**GLA UNIVERSITY**



**Department of Computer**

**Science and Engineering**

**AWS – Project Report**

**Project Title**

**User Data Management with EC2 and S3 Integration**

**TABLE OF CONTENTS**

1. **Introduction**

* Overview
* Benefits

1. **Prerequisites**

* AWS Account Setup
* Required Software and Tools

1. **Setting up AWS S3**

* Creating Bucket
* Editing its policies

1. **Setting Up AWS Lambda**

* Creating function
* Adding IAM role in Lambda
* Writing Code

1. **Setting up API Gateway**

* Creating API
* Configuring routes and stage
* Configure CORS and deploy

1. **Setting up EC2 Instance**

* Creating an EC2 instance
* Installing Apache
* Creating an index.html file

1. **Conclusion**

* Key benefits of this solution
* Key achievements

**Introduction**

The project “**User Data Management with EC2 and S3 Integration**” demonstrates a cloud-based solution that enables users to submit data via a web-based form hosted on an Amazon EC2 instance. The submitted data is securely stored in an Amazon S3 bucket, ensuring high availability and durability. This solution leverages the scalability and flexibility of AWS services to efficiently collect, store, and manage user data.

Overview: -

The primary objective of this project is to create a seamless mechanism for capturing user input through a simple HTML form hosted on an EC2 instance. When a user submits the form, the data is sent directly to an S3 bucket for storage. This process involves the following key steps:

1. Hosting a static HTML and CSS form on an EC2 instance, making it publicly accessible.
2. Configuring S3 to store form submissions securely.
3. Implementing mechanisms to manage the stored data as needed.

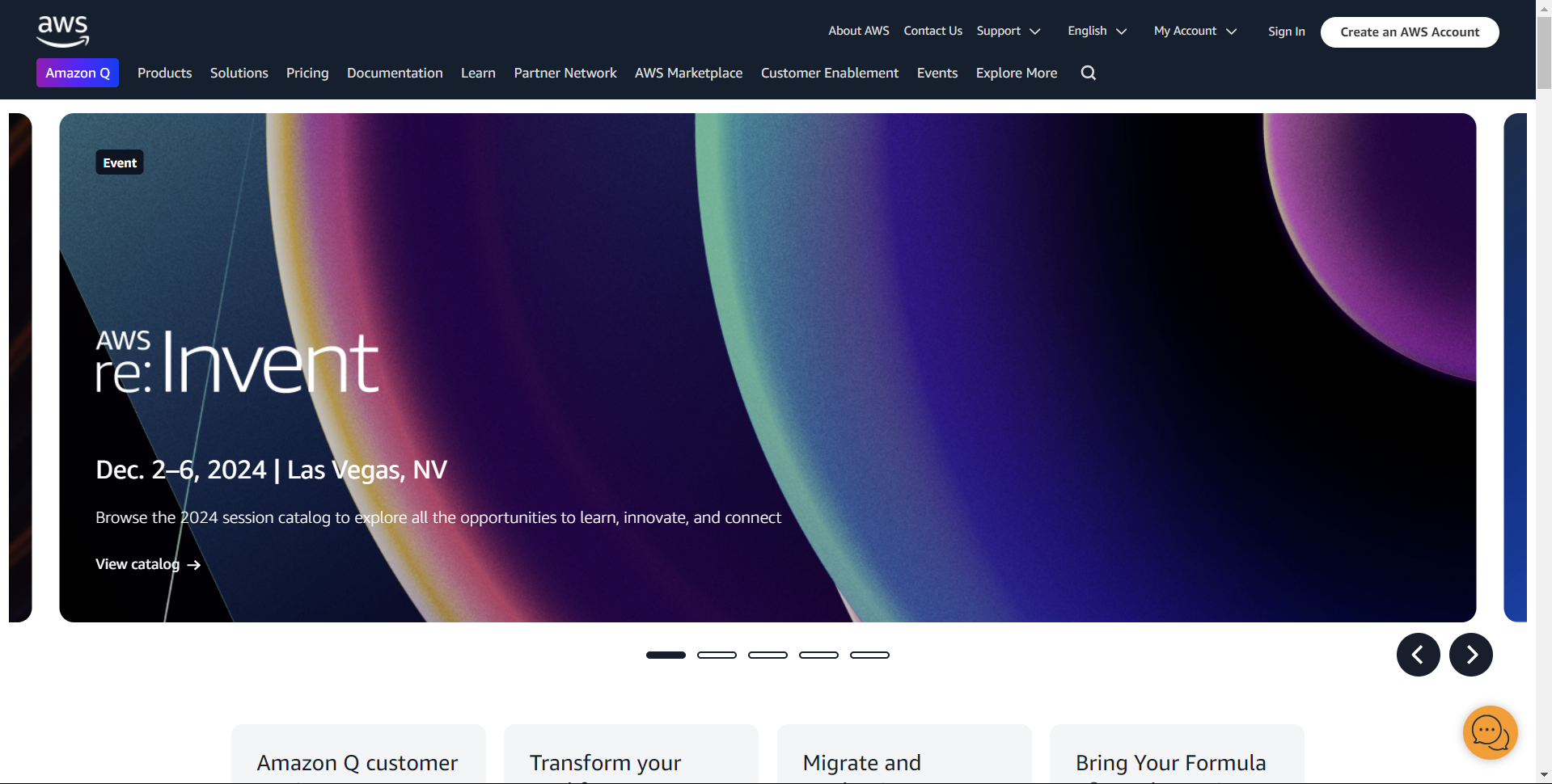
Benefits: -

* **Scalability**: The use of AWS EC2 and S3 ensures that the system can handle an increasing number of form submissions without performance degradation.
* **High Availability:** AWS services provide redundancy and failover capabilities, ensuring the form and data storage remain available at all times.
* **Security**: S3 offers robust access control mechanisms to protect user data, while HTTPS can be used to secure data transmission.
* **Cost-Efficiency**: Using S3 for storage is a cost-effective solution, as you only pay for what you use.
* **Simplicity**: The project demonstrates how to build and deploy a basic cloud-based data collection system with minimal infrastructure complexity.
* **Automation**: With proper configurations, the process of collecting and storing data is fully automated, reducing manual intervention.

**Prerequisites**

**AWS Account Setup: -**

Before you can start hosting websites on AWS, you need to set up an AWS account. Visit the [**AWS website**](https://aws.amazon.com/) and follow the sign-up process. Ensure you have a valid payment method and complete the necessary identity verification steps.



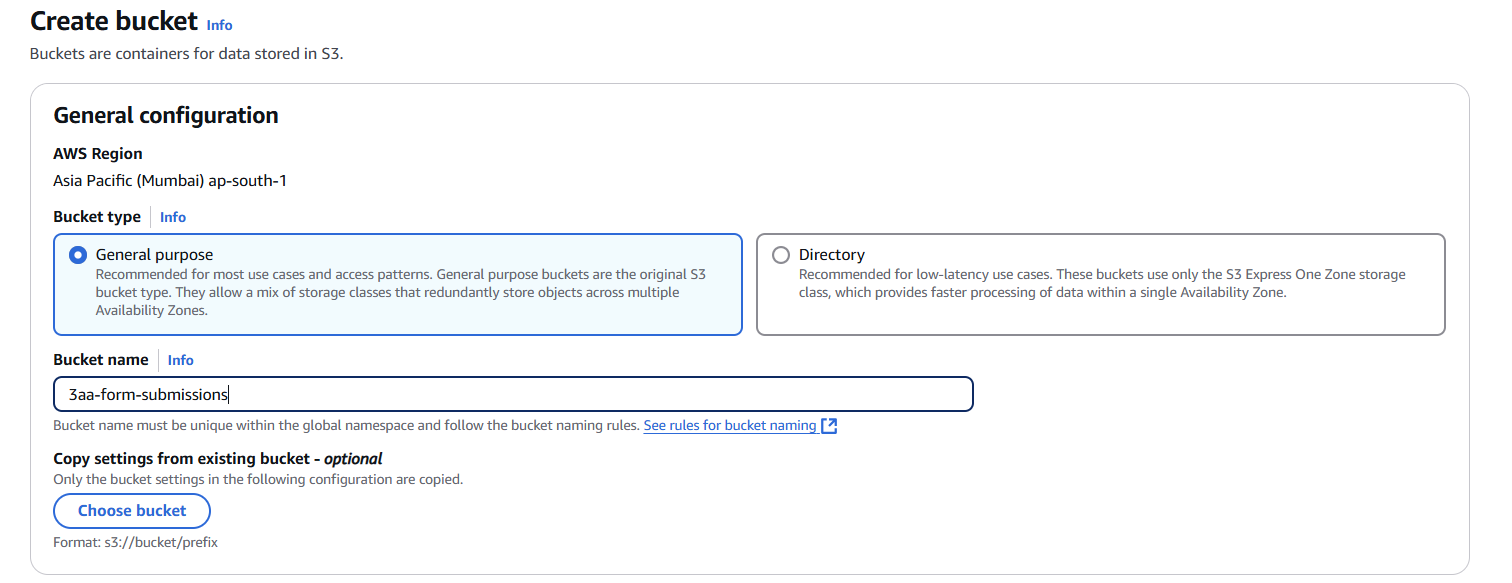
**Required Software and Tools:**

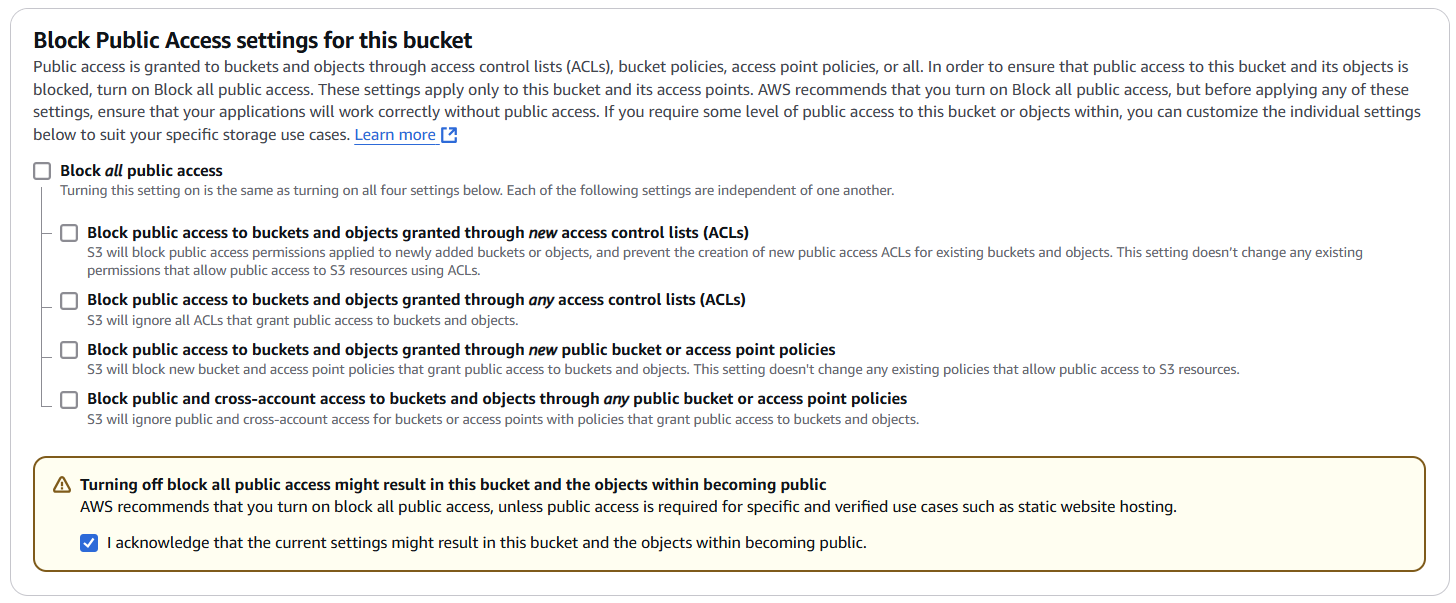
**AWS Services:**

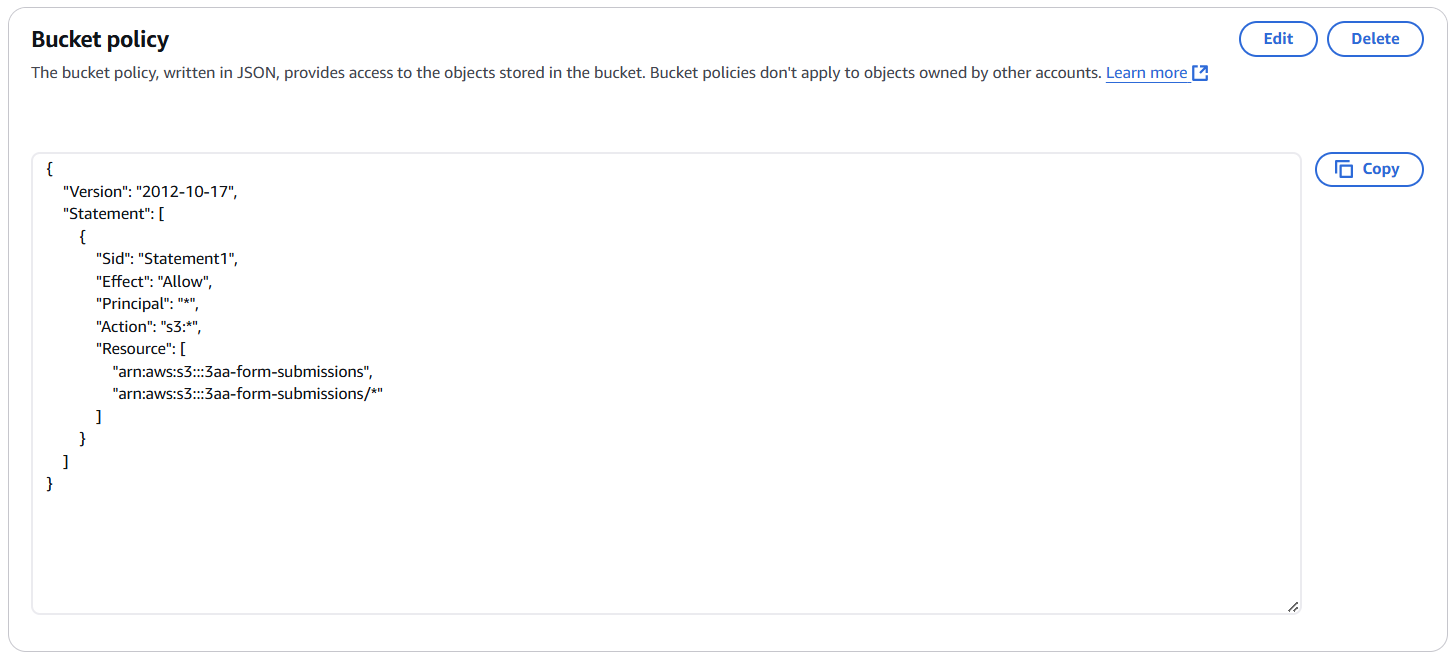
* **EC2 (Elastic Compute Cloud):**
  + Create and manage Amazon Linux instances.
  + Choose an appropriate instance type based on your workload (e.g., t2.micro for light loads, m5.large for more demanding applications).
  + Set up security groups to control inbound and outbound traffic.
* **Amazon S3 (Simple Storage Service):**
* Store user data securely in buckets.
* Configure bucket policies and Access Control Lists (ACLs) for managing access permissions.
* Enable versioning and lifecycle policies to optimize storage and costs.
* **AWS Lambda:**
  + Run backend logic to process form submissions without managing servers.
  + Set up triggers using API Gateway or directly from S3 events.
  + Automatically scale based on the number of incoming requests.
* **Amazon API Gateway:**
  + Create RESTful APIs to handle form submissions and route them to backend services.
  + Enable Cross-Origin Resource Sharing (CORS) to allow requests from the web form.
  + Configure security and throttling to control API usage.
* **IAM Roles and Policies:**
  + Create roles to grant EC2 and Lambda access to S3.
  + Use policies to define permissions for accessing specific resources.
  + Avoid hardcoding credentials by using role-based access control.

**Setting Up AWS S3**

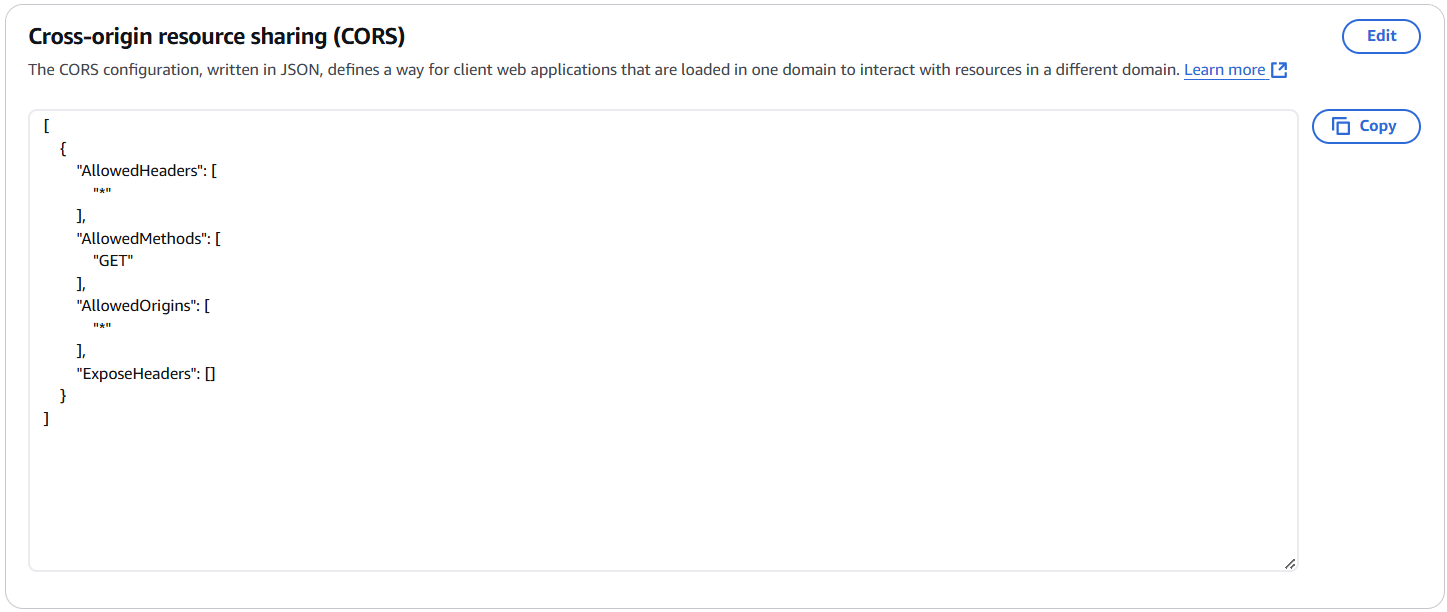
**Create a s3 bucket:**

[Name](https://aws.amazon.com/console/) the S3 bucket

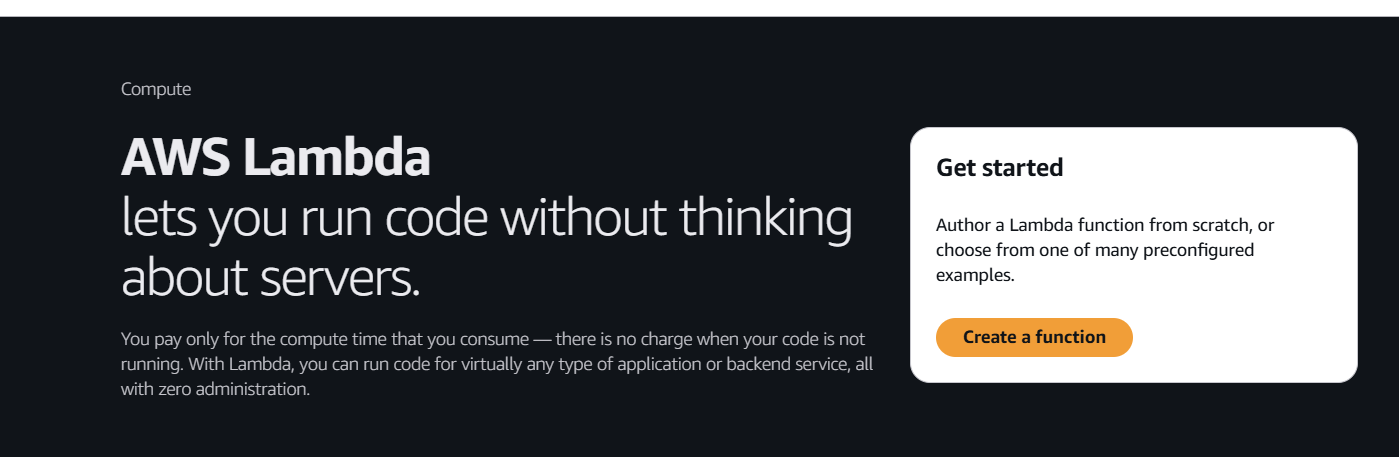
Unblock Public Access

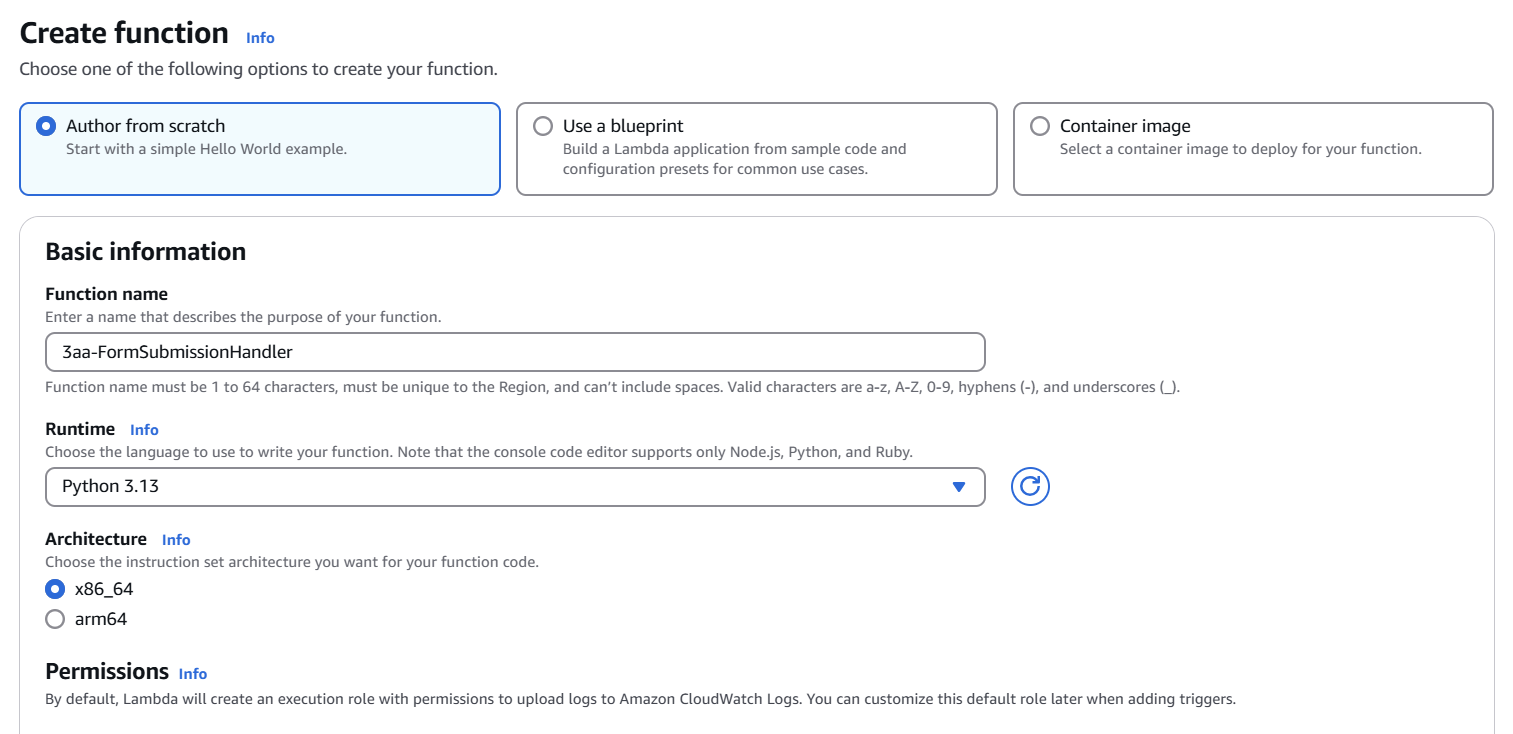
Edit Bucket Policy

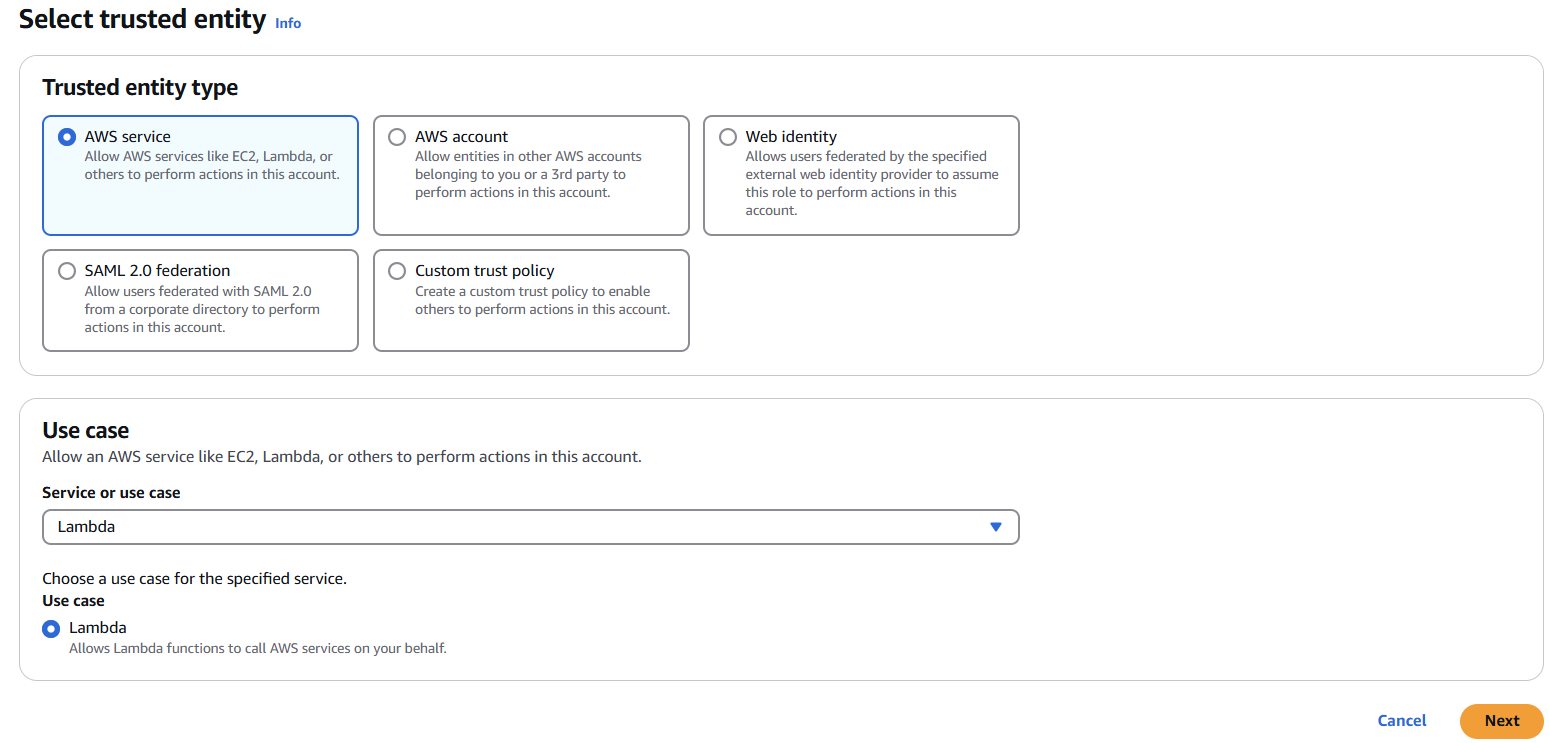
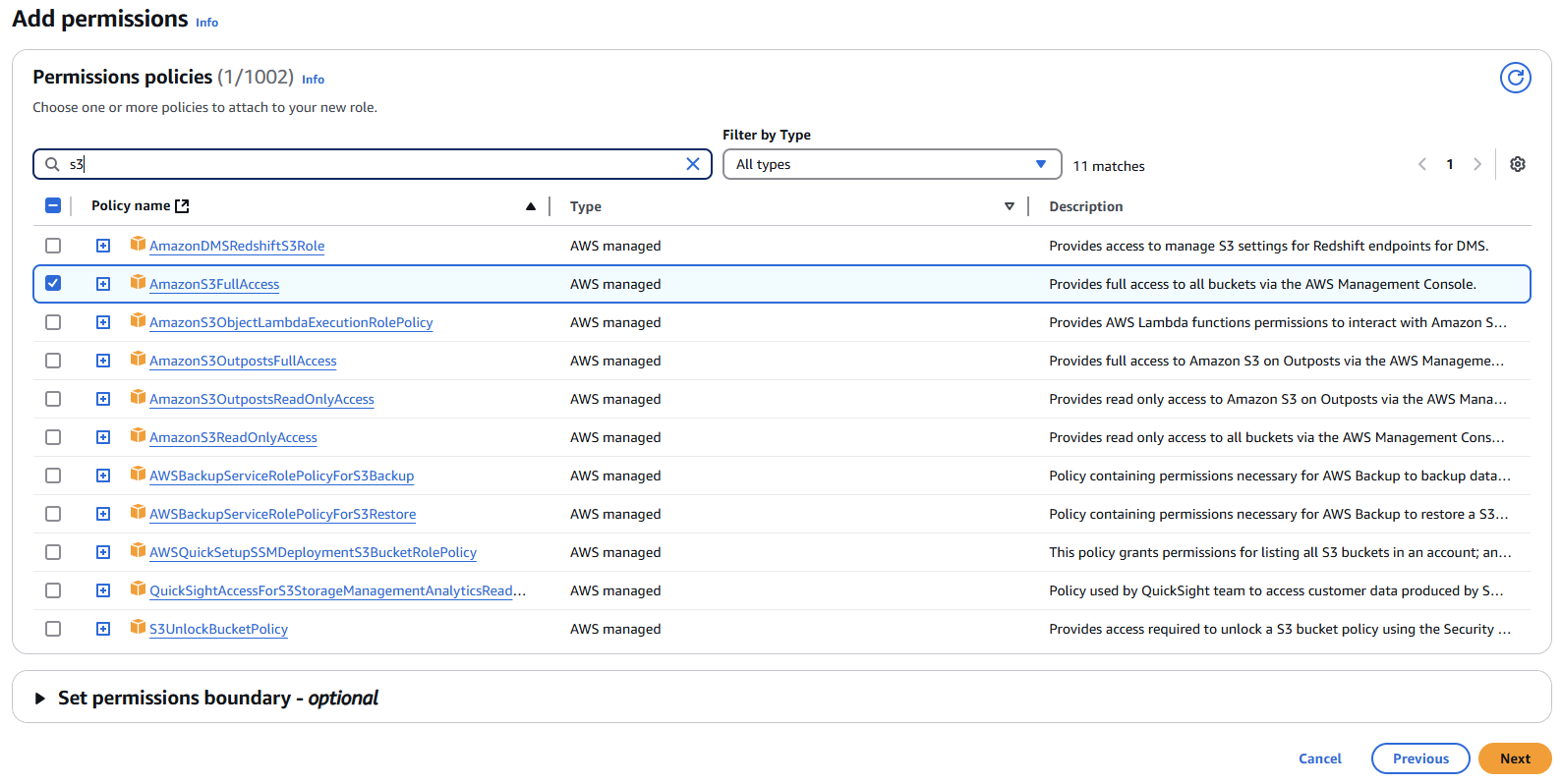
Below that edit CORS

In case of if you want to retrieve the data out of your AWS account (e.g. - To display the S3 data in a webpage).

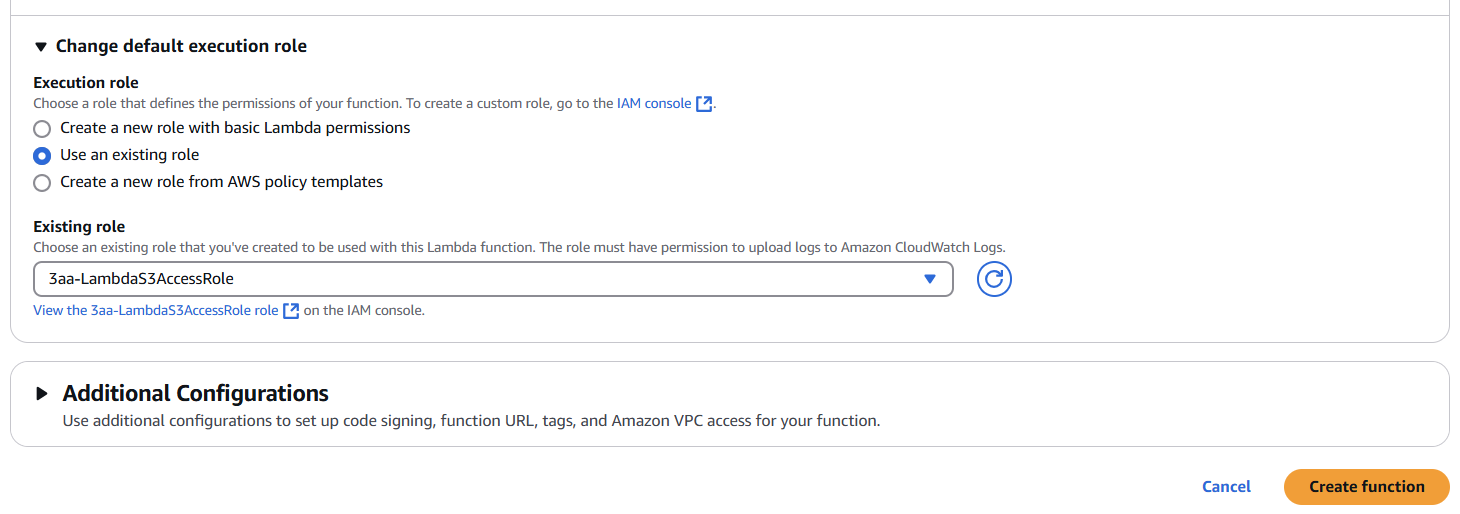
**Setting Up AWS Lambda**

Create a function

Name the function and choose the programming language

Create an IAM role to give lambda the access of S3



Select Execution role and create function

Add This code in Lambda\_function code and deploy

import json

import boto3

s3 = boto3.client('s3')

def lambda\_handler(event, context):

    bucket\_name = 'Yous S3 Bucket name'  # Replace with your bucket name

    try:

        body = event['body']

        data = json.loads(body)

         key = f"submissions/{data['name']}-{data['email']}.json"

        s3.put\_object(

            Bucket=bucket\_name,

            Key=key,

            Body=json.dumps(data),

            ContentType='application/json'

        )

        return {

        'statusCode': 200,

        'headers': {

            'Access-Control-Allow-Origin': '\* ',

            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET',

            'Access-Control-Allow-Headers': 'Content-Type, Authorization'

        },

        'body': json.dumps({'message': 'Success'})

    }

    except Exception as e:

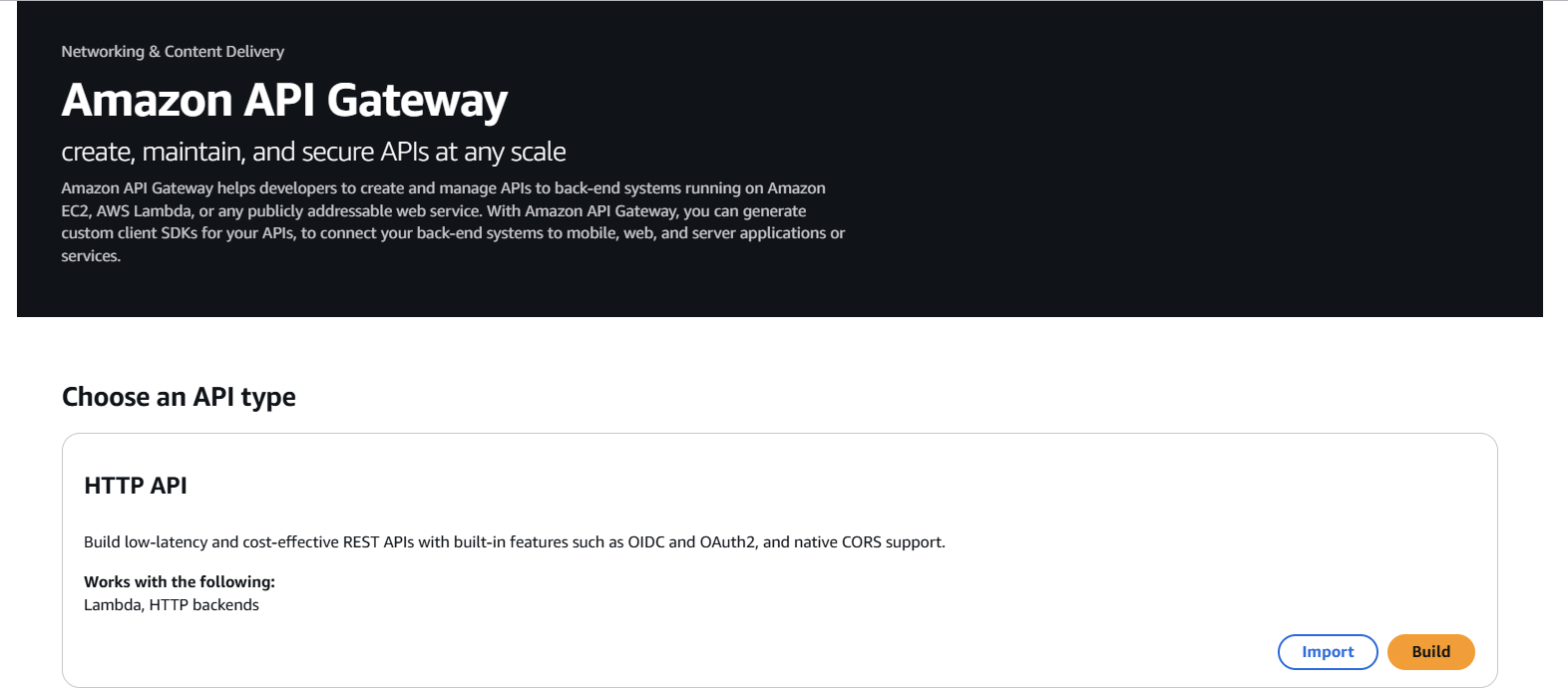
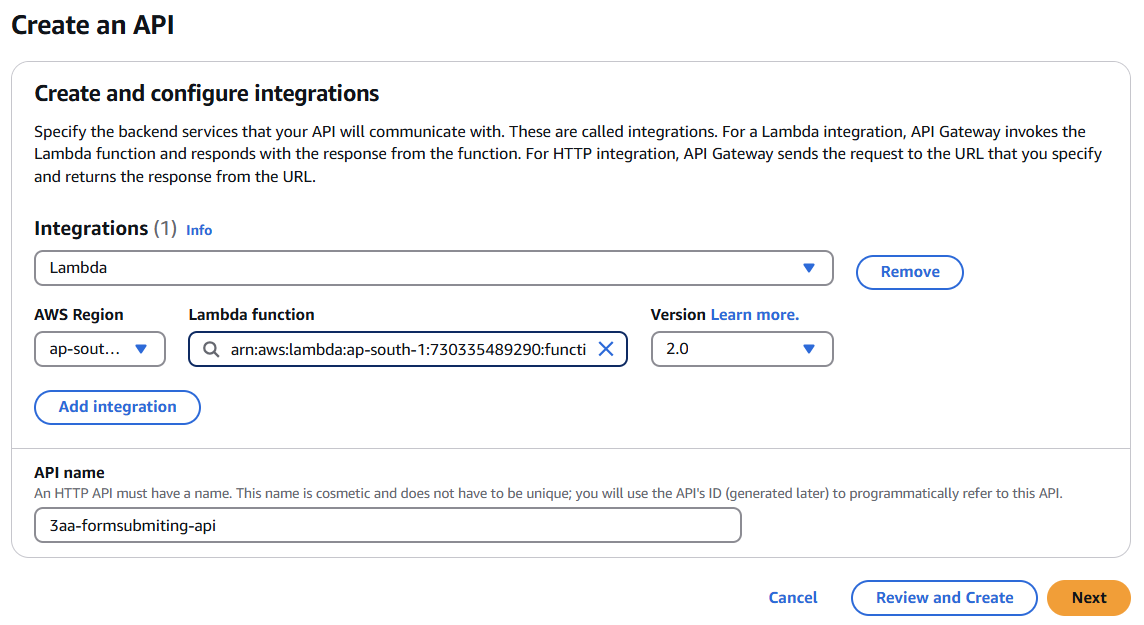
        return {

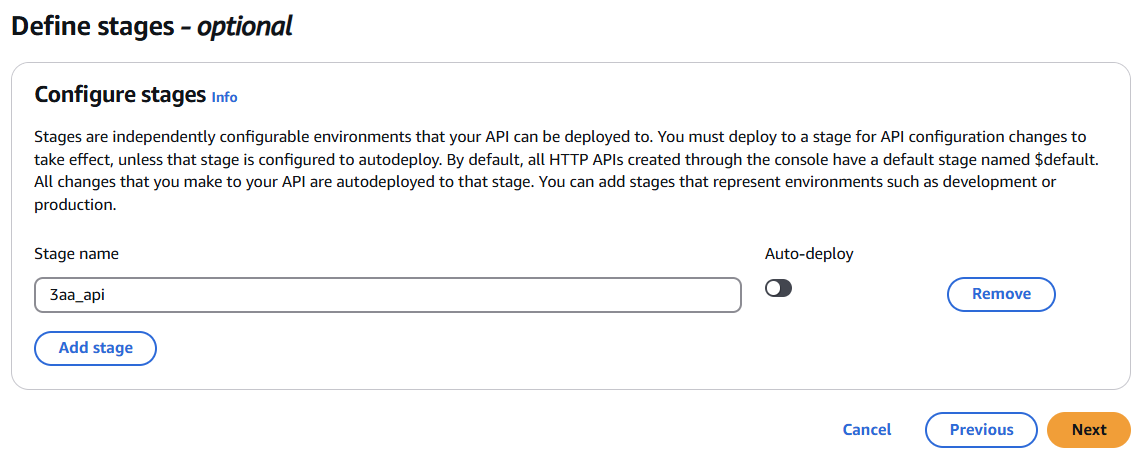
            'statusCode': 500,

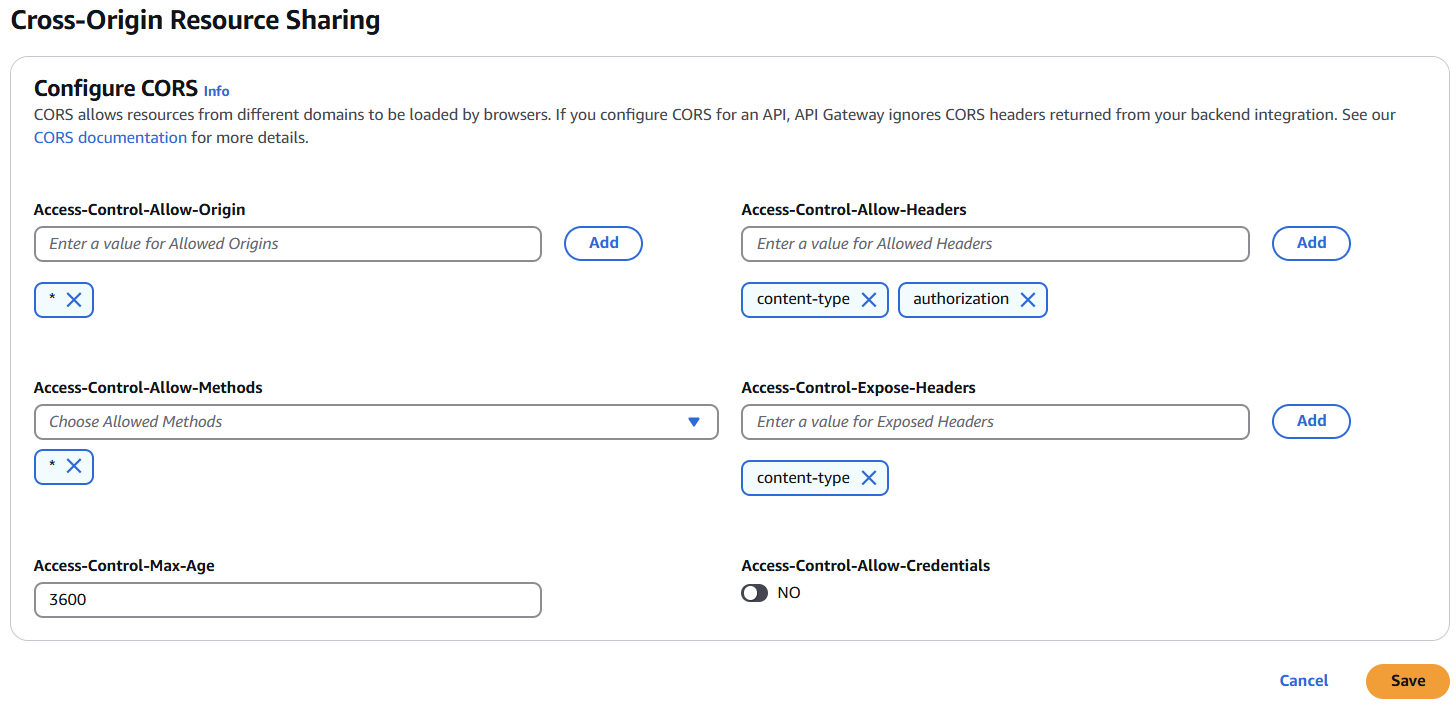
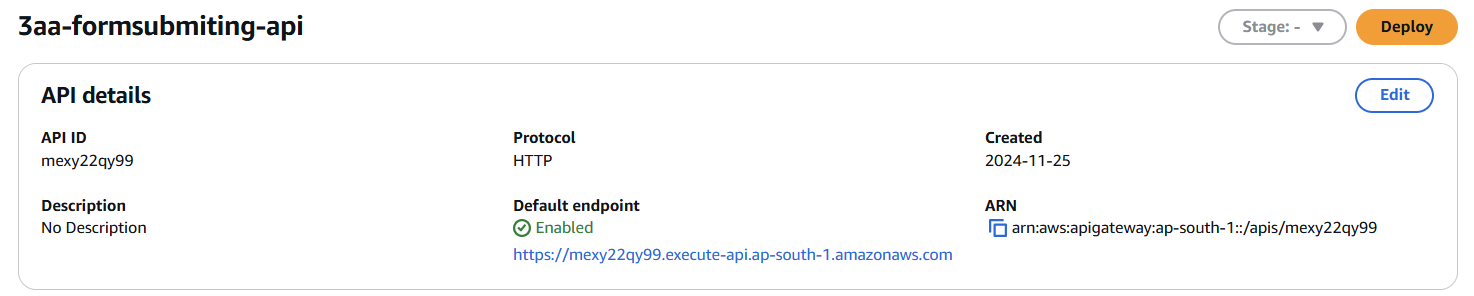
            'body': json.dumps({'error': str(e)})

        }

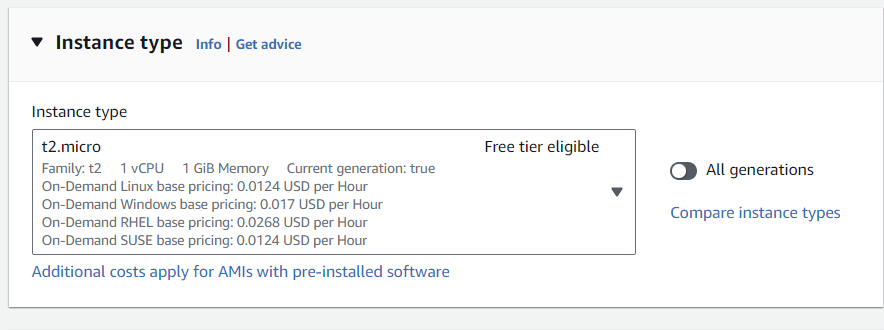
**Setting Up API Gateway**

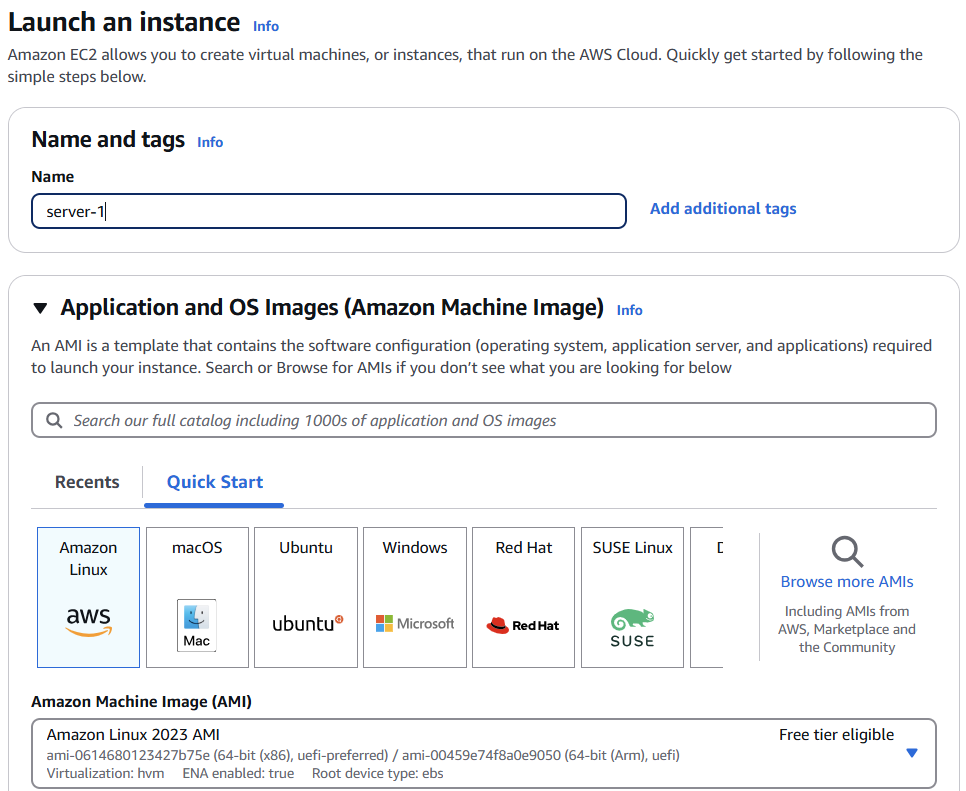
Create a http API

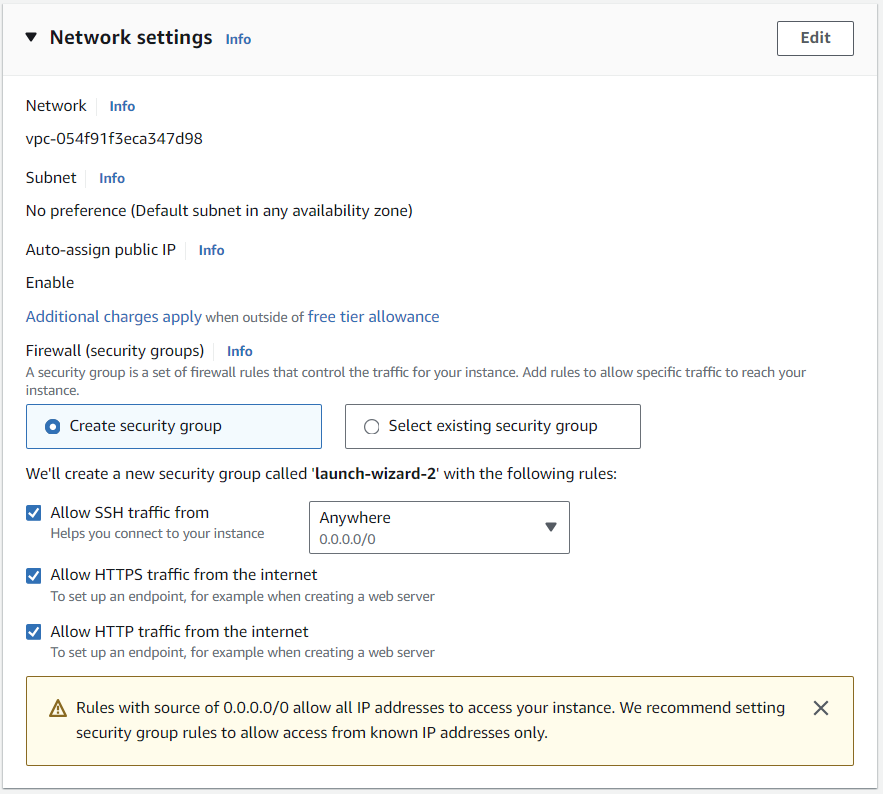
**Configure routes** and stage

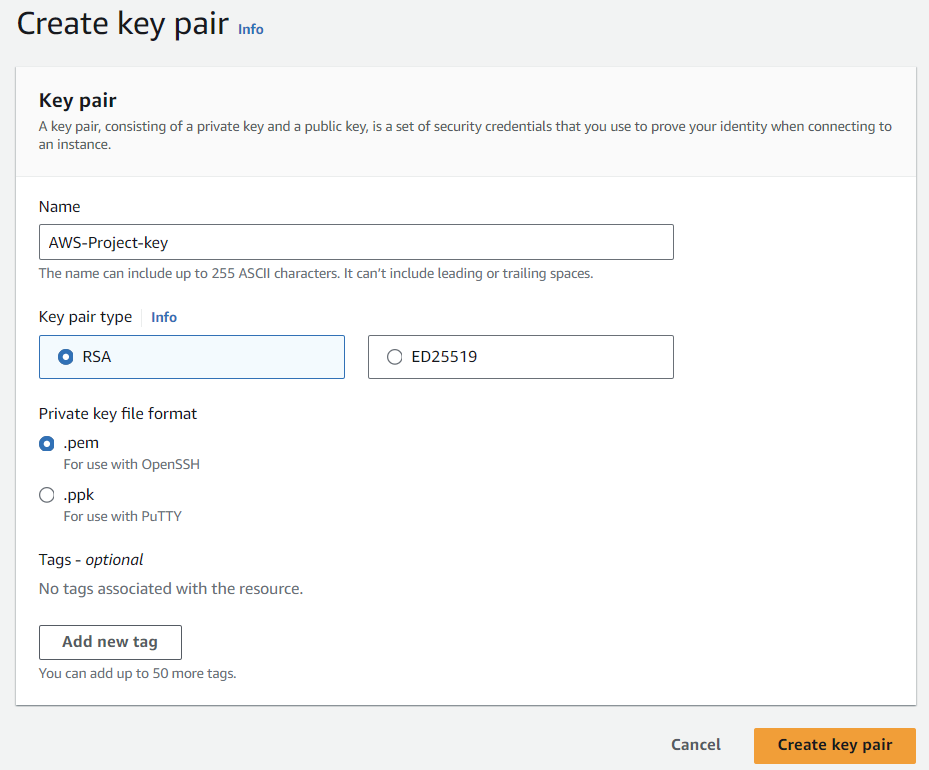
Configure the CORS and deploy

**Setting Up EC2 instance**

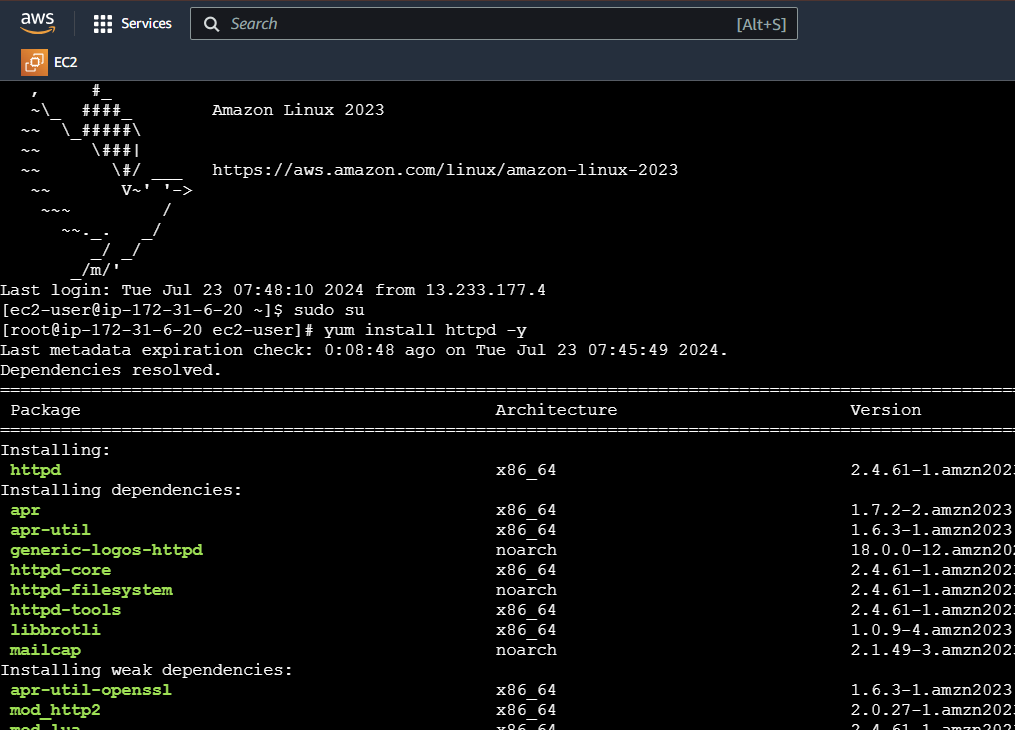
****Create An EC2 instance

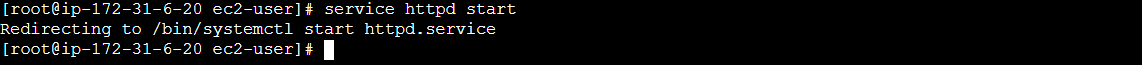
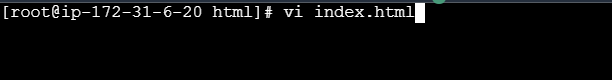


****



**Install Apache**



****

**Write the html code in index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Submit Data</title>

    <style>

        /\* Add style as you like \*/

    </style>

</head>

<body>

    <form id="dataForm">

        <label for="name">Name:</label>

        <input type="text" id="name" name="name" required><br>

        <label for="email">Email:</label>

        <input type="email" id="email" name="email" required><br>

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

    </form>

    <script>

       const form = document.getElementById('dataForm');

        form.addEventListener('submit', async (event) => {

            event.preventDefault();

            const formData = new FormData(form);

            const data = Object.fromEntries(formData.entries());

            // Send data to API Gateway

            const response = await fetch('Your API endpoint', {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/json'

                },

                body: JSON.stringify(data)

            });

            const result = await response.json();

            alert(result.message || result.error);

            form.reset();

        });

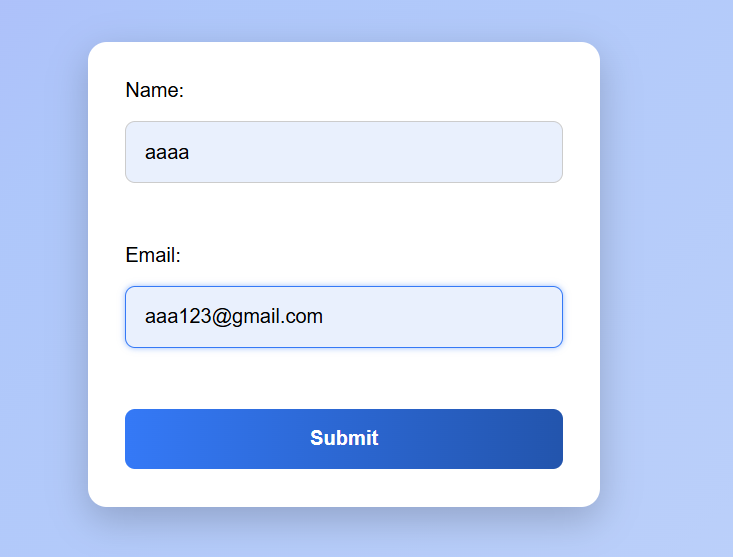
    </script>

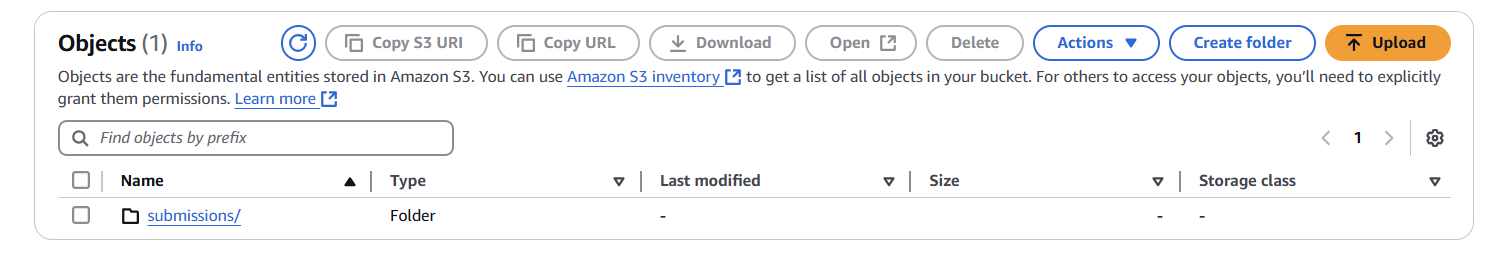
</body>

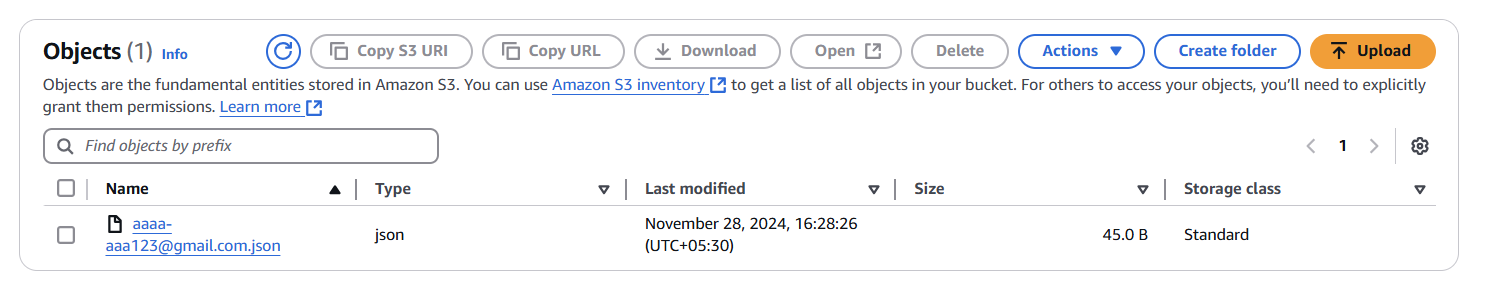
</html>

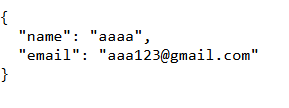
Replace the Your API endpoint with the endpoint URL of your API.

**fill the form and get the data into your S3**

****

****

****

****

**Conclusion**

**The User Data Management with EC2 and S3 Integration project demonstrates a scalable, secure, and cost-effective way to manage user data using AWS services. By hosting a static HTML form on EC2 and integrating it with S3 for data storage, the solution leverages AWS’s robust infrastructure to ensure high availability and durability.**

**Key benefits of this solution include:**

* **Scalability: Using EC2 and Auto Scaling, the solution can handle varying workloads efficiently.**
* **Security: IAM roles and policies enforce strict access control, ensuring data is protected.**
* **Cost-Effectiveness: Serverless options like Lambda and the pay-as-you-go model optimize costs.**
* **Reliability: S3 ensures data durability with redundancy across multiple availability zones.**

**This approach can be extended and customized for various use cases, making it a versatile solution for modern web applications requiring secure data handling.**

**Key achievements include:**

* **Successful Hosting of a Public Web Form**:  
  Deployed a static HTML form on an Amazon EC2 instance, accessible to users for data submission.
* **Secure Data Storage in Amazon S3**:  
  User-submitted data is securely stored in an S3 bucket with appropriate access controls and policies in place.
* **Seamless Integration of AWS Services**:  
  Achieved seamless integration between EC2, S3, API Gateway, and Lambda (if applicable) to handle user data efficiently.
* **Implementation of Access Control and Security**:  
  Configured IAM roles and policies to ensure secure communication and resource access between AWS services, avoiding the use of hardcoded credentials.
* **Scalability and Reliability**:  
  Leveraged AWS’s scalable infrastructure to handle increasing workloads, ensuring reliability and high availability of the system.
* **Cost Optimization**:  
  Utilized serverless computing (Lambda) and S3’s storage tiers to optimize costs, paying only for resources used.

**Lessons learned during the project highlight the importance of:**

* **Proper Resource Configuration**:  
  Setting up security groups, IAM roles, and policies correctly is crucial to ensure secure and seamless access between AWS services. Misconfigurations can lead to access issues and potential security risks.
* **CORS and API Integration**:  
  Managing **CORS (Cross-Origin Resource Sharing)** settings in API Gateway is vital for enabling cross-domain requests from the frontend to backend services. This ensures a smooth user experience without errors.
* **Data Privacy and Security**:  
  Storing user data in **S3** requires implementing proper access controls, encryption, and lifecycle policies to protect sensitive information and comply with security standards.
* **Automation and Scalability**:  
  Automating tasks using **Lambda** and leveraging **Auto Scaling** helps handle workload spikes efficiently, ensuring high availability and performance without manual intervention.
* **Cost Management**:  
  Understanding AWS's pricing models and using services like S3 lifecycle policies and Lambda's pay-as-you-go model can significantly reduce operational costs while maintaining efficiency.
* **Testing and Debugging**:  
  Thorough testing of form submissions, API endpoints, and data handling processes is critical to identify and resolve issues like access denial, incorrect configurations, or network-related errors.