Serverless Web Application in AWS

Venkatgiri Sasanapuri · Follow
10 min read · Apr 21, 2024

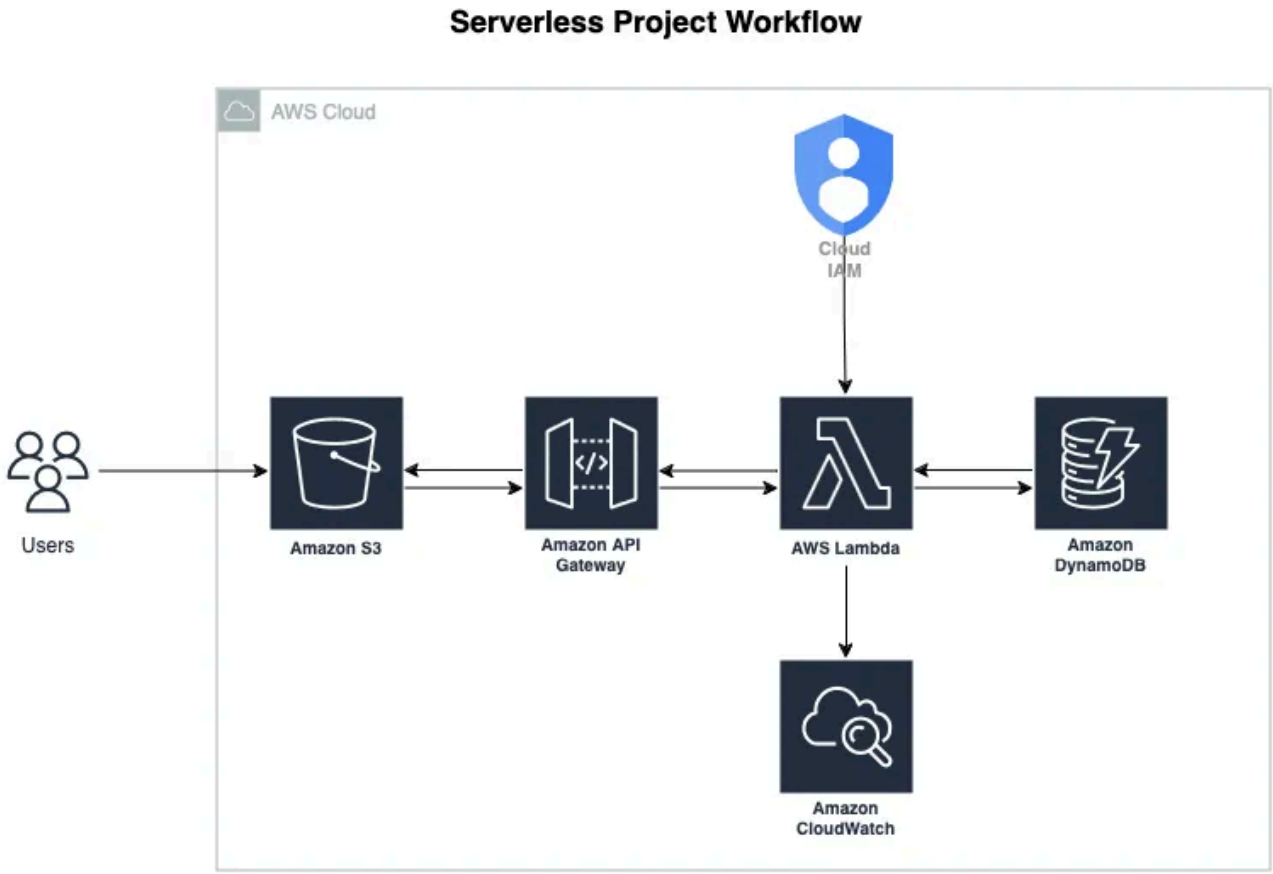


Image created using draw.io

The idea of using serverless comes when we want to get rid of the hardware limitations, server provisioning, and software installations needed to run our application.

Before jumping into the project, the prerequisites are:

1. AWS Account.
2. Visual Studio Code or any similar code editors.

We are creating a serverless web application using AWS services. As mentioned in the above diagram, we are using:

Amazon s3: To store static files. i.e., html, javascript files. We enable static hosting with the necessary permissions.

API Gateway: An API gateway manages incoming requests and routes them based on key factors such as request path, headers, and query parameters. In short, when we submit our details on the webpage, the API gateway manages to send this data to backend services like Lambda.

AWS Lambda: AWS lambda contains the backend code required for our application. This service is the backbone, and the name serverless is said because of this. We are neither creating servers nor installing any software here. We select the software environment needed and add our code.

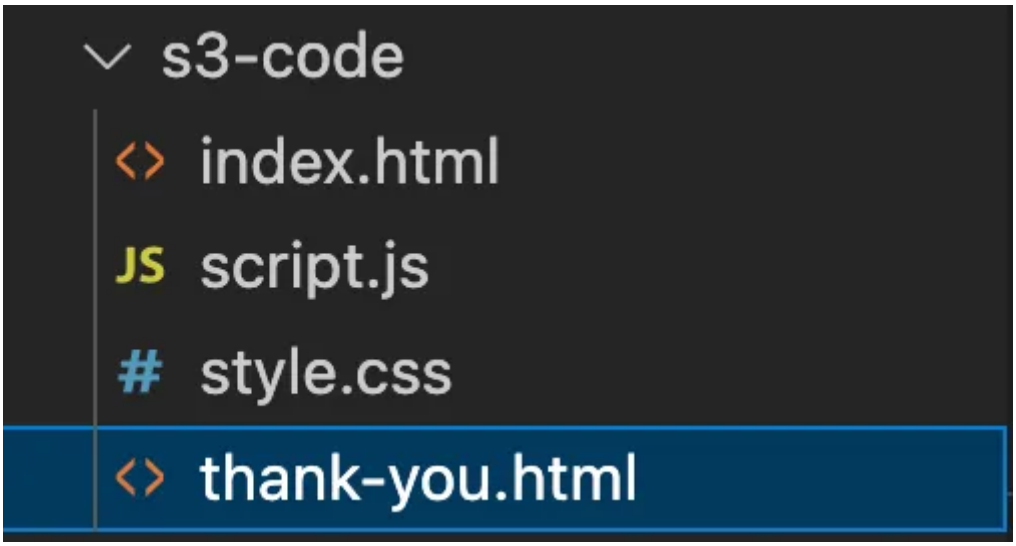
DynamoDB: This is a no-SQL(non-relational) database where the data is stored. For ex., When we fill out the HTML form, the data is processed as shown in the diagram and stored in the database.

Identity Access and Management (IAM): IAM is a service that helps manage resource access. We can perform user administration, resource administration, etc., here.

Cloudwatch: This is a monitoring tool that helps manage and monitor resources effectively.

Step 1: Create a Static Website using S3:

Let's start creating our bucket. I will list the files and the code I used to store them in s3.



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

```css
body {
  margin: 0;
  padding: 0;
  font-family: Arial, sans-serif;
  background-size: cover; /* Cover the entire background */
  background-position: center; /* Center the background image */
}

.container {
  max-width: 500px;
  margin: 50px auto; /* Center the container vertically and horizontally */
  padding: 20px;
  background-color: rgba(255, 255, 255, 0.8); /* Add a semi-transparent white b
  border-radius: 8px; /* Add some rounded corners */
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Add a subtle shadow */
}

h1 {
  text-align: center;
}

form label {
  display: block;
  margin-bottom: 8px;
}

form input,
form textarea {
  width: 100%;
  padding: 10px;
  margin-bottom: 15px;
  border: 1px solid #ccc; /* Add a border to input fields */
  border-radius: 4px; /* Add some rounded corners to input fields */
  box-sizing: border-box; /* Ensure padding is included in the width */
}

form button {
  width: 100%;
  padding: 10px 20px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

form button:hover {
  background-color: #0069d9;
}
```

script.js

```
document.getElementById('contact-form').addEventListener('submit', function(eve
    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('https://xxxxxxxx.execute-api.us-east-1.amazonaws.com/dev/submit', {
      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.');
```

```
    });
  }
```

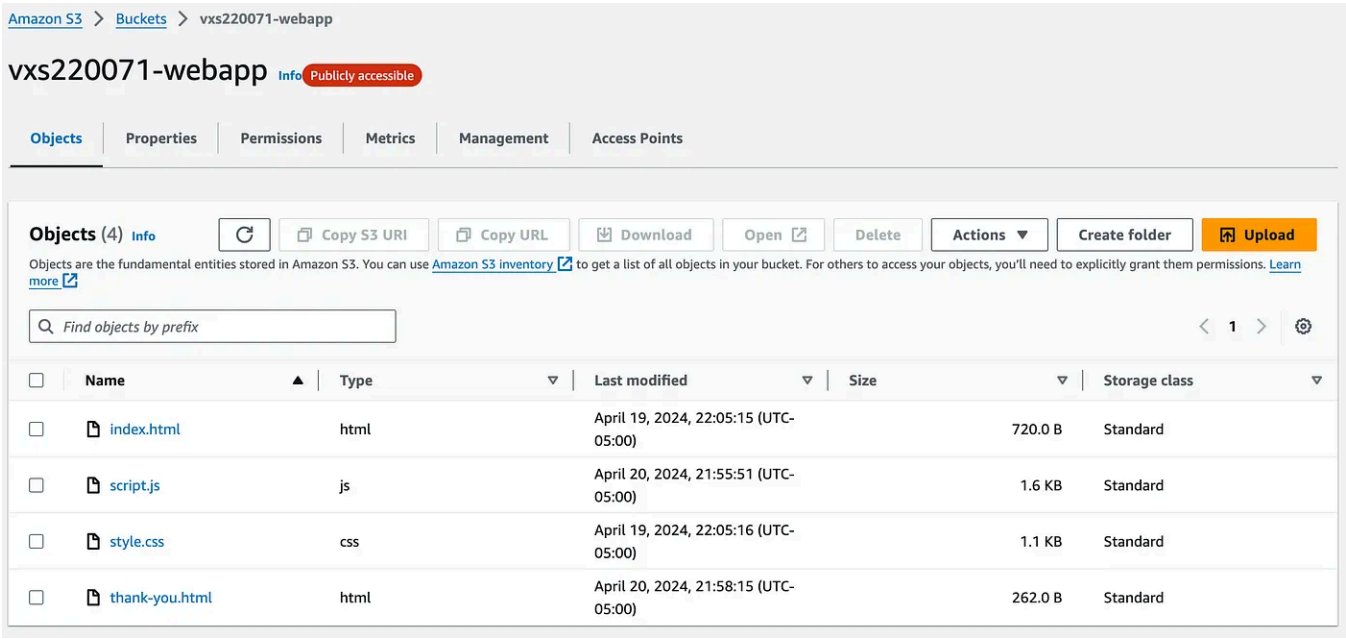
thank-you.html

```
<!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>
```

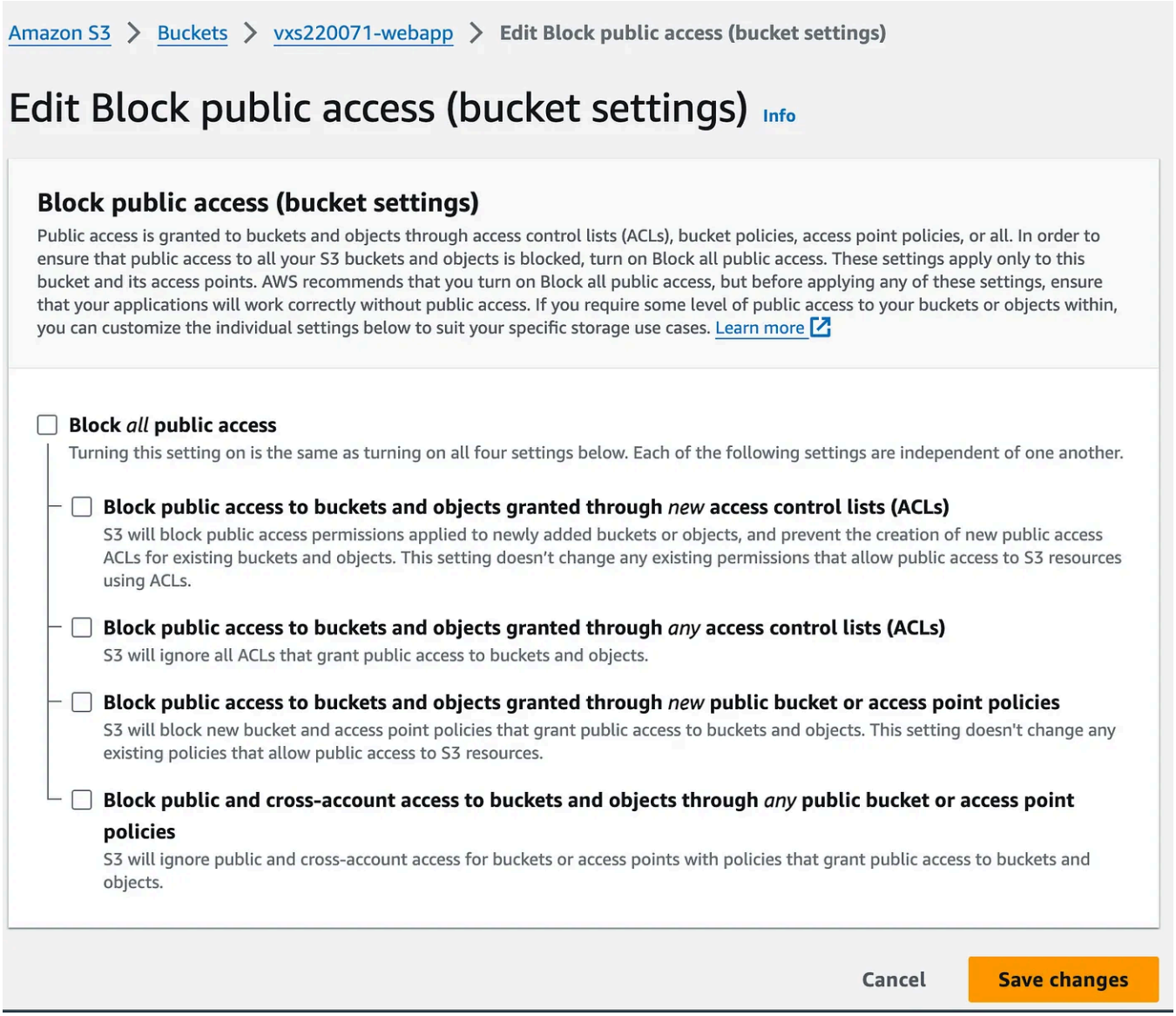
Let's create an s3 bucket and then enable the static website.

1. Open s3 in the AWS Console. Create a bucket with the name of your choice, leaving rest of the options to default.

2. Once the bucket is created, upload the index.html, script.js, style.css and thank-you.html files.



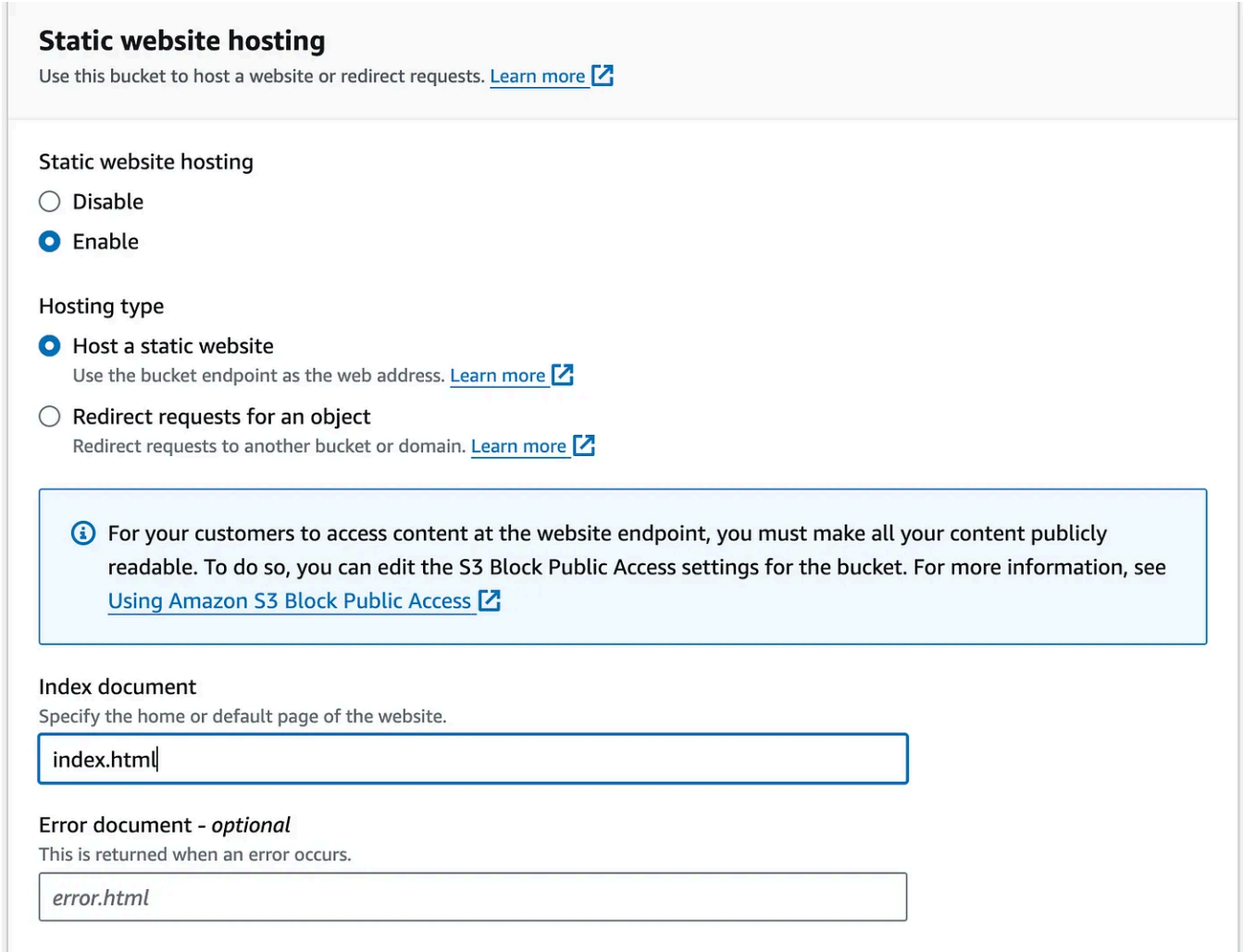
3. Open the permissions tab and scroll down to unblock the public access.



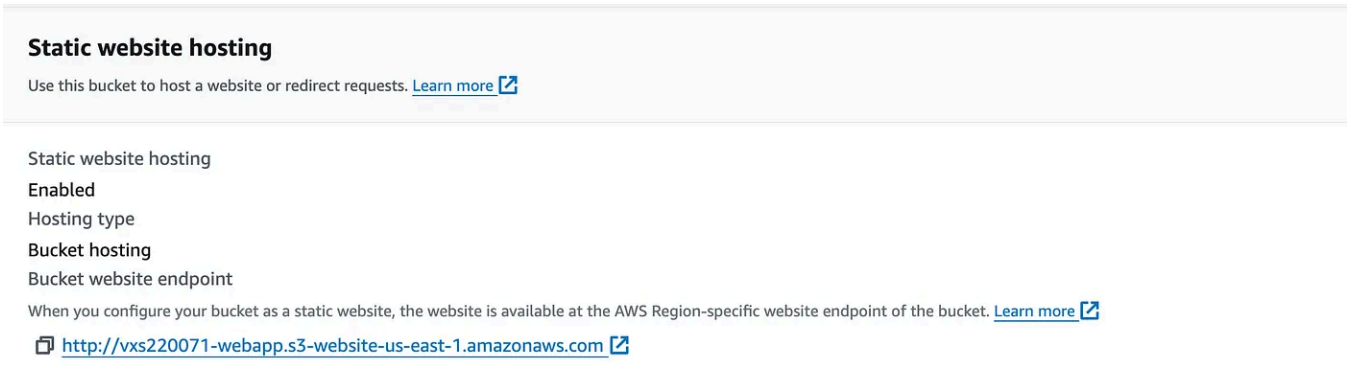
4. Edit the bucket policy and add the bucket below code. Make sure to change the bucket name.

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

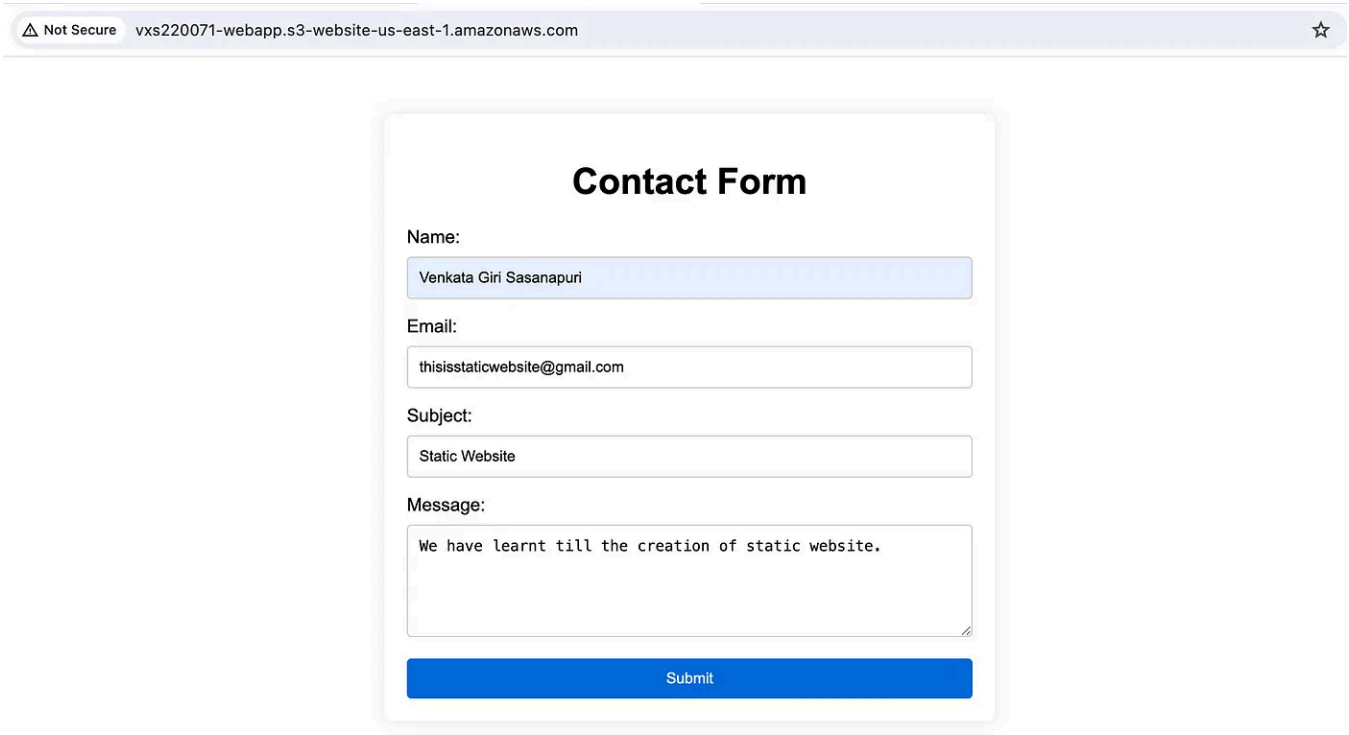
5. Go to the properties tab and scroll down till the end. Enable static website hosting. Give the name of your index document. It's index.html. Save the changes.



6. Click on the url created after enabling the static website.



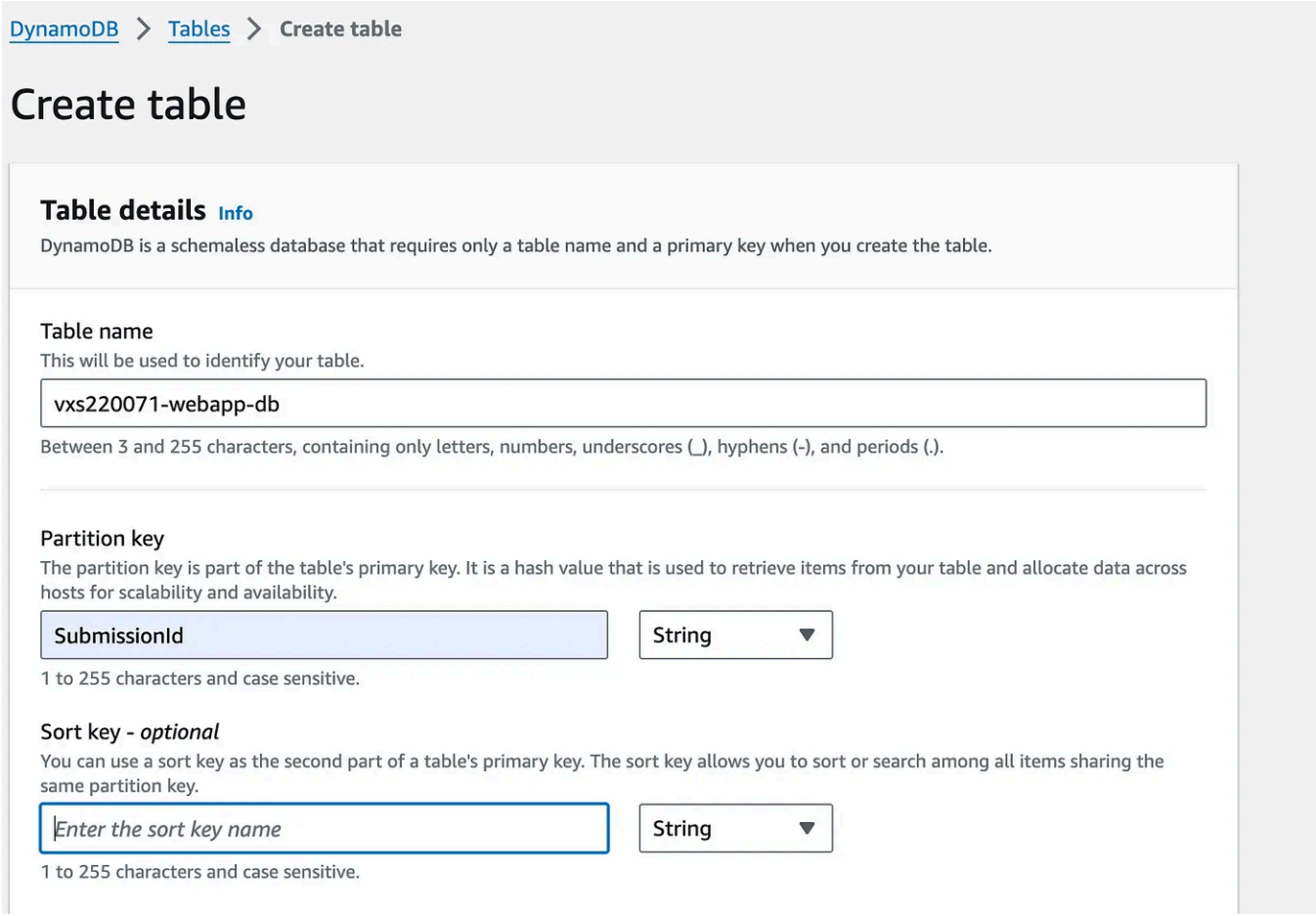
7. Congrats, we have created our static website. We have not set up the backend framework to store data in the database. So, even though I try to submit, the data won't be stored.



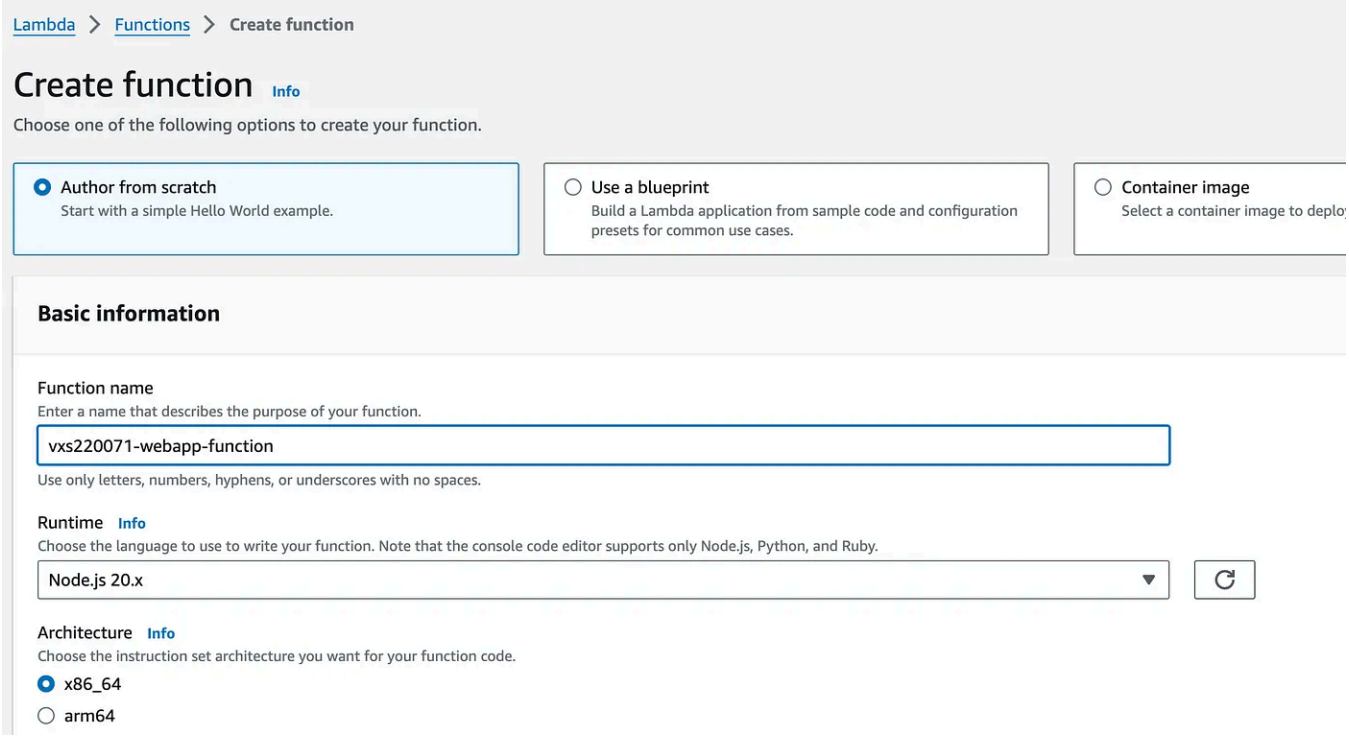
Step 2: Working on lambda and Dynamo DB

1. In this step, we create a database to store the form data.

2. Let’s create a DynamoDB with the name of your choice and the Partition key as mentioned. Leave the rest of the options to default and create the table.



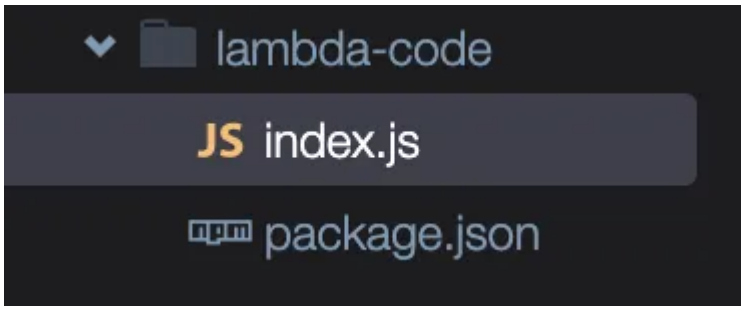
2. Let’s create the lambda function. Give a name of your choice and leave the rest of the options to default. This will also create a lambda role with basic execution permissions.



3. We must do certain steps to upload the code to lambda.

Let's add our code to lambda. We have two options for uploading our code: s3 and direct upload. Let's use s3 to follow best practices.

Before doing that, open the VS code and save the code as shown below.



index.js

Make sure to change the DynamoDB table name.

```
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 inpu
```

```
    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: '<dynamodbtable>',
    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"
    }
}
```

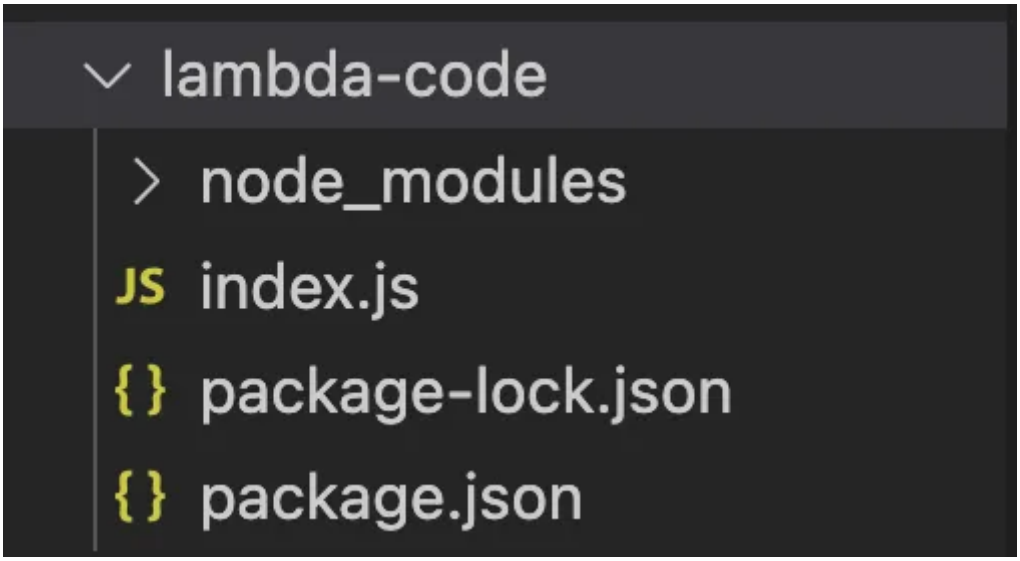
Open the terminal in VS code and make sure you are in the *lambda-code* directory. I had to move to lambda-code by typing the cd command as my parent directory is different.

```
venkatgirisasanapuri@Venkatgiris-MacBook-Air Javascript-testing.js % cd lambda-code
venkatgirisasanapuri@Venkatgiris-MacBook-Air lambda-code %
```

Run the npm install command to install the modules required to run our code.

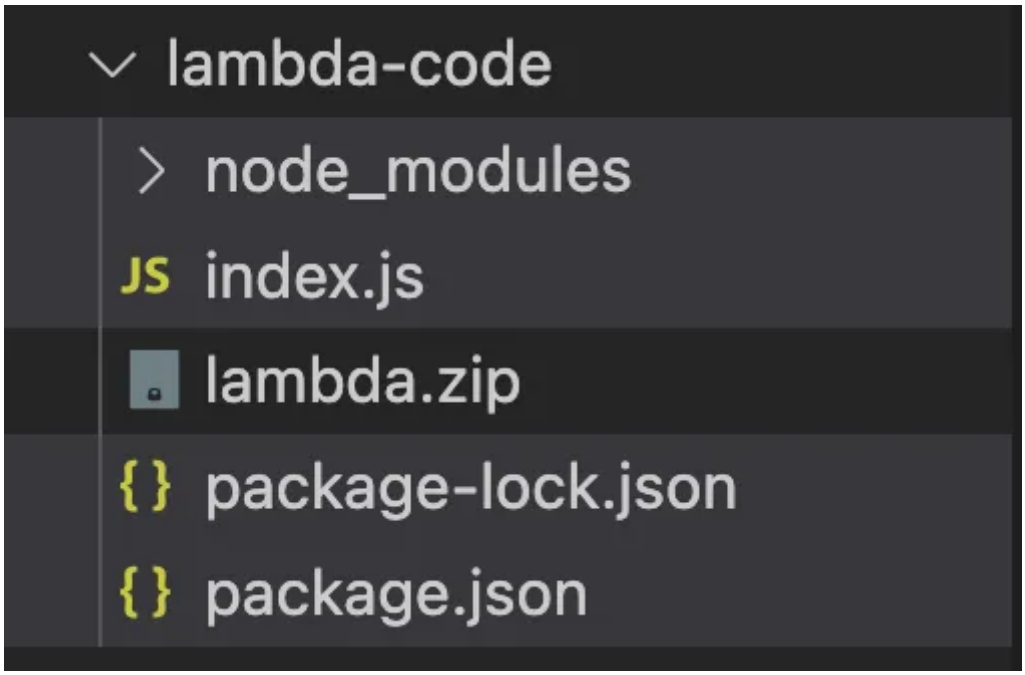
```
npm install
```

After installation, the folder structure should be like,

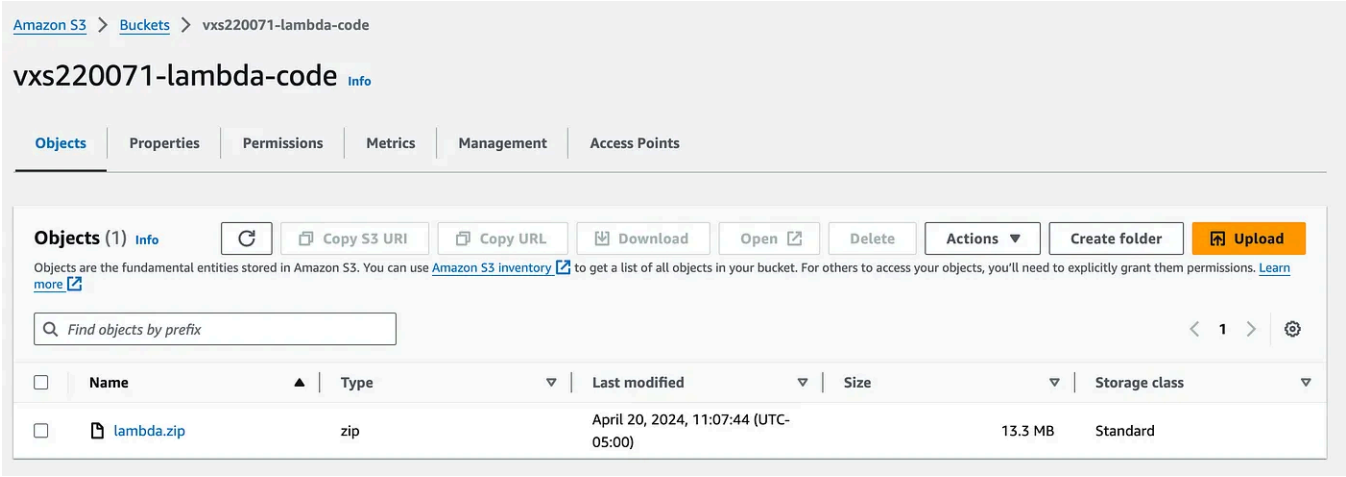


We should zip all the files and folders within the lambda code. We can use the command given below: (Make sure you are in the lambda-code directory)

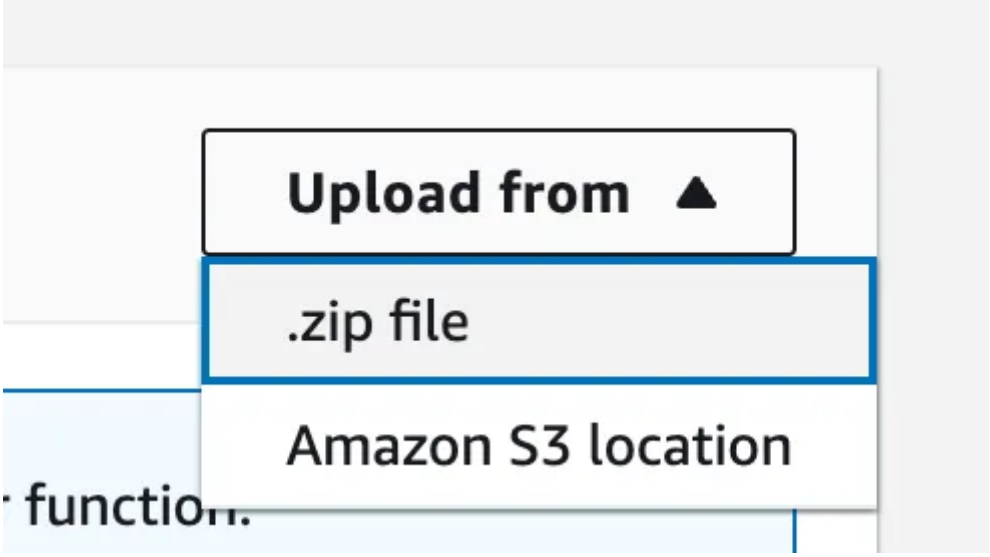
```
zip -r lambda.zip .
```



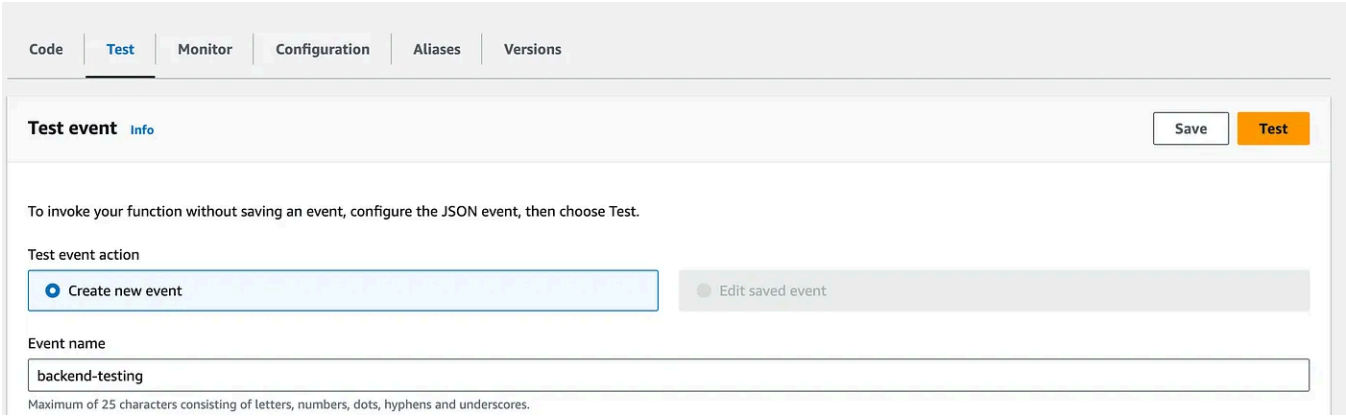
4. Let's create an s3 bucket to upload this code. Create the bucket with the name of your choice and with the default options, and upload the zip file.



5. Copy the S3 URI, open lambda, and select upload from the Amazon s3 location.

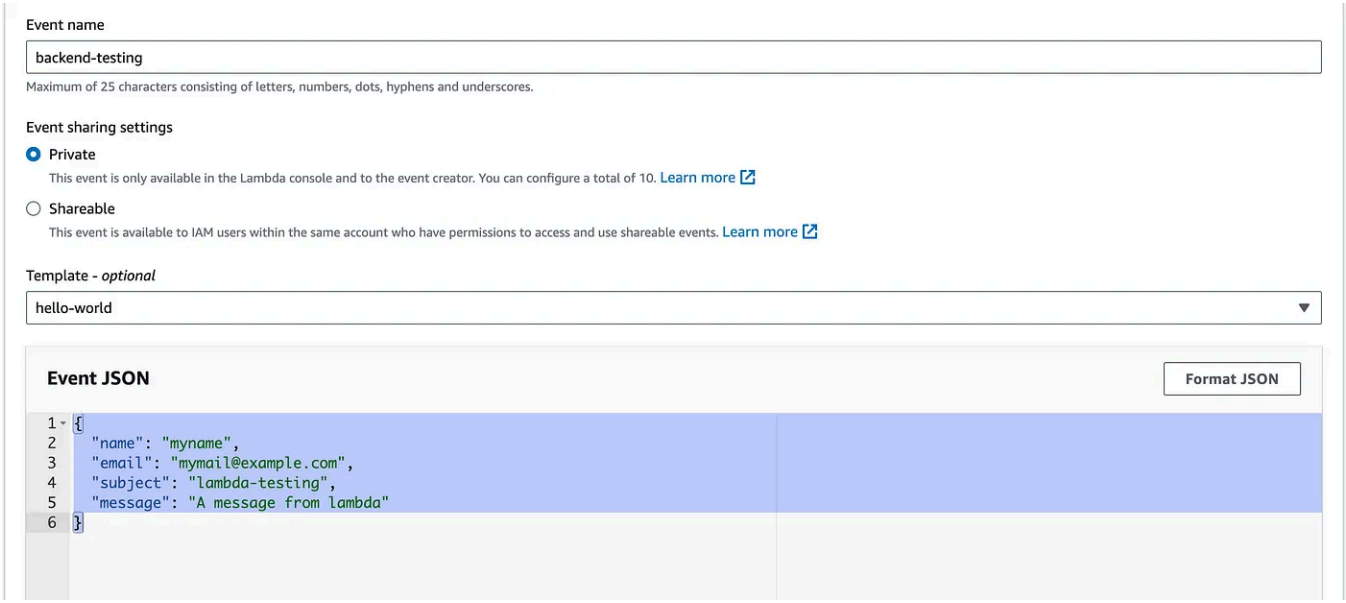


Let's test our backend code by saving our data manually using the test option in Lambda. We will create a test event.

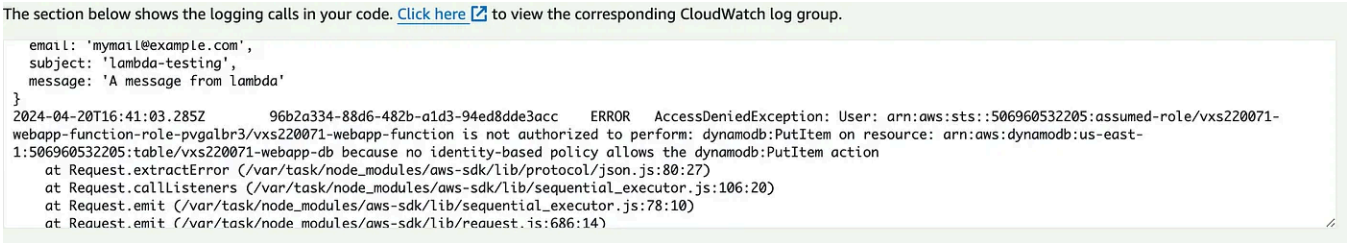


Add the below json-code and save the test event for future uses.

```
{
  "name": "myname",
  "email": "mymail@example.com",
  "subject": "lambda-testing",
  "message": "A message from lambda"
}
```

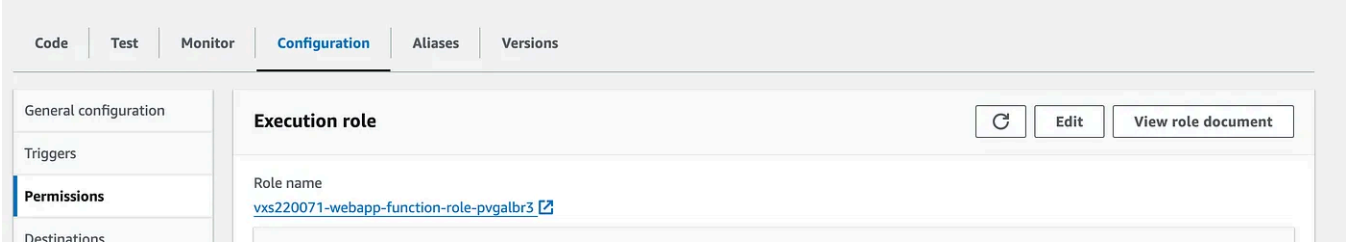


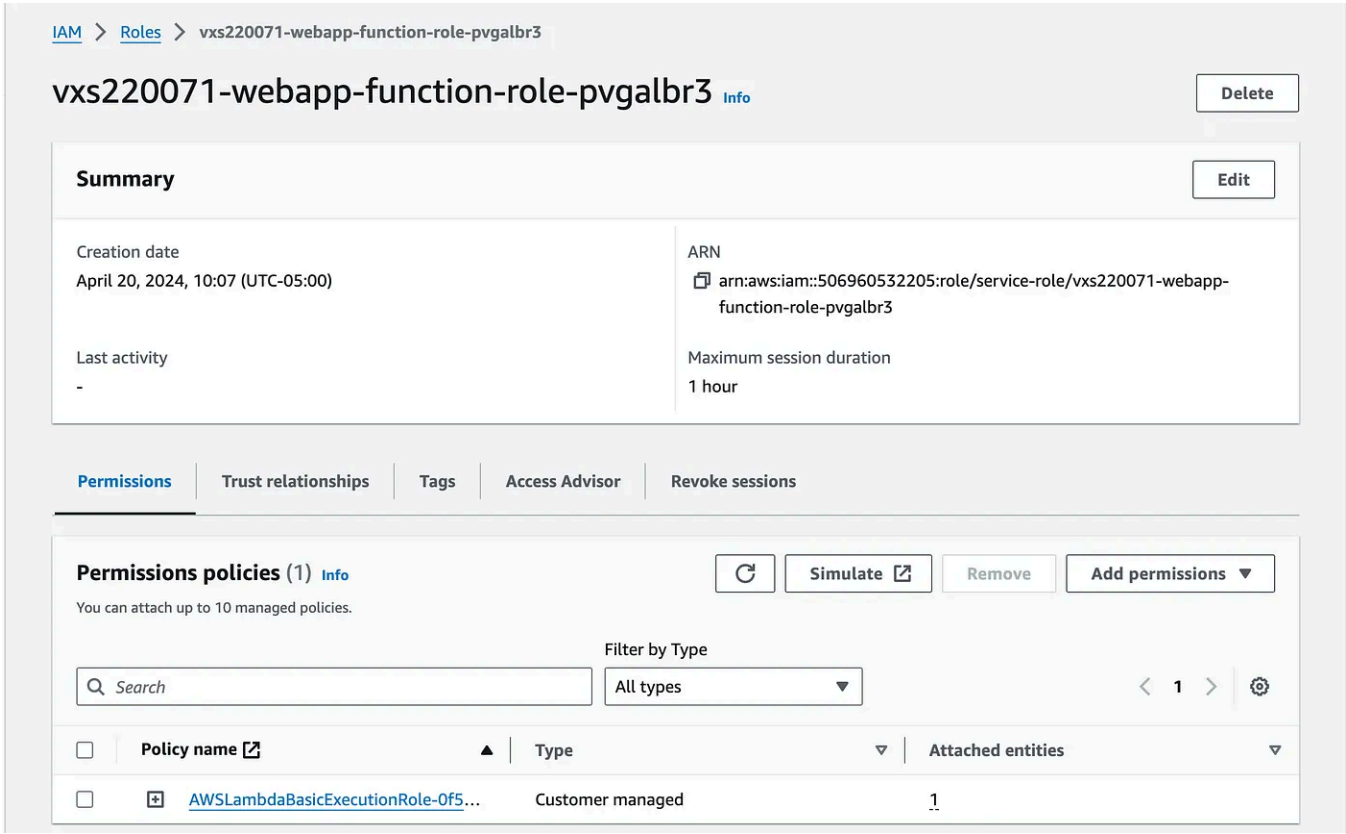
6. Test the event. This should save the details in the DynamoDB, but due to permission issues, I got an error saying the lambda role doesn't have permission to save data.



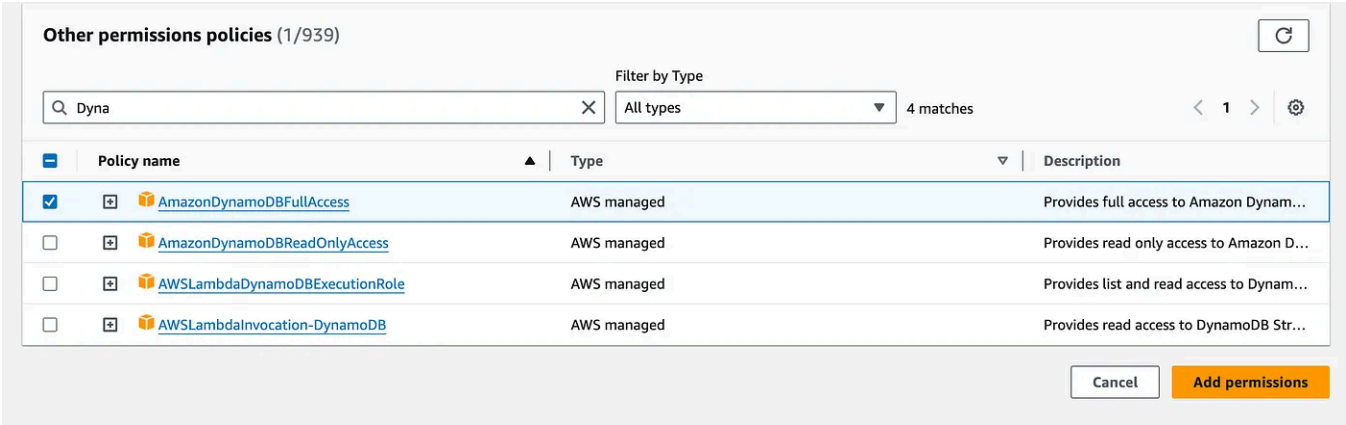
Let's modify the permissions of the lambda role.

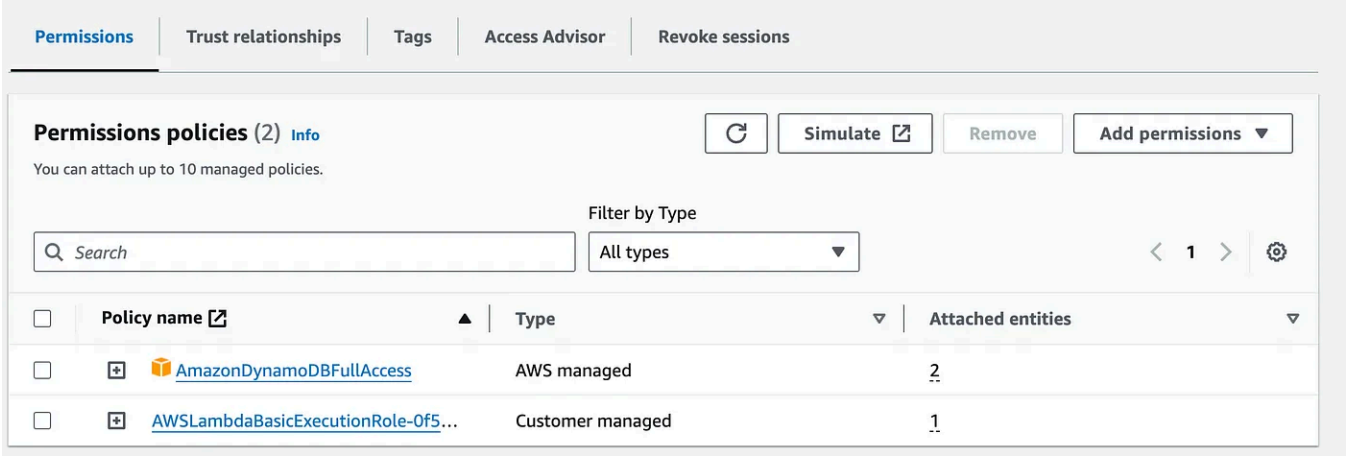
Click on the Configuration tab and then on Permissions from the left menu to view the role. Click on the role name.



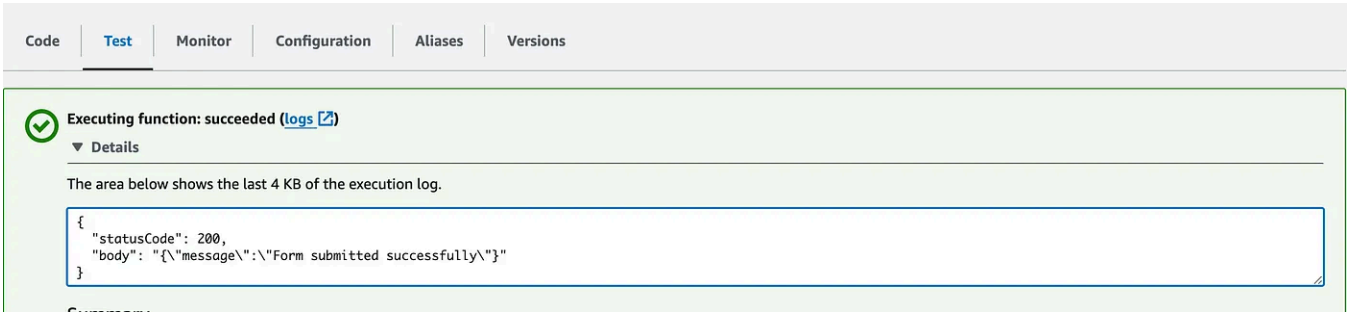


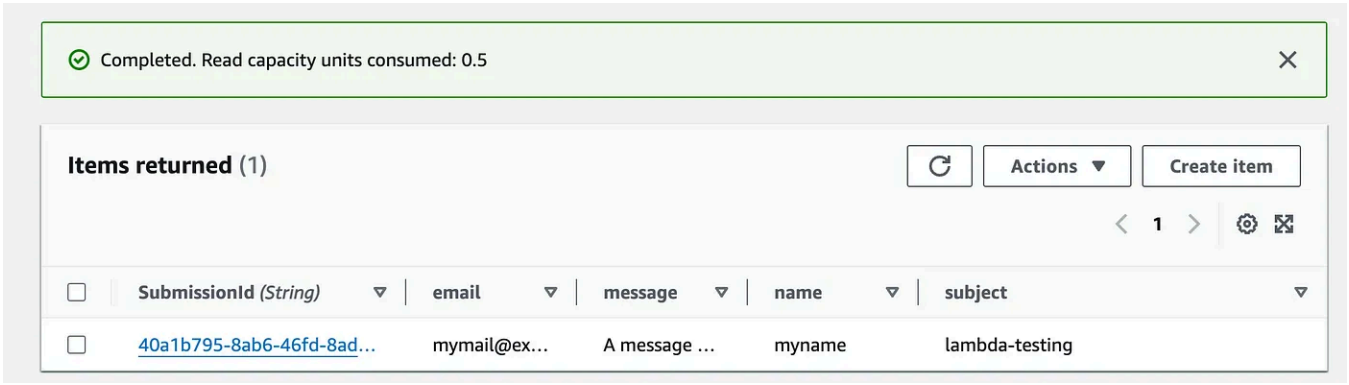
Click on Add Permissions and select Attach Policies. Search for Amazon DynamoDBFullAccess and Add permissions.





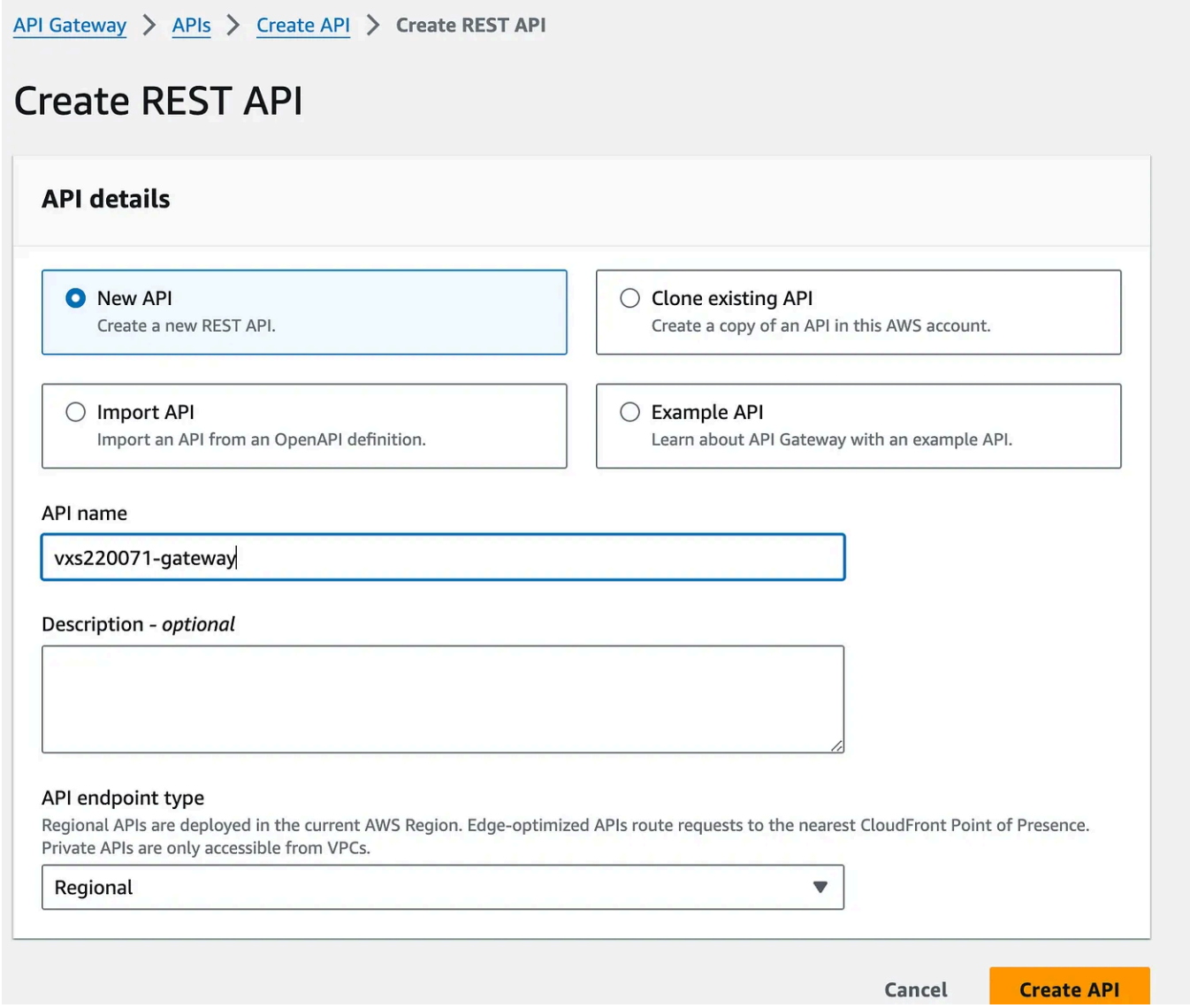
Let's open lambda to test whether the data is being saved. This time, my data is saved in DynamoDB.



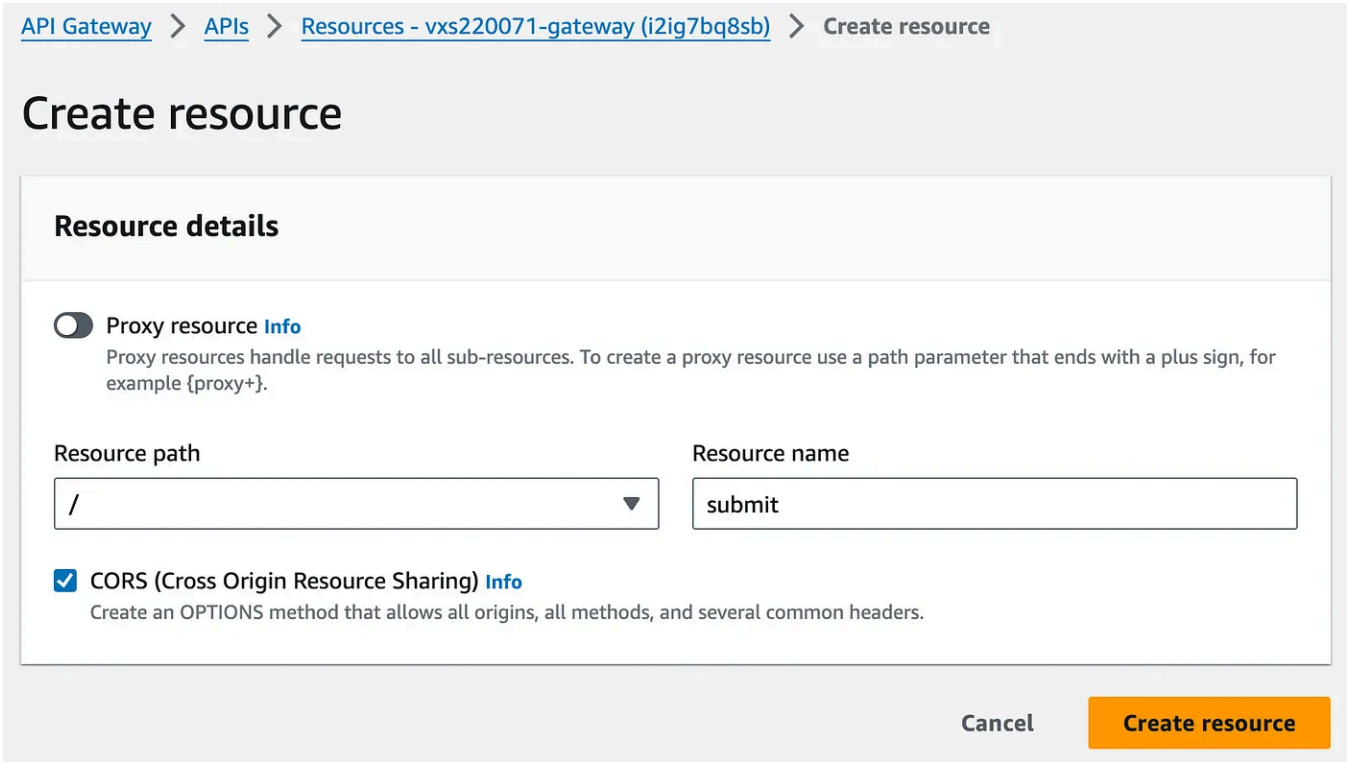


Step 3: Let's create the bridge to bring the data from S3 to Lambda. That is API Gateway.

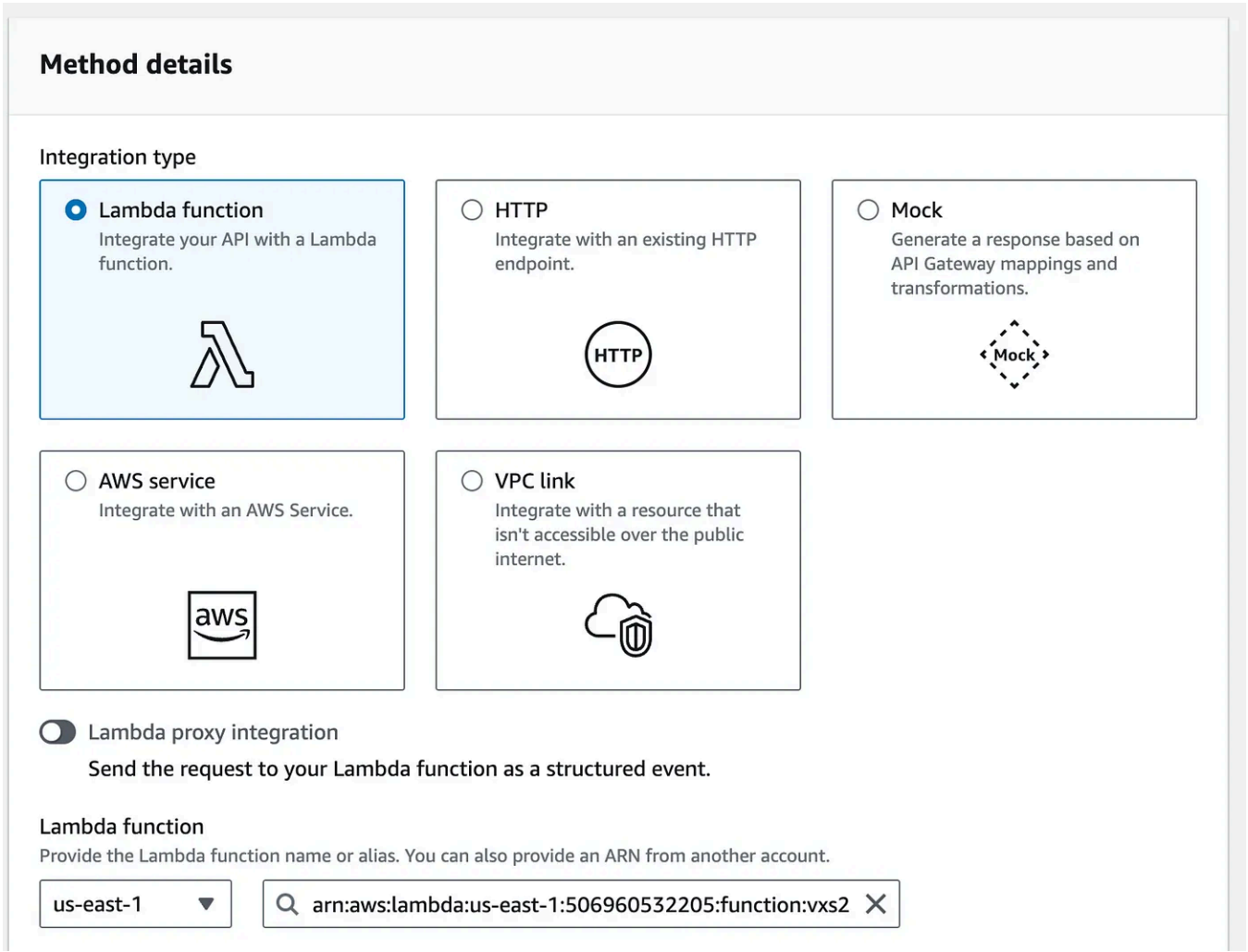
1. Open API gateway in the console and choose the REST API type. Give the name of your choice and leave other options to default.



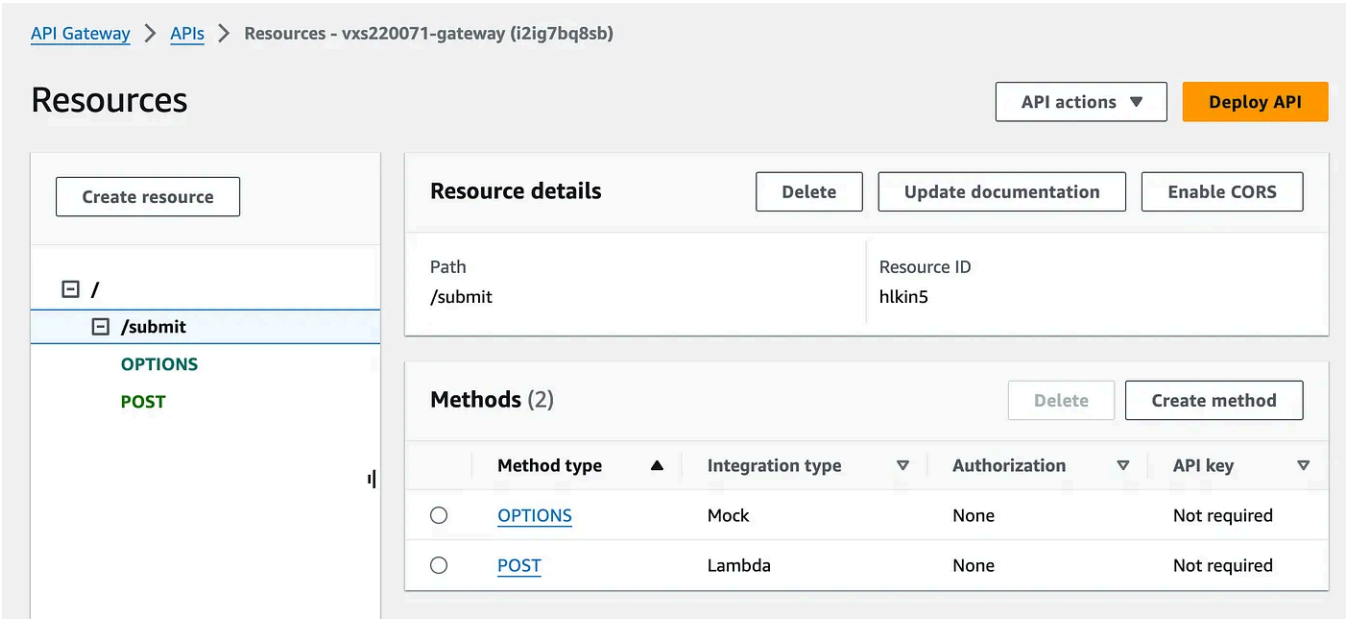
2. Click on Create resource and give the resource name. Enable CORS for the enhanced security.

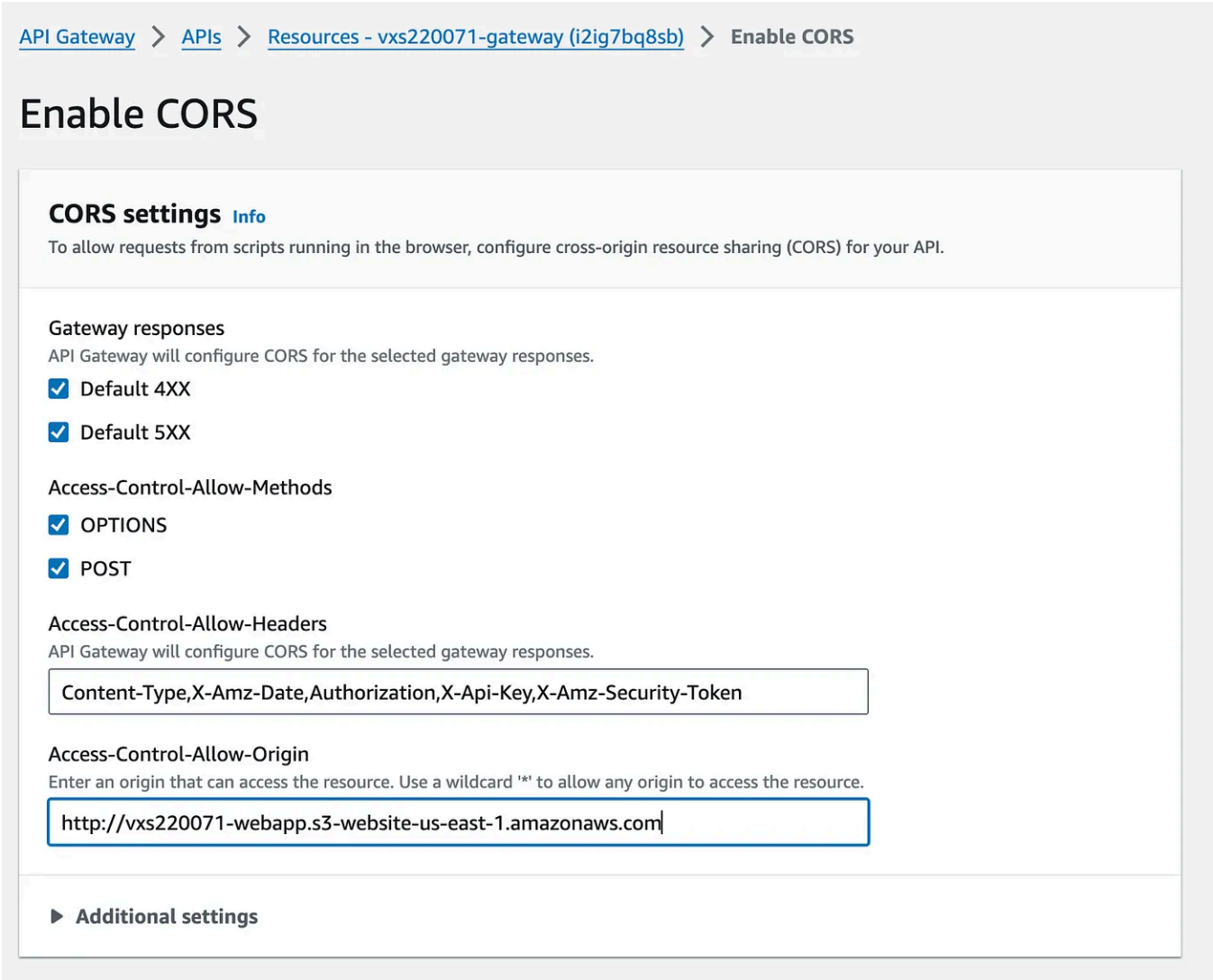


3. Create method and select the method type POST, and select your lambda function. Leave the rest of the options to default and Create method.

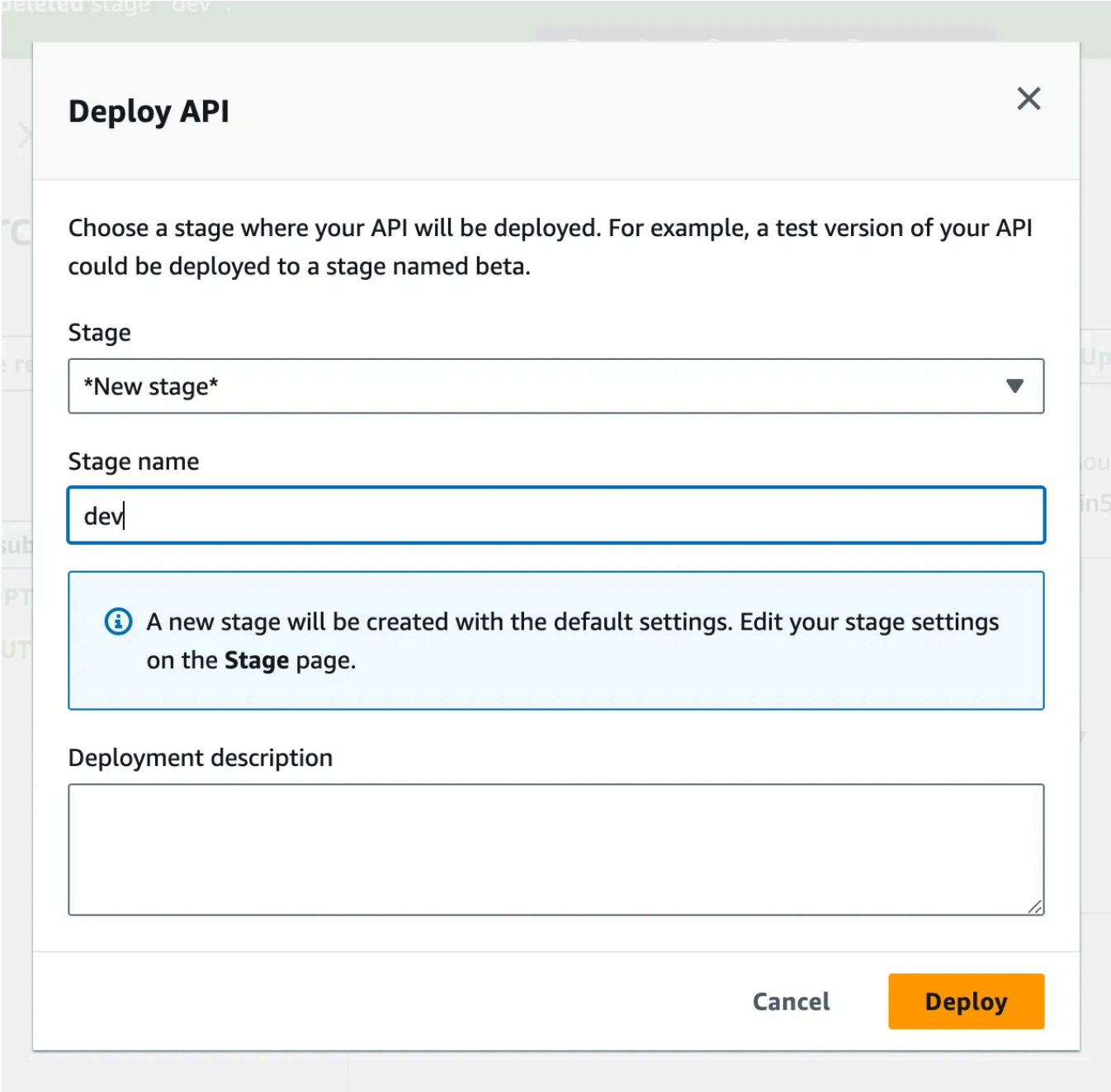


4. Select the Enable CORS option(click on the resource submit). Check the boxes as shown and provide your s3 static website URL. We have to update the CORS policy in s3. We will do it later.

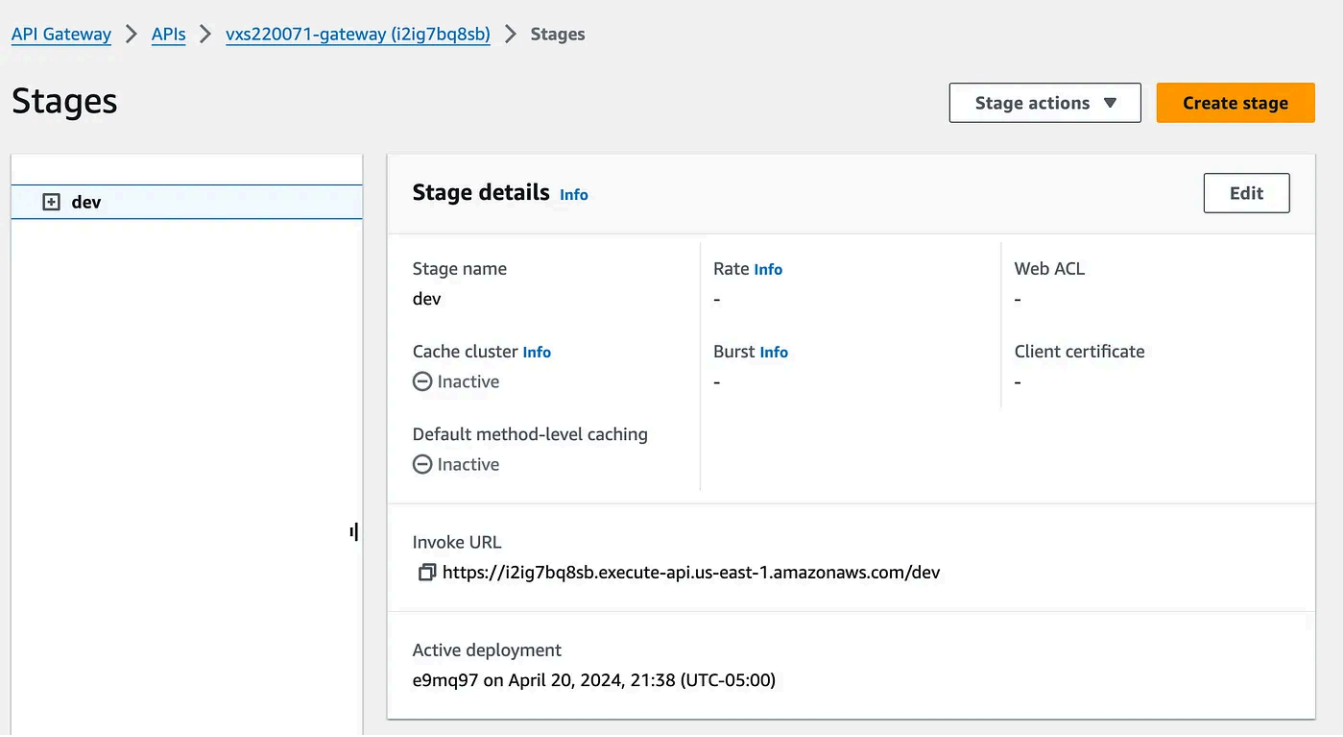




5. Click on Deploy API, select a New stage and give your stage name.



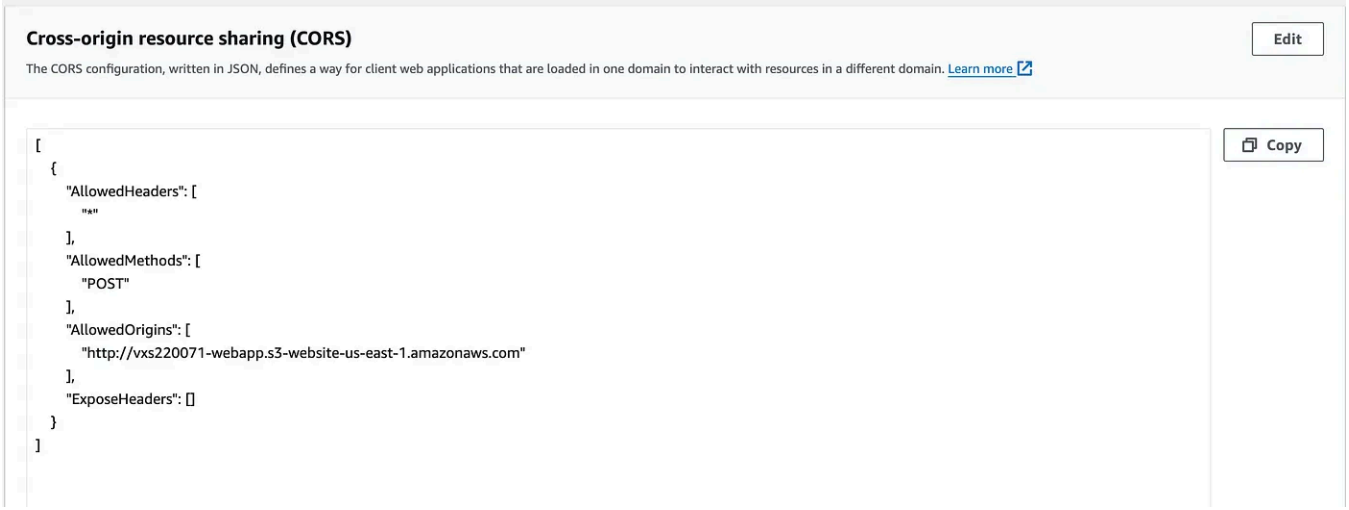
6. Once deployed, we can see the invoke URL below. This URL is responsible for receiving a response when the form is submitted. So, we have to modify the script.js file in the s3 bucket and upload the same file again.



Below is the part where you have to update the file. Once updated, upload the file again.

```
function submitForm(formData) {
    // Make an API request to the backend (API Gateway) for form submission
    fetch('<invoke-url>/submit', { // URL that represents the backend API endpc
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(formData)
    })
```

7. We have to update the CORS policy in s3. Open the s3 bucket and scroll down to the CORS section in the permissions tab. Replace the static URL.

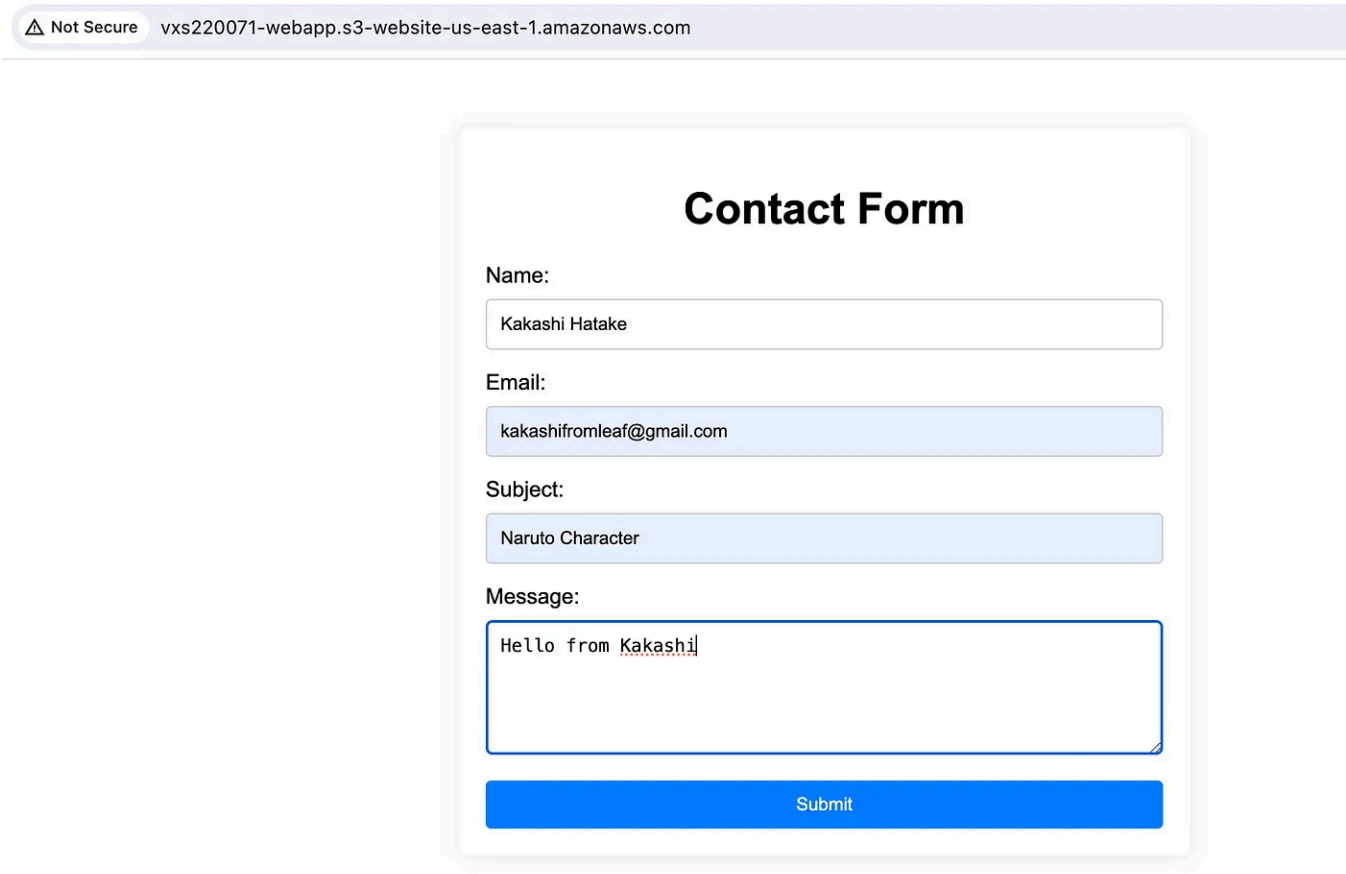


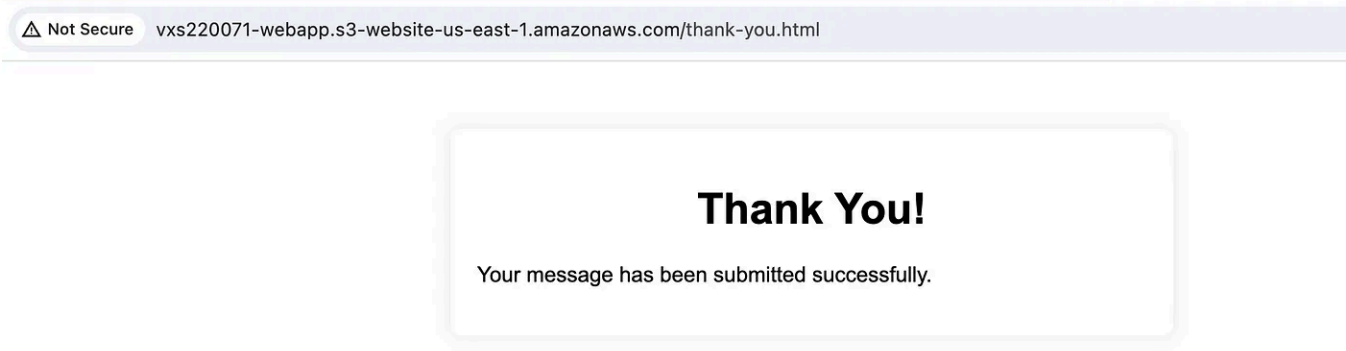
Copy the CORS policy below.

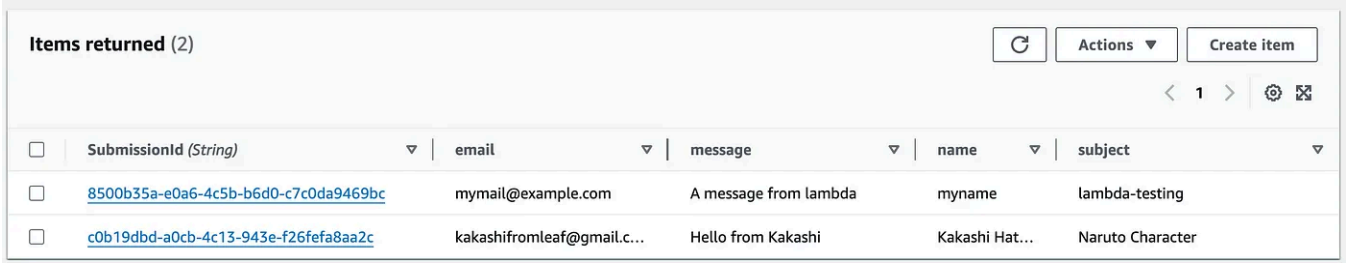
```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "POST"
        ],
        "AllowedOrigins": [
            "your-static-website-url"
        ],
        "ExposeHeaders": []
    }
]
```

Step 4: Testing

1. Let's see whether the submitted form is being saved in DynamoDB. Hit the static website in the browser and fill in the data. Once the data is submitted, we can open the db and explore items to see the data.







Congrats on the completion of the project!

Connect with me on Linkedin: https://www.linkedin.com/in/venkatgiris/

This blog is based on Ulises's blog. Everything is pretty similar except for a few steps where I struggled.
Thank you Ulises Magana.

DevOps AWS Cloud Computing Serverless Cloud

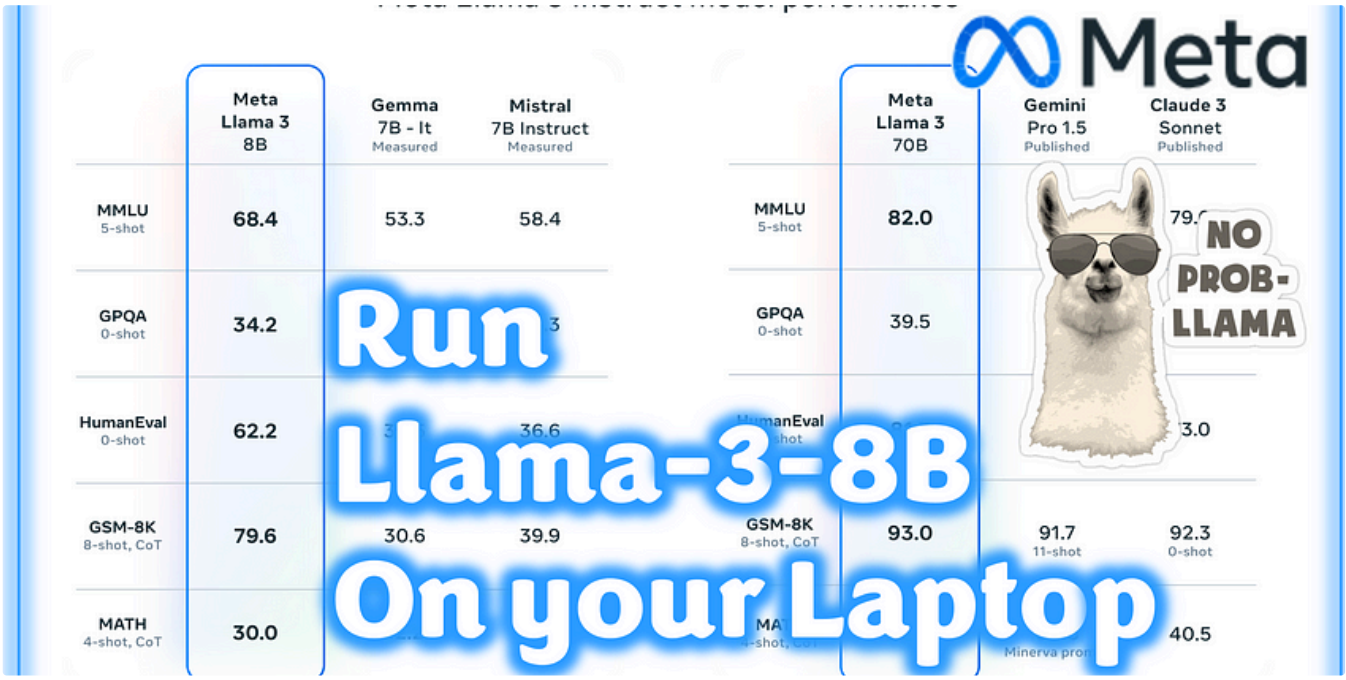
V

Follow

Written by Venkatgiri Sasanapuri

4 Followers

Recommended from Medium



Fabio Matricardi in Generative AI

Llama3 is out and you can run it on your Computer!

After only 1 day from the release, here is how you can run even on your Laptop with CPU only the latest Meta-AI model.

· 8 min read · Apr 20, 2024

1.5K 16

	Period	Note
	1 week	Simple landing page connected to domain of the venture. Collect ema
	2 week	Send emails, get feedback, and try the real need
	3 week	Start the building stage, with 1 sing focus
	4 week	Launch in public

Domenico Gagliardi

How I built and sold a micro-saas in 45 days

Hello everyone,

· 3 min read · Apr 20, 2024

271 9

Lists

General Coding Knowledge
20 stories · 1154 saves

Leadership
50 stories · 305 saves

Natural Language Processing

1409 stories · 906 saves

Productivity

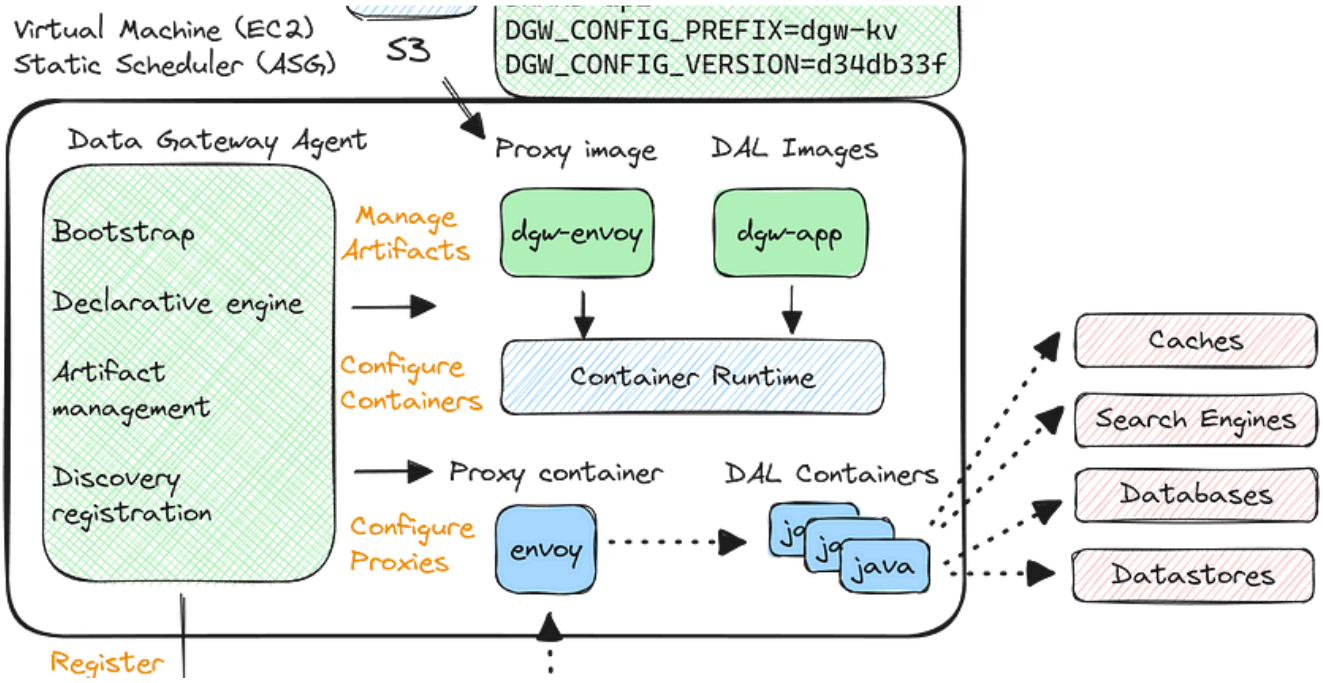
240 stories · 408 saves

Aryan Mehrotra in Stackademic

Why use GoFr for Golang Backend?

5 min read · Apr 21, 2024

344 1



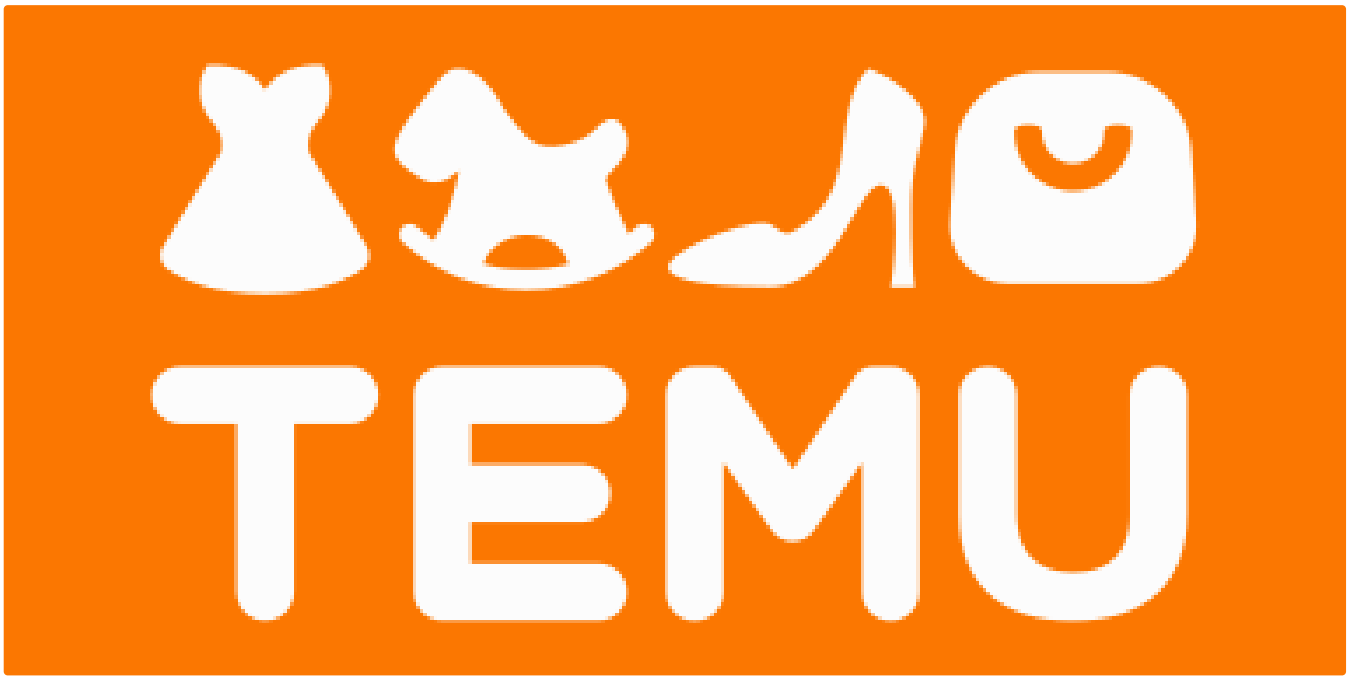
Netflix Technology Blog

Data Gateway—A Platform for Growing and Protecting the Data Tier

Shahar Zimmerman, Vidhya Arvind, Joey Lynch, Vinay Chella

13 min read · 5 days ago

312 3



Economics & Tech in Summit

The Secrets Behind Temu's Unbeatable Prices

How a Chinese e-commerce giant is using billions of dollars to disrupt the U.S. market

· 7 min read · Apr 21, 2024

1.3K 31

Alireza Yavari

The Observability Stack Part1

In the DevOps world when we talk about observability, we mean to generate, collect, and gather, perform some manipulation on it, centralize...

9 min read · Apr 20, 2024

345 1

See more recommendations