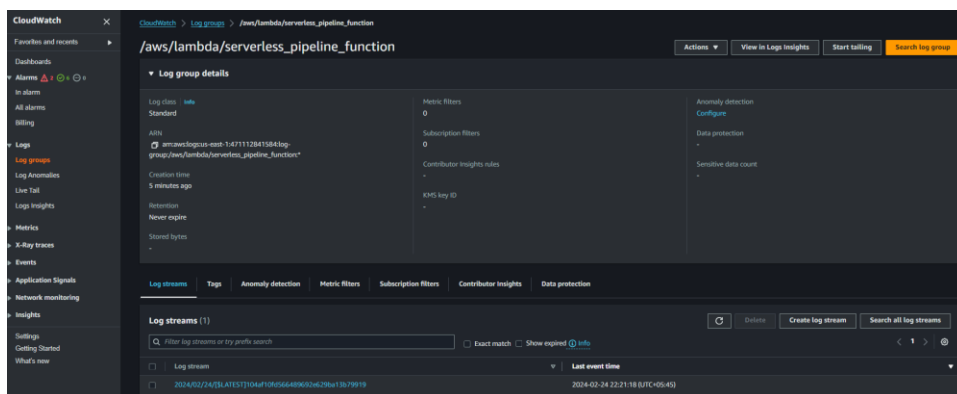
Step 7: Check same is replicated in second bucket



Step 8: Verify same file from the bucket one is replicated in bucket 3 or not



Step 9: Same file is converted into upper case after replication



Step 10: Go to the cloud watch and check the logs of the lambda function

