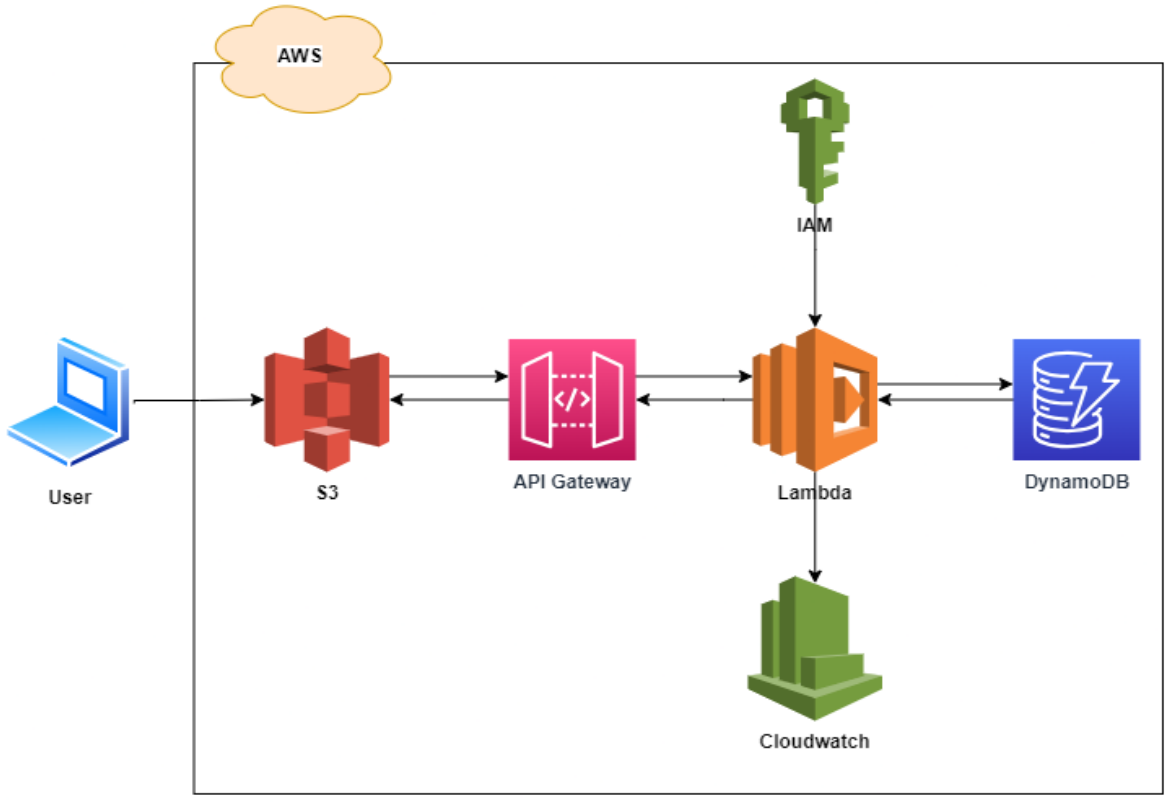
Creating a Serverless Contact Form using AWS API Gateway and Lambda

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;
}
```

---

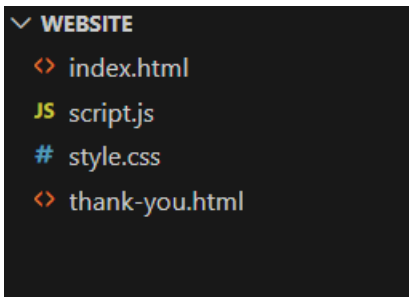
script.js ----------

```
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.');
  });
}
```

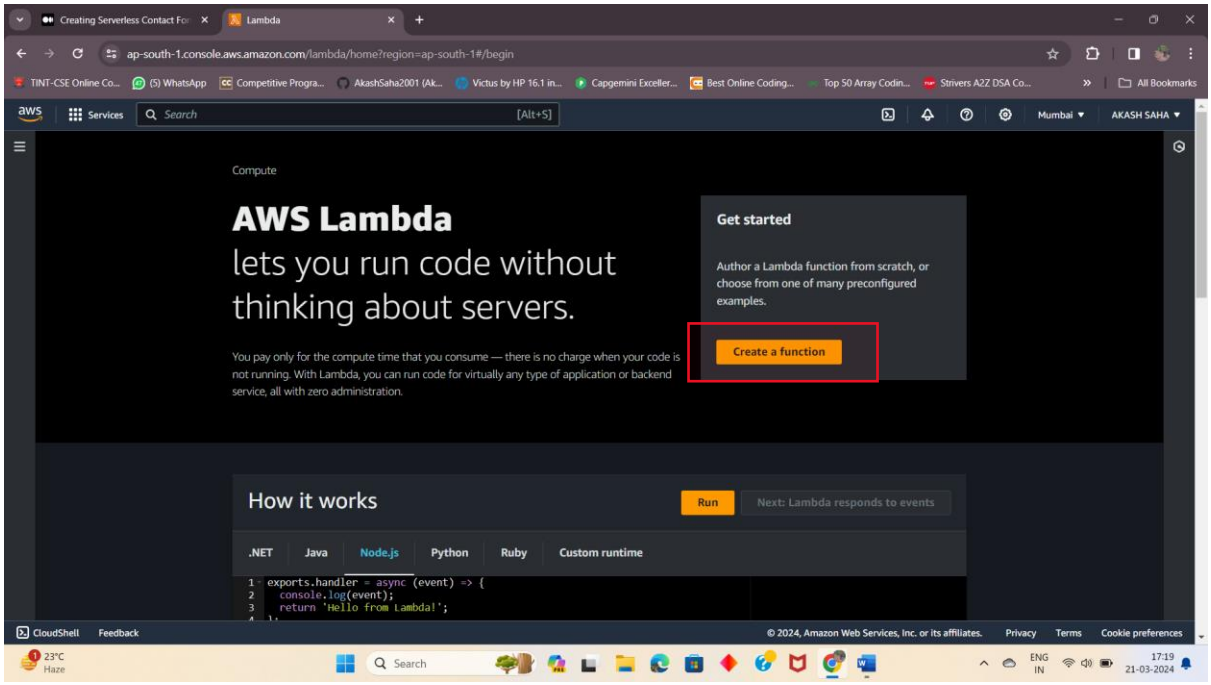
thank-you.html ----------

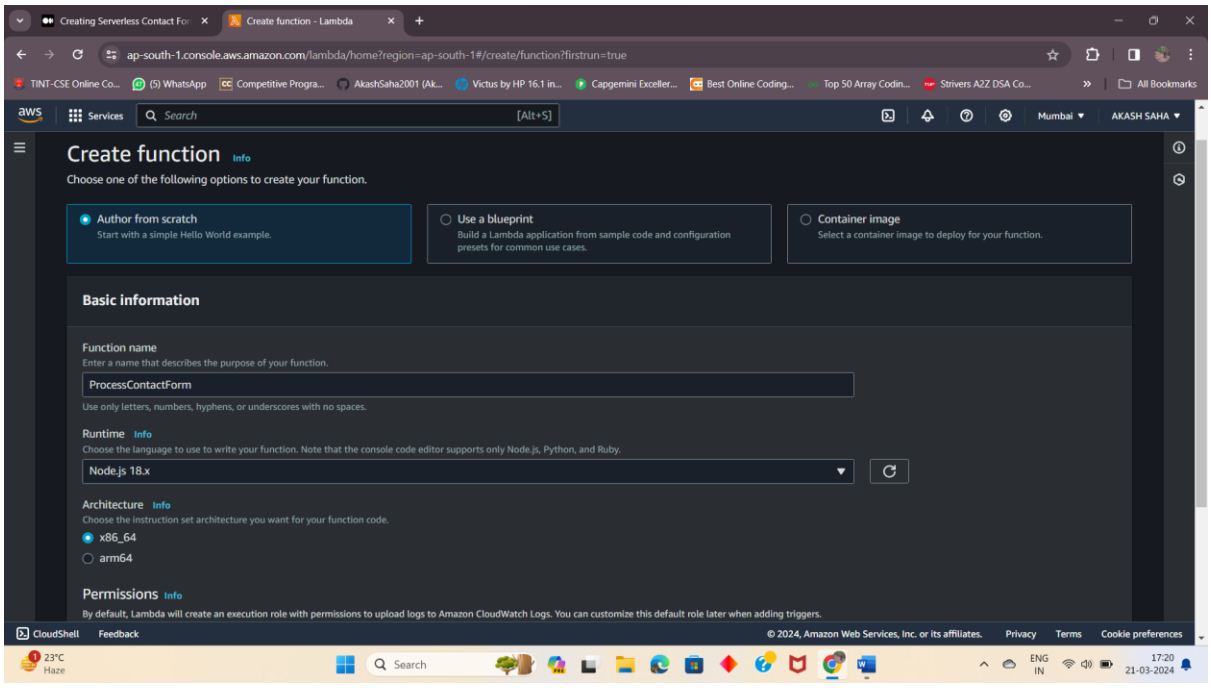
```
<!DOCTYPE html>
<html>
<head>
  <title>Thank You</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <div class="container">
    <h1>Thank You!</h1>
    <p>Your message has been submitted successfully.</p>
  </div>
</body>
</html>
```

The structure will be this :



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();
}
```

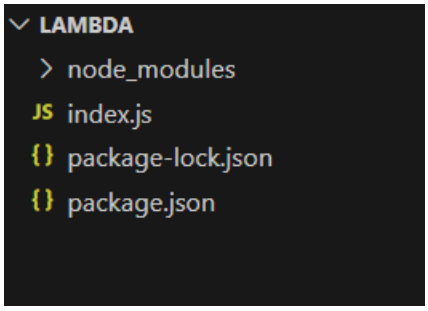
package.json ----------

```
{
  "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.

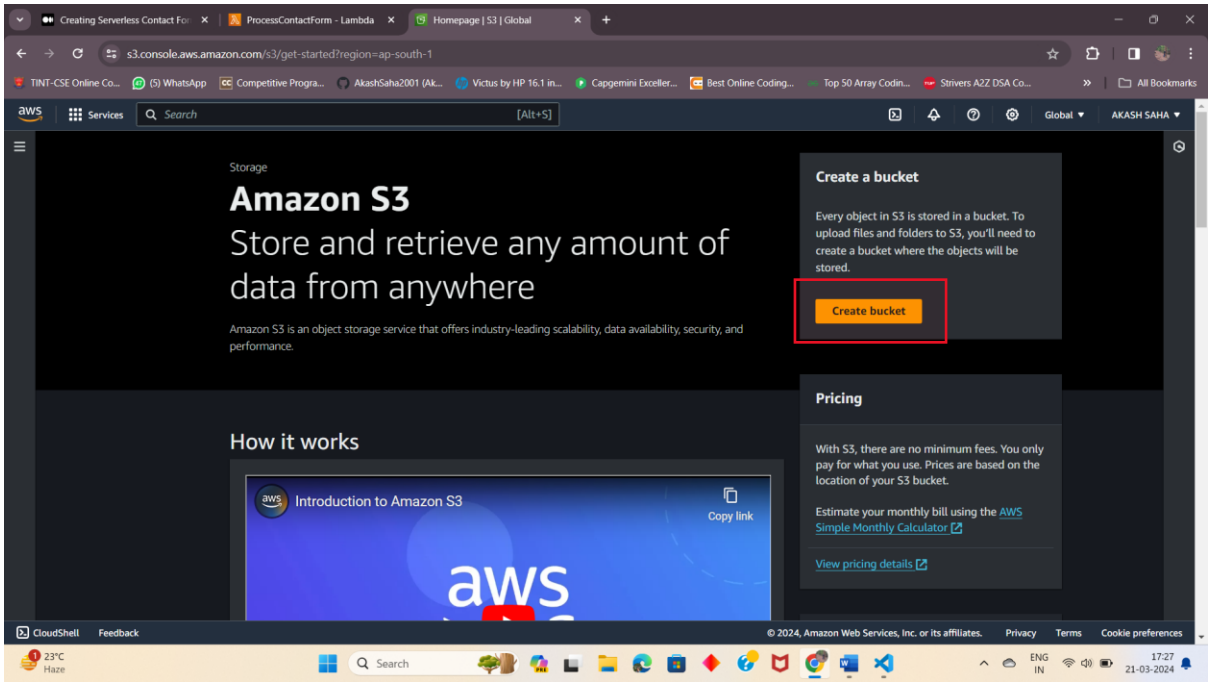
npm install

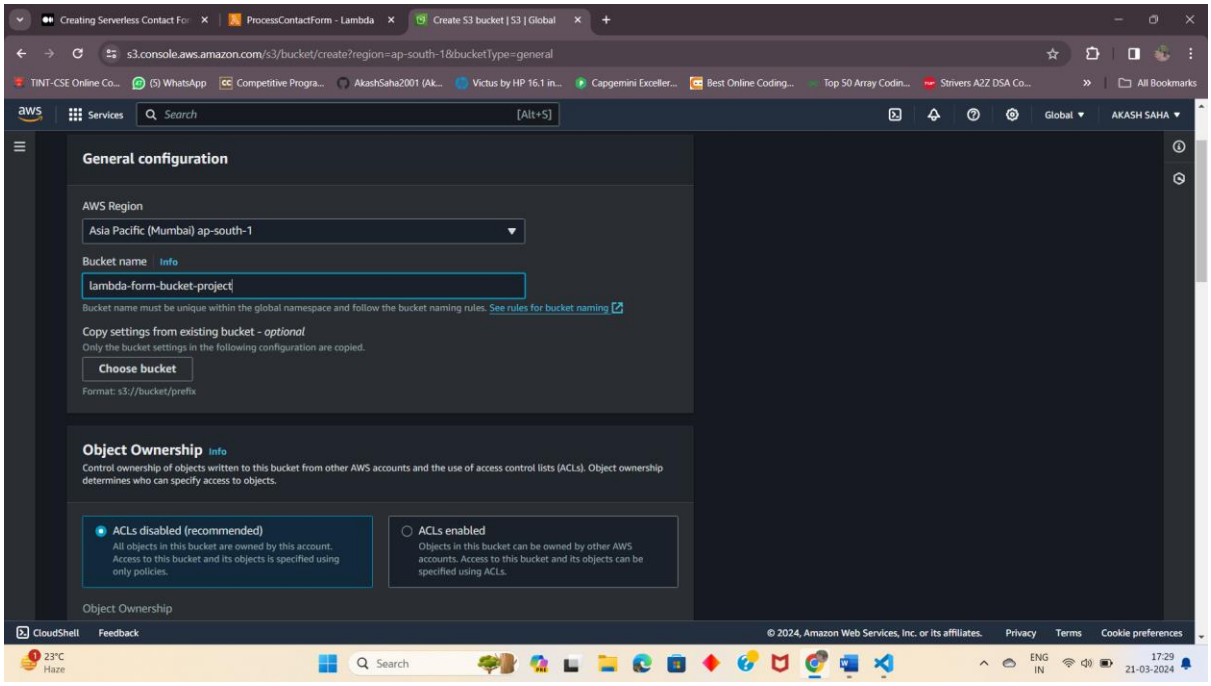
This will be the structure :

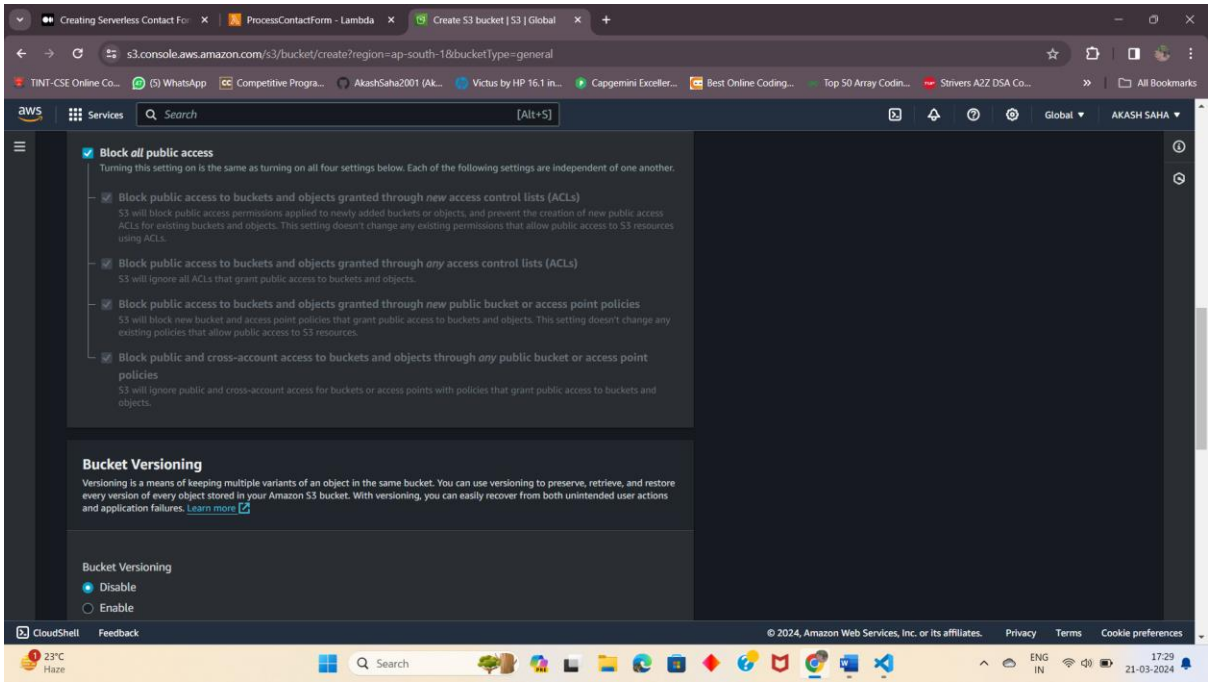


Then zip the folder. [lambda.zip]

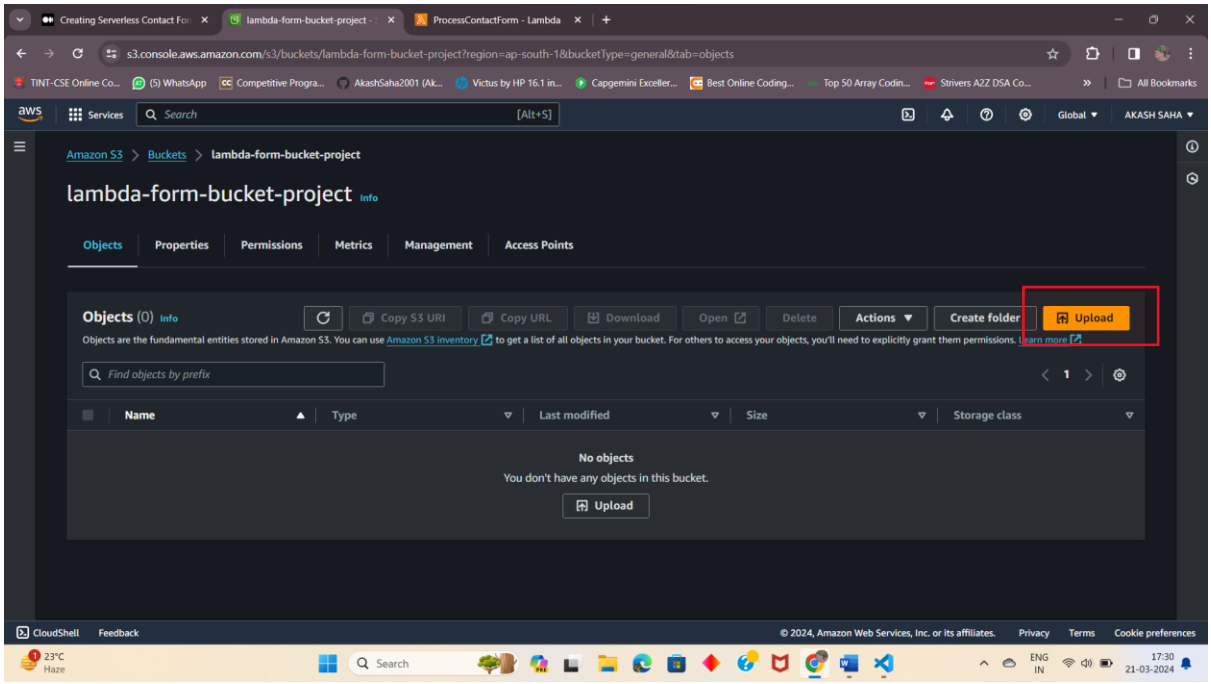
Activity 2: Create a Bucket

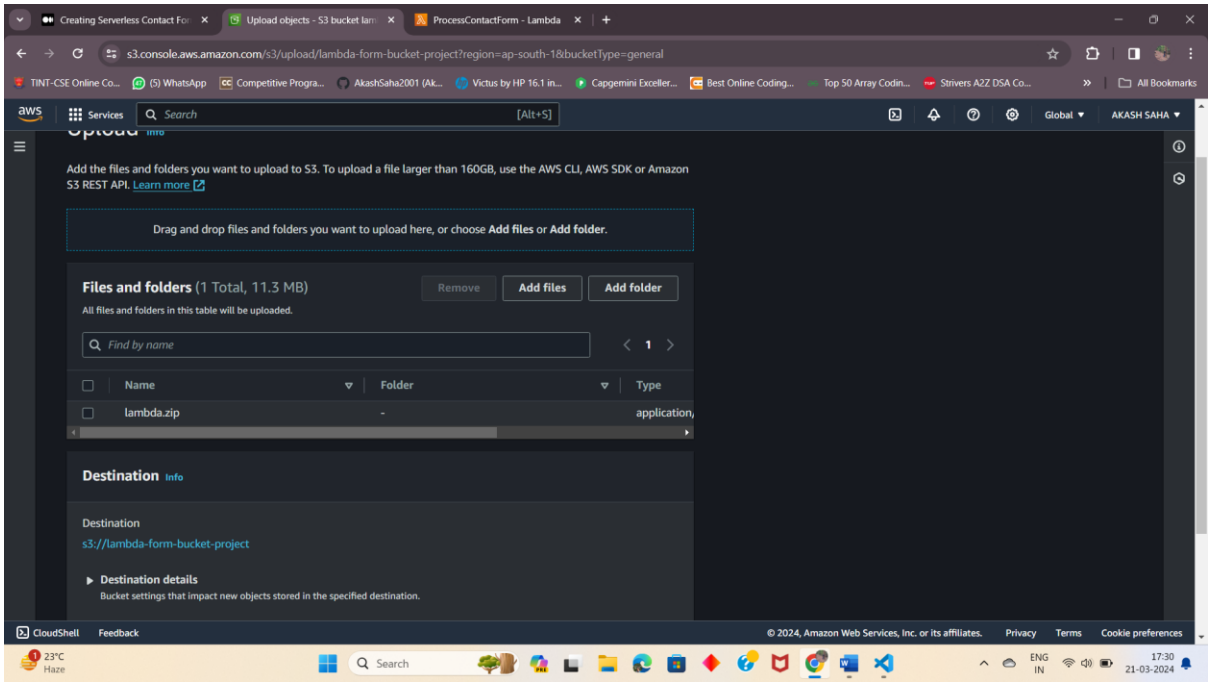






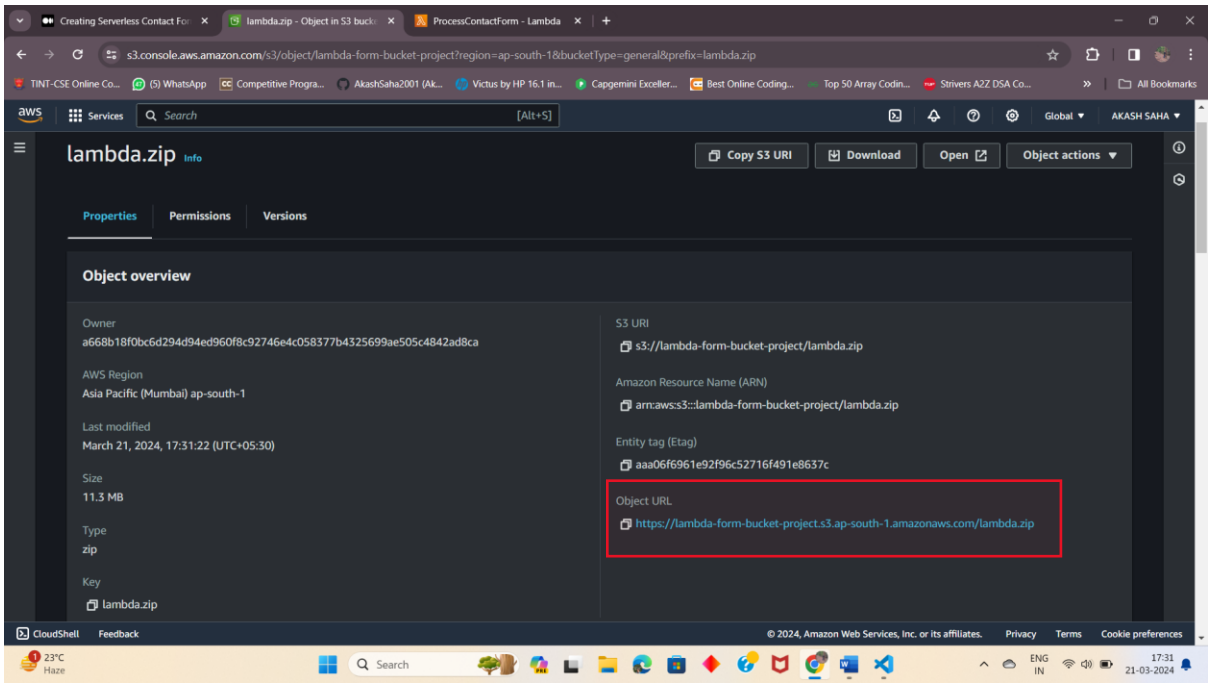
Upload the lambda.zip here.



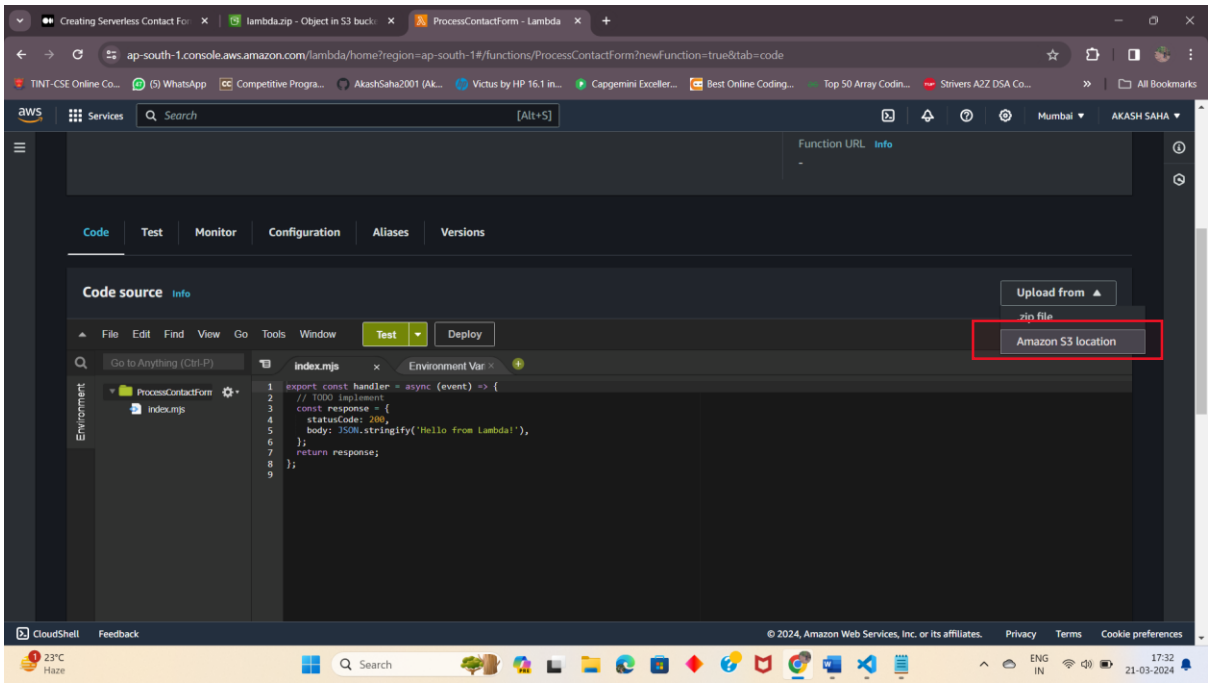


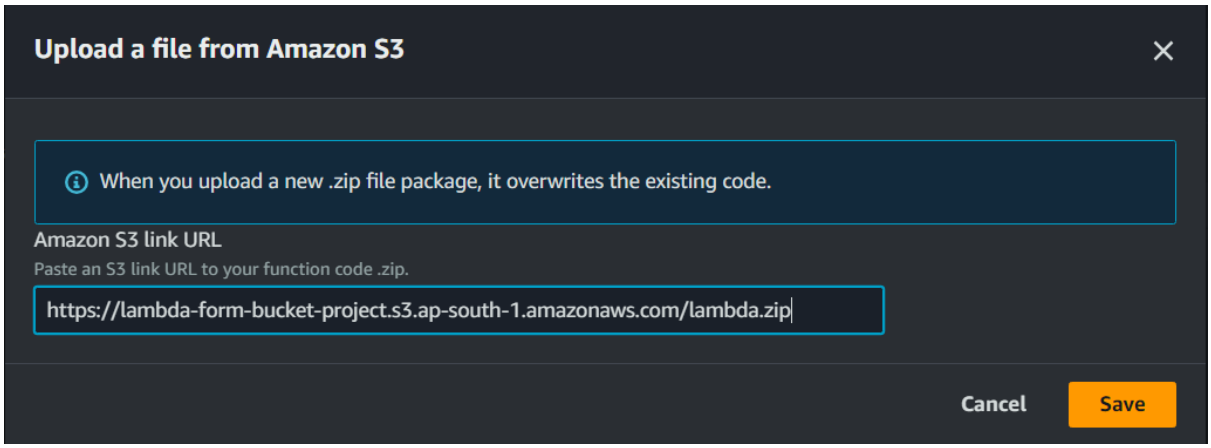
Upload the file.

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

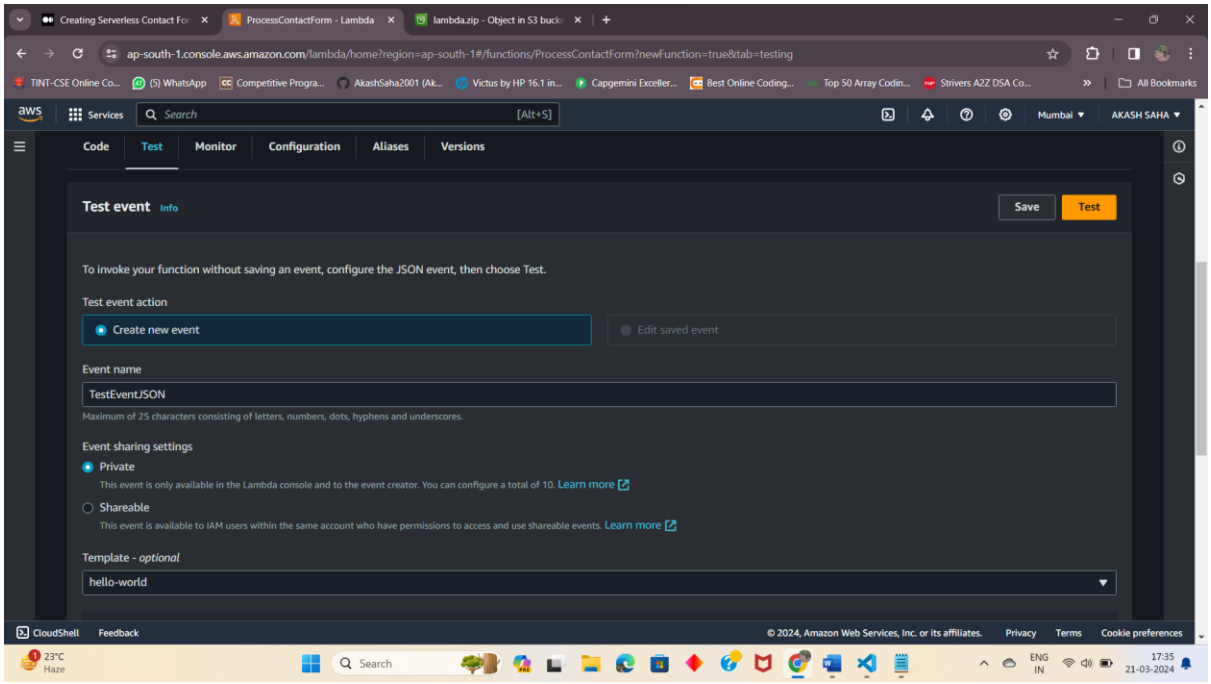


Now go to the lambda function and upload the zip by attaching the S3 location here.





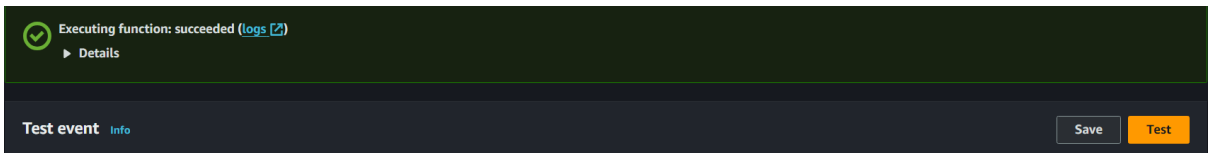
Now go to the Test



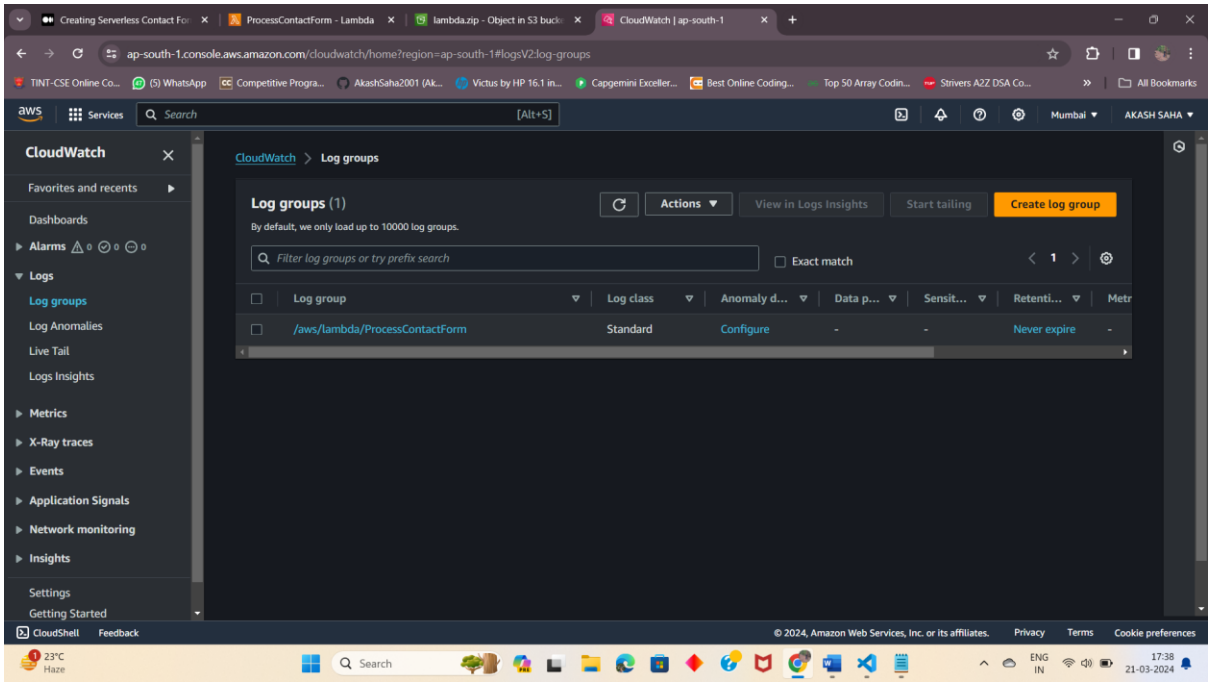
Edit the code :

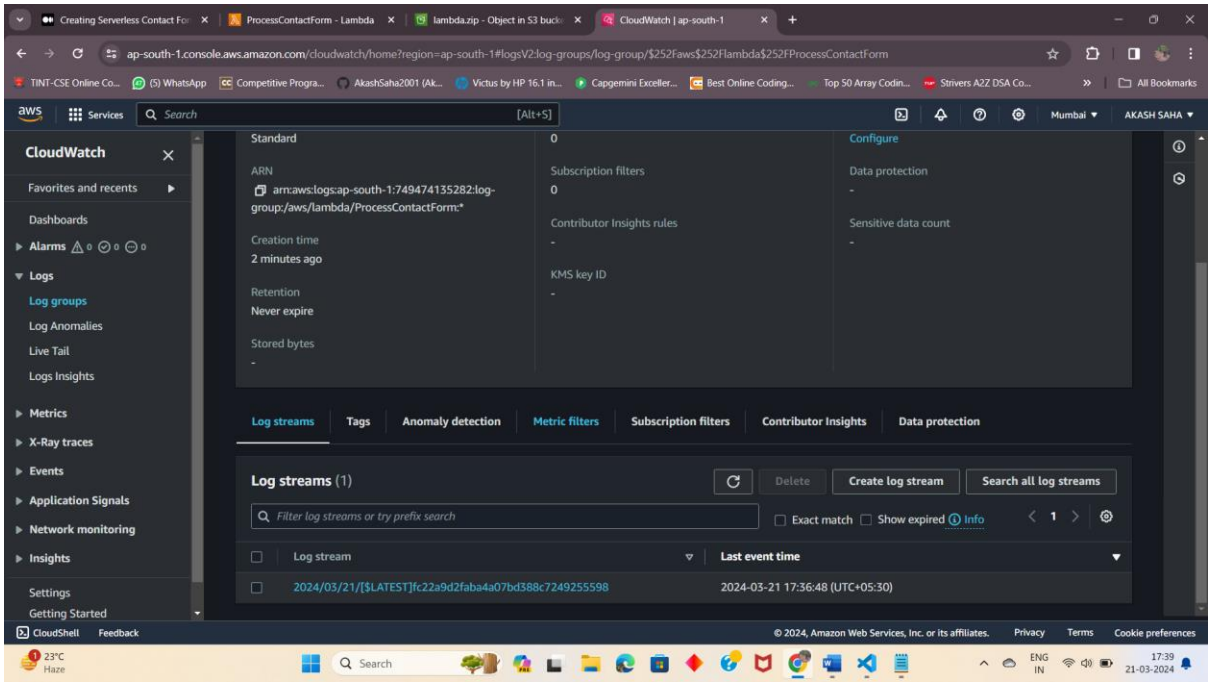

```
{
  "name": "Akash Saha",
  "email": "mr.akashsaha574@gmail.com",
  "subject": "It's Me",
  "message": "Testing"
}
```

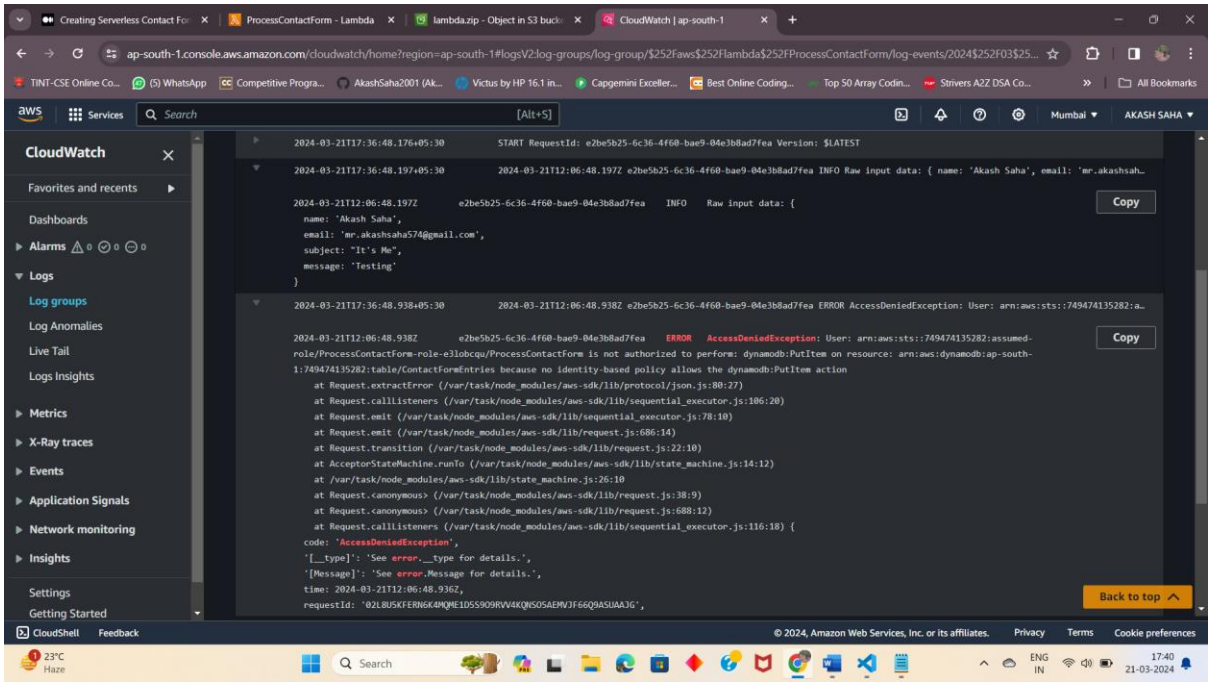
Click on the Test to check if it is getting successfully executed or not.



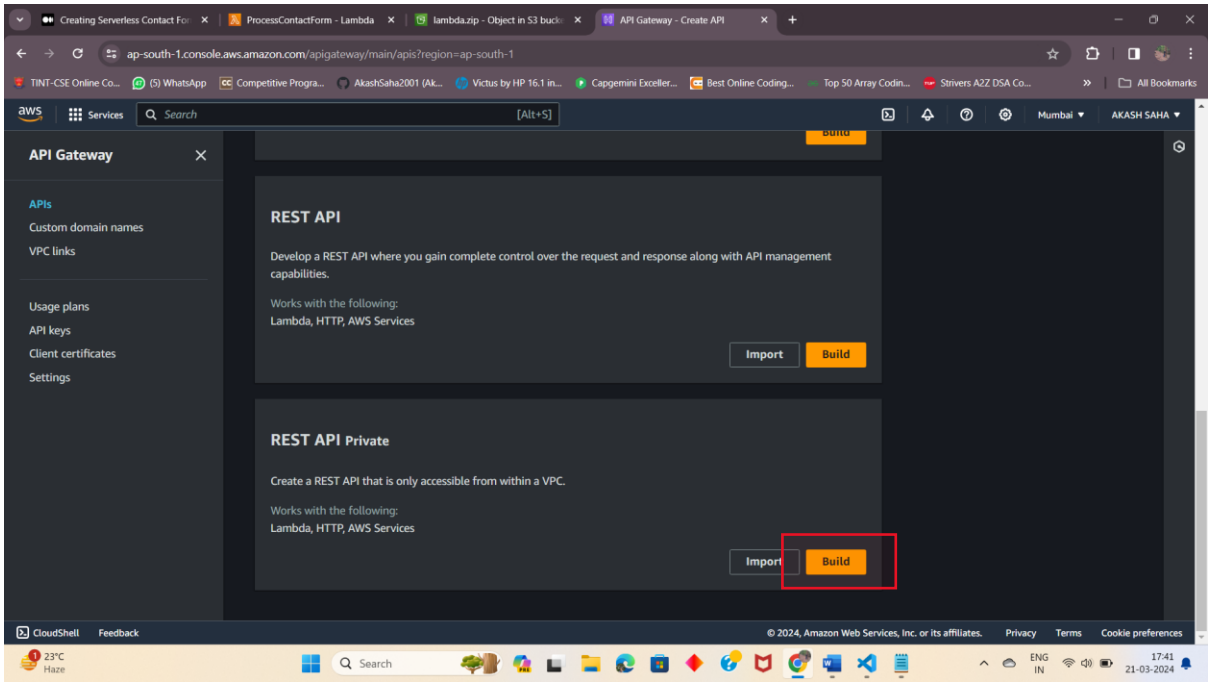
Activity 3: CloudWatch (Optional)

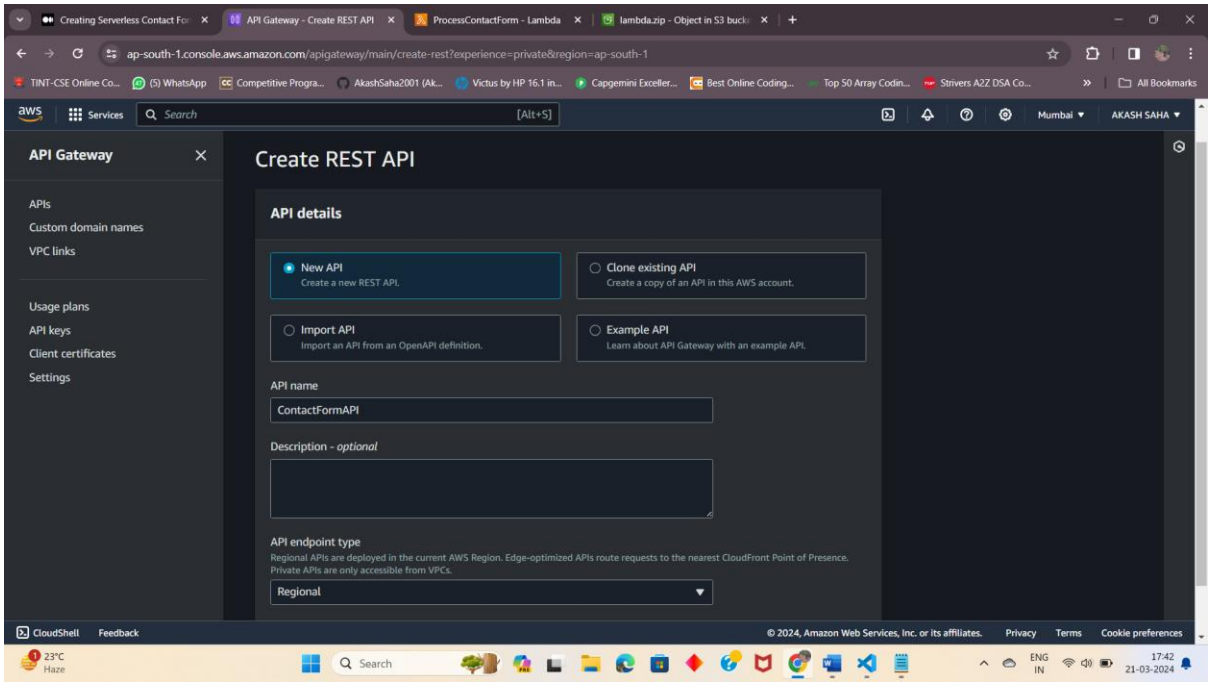




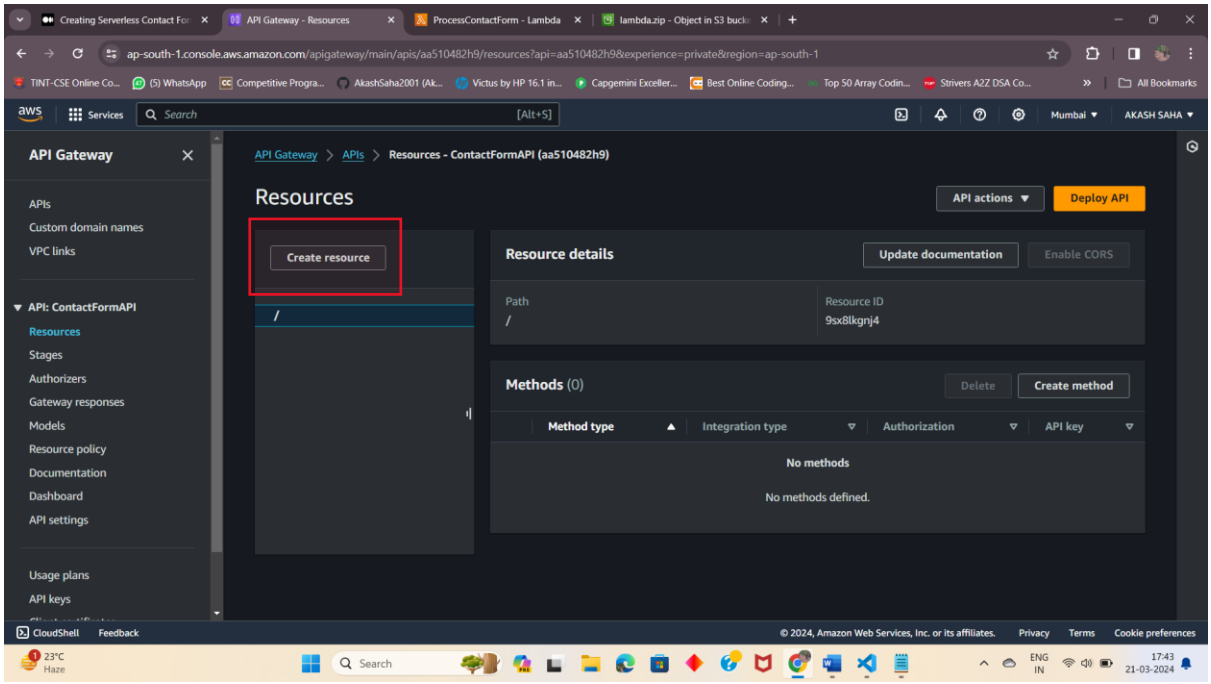


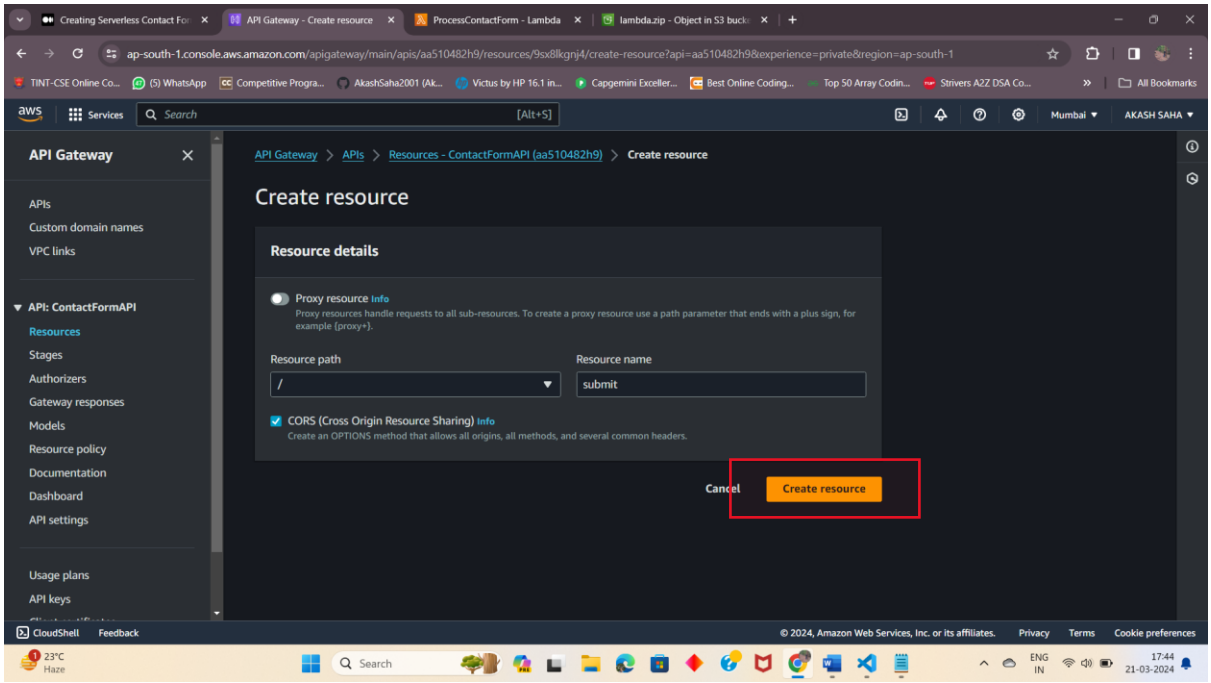
Activity 4: Crating API gateway



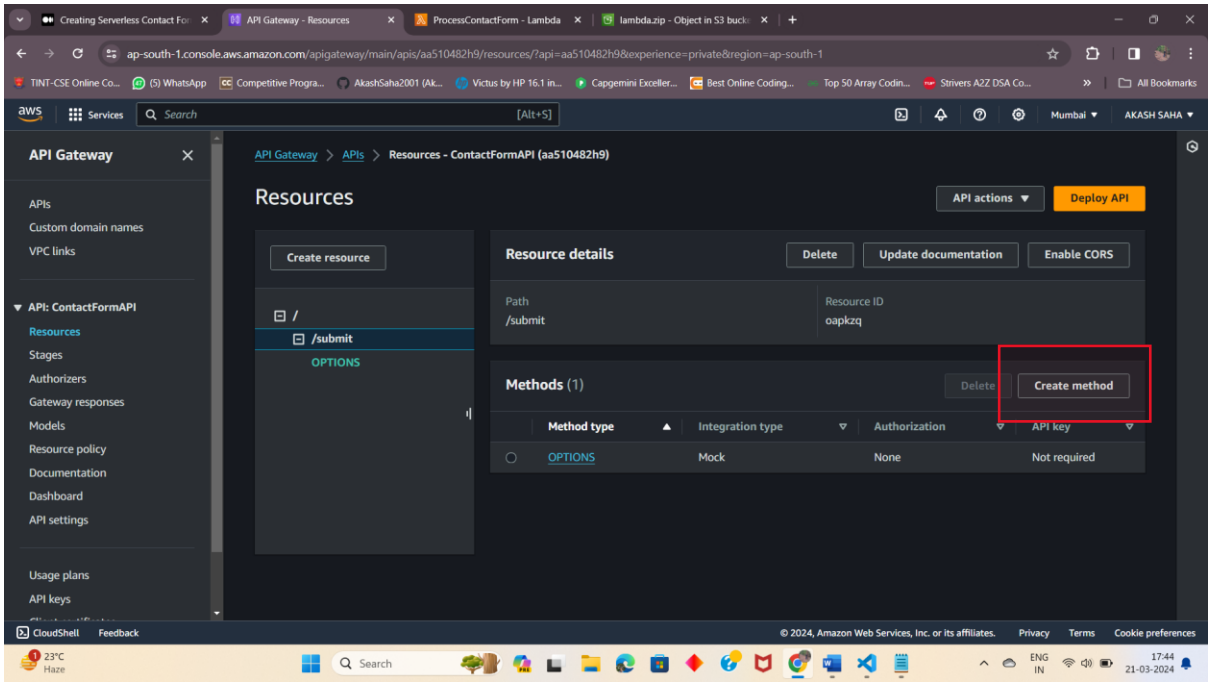


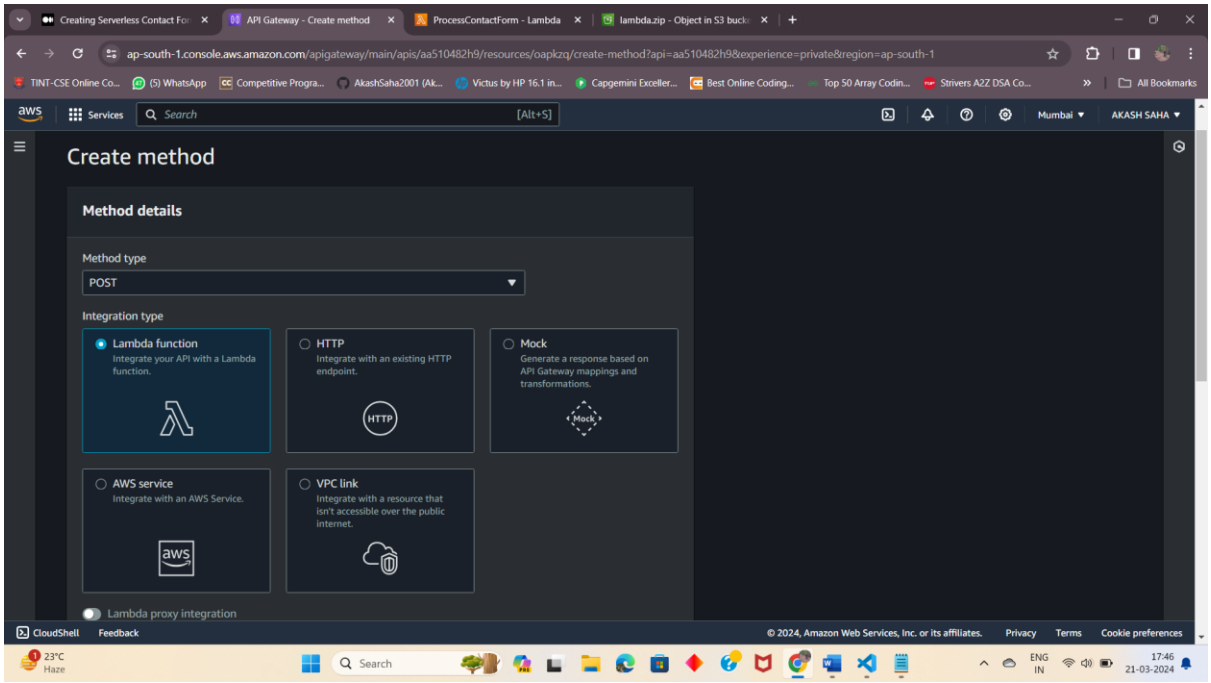
Create a resource.

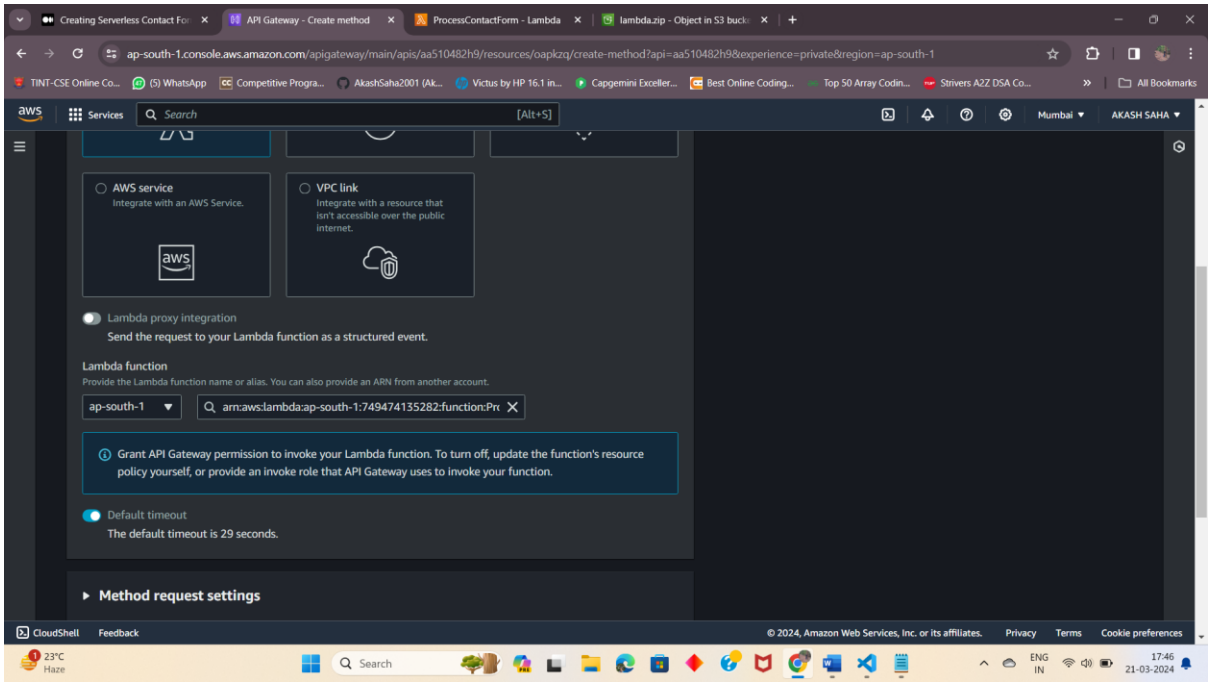




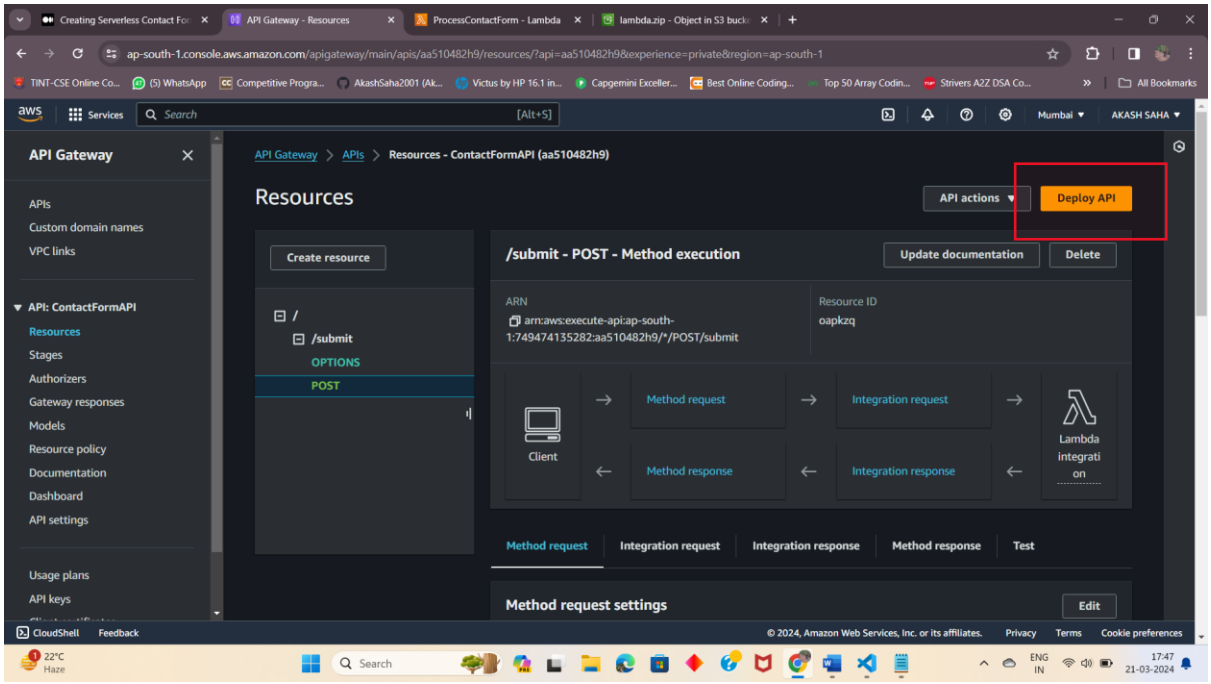
Create a method

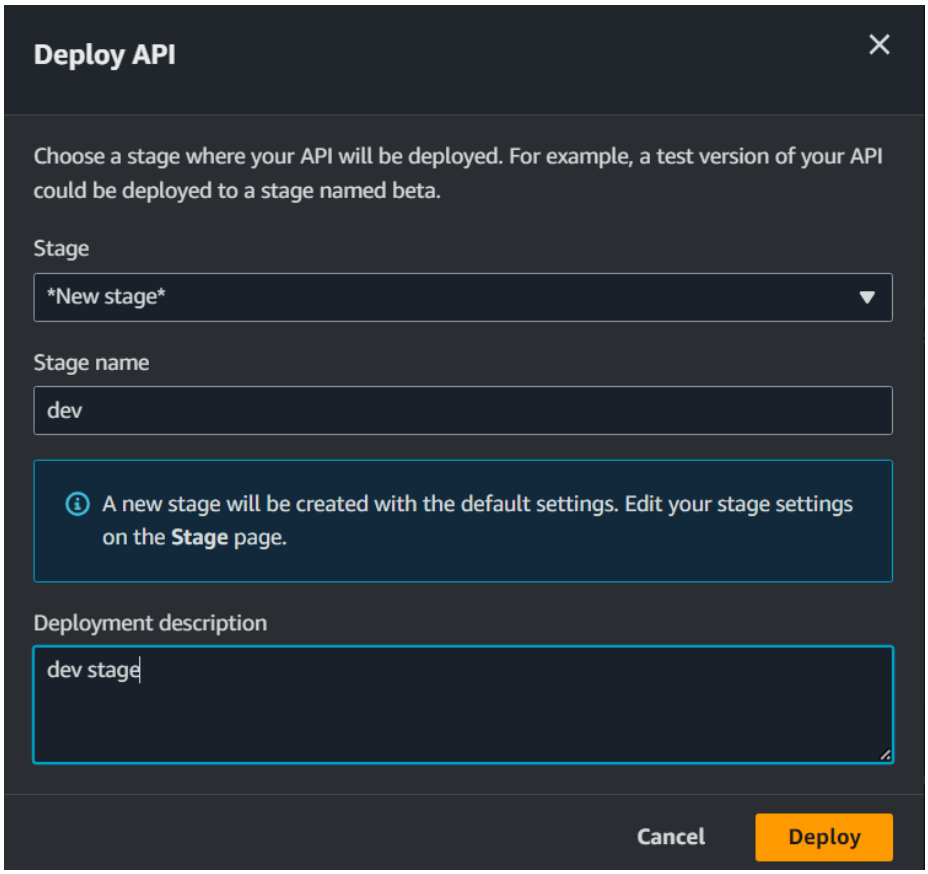



Creating Serverless Contact For
API Gateway - Create method
ProcessContactForm - Lambda
lambda.zip - Object in S3 buck
ap-south-1.console.aws.amazon.com/apigateway/main/apis/aa510482h9/resources/oapkzq/create-method?api=aa510482h9&experience=private&region=ap-south-1
Services
Search
[Alt+S]
Mumbai
AKASH SAHA
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
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
23°C
Haze
ENG
IN
17:46
21-03-2024

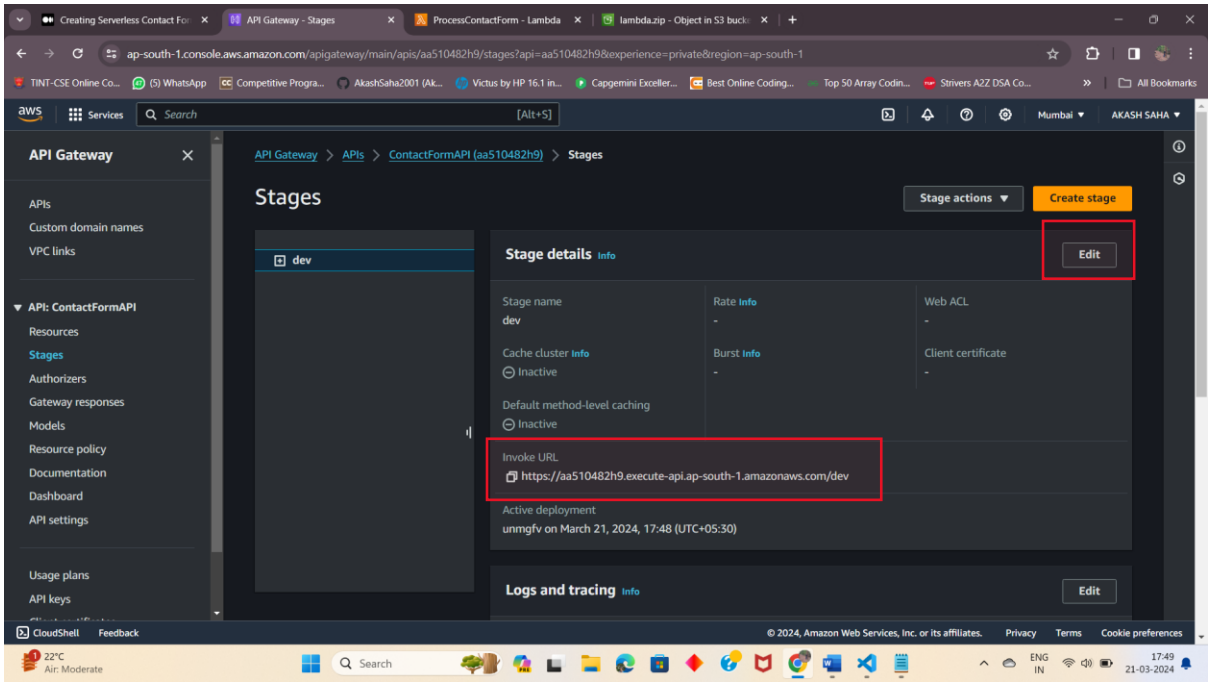

Creating Serverless Contact For
API Gateway - Create method
ProcessContactForm - Lambda
lambda.zip - Object in S3 buck
ap-south-1.console.aws.amazon.com/apigateway/main/apis/aa510482h9/resources/oapkzq/create-method?api=aa510482h9&experience=private&region=ap-south-1
Services
Search
[Alt+S]
Mumbai
AKASH SAHA
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
arn:aws:lambda:ap-south-1:749474135282:function:Pr
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.

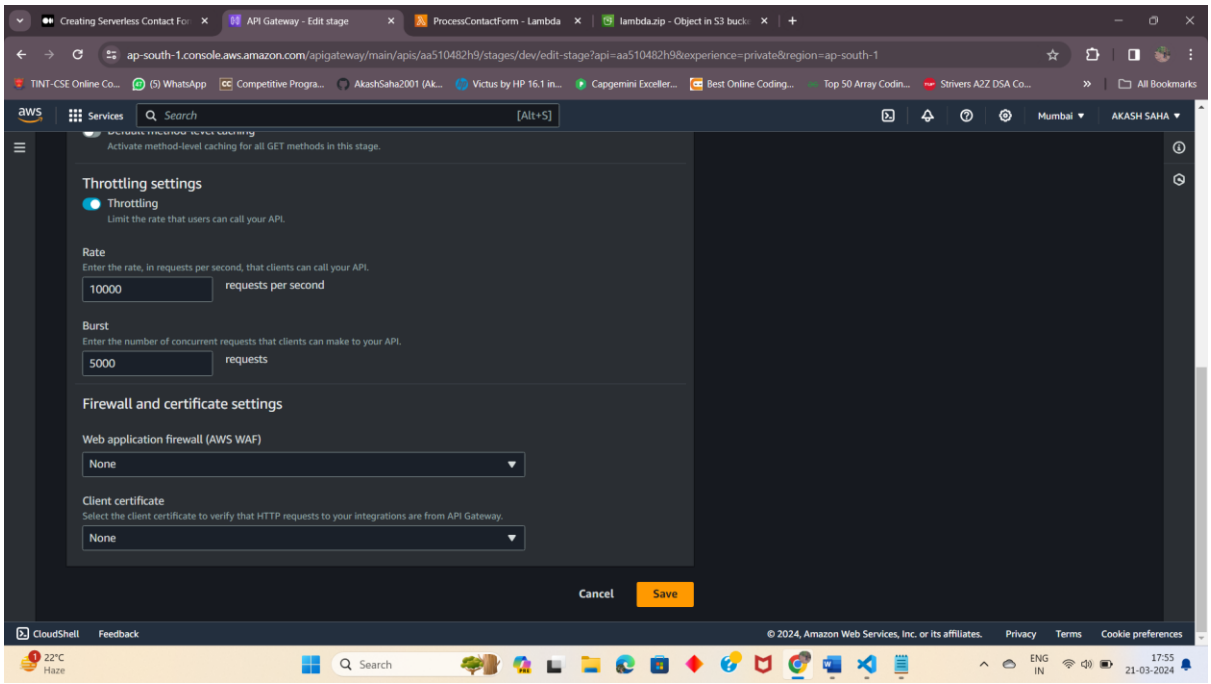




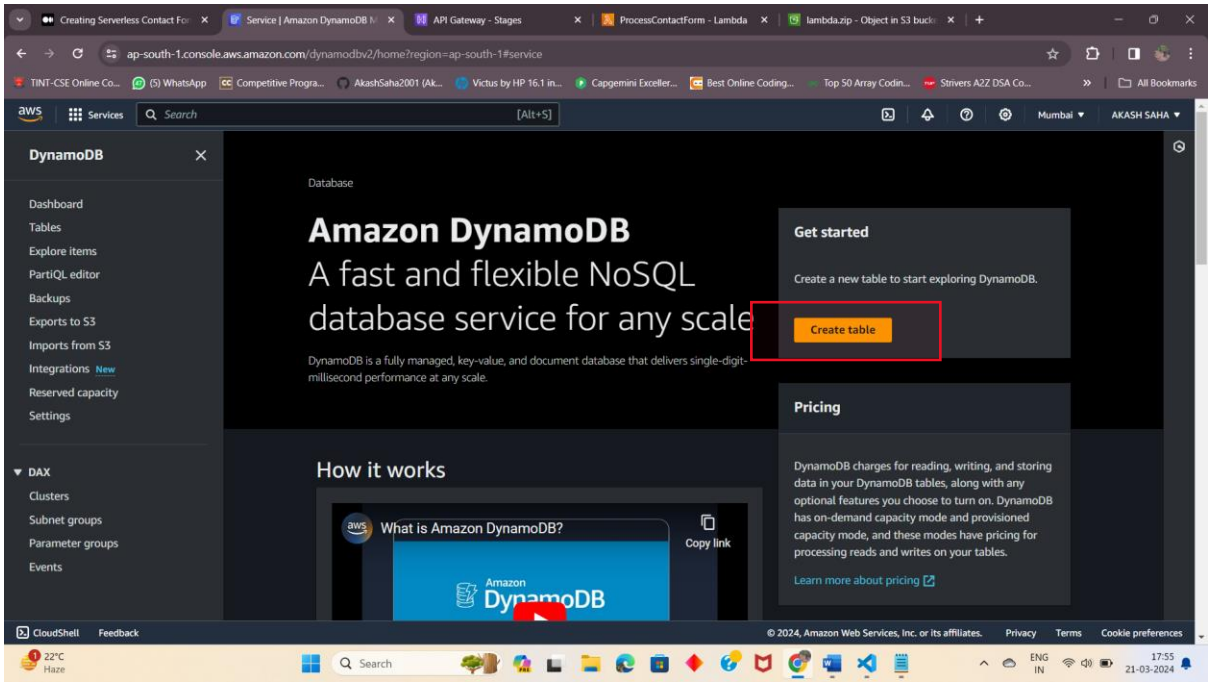
Copy the invoke URL

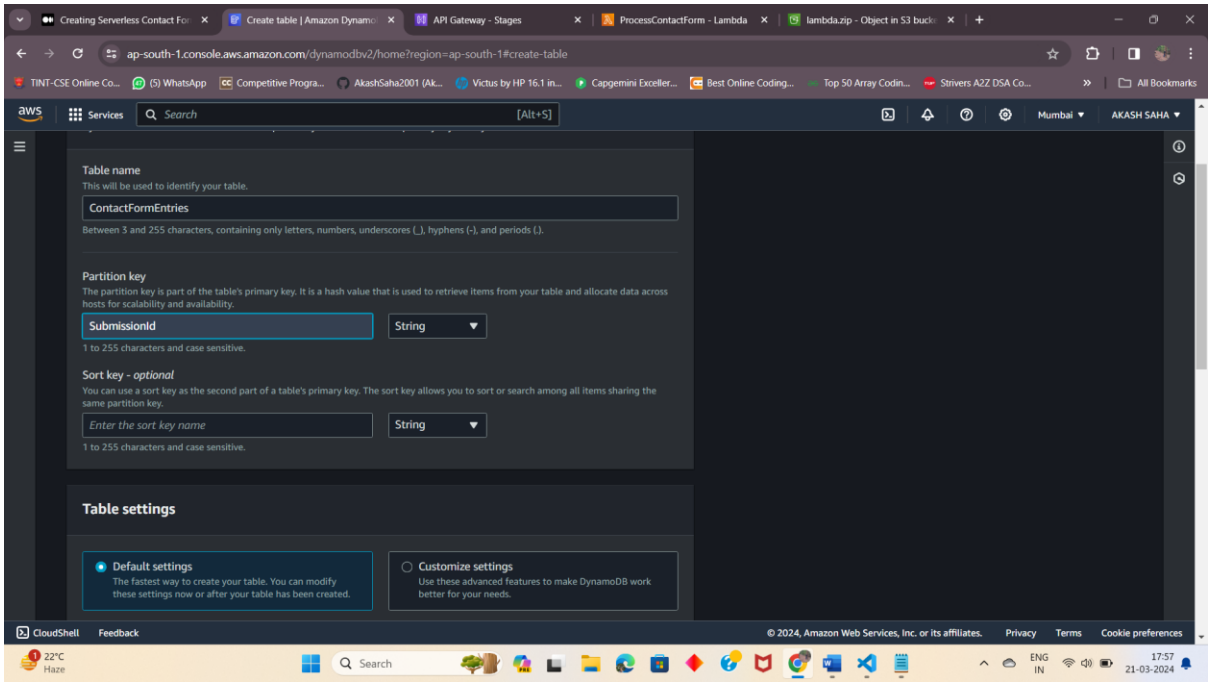


Go to the Edit and set the following settings.



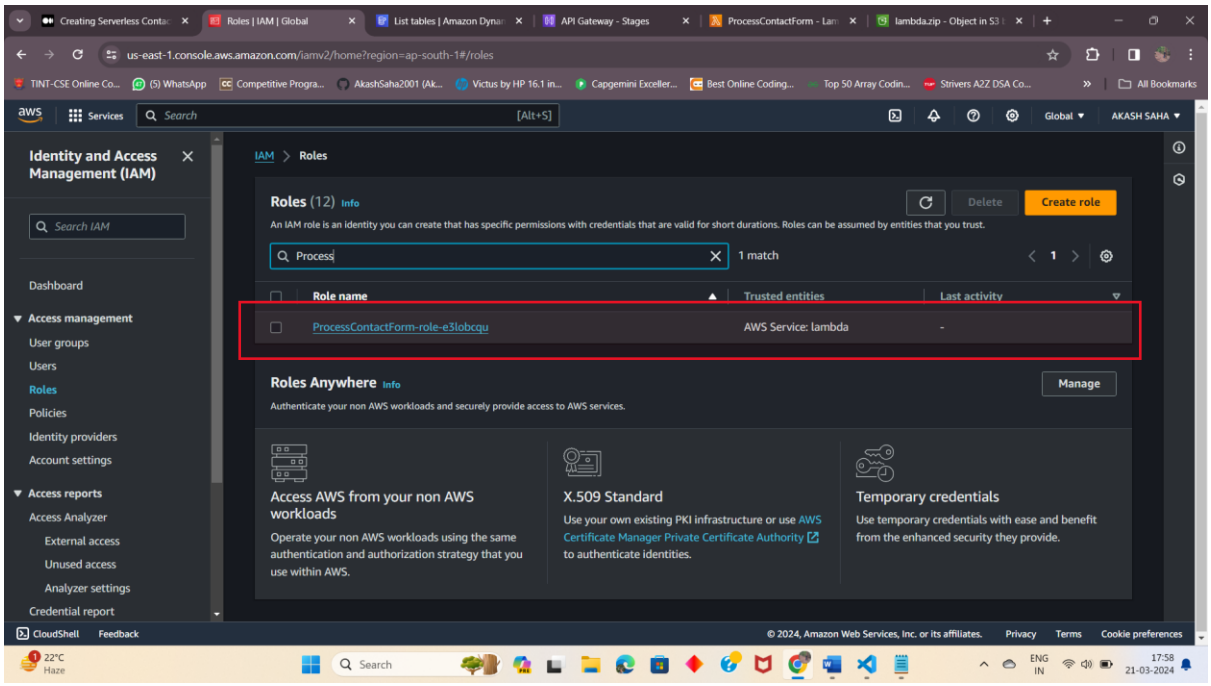
Activity 5: Create a table

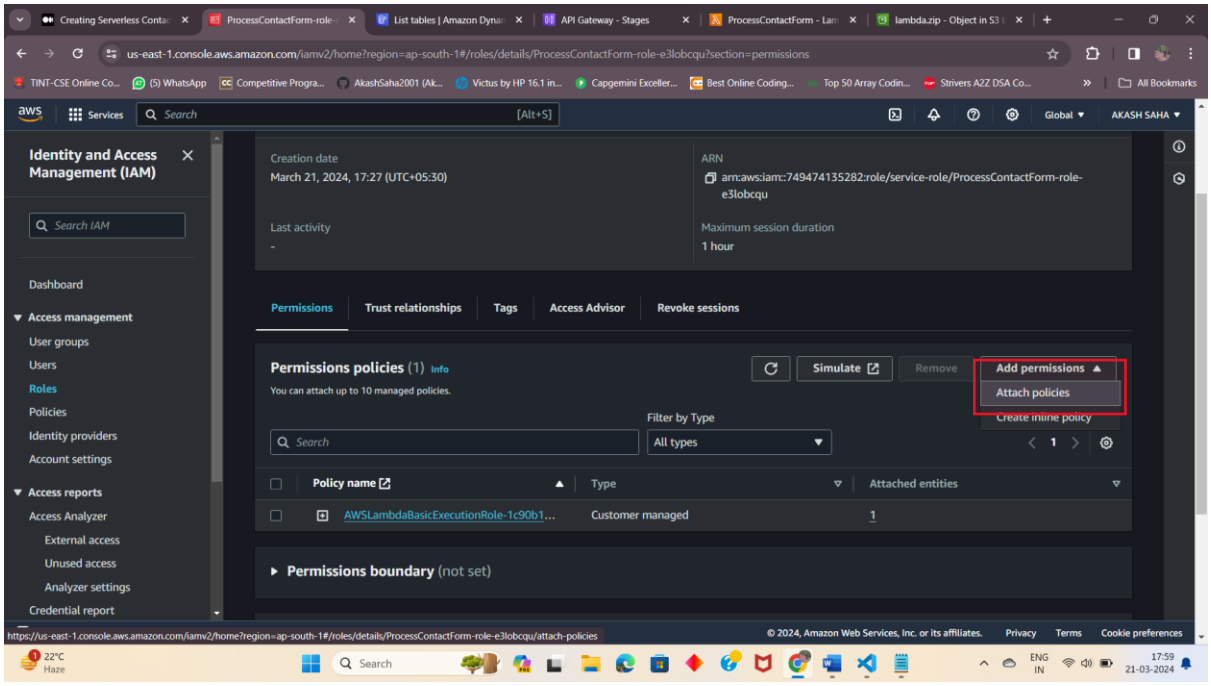


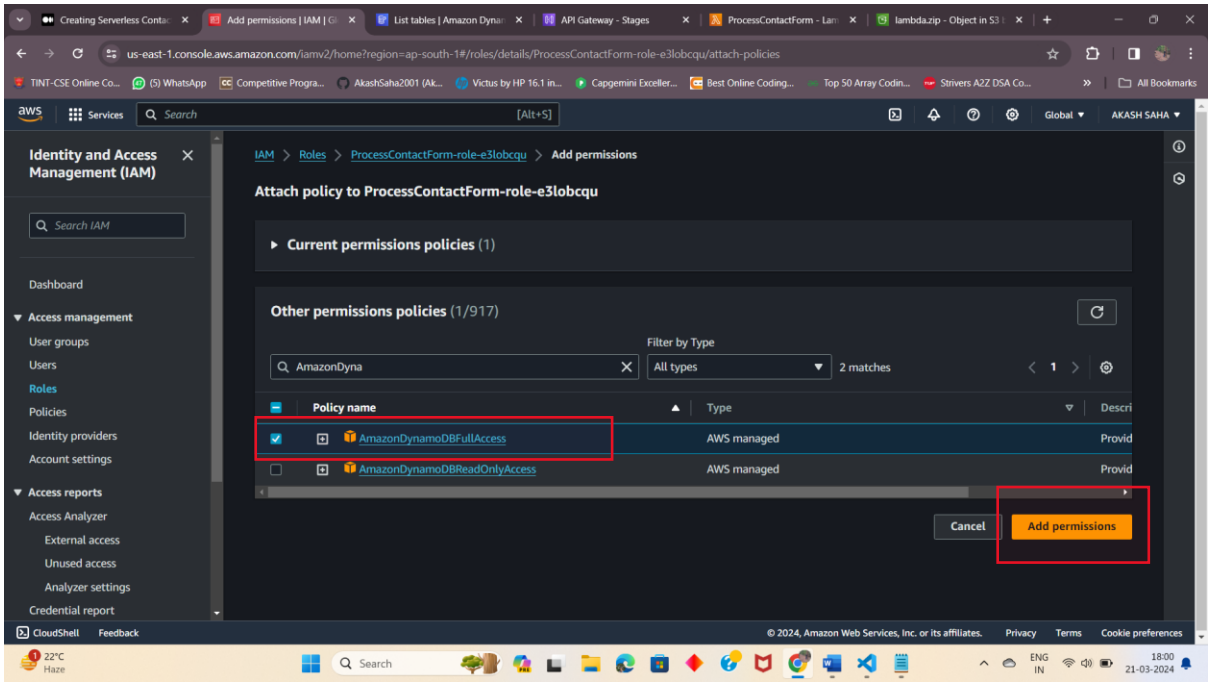


From the API Gateway role created when adding a permission to it to invoke a Lambda function, select it to add a DynamoDB policy that can allow us to put new items into the database.

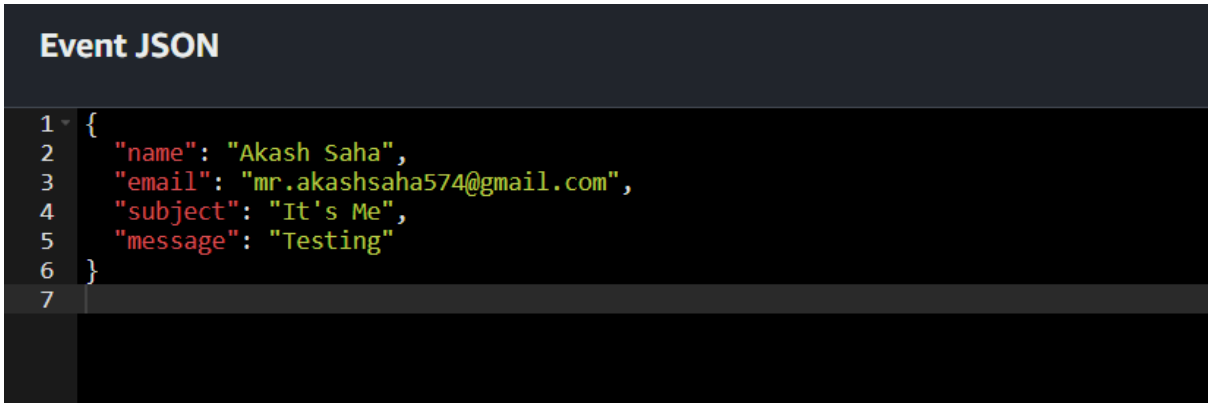
Go to the IAM Roles, search for your created AWS lambda function (ProcessContactForm)

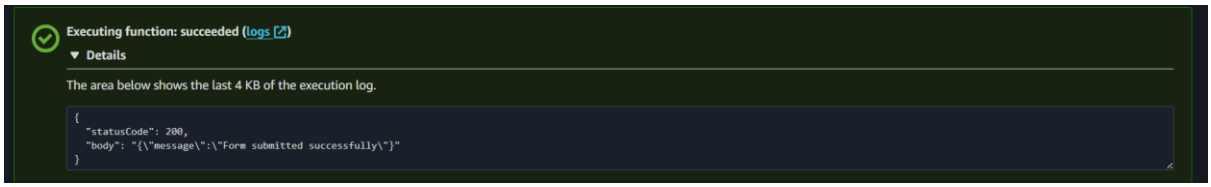




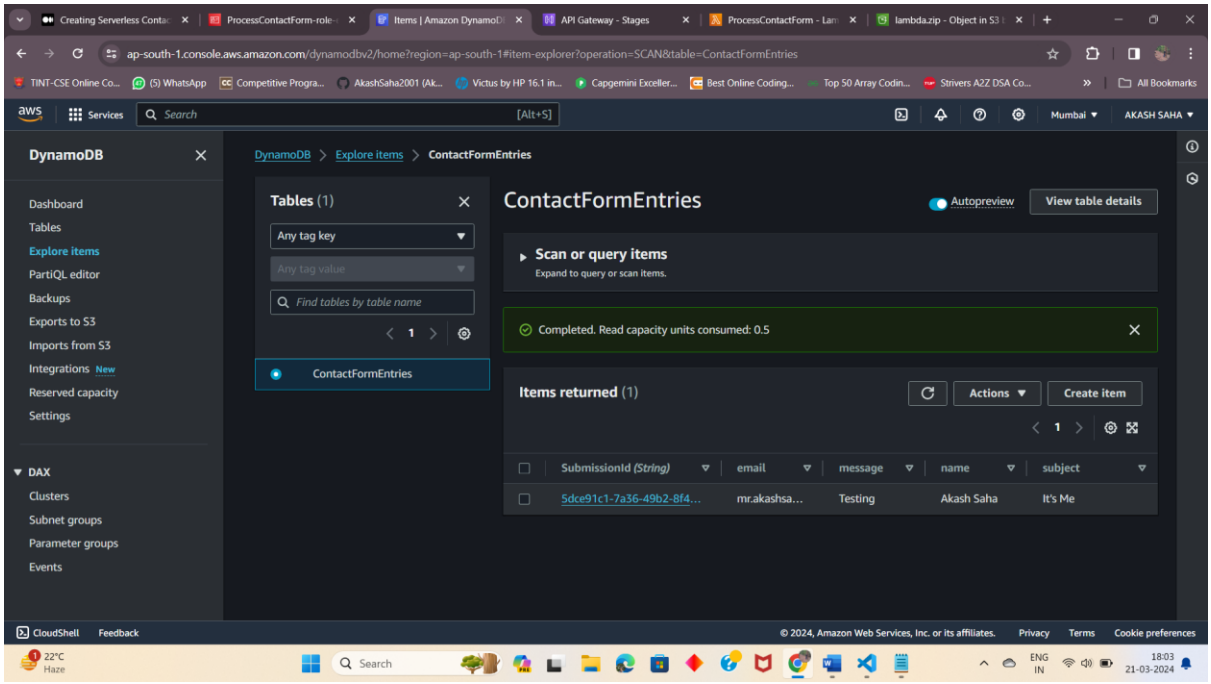


Test again your Lambda function with the test event you previously created. Now you'll see how the data is inserted into the DynamoDB table.





Check your database if it is inserted or not.



Upload the website to S3:

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

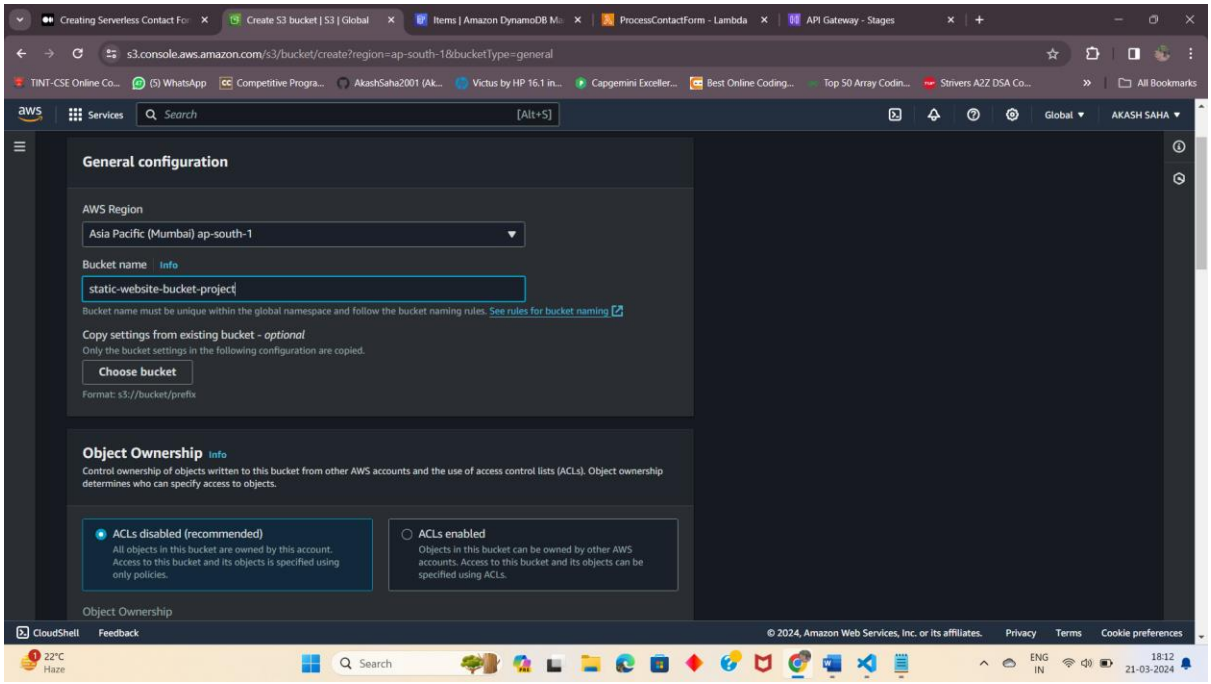
- Replace the `fetch` line of code in your `script.js` file with the Invoke URL of your API Gateway adding at the end the `/submit` endpoint.

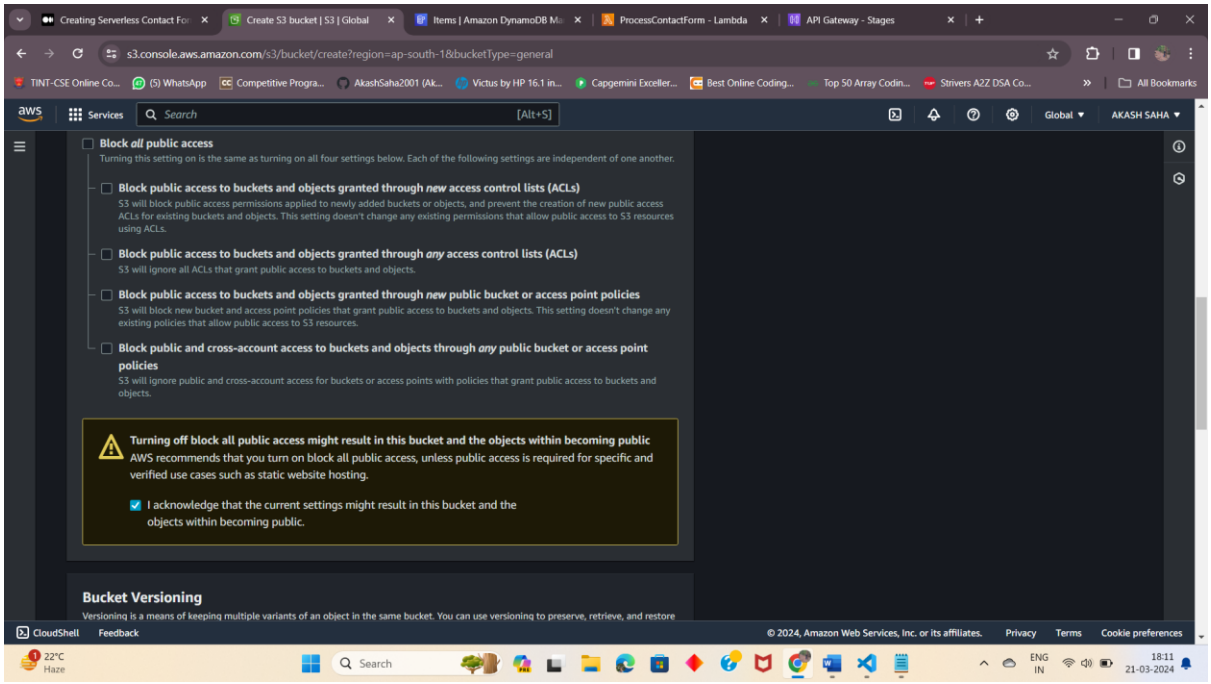
```
// Before
fetch('your-api-gateway-url', {

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

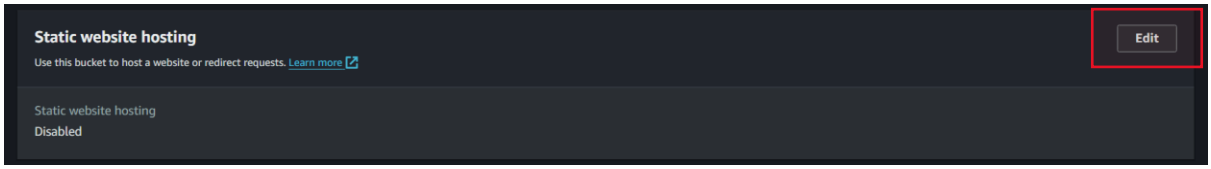
```
fetch('https://aa510482h9.execute-api.ap-south-1.amazonaws.com/dev/submit', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(formData)
})
```

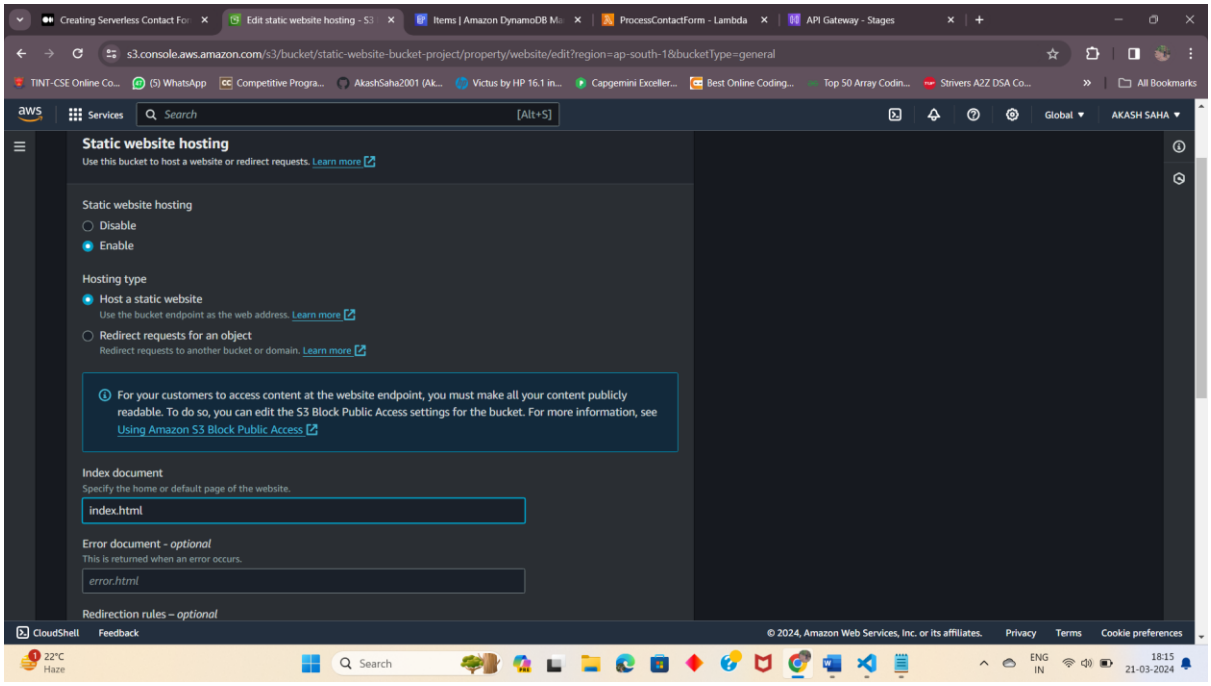
Create a new S3 bucket



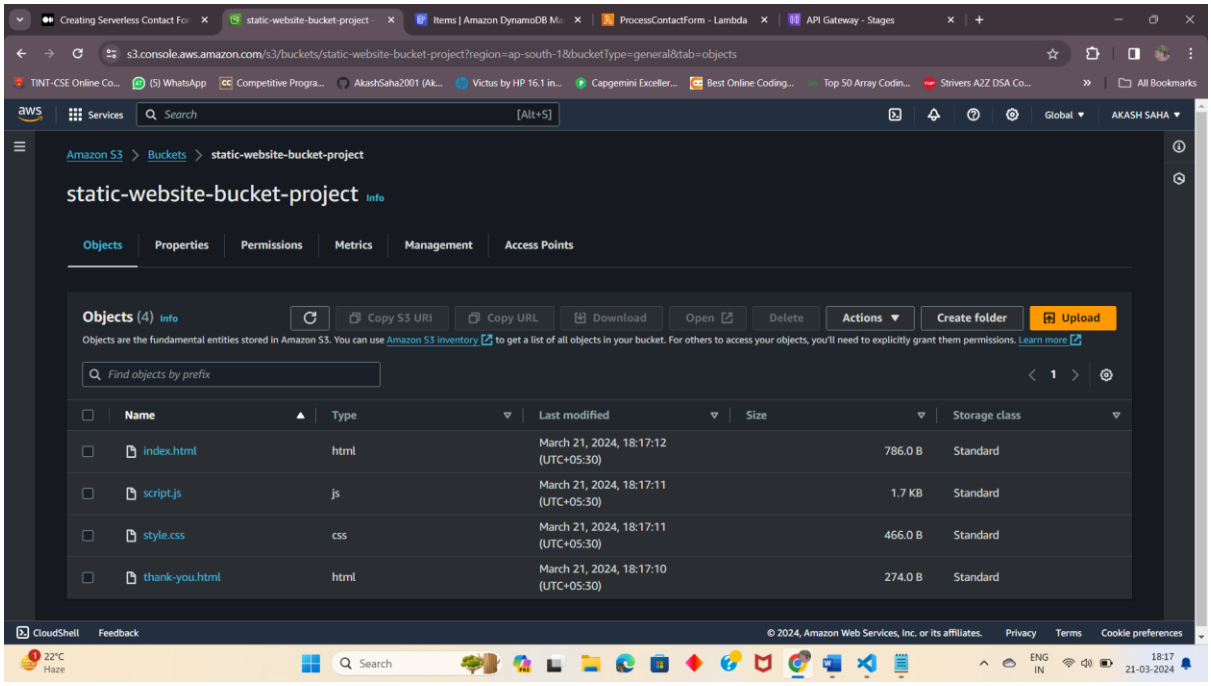


Now select the bucket and go to the Properties section scroll down to the last and search for Static website hosting.

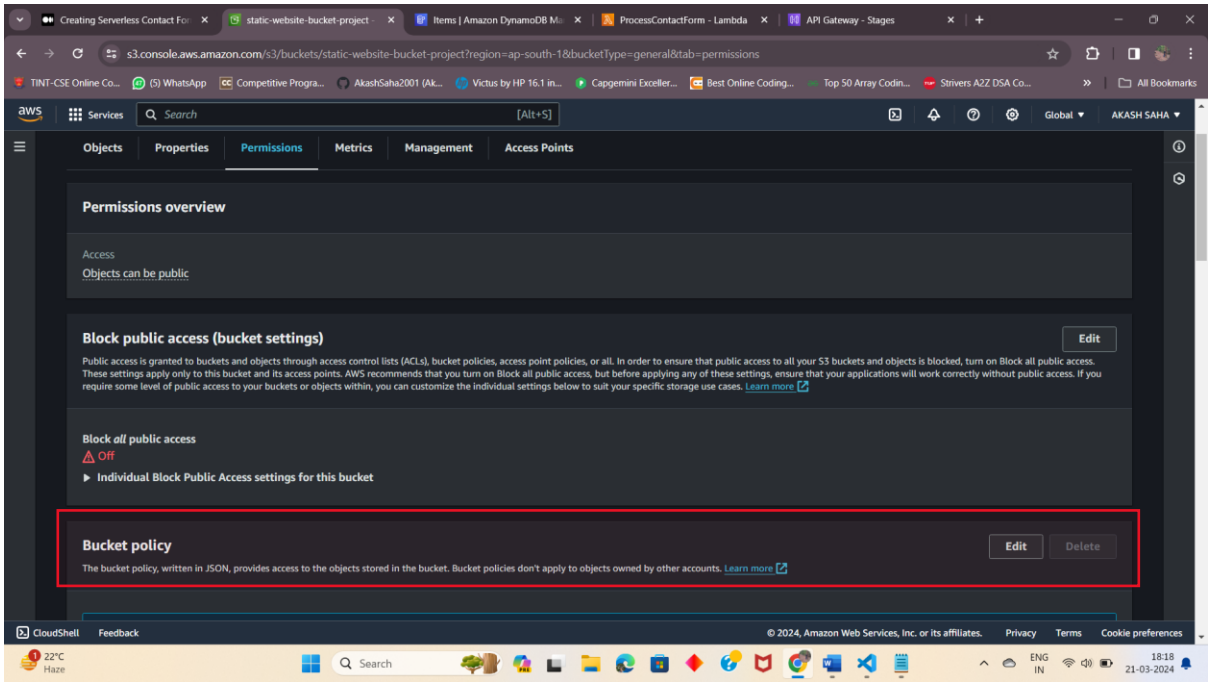




Upload all the files.



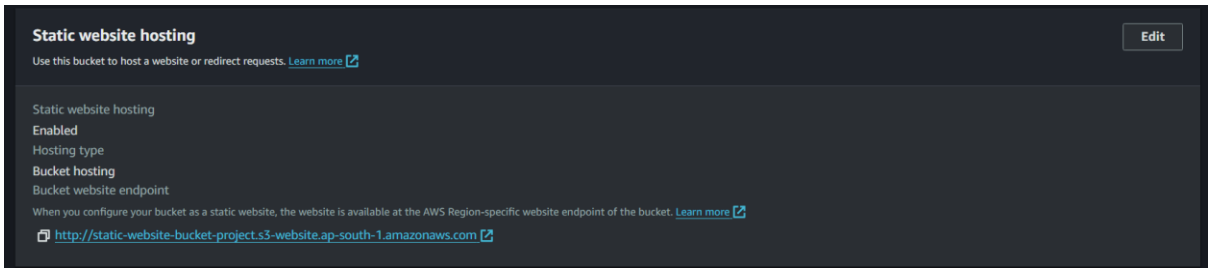
Under the Permissions section Go to the Bucket Policy

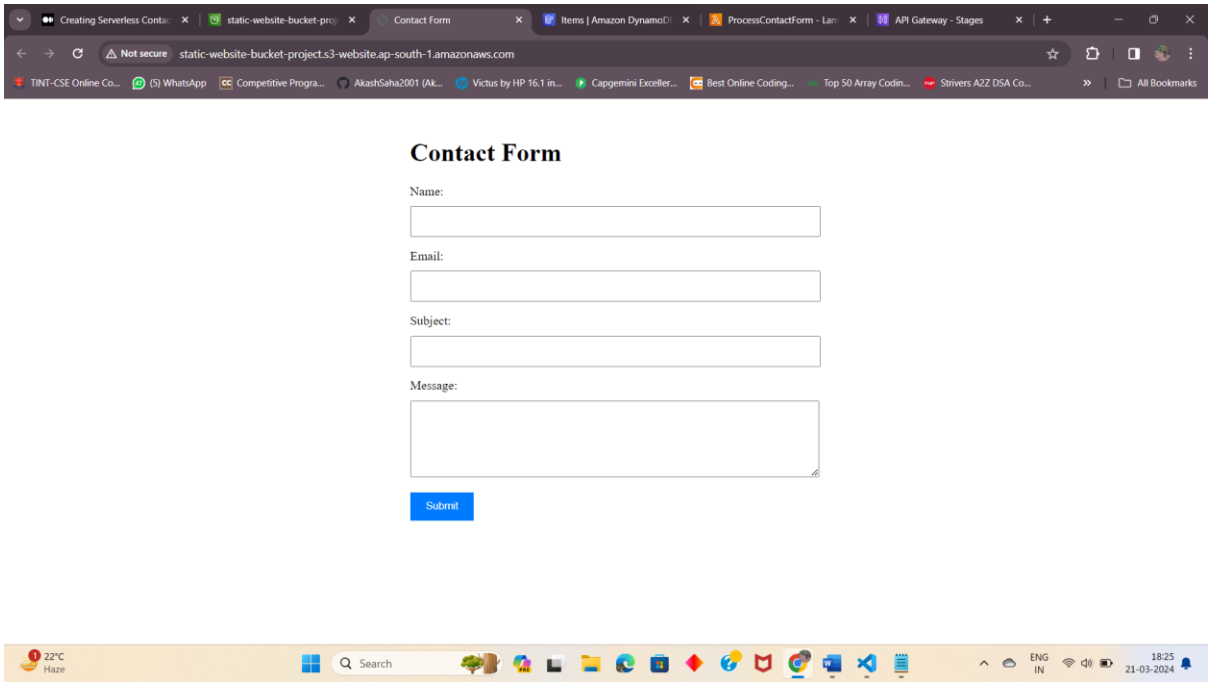


- Add the following as the bucket policy.
- **Note:** Modify the Resource line according to your S3 bucket name.

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

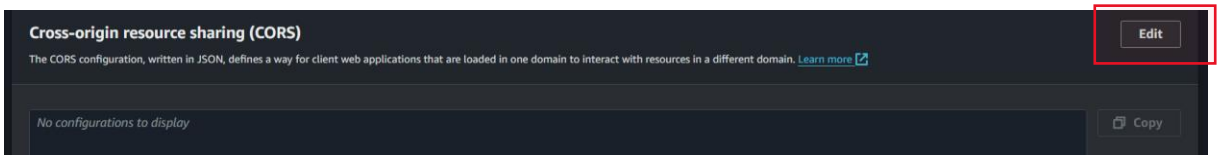
Now go to the Properties section and scroll down search for Static website hosting, click on the link to view your webpage.





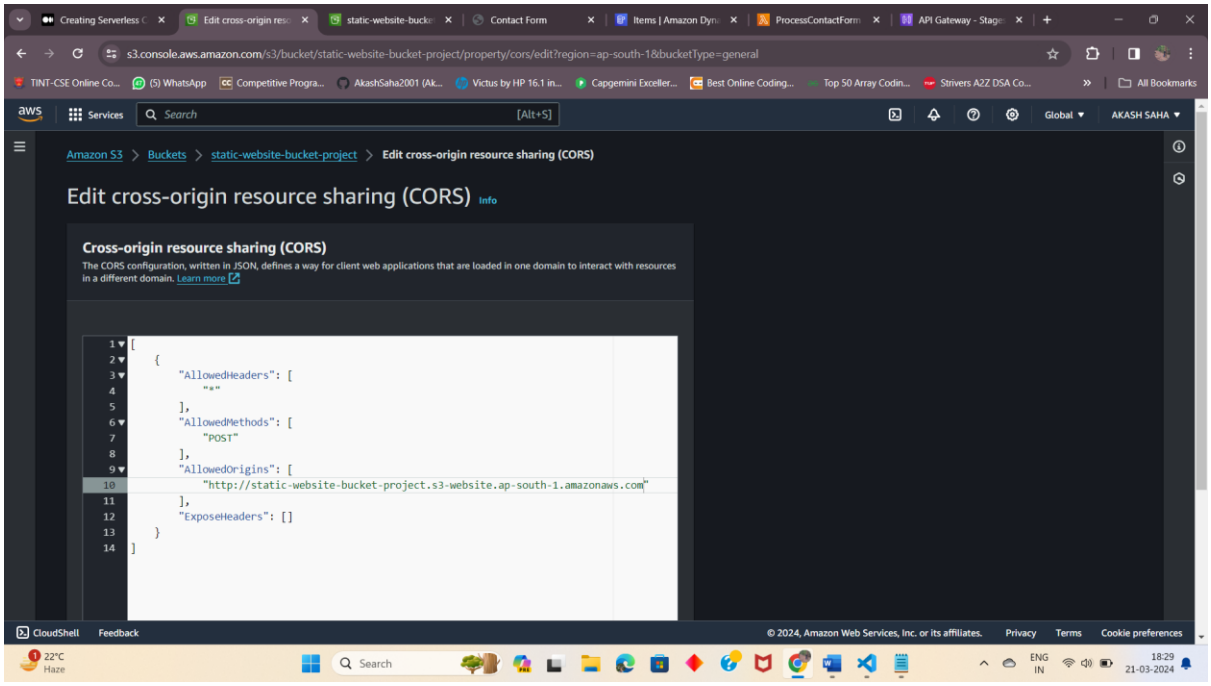
Configure CORS in the API Gateway and in the S3 bucket.

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

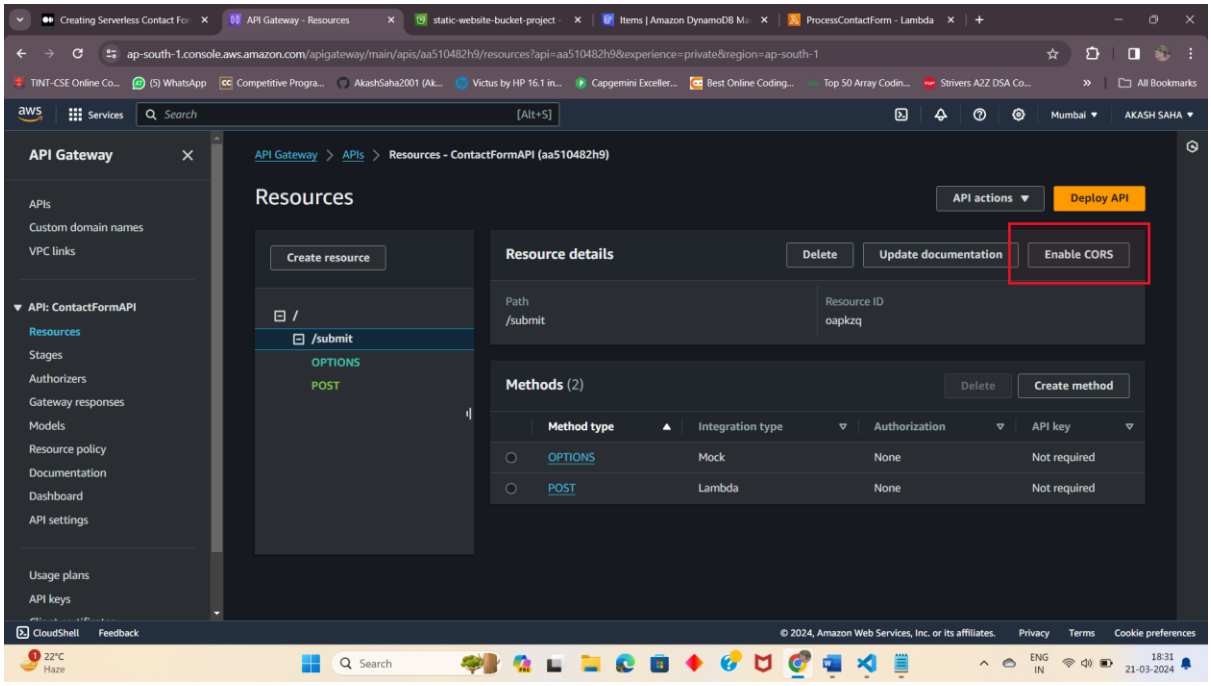


Add the following JSON, modifying the AllowedOrigins value to your static website URL to restrict access to the bucket resources from other origins, thereby enhancing security.

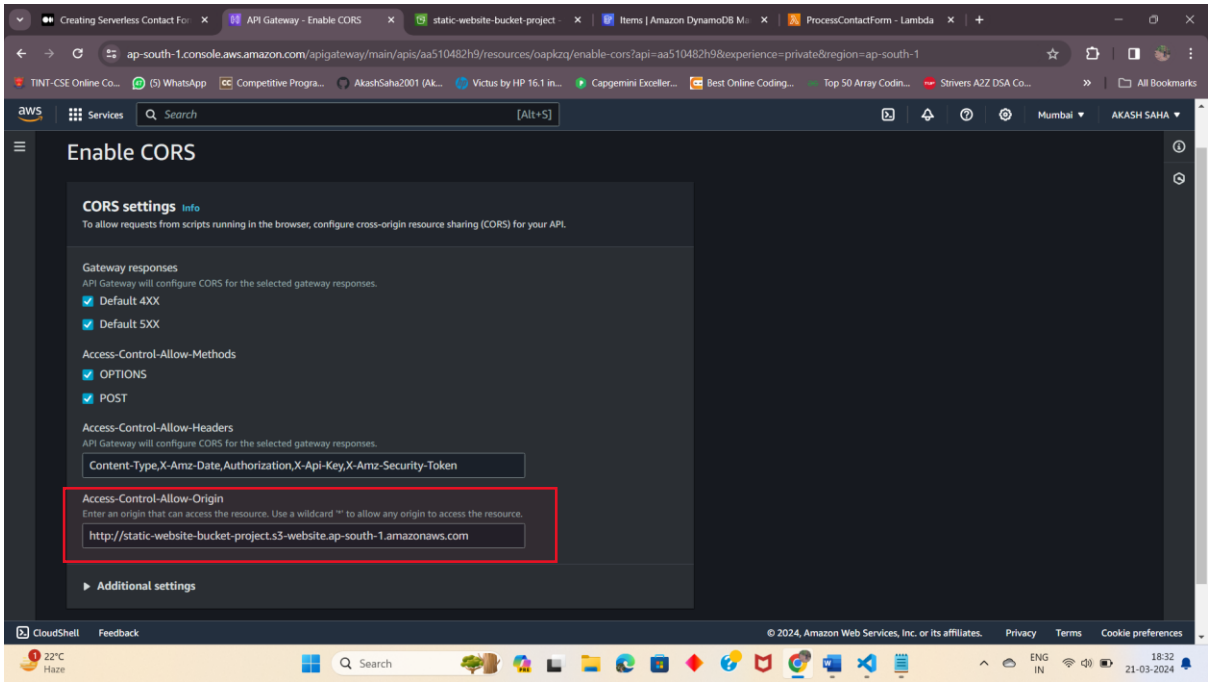
```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "POST"
    ],
    "AllowedOrigins": [
      "http://static-website-bucket-project.s3-website.ap-south-1.amazonaws.com"
    ],
    "ExposeHeaders": []
  }
]
```



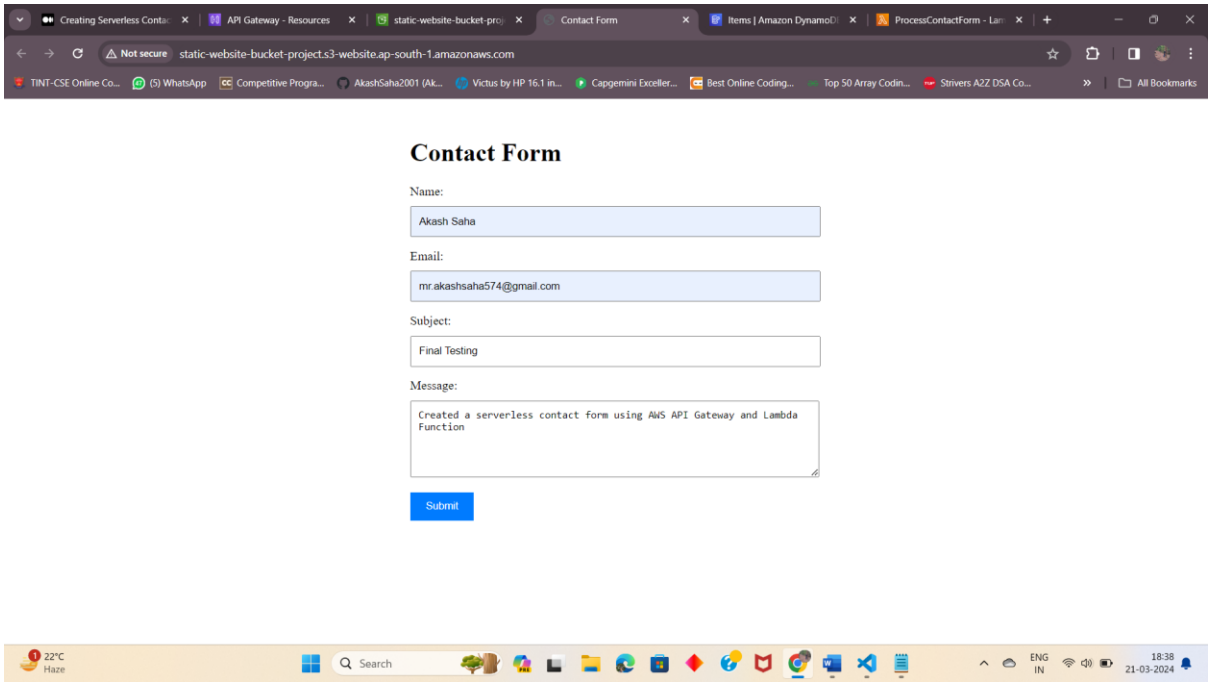
API Gateway: It also needs to be configured to explicitly allow requests coming from the S3 website domain.

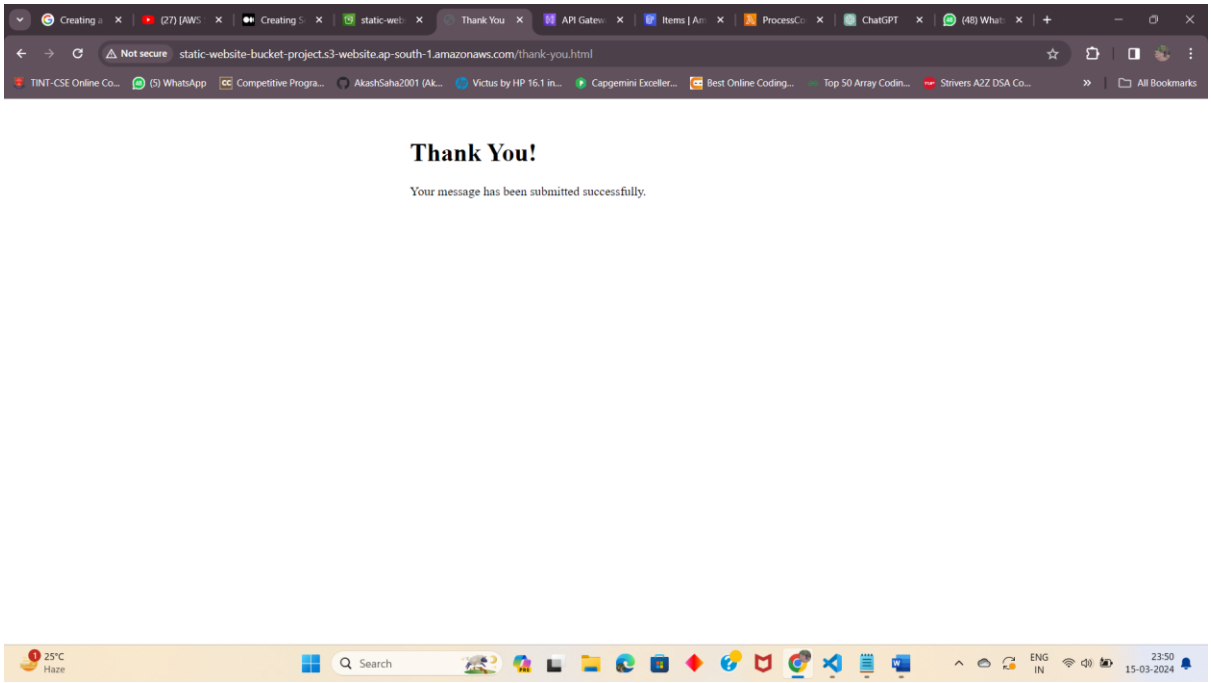


Under the Access-Control-Allow-Origin paste your static website URL

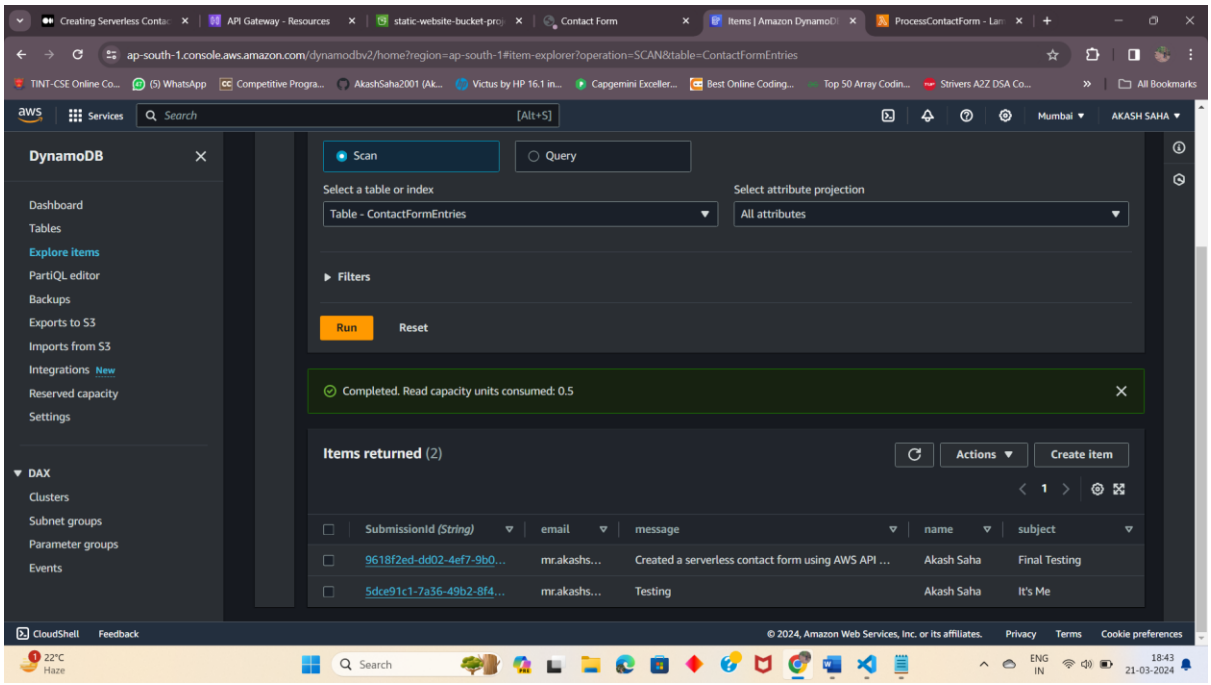


Now finally, Go to the static website, Fill the form and submit it.





Cheak the Database if it is Inserted or not.



Done.