

1. Building a Serverless Web Application

Objective: Create a serverless web application using AWS Lambda, API Gateway, S3, and DynamoDB.

Approach:

Set Up Backend: Create Lambda functions to handle backend logic. These functions will interact with a DynamoDB table for data storage.

API Gateway: Set up API Gateway to create RESTful endpoints that trigger the Lambda functions.

Frontend Hosting: Host a static website on S3 that interacts with the backend via API Gateway.

Integration: Ensure that the frontend can successfully send requests to the backend and display responses.

Goal: Understand the basics of building and connecting serverless backend services with a static frontend, enabling a fully serverless web application.\

1.1. Create a table in DynamoDb

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The path 'DynamoDB > Tables > Create table' is visible at the top. The main title is 'Create table'. Below it, the 'Table details' section is active, indicated by a blue border. It contains fields for 'Table name' (set to 'enquiry_form') and 'Partition key' (set to 'date'). There is also a 'Sort key - optional' field (set to 'name'). A note states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the 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.

<input type="text" value="date"/>	<input type="button" value="String"/> ▾
-----------------------------------	---

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.

<input type="text" value="name"/>	<input type="button" value="String"/> ▾
-----------------------------------	---

1 to 255 characters and case sensitive.

1.2. SNS Topic

Amazon SNS > Topics > Create topic

Create topic

Details

Type [Info](#)
Topic type cannot be modified after topic is created

FIFO (first-in, first-out)
• Strictly-preserved message ordering
• Exactly-once message delivery
• High throughput, up to 300 publishes/second
• Subscription protocols: SQS

Standard
• Best-effort message ordering
• At-least once message delivery
• Highest throughput in publishes/second
• Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Name

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - optional [Info](#)
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

Maximum 100 characters.

Amazon SNS > Topics > serverless-sns-topic

serverless-sns-topic

[Edit](#) [Delete](#) [Publish message](#)

Details

Name serverless-sns-topic	Display name -
ARN arn:aws:sns:us-east-1:612362567483:serverless-sns-topic	Topic owner 612362567483
Type Standard	

[Subscriptions](#) [Access policy](#) [Data protection policy](#) [Delivery policy \(HTTP/S\)](#) [Delivery status logging](#) [Encryption](#) [Tags](#) [Integrations](#)

Subscriptions (0) [Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

1.3. Create lambda

Lambda function is created using existing user role.

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.
 ▼ ⟳

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

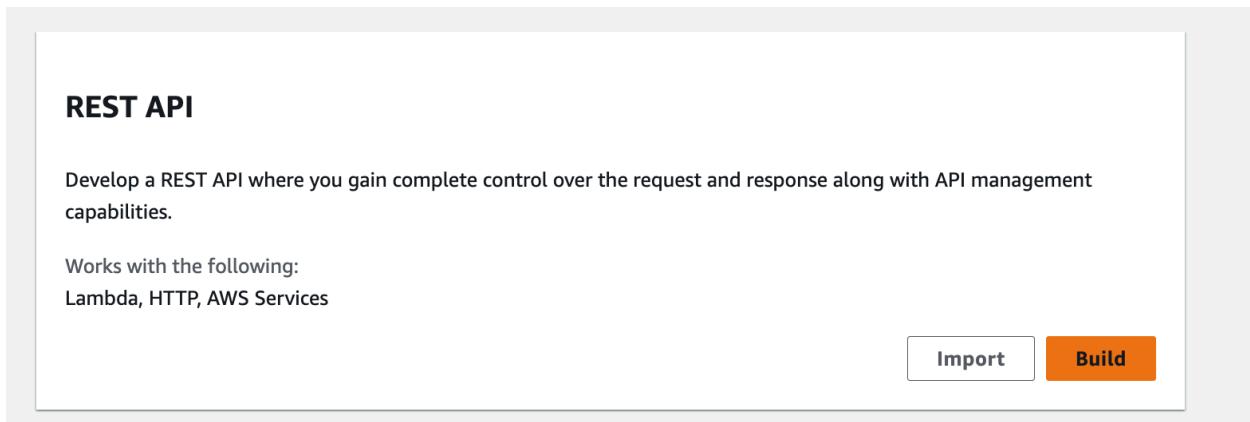
Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
 ▼ ⟳
[View the LabRole role](#) on the IAM console.

After that, post request implementation is done

```
1 import json
2 import boto3
3 import datetime
4
5 dynamodb = boto3.resource('dynamodb')
6 table = dynamodb.Table('enquiry_form')
7
8 def sns_notification(sns_arn, sns_message):
9     client = boto3.client('sns')
10    response = client.publish(TopicArn=sns_arn, Message=sns_message)
11    print('response:', response)
12    return response
13
14
15 def lambda_handler(event, context):
16     try:
17         print('Received event:', json.dumps(event))
18         current_date = datetime.datetime.now().strftime("%Y-%m-%d")
19         name = event['name']
20         email = event['email']
21         message = event['message']
22         item ={
23             'date':current_date,
24             'name':name,
25             'email':email,
26             'message':message
27         }
28
29         table.put_item(Item=item)
30         response = {
31             "statusCode": 200,
32             "data": json.dumps({"message": "Data is added successfully"})
33         }
34         #sns implementation
35         # arn = "arn:aws:sns:us-east-1:612362567483:serverless-sns-topic"
36         # message=f"New data stored"
37         # resource = sns_notification(arn,message)
38     except Exception as e:
39         print("Error:", e)
40         response = {
41             "statusCode": 500,
42             "data": json.dumps({"message": "Internal Server Error"})
43         }
44     return response
45
```

1.4. Create API Gateway

Required REST API configuration is added



The screenshot shows the 'Create REST API' wizard. It starts with a breadcrumb navigation: API Gateway > APIs > Create API > Create REST API. The main section is titled 'API details'.

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
enquiry-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

Create API

Steps implemented in API gateways are mentioned below:

i. Creating Resource

The screenshot shows the 'Create resource' dialog in the AWS API Gateway console. The path in the top navigation bar is: API Gateway > APIs > Resources - enquiry-api (spz2ocun7c) > Create resource. The main title is 'Create resource'. Below it is a section titled 'Resource details'. It contains two input fields: 'Resource path' with the value '/' and 'Resource name' with the value 'enquiry'. There are also two configuration sections: 'Proxy resource' (radio button selected) with a note about proxy resources handling requests to all sub-resources, and 'CORS (Cross Origin Resource Sharing)' (checkbox checked) with a note about creating an OPTIONS method. At the bottom right are 'Cancel' and 'Create resource' buttons.

- ii. Create a POST Method and add recently created lambda function in the configuration
iii. Deploy the API. For this purpose a new stage is created and then deployed.

The screenshot shows the 'Deploy API' dialog. It asks to choose a stage for deployment, with 'New stage' selected. The 'Stage name' field is set to 'enquiry-stage'. A note indicates that a new stage will be created with default settings. Below this is a 'Deployment description' field, which is currently empty. At the bottom are 'Cancel' and 'Deploy' buttons. Other sections visible include 'Request validator' (None) and 'Request paths (0)'.

iv. Invoke URL of the post method is copied from the created stage

The screenshot shows the AWS API Gateway Stages page. In the left sidebar, under the 'enquiry-stage' section, there is a 'POST' method listed under the '/enquiry' resource. A tooltip 'Copied' is shown over the URL link. The main content area is titled 'Method overrides' with the sub-instruction: 'This method inherits its settings from the 'enquiry-stage' stage.' Below this, the copied URL is displayed: <https://spz2ocun7c.execute-api.us-east-1.amazonaws.com/enquiry-stage/enquiry>.

v. Now in lambda function, we can observe that API Gateway is added.

The screenshot shows the AWS Lambda Function overview page for the 'serverless_api' function. Under the 'Triggers' section, there is a box labeled 'API Gateway' which is currently selected. Below this box is a button '+ Add trigger'. On the right side of the page, there is a detailed sidebar with the following information:

- Description: -
- Last modified: 8 minutes ago
- Function ARN: arn:aws:lambda:us-east-1:612362567483:function:serverless_api
- Function URL: Info

vi. The API CORS is enabled.

API Gateway > APIs > Resources - enquiry-api (spz2ocun7c) > Enable CORS

Enable CORS

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API.

Gateway responses
API Gateway will configure CORS for the selected gateway responses.

Default 4XX
 Default 5XX

Access-Control-Allow-Methods

OPTIONS
 POST

Access-Control-Allow-Headers
API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

Access-Control-Allow-Origin
Enter an origin that can access the resource. Use a wildcard '*' to allow any origin to access the resource.

*

▶ Additional settings

Cancel **Save**

1.5. S3 bucket creation and uploading the html file

Adding all the required configuration is S3 bucket. “ACLs enabled” is chosen in object ownership so that bucket is publicly available with the url.

Amazon S3 > Buckets > Create bucket

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

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](#)

enquiry-bucket

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 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.

Block all public access is removed for getting access through website.

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.

After creation of S3 bucket, html file containing enquiry form is uploaded.

Amazon S3 > Buckets > [enquiry-bucket](#) > 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, 1.6 KB)		Remove	Add files	Add folder
All files and folders in this table will be uploaded.				
<input type="text"/> Find by name				
<input type="checkbox"/>	Name	▼	Folder	< 1 >
<input type="checkbox"/>	index.html	-		

Upload: status

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

Summary		Failed
Destination	Succeeded	Failed
s3://enquiry-bucket	1 file, 1.6 KB (100.00%)	0 files, 0 B (0%)

Files and folders (1 Total, 1.6 KB)

Name	Folder	Type	Size	Status	Error
index.html	-	text/html	1.6 KB	Succeeded	-

When the url of uploaded file is browsed in web, access denied message is shown.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>NF2G8MX26025BMA7</RequestId>
  <HostId>w3CNLIVDsvcGJJiB7G+E5+A25Ki4TILsBqqAFSRgzy4cibLjseBbEiWY7ZdpLonVEmEt/5r4X/CzJE/g3Ja4g==</HostId>
</Error>
```

So, now the website is hosted from properties tab of created S3 bucket. Initially the static website hosting is disabled and for hosting it's status is updated to "Enable".

Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

Static website hosting
Disabled

[Amazon S3](#) > [Buckets](#) > [enquiry-bucket](#) > Edit static website hosting

Edit static website hosting [Info](#)

Static website hosting
Use this bucket to host a website or redirect requests. [Learn more](#)

Static website hosting
 Disable Enable

Hosting type
 Host a static website Use the bucket endpoint as the web address. [Learn more](#)
 Redirect requests for an object Redirect requests to another bucket or domain. [Learn more](#)

For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see Using Amazon S3 Block Public Access

Index document
Specify the home or default page of the website.
index.html

After that, the uploaded file should give permission to make public using ACL.

The screenshot shows the AWS S3 console with the bucket 'enquiry-bucket'. The 'Objects' tab is selected, displaying one object: 'index.html'. The object details show it is an 'html' file last modified on February 20, 2024, at 23:51:11 (UTC+05:45). A context menu is open on the right, with the 'Actions' section expanded. The 'Make public using ACL' option is highlighted with a blue border.

Now, the hosted website can be accessed.

The screenshot shows a web browser window with the URL 'enquiry-bucket.s3-website-us-east-1.amazonaws.com'. The page title is 'Enquiry Form'. The form contains three fields: 'Name:' with an input field, 'Email:' with an input field, and 'Message:' with a text area. A 'Submit' button is at the bottom.

Name:

Email:

Message:

Submit

The testing of implemented task is checked. Below we can observe the user input and the response of the integrated API. I have checked by inspecting the request in browser.



Enquiry Form

Name:

Utsha Shrestha

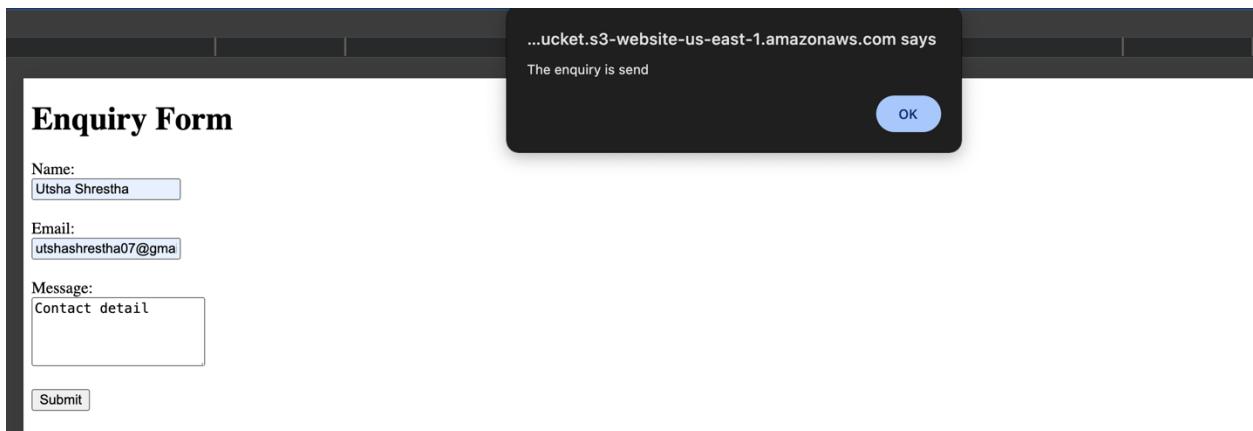
Email:

utshashrestha07@gma

Message:

Contact detail

Submit



The screenshot shows the Network tab in the Chrome DevTools. A single request named "enquiry" is listed. The response tab is selected, showing the JSON response:

```
{statusCode: 200, data: {"message": "Data is added successfully"}  
data: "{\"message\": \"Data is added successfully\"}"  
statusCode: 200}
```

The screenshot shows the Network tab in the Chrome DevTools. The request payload for the "enquiry" call is displayed:

```
▼ Request Payload  
▼ {name: "Utsha Shrestha", email: "utshashrestha07@gmail.com", message: "Contact detail"}  
email: "utshashrestha07@gmail.com"  
message: "Contact detail"  
name: "Utsha Shrestha"
```

The items in DynamoDB is inserted too.

The screenshot shows the AWS DynamoDB console under the "Explore items" section for the "enquiry_form" table. The table has two items:

- enquiry_form (selected)
- user-table

The "Scan or query items" section is active, showing the "Scan" option selected. The results table shows one item returned:

	date (String)	name (String)	email	message
2024-02-20	Utsha Shrestha	utshashrest...	Contact detail	

A green notification at the bottom indicates: "Completed. Read capacity units consumed: 0.5".

The logs of implemented task can be observed from cloud watch logs of lambda function.

CloudWatch > Log groups > /aws/lambda/serverless_api > 2024/02/20/[\${LATEST}]45eeefcf9a5b47f29c26b04ed7d576ae

Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Timestamp	Message
No older events at this moment. Retry	
2024-02-21T01:50:44.018+05:45	INIT_START Runtime Version: python:3.12.v18 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:776a3759221679a634181f858871d5514dc74a176f78bc535f822a932845ae5a
2024-02-21T01:50:44.523+05:45	START RequestId: 013a8f0d-36da-4407-9929-2ad146d228ff Version: \${LATEST}
2024-02-21T01:50:44.524+05:45	Received event: {"name": "Utsha Shrestha", "email": "utshashrestha07@gmail.com", "message": "Contact detail"} itme {'date': '2024-02-20', 'name': 'Utsha Shrestha', 'email': 'utshashrestha07@gmail.com', 'message': 'Contact detail'}
2024-02-21T01:50:45.118+05:45	END RequestId: 013a8f0d-36da-4407-9929-2ad146d228ff
2024-02-21T01:50:45.118+05:45	REPORT RequestId: 013a8f0d-36da-4407-9929-2ad146d228ff Duration: 595.42 ms Billed Duration: 596 ms Memory Size: 128 MB Max Memory Used: 76 MB Init Duration: 512.31 ms
No newer events at this moment. Auto retry paused . Resume	

2. Creating a Serverless API

****Objective**:** Develop a serverless API using AWS Lambda and API Gateway.

****Approach**:**

- ****Define API**:** Design a simple RESTful API (e.g., for a todo list application).
- ****Lambda Functions**:** Create Lambda functions for each API method (GET, POST, PUT, DELETE).
- ****API Gateway Setup**:** Use API Gateway to set up the API endpoints, connecting each endpoint to the corresponding Lambda function.
- ****Testing**:** Test the API using tools like Postman or AWS API Gateway test functionality.

****Goal**:** Gain hands-on experience in building and deploying a serverless API, understanding the integration between Lambda and API Gateway.

2.1. Create lambda function

Lambda functions with required configuration is created

Create function [Info](#)

Choose one of the following options to create your function.

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.

Basic information

Function name [Info](#)
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.
 [▼](#) [C](#)

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.

[▼](#) [C](#)

[View the LabRole role](#) on the IAM console.

2.2. Create a table in DynamoDB

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.

2.3. API Gateway configuration

Create a rest api

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

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

Cancel **Create API**

Creating a resource for the API

API Gateway > APIs > Resources - todo-api (98e7x5s42f) > Create resource

Create resource

Resource details

Proxy resource Info
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path

Resource name

CORS (Cross Origin Resource Sharing) Info
Create an OPTIONS method that allows all origins, all methods, and several common headers.

Cancel **Create resource**

Creating GET, POST, PUT, DELETE methods of API

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 ▾ X

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.

Create method

Create method

Method details

Method type

POST

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-1:612362567483: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

Create method

Method details

Method type

PUT

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-1:612362567483: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

List of created api methods are shown below

The screenshot shows the AWS API Gateway interface. On the left, there's a sidebar with a 'Create resource' button. The main area is titled 'Resources' and shows a tree structure with a root node '/'. Underneath it is a node '/todo'. To the right of the tree, there's a 'Resource details' section with a 'Path' field containing '/todo' and a 'Resource ID' field containing 'sp7h1t'. Below this is a table titled 'Methods (5)' listing the following information:

Method type	Integration type	Authorization	API key
DELETE	Lambda	None	Not required
GET	Lambda	None	Not required
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required
PUT	Lambda	None	Not required

At the top right of the main area, there are buttons for 'API actions' (with a dropdown arrow), 'Deploy API' (in orange), 'Delete', 'Update documentation', and 'Enable CORS'.

The API CORS is enabled.

The screenshot shows the 'Enable CORS' configuration dialog. At the top, there's a breadcrumb navigation: API Gateway > APIs > Resources - todo-api (98e7x5s42f) > Enable CORS. The main title is 'Enable CORS'.

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API.

Gateway responses
API Gateway will configure CORS for the selected gateway responses.

Default 4XX
 Default 5XX

Access-Control-Allow-Methods

DELETE
 GET
 OPTIONS
 POST
 PUT

Access-Control-Allow-Headers
API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

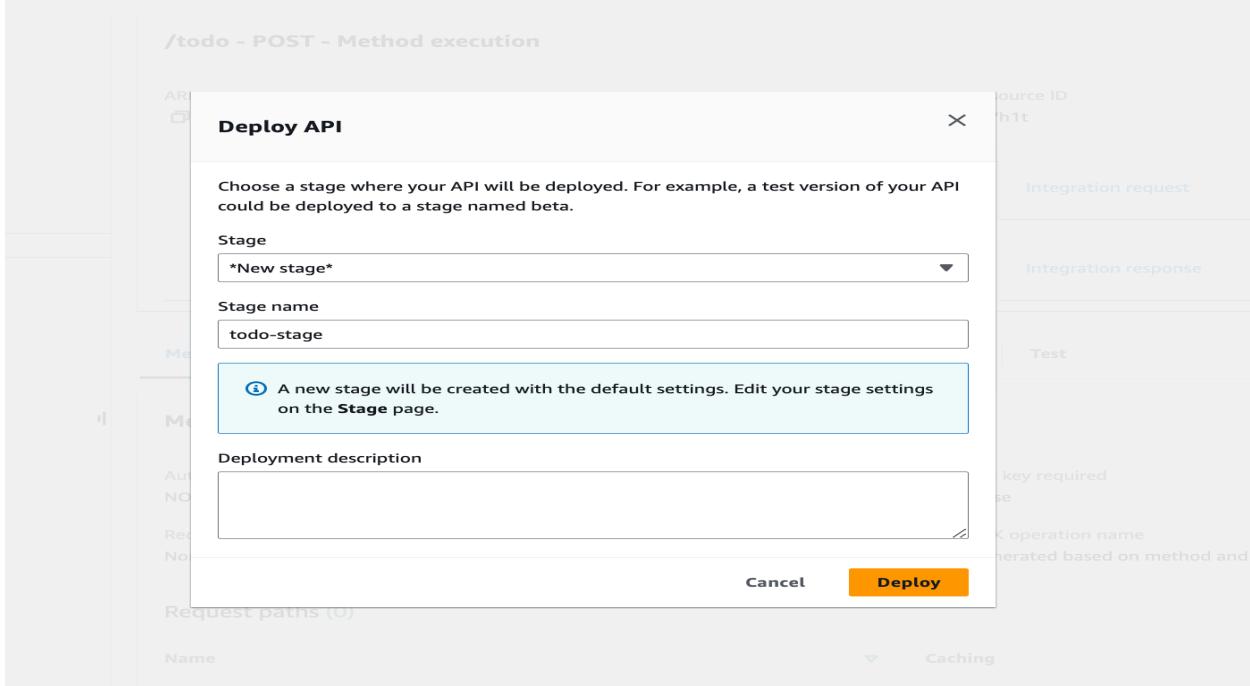
Access-Control-Allow-Origin
Enter an origin that can access the resource. Use a wildcard '*' to allow any origin to access the resource.

*

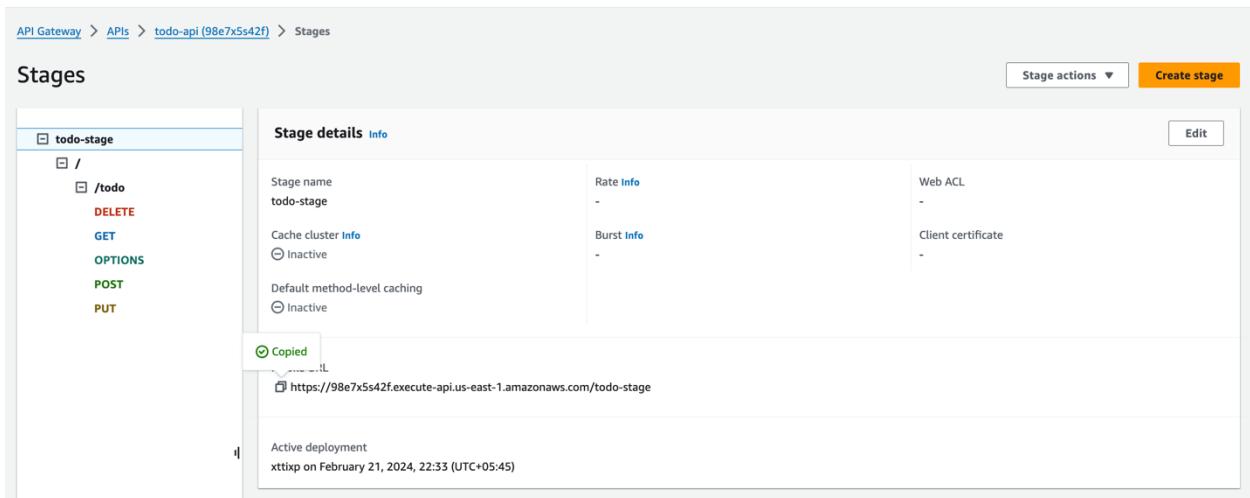
Additional settings

Cancel Save

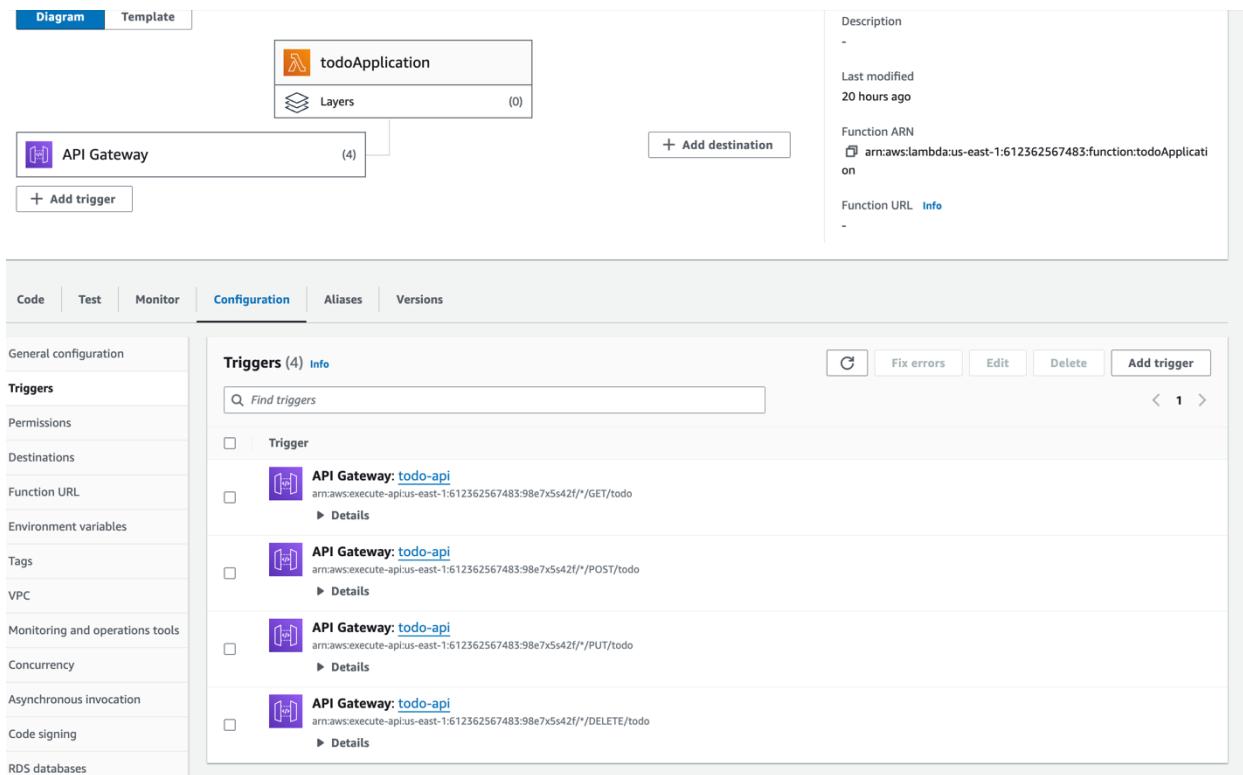
After completion of basic configurations for API gateway, Lambda functions, tables the API is deployed. During the process, new stage is created and deployed.



The invoke url is used to access the API



Now, getting back to created lambda function. In the configuration tab the list of triggers of created API Gateway can be viewed.



Functions for each request and its operation is created. After that source code is deployed and tested in Postman.

i. POST Request code snippet and postman testing

```

25
26 def post_request(event):
27     try:
28         item={
29             'id':event['id'],
30             'task':event['task'],
31             'status':event['status']
32         }
33         table.put_item(Item=item)
34         response= {
35             "statusCode":200,
36             "message":"todo added"
37         }
38     except ClientError as e:
39         response= {
40             "statusCode":500,
41             "message":"Something went wrong"
42             "data":str(e)
43         }
44     return response
45

```

POST https://98e7x5s42f.execute-api.us-east-1.amazonaws.com/todo-stage/todo

Body

```

1 {
2   "httpMethod": "POST",
3   "id": "2",
4   "task": "Serverless lab",
5   "status": "Pending"
6 }
```

Body

```

1 {
2   "statusCode": 200,
3   "message": "todo added"
4 }
```

POST https://98e7x5s42f.execute-api.us-east-1.amazonaws.com/todo-stage/todo

Body

```

1 {
2   "httpMethod": "POST",
3   "id": "1",
4   "task": "Basic lab",
5   "status": "Pending"
6 }
```

Body

```

1 {
2   "statusCode": 200,
3   "message": "todo added"
4 }
```

Items list in todo table

todo

Autopreview View table details

Scan or query items

Scan Query

Select a table or index Table - todo Select attribute projection All attributes

Filters

Run Reset

Completed. Read capacity units consumed: 0.5

Items returned (2)

	id (String)	status	task
<input type="checkbox"/>	2	Pending	Serverless lab
<input type="checkbox"/>	1	Pending	Basic lab

ii. Get request code snippet and postman testing

```
45 def get_request(event):
46     try:
47         data = table.scan()
48         items = data.get("Items", [])
49         print(items)
50         response= {
51             "statusCode":200,
52             "message":"Sucess",
53             "data":items
54         }
55     except ClientError as e:
56         response={
57             "statusCode":500,
58             "message":"Something went wrong",
59             "data":str(e)
60         }
61     return response
62
```

The screenshot shows a Postman interface with the following details:

- URL:** https://98e7x5s42f.execute-api.us-east-1.amazonaws.com/todo-stage/todo
- Method:** GET
- Response Headers:** 200 OK, 1240 ms, 532 B
- Body (Pretty):**

```
1 {
2     "statusCode": 200,
3     "message": "Sucess",
4     "data": [
5         {
6             "task": "Serverless lab",
7             "id": "2",
8             "status": "Pending"
9         },
10        {
11            "task": "Basic lab",
12            "id": "1",
13            "status": "Pending"
14        }
15    ]
16 }
```

iii. PUT request code snippet and postman testing

```
81 def put_request(event):
82     try:
83         id = event['id']
84         update_key = event['key']
85         update_value = event['new_value']
86
87         table.update_item(
88             Key={'id': id},
89             UpdateExpression=f'SET #attr = :value',
90             ExpressionAttributeNames={'#attr': update_key},
91             ExpressionAttributeValues={':value': update_value},
92             ReturnValues='UPDATED_NEW'
93         )
94
95         response = {
96             "statusCode": 200,
97             "message": "Task Updated"
98         }
99     except ClientError as e:
100        response = {
101            "statusCode": 500,
102            "message": "Something went wrong",
103            "data": str(e)
104        }
105
106 return response
```

The screenshot shows a POST request to the URL `https://98e7x5s42f.execute-api.us-east-1.amazonaws.com/todo-stage/todo`. The request body is a JSON object with the following structure:

```
1 {
2   ...
3   "httpMethod": "PUT",
4   ...
5   "id": "1",
6   ...
7   "key": "status",
8   ...
9   "new_value": "completed"
10 }
```

The response status is 200 OK, with a response time of 1589 ms and a body size of 412 B. The response body is:

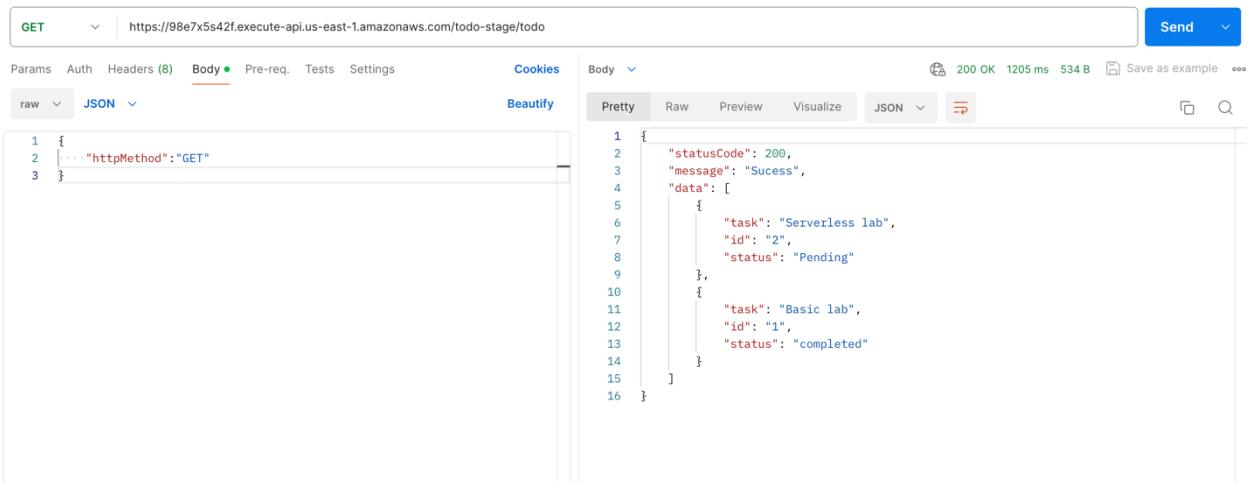
```
1 {
2   "statusCode": 200,
3   "message": "Task Updated"
4 }
```

Checking the item data after PUT request

The screenshot shows the AWS Lambda function's CloudWatch Logs. A green success message is displayed: `Completed. Read capacity units consumed: 0.5`.

The table below shows the items returned from the database:

Items returned (2)			
	id (String)	status	task
<input type="checkbox"/>	2	Pending	Serverless lab
<input type="checkbox"/>	1	completed	Basic lab



iv. Delete request code snippet and postman testing

```
63
64     def delete_request(event):
65         try:
66             id_value = event['id']    # Get id from event
67             if id_value is None:
68                 raise ValueError("ID is missing in the event")
69
70             response = table.delete_item(Key={'id': id_value})
71             return {
72                 "statusCode": 200,
73                 "message": "Data is deleted successfully",
74             }
75         except Exception as e:
76             return {
77                 "statusCode": 500,
78                 "message": "An error occurred while deleting data",
79                 "error": str(e)
80             }
81
```

DELETE https://98e7x5s42f.execute-api.us-east-1.amazonaws.com/todo-stage/todo

Body

```

1 {
2   ...
3   "httpMethod": "DELETE",
4   ...
5   "id": "2"
6 }

```

Body

```

1 {
2   "statusCode": 200,
3   "message": "Data is deleted successfully"
4 }

```

Checking the list of items after deleting in postman and items of table

GET https://98e7x5s42f.execute-api.us-east-1.amazonaws.com/todo-stage/todo

Body

```

1 {
2   ...
3   "httpMethod": "GET",
4   ...
5 }

```

Body

```

1 {
2   "statusCode": 200,
3   "message": "Success",
4   "data": [
5     {
6       "task": "Basic lab",
7       "id": "1",
8       "status": "completed"
9     }
10   ]
11 }

```

Items returned (1)				<input type="button" value="C"/>	<input type="button" value="Actions ▾"/>	<input type="button" value="Create item"/>
	<input type="checkbox"/> id (String)	▼	status	▼	task	▼
	<input type="checkbox"/> 1		completed		Basic lab	

3. Serverless Data Processing Pipeline

Objective: Build a serverless pipeline for processing data (e.g., log processing or ETL jobs).

Approach:

- **Data Ingestion:** Use AWS services like S3 or Kinesis to ingest data.
- **Processing:** Create Lambda functions to process the ingested data.
- **Storage:** Store the processed data in an appropriate AWS service, like S3 or DynamoDB.
- **Monitoring:** Set up CloudWatch to monitor the pipeline's performance and to log any issues.

Goal: Learn to build a serverless data processing pipeline, understanding the flow of data through various AWS services.

3.1. S3 bucket creation

Creating a two different S3 buckets with required configurations

The screenshot shows the AWS Create Bucket wizard. In the General configuration section, the AWS Region is set to 'US East (N. Virginia) us-east-1'. The Bucket type is set to 'General purpose', which is described as recommended for most use cases and access patterns. It also notes that general purpose buckets are the original S3 bucket type and allow a mix of storage classes. Another option, 'Directory - New', is available but not selected. The Bucket name is 'usthalab3bucket'. A note states that the bucket name must be unique within the global namespace and follow the bucket naming rules, with a link to 'See rules for bucket naming'. In the Object Ownership section, the 'ACLs disabled (recommended)' option is selected, indicating that all objects in the bucket are owned by the account and access is specified using only policies. The 'ACLs enabled' option is also shown but not selected.

Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3. [Learn more](#)

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

usthalab3bucket

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.

Object Ownership

Bucket owner enforced

Create bucket Info

Buckets are containers for data stored in S3. [Learn more](#)

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

utshalab3bucket2

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.

Object Ownership

Bucket owner enforced

3.2. Lambda function implementation

Create a lambda function with runtime Python3.12

The screenshot shows the 'Create function' wizard in the AWS Lambda console. The top navigation bar includes 'Lambda > Functions > Create function'. The main section is titled 'Create function' with an 'Info' link. It asks to choose one of three options: 'Author from scratch' (selected), 'Use a blueprint', or 'Container image'. The 'Basic information' tab is active, showing fields for 'Function name' (set to 'ingest_data'), 'Runtime' (set to 'Python 3.12'), 'Architecture' (set to 'x86_64'), and 'Permissions' (set to 'Change default execution role'). Other tabs like 'Advanced settings' are also visible. At the bottom right are 'Cancel' and 'Create function' buttons.

Below code snippet is the implementation of ingested data in lambda function. After lambda function implementation, the code is deployed.

```
1 import json
2 import boto3
3
4 s3 = boto3.client('s3')
5
6 def lambda_handler(event, context):
7     # Get the bucket names from the event
8     source_bucket = event['Records'][0]['s3']['bucket']['name']
9     key = event['Records'][0]['s3']['object']['key']
10
11    # Download the file from the source bucket
12    response = s3.get_object(Bucket=source_bucket, Key=key)
13    content = response['Body'].read().decode('utf-8')
14
15    # Convert content to uppercase
16    uppercase_content = content.upper()
17
18    # Upload the uppercase content to the next bucket
19    uppercase_bucket = "utshalab3bucket2"
20    s3.put_object(Body=uppercase_content, Bucket=uppercase_bucket, Key=key)
21
22    # Return a response
23    return {
24        'statusCode': 200,
25        'body': json.dumps('Data of the file transformed and uploaded successfully')
26    }
27
```

Adding trigger in the created lambda function with required configurations. So, that whenever a text file is uploaded triggered S3 bucket, its processed the data of the text file and uploads in specified S3 bucket location.

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.
 [X](#) [C](#)
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.

All object create events [X](#)

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.

The created S3 trigger can be viewed.

The trigger `usthalab3bucket` was successfully added to function `ingest_data`. The function is now receiving events from the trigger.

Function overview [Info](#) [Export to Application Composer](#) [Download ▾](#)

[Diagram](#) [Template](#)

ingest_data

S3

[+ Add destination](#)

[+ Add trigger](#)

Description
-

Last modified
7 minutes ago

Function ARN
`arn:aws:lambda:us-east-1:612362567483:function:ingest_data`

Function URL [Info](#)
-

[Code](#) [Test](#) [Monitor](#) [Configuration](#) [Aliases](#) [Versions](#)

Triggers (1) [Info](#) [C](#) [Fix errors](#) [Edit](#) [Delete](#) [Add trigger](#)

[Find triggers](#) [<](#) [1](#) [>](#)

Trigger

S3: usthala3bucket
`arn:aws:s3:::usthala3bucket` [Details](#)

3.3. Testing the implementation and monitoring in cloud watch logs

A text file with some content is uploaded in a S3 bucket.

Amazon S3 > Buckets > [usthalab3bucket](#) > 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, 32.0 B)				
All files and folders in this table will be uploaded.				
<input type="checkbox"/>	Name	Folder	Type	Size
<input type="checkbox"/>	test_text_file.txt	-	text/plain	32.0 B

Destination [Info](#)

Destination
[s3://usthalab3bucket](#)

▶ 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](#)

Amazon S3 > Buckets > [usthalab3bucket2](#)

utshalab3bucket2 [Info](#)

[Objects](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (1) [Info](#)

[Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Find objects by prefix](#) Show versions

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	test_text_file.txt	txt	February 22, 2024, 22:34:15 (UTC+05:45)	32.0 B	Standard

Now, navigating to Cloud watch and checking the log groups. The newly created lambda function 'ingest_data' is in the list of log group.

The screenshot shows the AWS CloudWatch interface. On the left, there's a sidebar with various navigation options like Dashboards, Alarms, Logs, Metrics, X-Ray traces, Events, Application Signals, Network monitoring, and Insights. Under Logs, 'Log groups' is selected. The main area is titled 'Log groups (7)' and lists the following log groups:

Log group	Type
/aws/lambda/RedshiftEventSubscription	Standard
/aws/lambda/RedshiftOverwatch	Standard
/aws/lambda/RoleCreationFunction	Standard
/aws/lambda/ingest_data	Standard
/aws/lambda/lambda_function	Standard
/aws/lambda/serverless_api	Standard
/aws/lambda/todoApplication	Standard

Opening the log group and in Log streams, there will be the list of the logs streams of the lambda functions.

The screenshot shows the AWS CloudWatch Log streams page. At the top, there are tabs for Log streams, Tags, Anomaly detection, Metric filters, Subscription filters, Contributor Insights, and Data protection. The 'Log streams' tab is selected. It displays a single log stream:

Log stream	Last event time
2024/02/22/[LATEST]425e54...	2024-02-22 22:34:14 (UTC+05:45)

Based upon the log events, the implementation performed has been executed successfully.

The screenshot shows the CloudWatch Log Events interface. The URL is [CloudWatch > Log groups > /aws/lambda/ingest_data > 2024/02/22/\[\\$LATEST\]425e54fb801c404a969177fbcc3723ec](#). The log group path is `/aws/lambda/ingest_data`. The log stream is `2024/02/22/[$LATEST]425e54fb801c404a969177fbcc3723ec`. The log events table shows the following entries:

Timestamp	Message
No older events at this moment. Retry	
2024-02-22T22:34:12.542+05:45	INIT_START Runtime Version: python:3.12.v19 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:a79oe1de439e89e7a1dc89465a221e8fe9bb3...
2024-02-22T22:34:13.110+05:45	START RequestId: abd5bd51-1176-4c3f-94c5-0f3302432aa2 Version: \$LATEST
2024-02-22T22:34:14.256+05:45	END RequestId: abd5bd51-1176-4c3f-94c5-0f3302432aa2
2024-02-22T22:34:14.256+05:45	REPORT RequestId: abd5bd51-1176-4c3f-94c5-0f3302432aa2 Duration: 1146.03 ms Billed Duration: 1147 ms Memory Size: 128 MB Max Memory Used...
No newer events at this moment. Auto retry paused . Resume	

Now, while checking in the S3 bucket location where processed data is saved, uploaded text file can viewed.

The screenshot shows the Amazon S3 Buckets interface. The URL is [Amazon S3 > Buckets > usthalab3bucket](#). The bucket name is `usthalab3bucket`. The [Info](#) tab is selected. The [Objects](#) tab is active. The objects table shows one item:

Name	Type	Last modified	Size	Storage class
test_text_file.txt	txt	February 22, 2024, 22:34:12 (UTC+05:45)	32.0 B	Standard

Initially text file content were in lower case.



Now, the content of the text file are in uppercase.



The content of the file can be also viewed from “Query with S3 select”.

A screenshot of the AWS S3 console. The left sidebar shows the bucket "utshalab3bucket2" with an "Info" link. The main area displays a single object named "test_text_file.txt" with a size of 32.0 B. In the top right corner, there is a context menu with several options: "Actions", "Create folder", "Upload", "Download as", "Share with a presigned URL", "Calculate total size", "Copy", "Move", "Initiate restore", and "Query with S3 Select". The "Query with S3 Select" option is highlighted with a blue border. Other options like "Edit actions", "Rename object", "Edit storage class", "Edit server-side encryption", "Edit metadata", "Edit tags", and "Make public using ACL" are also listed.

SQL query

Amazon S3 Select supports only the SELECT SQL command. Using the S3 console, you can extract up to 40 MB of records from an object that is up to 128 MB in size. To work with larger files or more records, use the AWS CLI, AWS SDK, or Amazon S3 REST API. For more complex SQL queries, use [Amazon Athena](#).

Add SQL from templates Run SQL query

```
1 /* To create reference point for writing SQL queries, you can display the first 5 records of input data by running the following SQL query: SELECT * FROM s3object s LIMIT 5 */  
2 SELECT * FROM s3object s LIMIT 5
```

SQL Ln 1, Col 1 Errors: 0 Warnings: 0

Query results

Query results are not available after you choose Close or navigate away. Choose Download results to download a copy of the following query results.

Status
Successfully returned 1 record in 2849 ms
Bytes returned: 33 B

```
1 HELLO EVERYONE. HAVE A NICE DAY.  
2
```

Download results