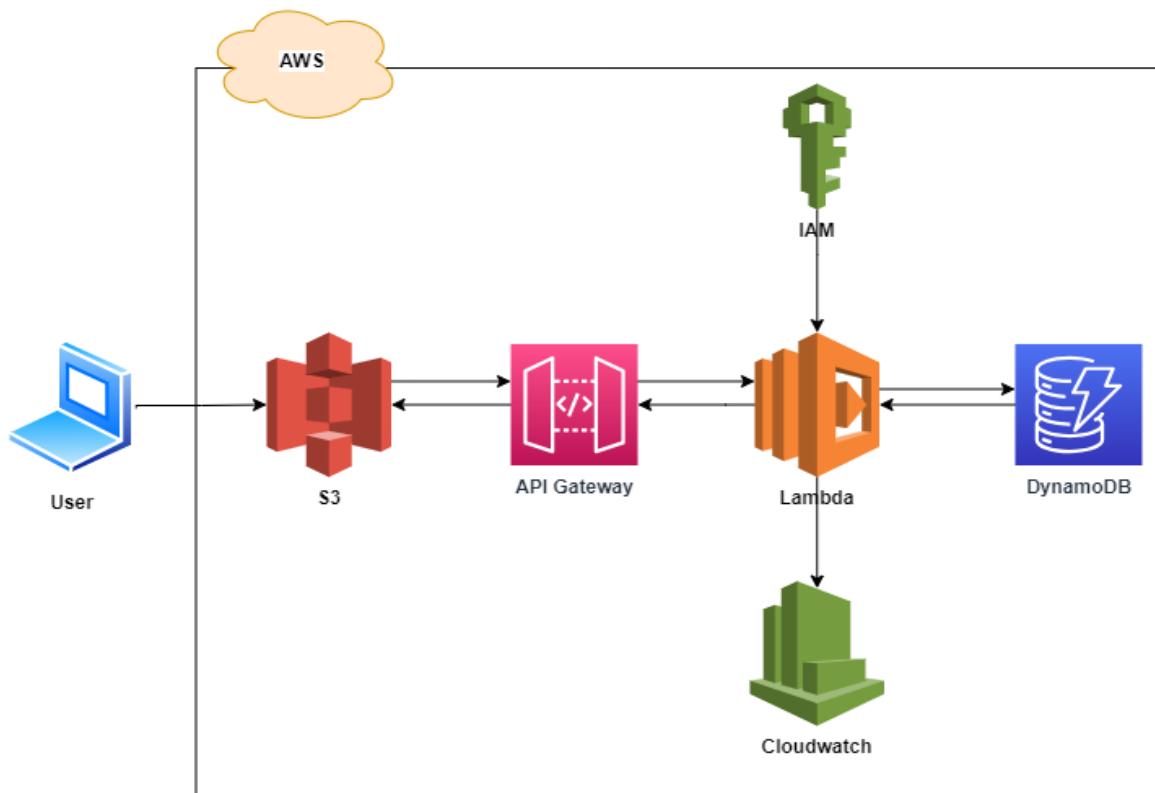


Creating a Serverless Contact Form using AWS API Gateway and Lambda

File Structure :

```
- /  
  - website/  
    - index.html  
    - style.css  
    - script.js  
  - lambda/  
    - index.js  
    - package.json
```

Flow of the Project :



Create the static website with the HTML form. Write JavaScript code to handle form submission and make an HTTP POST request to the API Gateway endpoint.

index.html -----

```
<!DOCTYPE html>
<html>
<head>
<title>Contact Form</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
<div class="container">
<h1>Contact Form</h1>
<form id="contact-form">
<label for="name">Name:</label>
<input type="text" id="name" name="name" required>

<label for="email">Email:</label>
<input type="email" id="email" name="email" required>

<label for="subject">Subject:</label>
<input type="text" id="subject" name="subject" required>

<label for="message">Message:</label>
<textarea id="message" name="message" rows="5" required></textarea>

<button type="submit">Submit</button>
</form>
</div>

<script src="script.js"></script>
</body>
</html>
```

style.css -----

```
.container {  
    max-width: 500px;  
    margin: 0 auto;  
    padding: 20px;  
}  
  
form label {  
    display: block;  
    margin-bottom: 10px;  
}  
  
form input,  
form textarea {  
    width: 100%;  
    padding: 10px;  
    margin-bottom: 15px;  
}  
  
form button {  
    padding: 10px 20px;  
    background-color: #007bff;  
    color: #fff;  
    border: none;  
    cursor: pointer;  
}  
  
form button:hover {  
    background-color: #0069d9;  
}
```

script.js

```
document.getElementById('contact-form').addEventListener('submit', function(event) {
  event.preventDefault(); // Prevent form submission
  // Validate form inputs
  var name = document.getElementById('name').value.trim();
  var email = document.getElementById('email').value.trim();
  var subject = document.getElementById('subject').value.trim();
  var message = document.getElementById('message').value.trim();
  // Email validation using a regular expression
  var emailRegex = /^[\S+@\S+\.\S+$/;
  if (!name || !email || !subject || !message || !emailRegex.test(email)) {
    alert('Please fill in all fields with valid inputs.');
    return;
  }
  // Construct the form data
  var formData = {
    name: name,
    email: email,
    subject: subject,
    message: message
  };
  // Perform form submission
  submitForm(formData);
});
function submitForm(formData) {
  // Make an API request to the backend (API Gateway) for form submission
  fetch('your-api-gateway-url', { // URL that represents the backend API endpoint to which the
    form data is going to be sent
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(formData)
  })
  .then(function(response) {
    if (response.ok) {
      // Redirect to the thank you page
      window.location.href = 'thank-you.html';
    } else {
      throw new Error('Form submission failed.');
    }
  })
  .catch(function(error) {
    console.error(error);
    alert('Form submission failed. Please try again later.');
  });
}
```

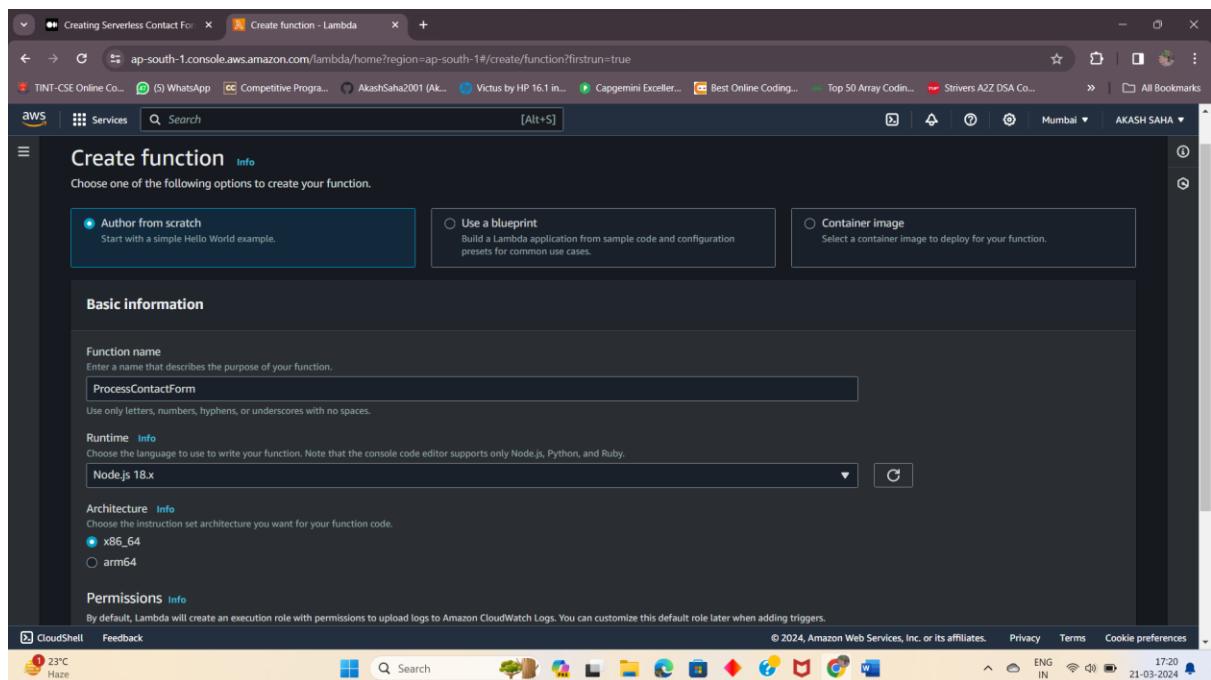
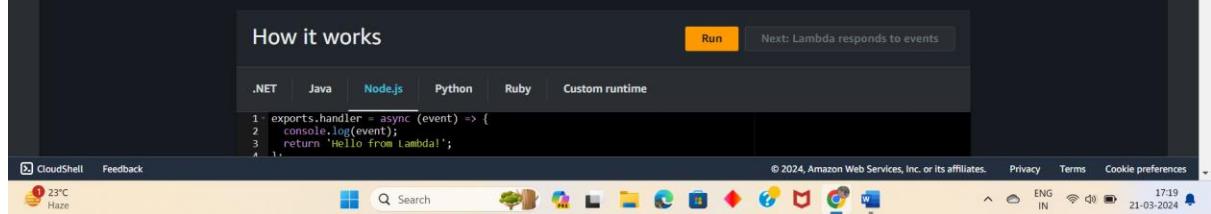
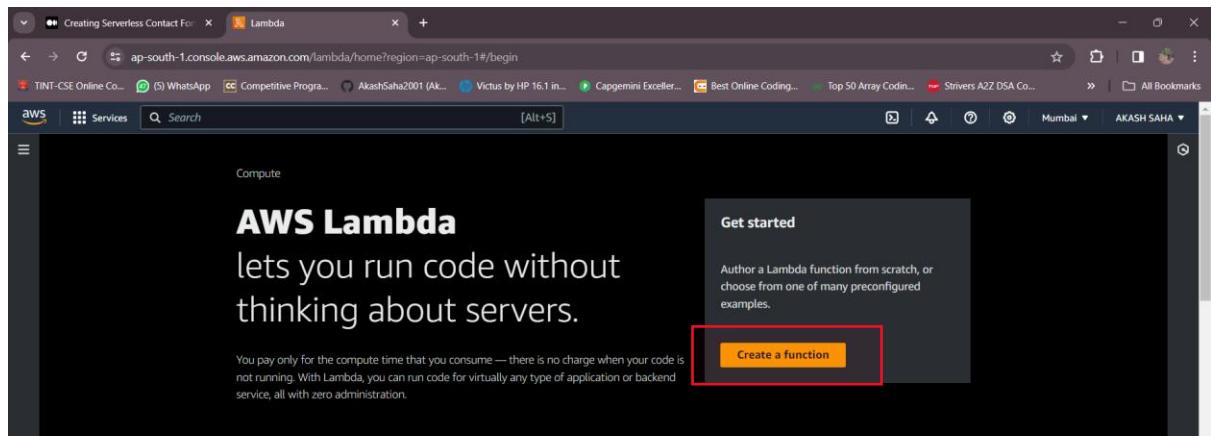
thank-you.html -----

```
<!DOCTYPE html>
<html>
<head>
<title>Thank You</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
<div class="container">
<h1>Thank You!</h1>
<p>Your message has been submitted successfully.</p>
</div>
</body>
</html>
```

The structure will be this :

```
▽ WEBSITE
  ◊ index.html
  JS script.js
  # style.css
  ◊ thank-you.html
```

Activity 1 : Create a Lambda function



Now click on Create.

Create another folder in your system named as lambda.

index.js -----

```
const AWS = require('aws-sdk');
const dynamodb = new AWS.DynamoDB.DocumentClient();
const { v4: uuidv4 } = require('uuid');

exports.handler = async (event) => {
  try {
    console.log('Raw input data:', event); // Add this line to log the raw input data

    const formData = {
      name: event.name,
      email: event.email,
      subject: event.subject,
      message: event.message,
    };

    const item = {
      SubmissionId: generateUUID(), // Generate a UUID
      ...formData, // Use the form data as attributes
    };

    // Store the form data in DynamoDB
    await storeFormData(item);

    return {
      statusCode: 200,
      body: JSON.stringify({ message: 'Form submitted successfully' }),
    };
  } catch (error) {
    console.error(error);
    return {
      statusCode: 500,
      body: JSON.stringify({ message: 'Error submitting the form' }),
    };
  }
};

async function storeFormData(item) {
  const params = {
    TableName: 'ContactFormEntries',
    Item: item,
  };
  await dynamodb.put(params).promise();
}

function generateUUID() {
  return uuidv4();
}
```

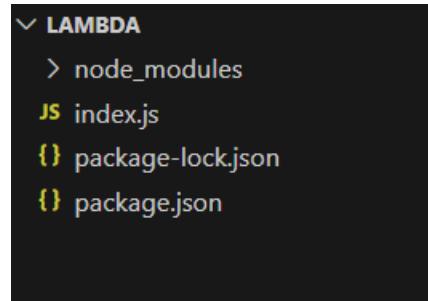
package.json -----

```
{  
  "name": "serverless-contact-form",  
  "version": "1.0.0",  
  "description": "Lambda function for handling the serverless contact form",  
  "main": "index.js",  
  "dependencies": {  
    "aws-sdk": "^2.1386.0"  
  }  
}
```

Move to the lambda directory and install the AWS SDK by running the command below. The npm install is used to install the dependencies listed in the package.json file of the project.

npm install

This will be the structure :



Then zip the folder. [lambda.zip]

Activity 2: Create a Bucket

The screenshot shows the Amazon S3 service page. On the right, a modal window titled "Create a bucket" is displayed. It contains a brief description: "Every object in S3 is stored in a bucket. To upload files and folders to S3, you'll need to create a bucket where the objects will be stored." Below the description is a prominent yellow "Create bucket" button, which is highlighted with a red box. The main S3 page features a large "Amazon S3" logo and the tagline "Store and retrieve any amount of data from anywhere". A section titled "How it works" includes a screenshot of a video player showing an "Introduction to Amazon S3" video.

The screenshot shows the "Create 53 bucket" wizard on the "General configuration" step. The "AWS Region" dropdown is set to "Asia Pacific (Mumbai) ap-south-1". The "Bucket name" field contains "lambda-form-bucket-project", which is underlined in blue, indicating it's the active input field. Below the bucket name, there's a note: "Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming." A "Choose bucket" button is available for copying settings from an existing bucket. The "Object Ownership" section shows two options: "ACLs disabled (recommended)" (selected) and "ACLs enabled". The "Object Ownership" heading has a small "Info" link. The bottom of the screen shows a Windows taskbar with various pinned icons and system status indicators.

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.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning

Disable
 Enable

Upload the lambda.zip here.

lambda-form-bucket-project [Info](#)

Objects (0) [Info](#)

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](#)

Name	Type	Last modified	Size	Storage class
No objects				
You don't have any objects in this bucket.				
<input type="button" value="Upload"/>				

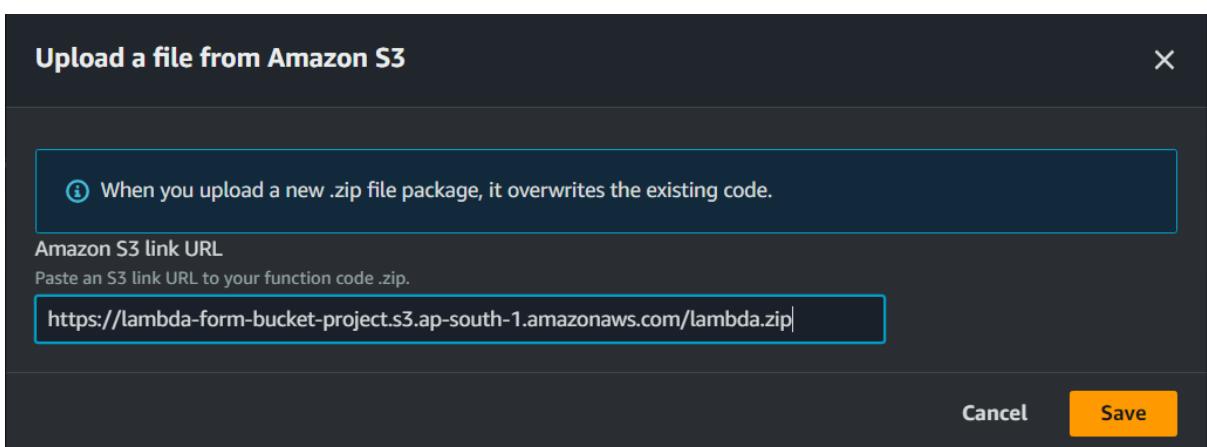
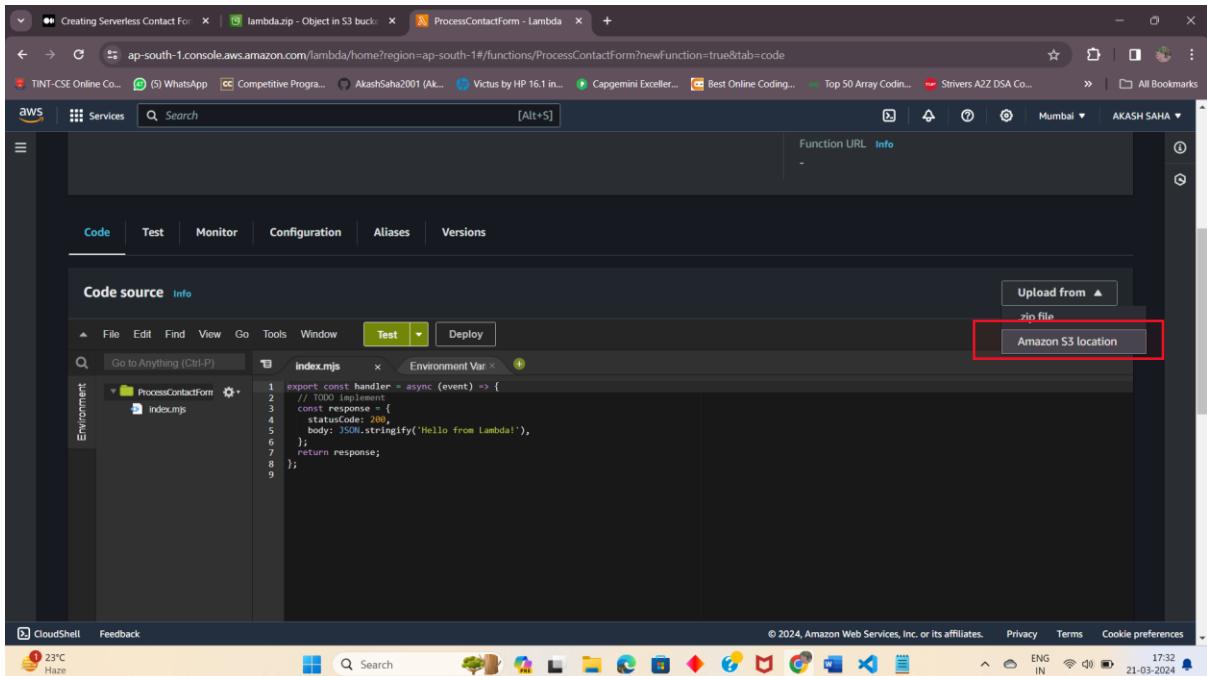
The screenshot shows the AWS S3 console interface. In the top navigation bar, there are three tabs: 'Creating Serverless Contact For...', 'Upload objects - S3 bucket lam...', and 'ProcessContactForm - Lambda'. The current view is under the 'Upload objects - S3 bucket lam...' tab. The main area displays a file upload form with a large input field for dragging files. Below it, a table lists the uploaded file 'lambda.zip' with details like name, type (application/zip), and size (11.3 MB). A 'Destination' section shows the destination as 's3://lambda-form-bucket-project'. At the bottom, there are links for 'CloudShell', 'Feedback', and a status bar showing 'ENG IN' and the date '21-03-2024'.

Upload the file.

Go to the lambda.zip and Copy the Object URL.

The screenshot shows the AWS S3 object details page for 'lambda.zip'. The top navigation bar has tabs for 'Creating Serverless Contact For...', 'lambda.zip - Object in S3 buck...', and 'ProcessContactForm - Lambda'. The current view is under the 'lambda.zip - Object in S3 buck...' tab. The main area is titled 'lambda.zip' and shows the 'Properties' tab selected. It displays various metadata: Owner (a668b18f0bc6d294d94ed960f8c92746e4c058377b4325699ae505c4842ad9ca), AWS Region (Asia Pacific (Mumbai) ap-south-1), Last modified (March 21, 2024, 17:31:22 (UTC+05:30)), Size (11.3 MB), Type (zip), and Key (lambda.zip). On the right, there are sections for 'S3 URI' (s3://lambda-form-bucket-project/lambda.zip), 'Amazon Resource Name (ARN)' (arn:aws:s3:::lambda-form-bucket-project/lambda.zip), 'Entity tag (Etag)' (aaa06f6961e92f96c52716f491e8637c), and 'Object URL' (https://lambda-form-bucket-project.s3.ap-south-1.amazonaws.com/lambda.zip). The 'Object URL' is highlighted with a red box. At the bottom, there are links for 'CloudShell', 'Feedback', and a status bar showing 'ENG IN' and the date '21-03-2024'.

Now go to the lambda function and upload the zip by attaching the S3 location here.



Now go to the Test

The screenshot shows the AWS Lambda console interface. The top navigation bar includes tabs for 'Code', 'Test' (which is selected), 'Monitor', 'Configuration', 'Aliases', and 'Versions'. Below this, a sub-header says 'Test event' with a 'Info' link. A 'Save' button and a large orange 'Test' button are on the right. The main area contains fields for 'Event name' (set to 'TestEventJSON'), 'Event sharing settings' (set to 'Private'), and a 'Template - optional' dropdown (set to 'hello-world'). A note at the top says 'To invoke your function without saving an event, configure the JSON event, then choose Test.' There are also 'Create new event' and 'Edit saved event' buttons.

Edit the code :

The screenshot shows the 'Event JSON' editor. It displays a JSON object with the following content:

```
1 {  
2   "name": "Akash Saha",  
3   "email": "mr.akashsaha574@gmail.com",  
4   "subject": "It's Me",  
5   "message": "Testing"  
6 }
```

```
{  
  "name": "Akash Saha",  
  "email": "mr.akashsaha574@gmail.com",  
  "subject": "It's Me",  
  "message": "Testing"  
}
```

Click on the Test to check if it is getting successfully executed or not.

The screenshot shows the test results for the function. It displays a green circular icon with a checkmark, followed by the text 'Executing function: succeeded (logs [2])'. Below this, there is a 'Details' link. At the bottom, there is another 'Test event' section with a 'Save' and 'Test' button.

Activity 3: CloudWatch (Optional)

The screenshot shows the AWS CloudWatch Log groups interface. The left sidebar navigation includes: Dashboards, Alarms, Logs (selected), Log groups, Log Anomalies, Live Tail, Logs Insights, Metrics, X-Ray traces, Events, Application Signals, Network monitoring, and Insights. Under Logs, Log groups is selected. The main content area displays the 'Log groups (1)' section. A table lists one log group: '/aws/lambda/ProcessContactForm'. The table columns are: Log group (checkbox), Log class (Standard), Anomaly detection (Configure), Data protection (Never expire), Sensitivity (Low), Retention (Never expire), and Metrics (checkbox). There are 'Actions' and 'Create log group' buttons at the top right.

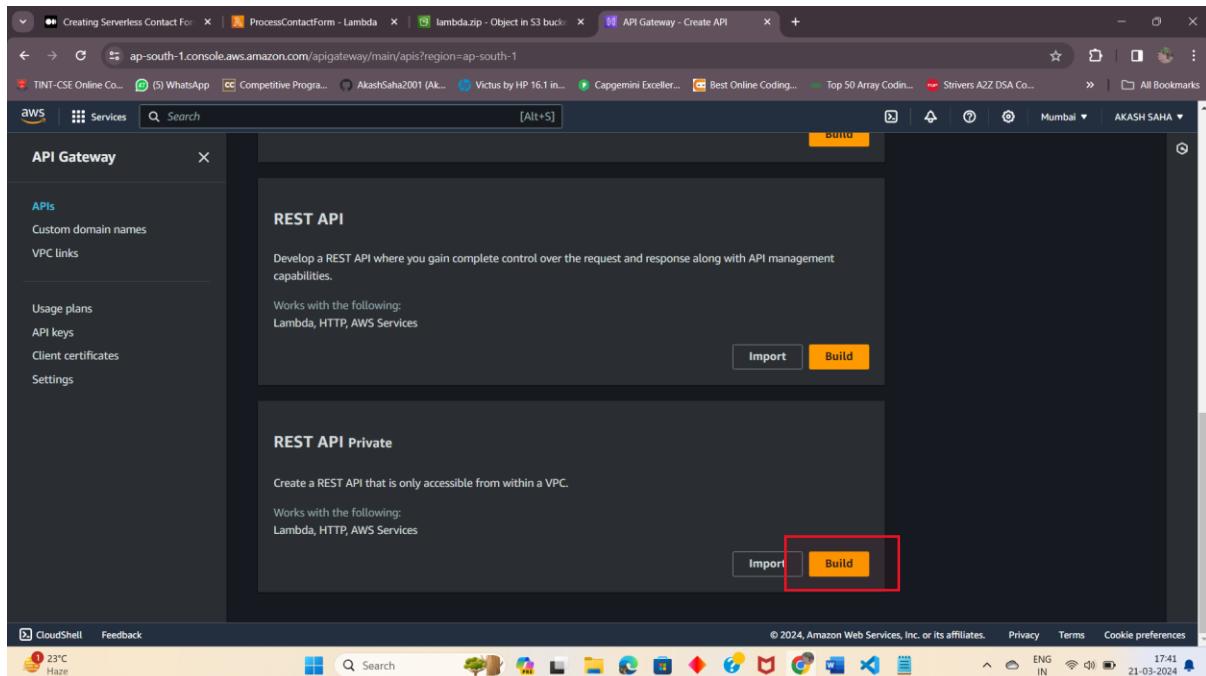
The screenshot shows the configuration details for the log group '/aws/lambda/ProcessContactForm'. The left sidebar navigation is identical to the previous screenshot. The main content area shows the log group's ARN: arn:aws:logs:ap-south-1:749474135282:log-group:/aws/lambda/ProcessContactForm. It also displays Creation time (2 minutes ago), Retention (Never expire), and Stored bytes. Below this, the 'Log streams' tab is selected, showing a table with one log stream entry: '2024/03/21/[LATEST]fc22a9d2fabaa4a07bd388c7249255598' with a Last event time of 2024-03-21 17:36:48 (UTC+05:30). Other tabs include Tags, Anomaly detection, Metric filters, Subscription filters, Contributor insights, and Data protection. There are 'Configure' and 'Data protection' buttons at the top right.

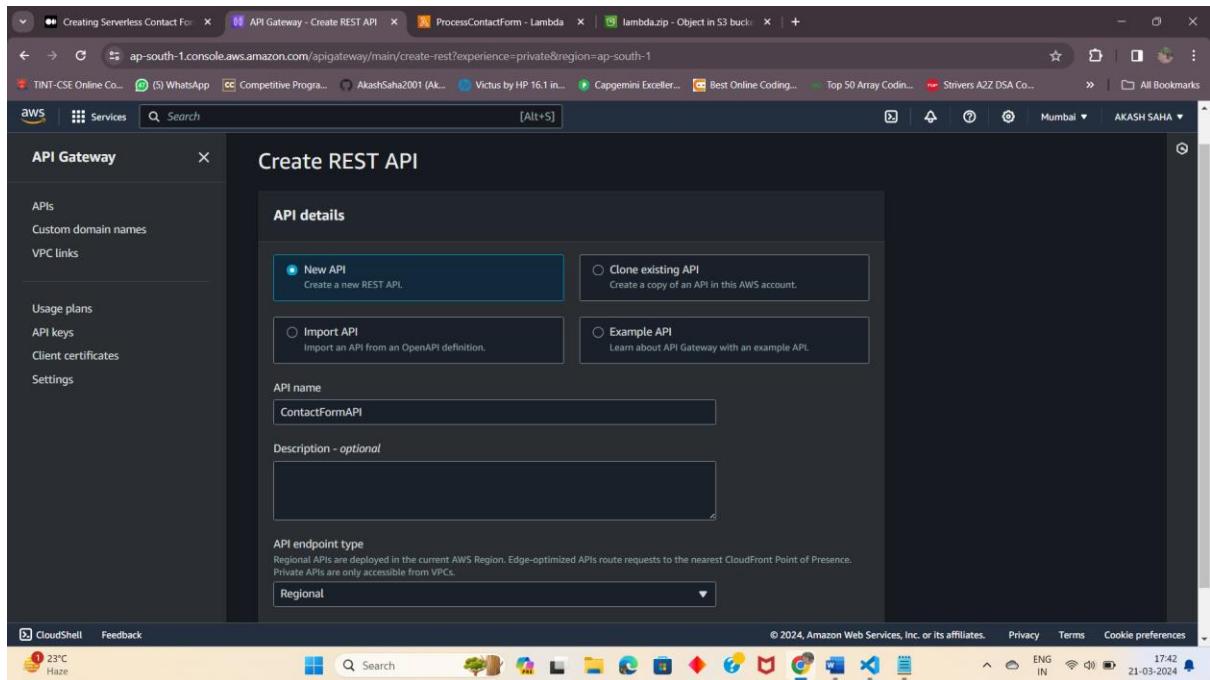
```

2024-03-21T17:36:48.176+05:30 START RequestId: e2be5b25-6c36-4f60-bae9-04e3b8ad7fea Version: $LATEST
2024-03-21T17:36:48.197+05:30 2024-03-21T12:06:48.197Z e2be5b25-6c36-4f60-bae9-04e3b8ad7fea INFO Raw input data: { name: 'Akash Saha', email: 'mr.akashsaha574@gmail.com' }
2024-03-21T12:06:48.197Z e2be5b25-6c36-4f60-bae9-04e3b8ad7fea INFO Raw input data: { name: 'Akash Saha', email: 'mr.akashsaha574@gmail.com', subject: 'It's Me', message: 'Testing' }
2024-03-21T17:36:48.938+05:30 2024-03-21T12:06:48.938Z e2be5b25-6c36-4f60-bae9-04e3b8ad7fea ERROR AccessDeniedException: User: arn:aws:sts::749474135282:assumed-role/ProcessContactForm-role-e1lobcqu/ProcessContactForm is not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:ap-south-1:749474135282:table/ContactFormEntries because no identity-based policy allows the dynamodb:PutItem action
at Request.extractError (/var/task/node_modules/aws-sdk/lib/protocol/json.js:80:27)
at Request.callListeners (/var/task/node_modules/aws-sdk/lib/sequential_executor.js:106:20)
at Request.emit (/var/task/node_modules/aws-sdk/lib/sequential_executor.js:78:10)
at Request.emit (/var/task/node_modules/aws-sdk/lib/request.js:686:14)
at Request.transition (/var/task/node_modules/aws-sdk/lib/request.js:22:10)
at AcceptorStateMachine.runTo (/var/task/node_modules/aws-sdk/lib/state_machine.js:14:12)
at /var/task/node_modules/aws-sdk/lib/state_machine.js:26:10
at Request._onAnonymous_ (/var/task/node_modules/aws-sdk/lib/request.js:38:9)
at Request._onAnonymous_ (/var/task/node_modules/aws-sdk/lib/request.js:688:12)
at Request.callListeners (/var/task/node_modules/aws-sdk/lib/sequential_executor.js:116:18) {
  code: 'AccessDeniedException',
  '__type': 'See error.__type for details.',
  '[Message]': 'See error.Message for details.',
  time: 2024-03-21T12:06:48.936Z,
  requestId: '02LBUSKFERN6K4QHE1D5S909RVW4KQNS0SAEMVJF6609ASUAAJG',
  requestID: '02LBUSKFERN6K4QHE1D5S909RVW4KQNS0SAEMVJF6609ASUAAJG',
}

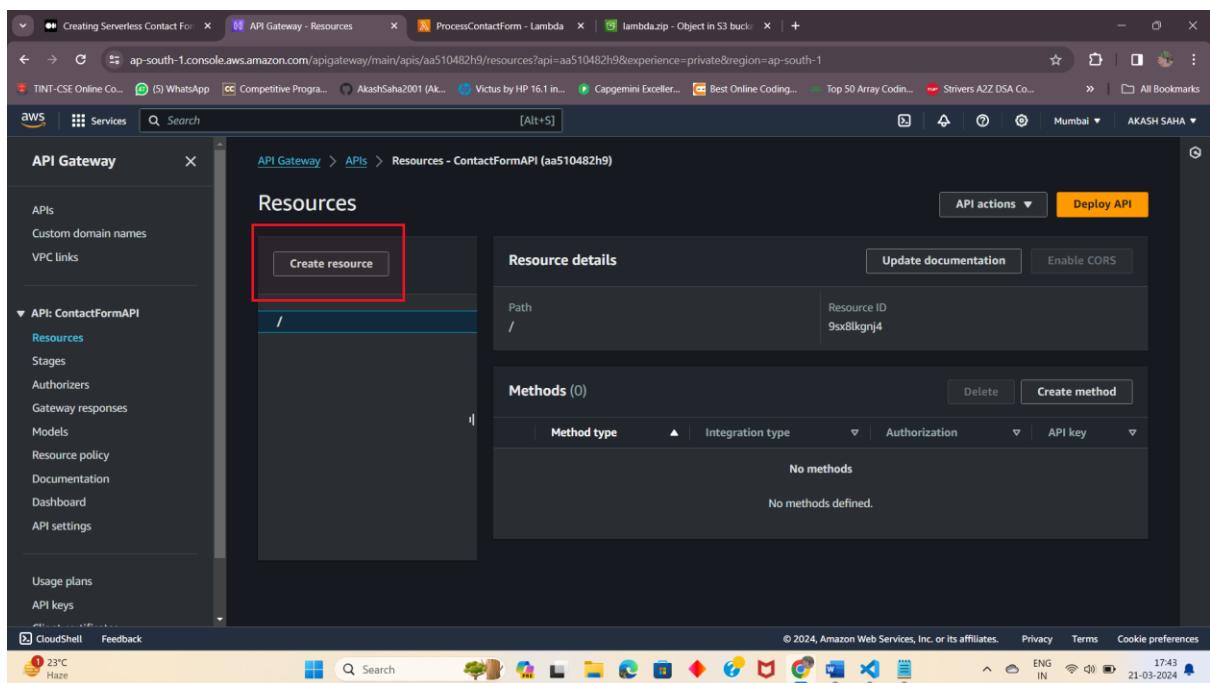
```

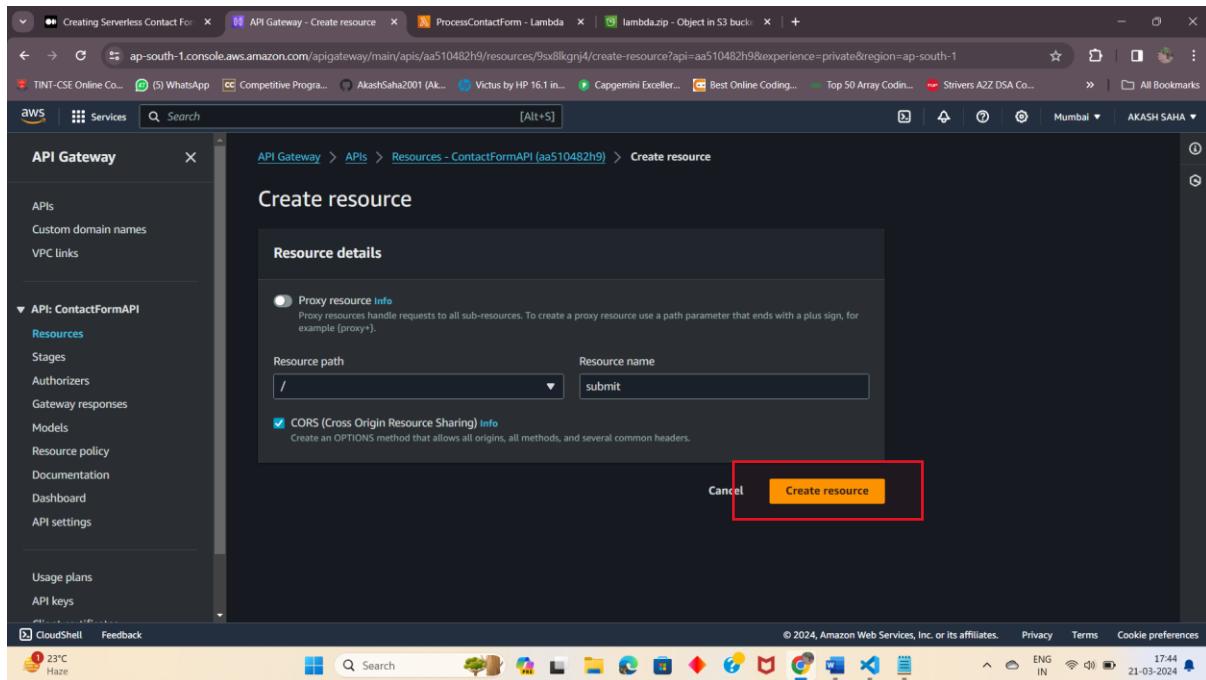
Activity 4: Creating API gateway



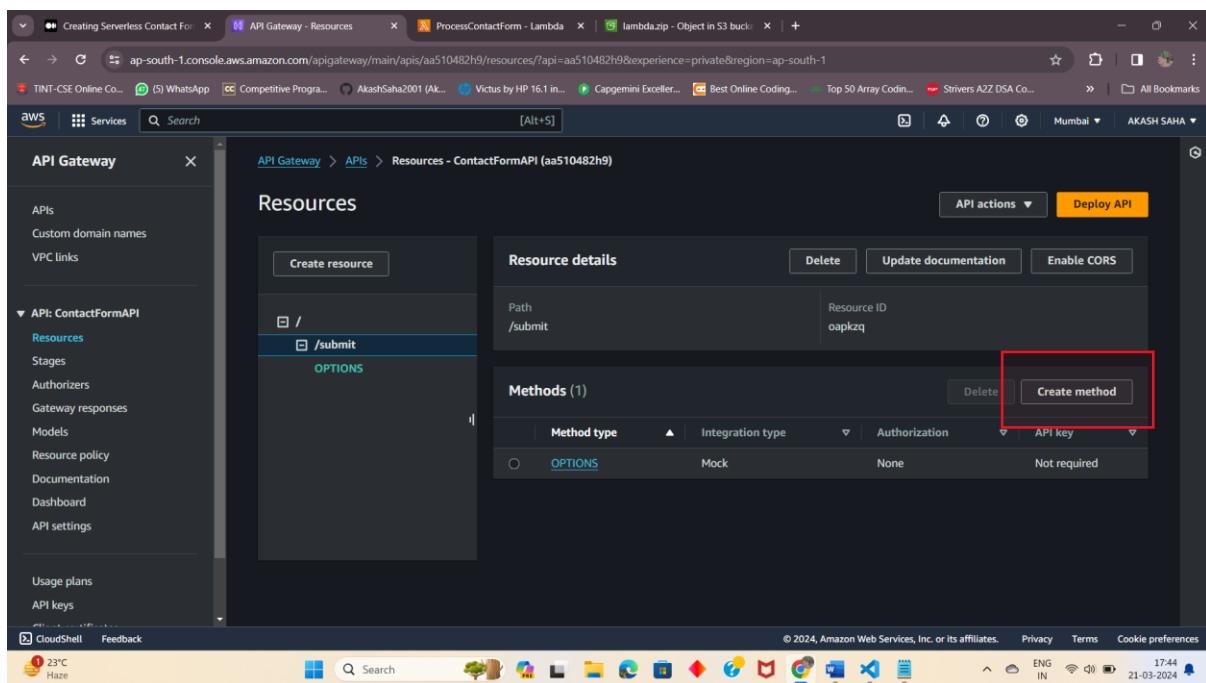


Create a resource.





Create a method



Creating Serverless Contact Form API Gateway - Create method ProcessContactForm - Lambda lambda.zip - Object in S3 buck...

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.

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

23°C Haze ENG IN 17:46 21-03-2024

Creating Serverless Contact Form API Gateway - Create method ProcessContactForm - Lambda lambda.zip - Object in S3 buck...

Create method

Method details

Integration type:

- 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.

ap-south-1

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.

Method request settings

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

23°C Haze ENG IN 17:46 21-03-2024

Now Deploy the API.

The screenshot shows the AWS API Gateway Resources page. On the left, there's a sidebar with 'APIs', 'Custom domain names', 'VPC links', and a expanded section for 'API: ContactFormAPI' containing 'Resources', 'Stages', 'Authorizers', 'Gateway responses', 'Models', 'Resource policy', 'Documentation', 'Dashboard', and 'API settings'. Below that are 'Usage plans' and 'API keys'. At the bottom of the sidebar are 'CloudShell' and 'Feedback' buttons. The main area is titled 'Resources' and shows a tree structure with a POST method under '/submit'. To the right of the tree is a detailed view of the '/submit - POST - Method execution' resource. It includes fields for 'ARN' (arn:aws:execute-api:ap-south-1:749474135282:aa510482h9/*/POST/submit) and 'Resource ID' (oapkzq). Below this is a flow diagram showing the request-response cycle: Client → Method request → Integration request → Lambda integration → Method response → Integration response → Method response. There are tabs for 'Method request', 'Integration request', 'Integration response', 'Method response', and 'Test'. At the top right of the main area are 'API actions' and a prominent orange 'Deploy API' button, which is highlighted with a red box. The status bar at the bottom right shows 'ENG IN' and the date '21-03-2024'.

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

dev

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

Deployment description

dev stage

Cancel Deploy

Copy the invoke URL

The screenshot shows the AWS API Gateway Stages page. On the left, there's a sidebar with 'APIs', 'Custom domain names', 'VPC links', and a expanded section for 'API: ContactFormAPI' containing 'Resources', 'Stages' (which is selected), 'Authorizers', 'Gateway responses', 'Models', 'Resource policy', 'Documentation', 'Dashboard', and 'API settings'. The main area is titled 'Stages' and shows a table with one row for 'dev'. The 'Stage details' section includes fields for 'Stage name' (set to 'dev'), 'Cache cluster' (set to 'Inactive'), 'Default method-level caching' (set to 'Inactive'), and a 'Logs and tracing' section. A red box highlights the 'Invoke URL' field, which contains the value <https://aa510482h9.execute-api.ap-south-1.amazonaws.com/dev>. The status bar at the bottom right shows the date as 21-03-2024.

Go to the Edit and set the following settings.

The screenshot shows the 'Edit stage' settings page for the 'dev' stage. It has several sections: 'Throttling settings' (with 'Throttling' selected), 'Rate' (set to 10000 requests per second), 'Burst' (set to 5000 requests), 'Firewall and certificate settings' (including 'Web application firewall (AWS WAF)' set to 'None' and 'Client certificate' also set to 'None'). At the bottom are 'Cancel' and 'Save' buttons. The status bar at the bottom right shows the date as 21-03-2024.

Activity 5: Create a table

The screenshot shows the Amazon DynamoDB home page. On the left, there's a sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations (which is highlighted in red), Reserved capacity, and Settings. The main content area features the text "Amazon DynamoDB: A fast and flexible NoSQL database service for any scale". Below this, there's a section titled "How it works" with a screenshot of a mobile app interface. To the right, there's a "Get started" box with a "Create table" button, which is also highlighted with a red box. Further down, there's a "Pricing" section. At the bottom of the page, there's a footer with links for CloudShell, Feedback, and various AWS services.

The screenshot shows the "Create table" wizard. Step 1 is "Set table details". It asks for the "Table name" (ContactFormEntries) and "Partition key" (SubmissionId). It also has sections for "Sort key - optional" and "Table settings". Under "Table settings", there are two radio button options: "Default settings" (selected) and "Customize settings". The status bar at the bottom indicates it's step 1 of 5.

From the API Gateway role created when adding a permission to it to invoke a Lambda function, select it to add a DynamoDB policy that can allow us to put new items into the database.

Go to the IAM Roles, search for your created AWS lambda function (ProcessContactForm)

The screenshot shows the AWS IAM Roles page. A search bar at the top contains the text "Process". Below the search bar, a table lists roles. One role, "ProcessContactForm-role-e3lobcqu", is highlighted with a red box. The table includes columns for "Role name", "Trusted entities", and "Last activity".

The screenshot shows the "Details" page for the "ProcessContactForm-role-e3lobcqu" role. The "Permissions" tab is selected. At the top right of the permissions section, there is a "Add permissions" button and an "Attach policies" button, both highlighted with a red box. Below these buttons is a "Create inline policy" link. The main area displays the ARN of the role and a table showing attached policies, with one policy named "AWSLambdaBasicExecutionRole-1c90b1..." listed.

The screenshot shows the AWS IAM 'Add permissions' dialog. On the left, the navigation pane is visible with 'Identity and Access Management (IAM)' selected. The main area shows the 'Attach policy to ProcessContactForm-role-e3lobcq' dialog. Under 'Other permissions policies (1/917)', the 'AmazonDynamoDBFullAccess' policy is selected and highlighted with a red box. At the bottom right of the dialog, the 'Add permissions' button is also highlighted with a red box.

Test again your Lambda function with the test event you previously created. Now you'll see how the data is inserted into the DynamoDB table.

The screenshot shows the 'Event JSON' configuration for a Lambda function. The JSON payload is as follows:

```
1 {  
2   "name": "Akash Saha",  
3   "email": "mr.akashsaha574@gmail.com",  
4   "subject": "It's Me",  
5   "message": "Testing"  
6 }  
7
```

The screenshot shows the execution log for a Lambda function. It indicates a successful execution with the message 'Executing function: succeeded'. The log output is:

```
{  
  "statusCode": 200,  
  "body": "{\"message\":\"Form submitted successfully\"}"  
}
```

Check your database if it is inserted or not.

The screenshot shows the AWS DynamoDB console. On the left, the navigation pane is visible with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Under the 'Tables' section, 'ContactFormEntries' is selected. The main area displays the 'ContactFormEntries' table with a single item listed:

SubmissionId	email	message	name	subject
5dce91c1-7a36-49b2-8f4...	mr.akasha...	Testing	Akash Saha	It's Me

Upload the website to S3:

Create an S3 bucket to host the static website. Upload the website files to the bucket.

- Replace the fetch line of code in your script.js file with the Invoke URL of your API Gateway adding at the end the /submit endpoint.

```
// Before
fetch('your-api-gateway-url', {

// Current
fetch('api-gateway-invoke-url/submit', {
```

```
fetch('https://aa510482h9.execute-api.ap-south-1.amazonaws.com/dev/submit', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(formData)
})
```

Create a new S3 bucket

The screenshot shows the AWS S3 console with the URL <https://s3.console.aws.amazon.com/s3/bucket/create?region=ap-south-1&bucketType=general>. The page is titled "Create S3 bucket | S3 | Global".
General configuration
AWS Region: Asia Pacific (Mumbai) ap-south-1
Bucket name: static-website-bucket-project (highlighted)
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
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
CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 22°C Haze ENG IN 1812 21-03-2024

The screenshot shows the AWS S3 console with the URL <https://s3.console.aws.amazon.com/s3/bucket/create?region=ap-south-1&bucketType=general>. The page is titled "Create S3 bucket | S3 | Global".
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.
Warning: 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.
Bucket Versioning
Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore
CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 22°C Haze ENG IN 1811 21-03-2024

Now select the bucket and go to the Properties section scroll down to the last and search for Static website hosting.

The screenshot shows the AWS S3 bucket properties for "static-website-bucket-project".
Static website hosting
Use this bucket to host a website or redirect requests. [Learn more](#)
Static website hosting
Disabled
Edit (button highlighted with a red border)

The screenshot shows the AWS S3 console with the 'Static website hosting' configuration page for a bucket named 'static-website-bucket-project'. The 'Host a static website' option is selected. Under 'Index document', 'index.html' is specified. Under 'Error document - optional', 'error.html' is specified. A note states: '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.' The status bar at the bottom indicates it's 18:15 IN 21-03-2024.

Upload all the files.

The screenshot shows the AWS S3 console with the 'Objects' tab selected for the 'static-website-bucket-project' bucket. There are four objects listed:

Name	Type	Last modified	Size	Storage class
index.html	html	March 21, 2024, 18:17:12 (UTC+05:30)	786.0 B	Standard
script.js	js	March 21, 2024, 18:17:11 (UTC+05:30)	1.7 KB	Standard
style.css	css	March 21, 2024, 18:17:11 (UTC+05:30)	466.0 B	Standard

The status bar at the bottom indicates it's 18:17 IN 21-03-2024.

Under the Permissions section Go to the Bucket Policy

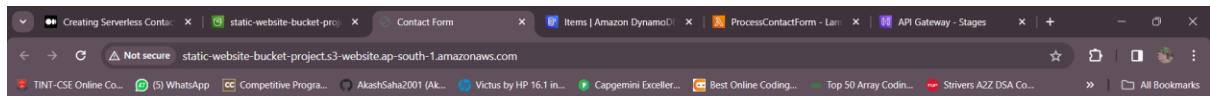
The screenshot shows the AWS S3 console with the 'Permissions' tab selected. Under 'Block public access (bucket settings)', 'Block all public access' is set to 'Off'. A red box highlights the 'Bucket policy' section, which contains the JSON policy code shown below.

- Add the following as the bucket policy.
- **Note:** Modify the Resource line according to your S3 bucket name.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "PublicReadGetObject",  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": "s3:GetObject",  
      "Resource": "arn:aws:s3:::static-website-bucket-project/*"  
    }  
  ]  
}
```

Now go to the Properties section and scroll down search for Static website hosting, click on the link to view your webpage.

The screenshot shows the 'Properties' section of the AWS S3 console. Under 'Static website hosting', 'Enabled' is checked. The 'Bucket website endpoint' field contains the URL <http://static-website-bucket-project.s3-website.ap-south-1.amazonaws.com>.



Contact Form

Name:

Email:

Subject:

Message:



Configure CORS in the API Gateway and in the S3 bucket.

Go to the Permissions section and search for Cross-origin resource sharing

Cross-origin resource sharing (CORS)

The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. [Learn more](#)

No configurations to display

Add the following JSON, modifying the AllowedOrigins value to your static website URL to restrict access to the bucket resources from other origins, thereby enhancing security.

```
[  
  {  
    "AllowedHeaders": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "POST"  
    ],  
    "AllowedOrigins": [  
      "http://static-website-bucket-project.s3-website.ap-south-1.amazonaws.com"  
    ],  
    "ExposeHeaders": []  
  }  
]
```

The screenshot shows the AWS S3 console with the path: Amazon S3 > Buckets > static-website-bucket-project > Edit cross-origin resource sharing (CORS). The page displays the CORS configuration in JSON format:

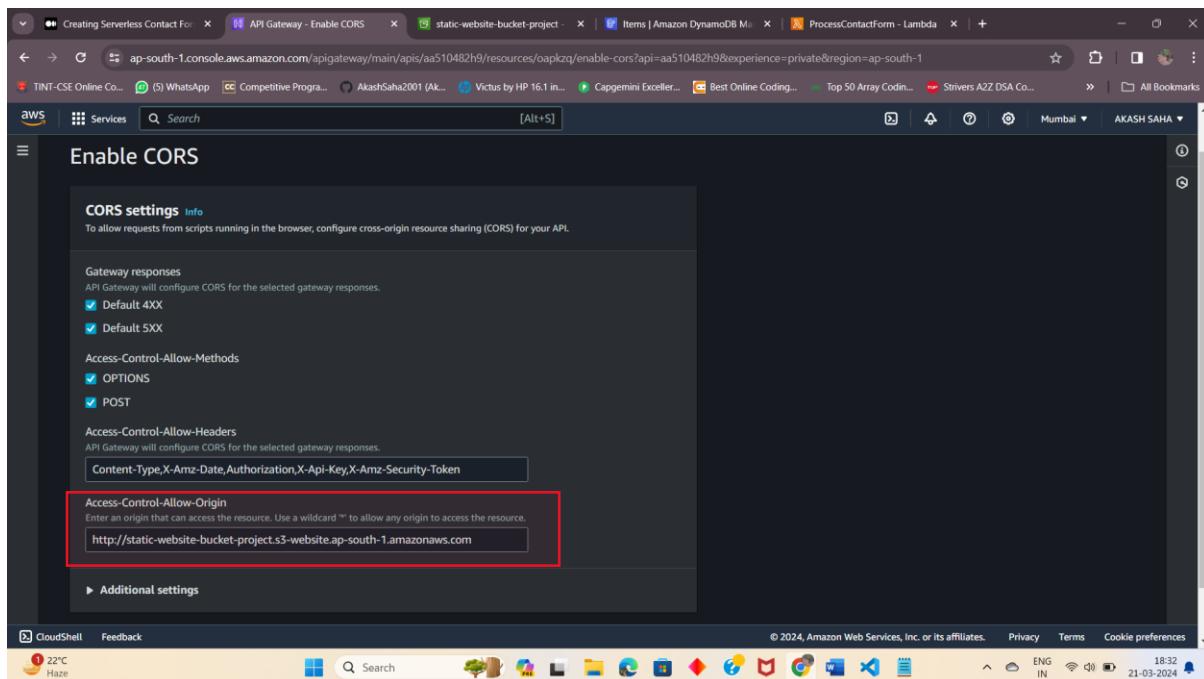
```
1 [  
2 {  
3     "AllowedHeaders": [  
4         "*"  
5     ],  
6     "AllowedMethods": [  
7         "POST"  
8     ],  
9     "AllowedOrigins": [  
10        "http://static-website-bucket-project.s3-website.ap-south-1.amazonaws.com"  
11    ],  
12    "ExposeHeaders": []  
13}  
14]
```

The browser's status bar at the bottom indicates the date and time as 21-03-2024 18:29.

API Gateway: It also needs to be configured to explicitly allow requests coming from the S3 website domain.

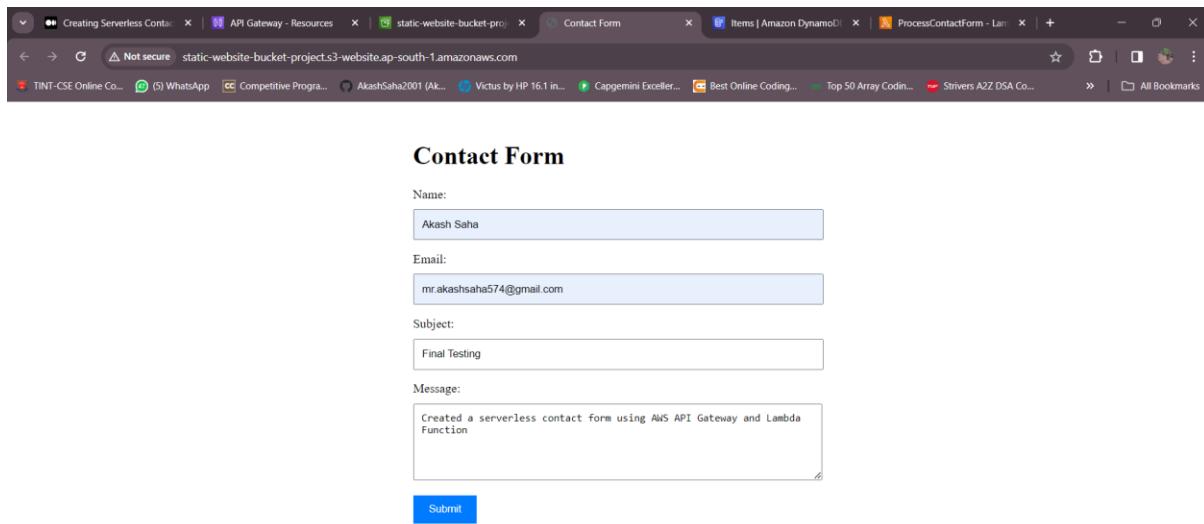
The screenshot shows the AWS API Gateway Resources page for the ContactFormAPI. The left sidebar shows the API structure: APi: ContactFormAPI > Resources > / > /submit. The main panel displays the Resource details for the /submit endpoint, including the Path (/submit), Resource ID (oapkzq), and Methods (OPTIONS and POST). The 'Enable CORS' button is highlighted with a red box. The browser's status bar at the bottom indicates the date and time as 21-03-2024 18:31.

Under the Access-Control-Allow-Origin paste your static website URL

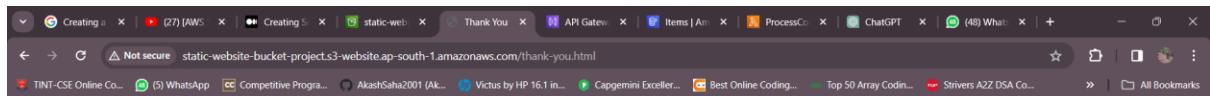


The screenshot shows the AWS Management Console with multiple tabs open. The active tab is 'Enable CORS' under the 'API Gateway - Enable CORS' section. The 'CORS settings' panel is displayed, showing configuration for 'Gateway responses' (Default 4XX and Default 5XX checked), 'Access-Control-Allow-Methods' (OPTIONS and POST checked), and 'Access-Control-Allow-Headers' (Content-Type, X-Amz-Date, Authorization, X-Api-Key, X-Amz-Security-Token). The 'Access-Control-Allow-Origin' field is highlighted with a red box and contains the URL 'http://static-website-bucket-project.s3-website.ap-south-1.amazonaws.com'. The status bar at the bottom right shows '18:32 21-03-2024'.

Now finally, Go to the static website, Fill the form and submit it.



The screenshot shows a web browser window with the URL 'static-website-bucket-project.s3-website.ap-south-1.amazonaws.com'. The page title is 'Contact Form'. The form has four fields: 'Name' (filled with 'Akash Saha'), 'Email' (filled with 'mr.akashsaha57@gmail.com'), 'Subject' (filled with 'Final Testing'), and 'Message' (containing the text 'Created a serverless contact form using AWS API Gateway and Lambda Function'). A blue 'Submit' button is at the bottom. The status bar at the bottom right shows '18:38 21-03-2024'.

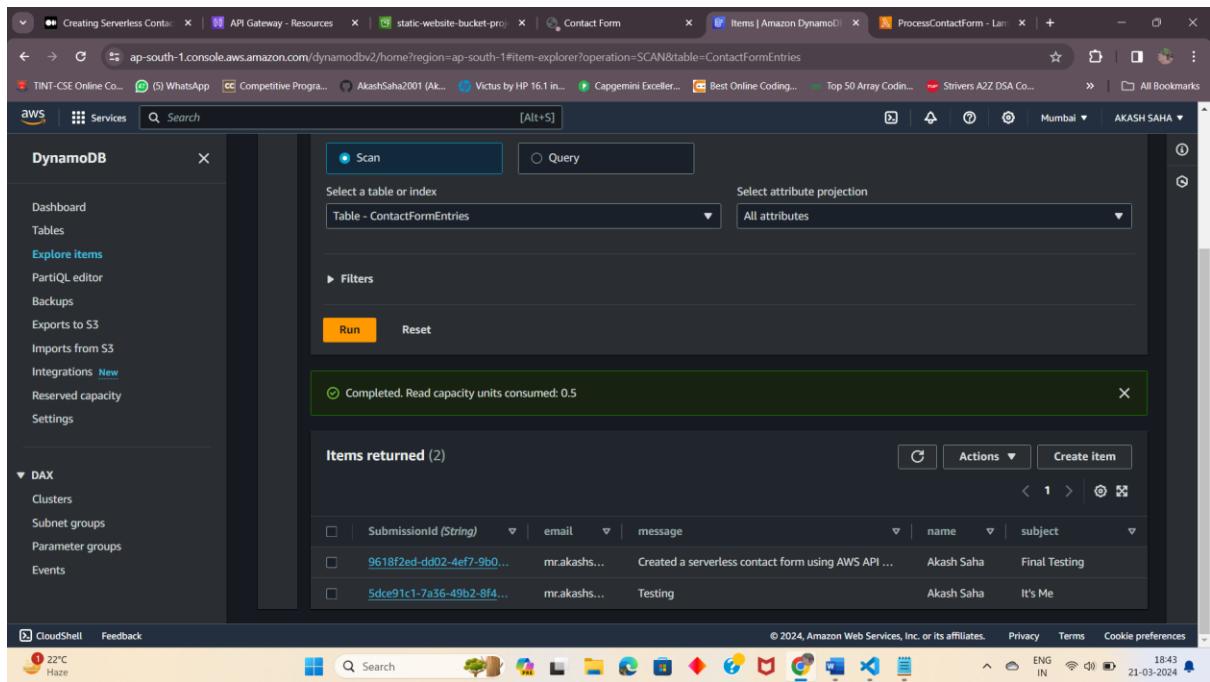


Thank You!

Your message has been submitted successfully.



Cheak the Database if it is Inserted or not.



Done.