SHRI G.S, INSTITUTE OF TECHNOLOGY & SCIENCE, INDORE- 452003

2025-26

CLOUD COMPUTING

IT-48330

SESSION ASSIGNMENT PROJECT

Submitted To

Ms. Sunita Verma

Ms. Puja Gupta

Submitted by

Name – Rohit Singh Rathor

Enrollment - 0801IT233D09

Name - Adarsh Agrawal

Enrollment – 0801IT221150

Serverless Contact Form

1. Abstract

This project implements a **serverless contact form** using **AWS Lambda**, **Amazon API Gateway**, **Amazon Simple Email Service (SES)**, and **Amazon S3**. Users submit a simple HTML form hosted on S3. A request to an API Gateway endpoint triggers a Lambda function that validates the input and sends the form contents as an email via SES. The design is cost-efficient, scalable, and requires minimal infrastructure management.

---

2. Introduction

Contact forms are a common feature for websites to collect user feedback, enquiries, or leads. A serverless approach removes the need to run or maintain servers, automatically scales with traffic, and lowers initial costs. AWS Lambda handles form processing, API Gateway securely exposes the Lambda function to the internet, S3 hosts the static form HTML, and SES sends notification emails.

---

3. Purpose of Work

- Provide a minimal, production-ready serverless contact form.
- Demonstrate integration between S3, API Gateway, Lambda, and SES.
- Ensure messages are delivered reliably to a configurable recipient email.
- Show best practices: validation, CORS, IAM least-privilege, and SES verification.

---

4. Working Steps Diagram (Flow)

```
[User Browser]
   ↓ (submit)
[S3-hosted HTML form w/ JS fetch]
   ↓ (HTTPS POST)
[API Gateway]
   ↓ (invoke)
[AWS Lambda Function]
   → (use aws-sdk / boto3)
[Amazon SES]
   ↓
[Recipient Email Inbox]
```

---

5. System Architecture

- **Frontend (Static):** S3 (with static website hosting) or CloudFront + S3 for secure hosting.
- **API:** Amazon API Gateway (REST or HTTP API) exposes the POST endpoint to the web and handles CORS.
- **Compute:** AWS Lambda processes incoming traffic and calls SES.
- **Email Delivery:** Amazon SES sends the email to the configured recipient.
- **IAM:** Role for Lambda with permissions to call SES (`SendEmail`, `SendRawEmail`).
- **Optional:** CloudWatch logs for Lambda, AWS WAF for API protection, Secrets Manager for credentials (if using SMTP).

---

6. Hardware / Software Requirements

Hardware: none (serverless) — developer needs a workstation to write code and deploy.

Software / Tools:

- AWS account with SES in a supported region.
- AWS CLI (optional but recommended).
- Node.js (for frontend / bundling) or plain HTML/JS.
- Python 3.9+ (or Node.js 18+) for Lambda code (examples provided).
- (Optional) Terraform / CloudFormation / SAM for infra-as-code.

---

7. Step-by-step Implementation (Condensed, ordered)

A. SES: Verify and configure

1. Open AWS Console → Amazon SES (choose correct region).
2. **If SES is in Sandbox:** request production access (otherwise you can only send to verified addresses).
3. **Verify sending identity**:
   - SES → Verified Identities → Create identity → Type: `email` or `domain`.
   - Enter the sender email (e.g., `no-reply@yourdomain.com`) and verify ownership by clicking the emailed link.
4. (Optional) Generate SMTP credentials if you plan to use SMTP instead of the SES API:
   - SES → SMTP settings → Create SMTP credentials (creates an IAM user and returns SMTP username/password). Save the `.csv`.

Important: Lambda using the SES API does *not* require SMTP credentials — it needs IAM permissions for SES actions on the Lambda execution role.

B. IAM role and policy for Lambda

Create an IAM policy that allows SES send actions, then attach to the Lambda execution role.

Example minimal policy (JSON):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSESSend",
      "Effect": "Allow",
      "Action": [
        "ses:SendEmail",
        "ses:SendRawEmail"
      ],
      "Resource": "*"
    }
  ]
}
```

C. Create Lambda function

1. AWS Console → Lambda → Create function.
2. Runtime: **Python 3.9+** or **Node.js 18+** (example below uses Python).
3. Assign the execution role created above.
4. Add environment variables:
    - `SENDER_EMAIL` — the verified SES sender email.
    - `RECIPIENT_EMAIL` — where form submissions will be delivered.
    - (Optional) `AWS_REGION` — SES region.

Python Lambda sample (handler.py):

```
import os
import json
import boto3
import re

ses = boto3.client('ses', region_name=os.environ.get('AWS_REGION', 'us-
east-1'))

SENDER = os.environ['SENDER_EMAIL']
RECIPIENT = os.environ['RECIPIENT_EMAIL']

def validate_email(email):
    return re.match(r"[^@]+@[^@]+\.[^@]+", email)

def lambda_handler(event, context):
    # For API Gateway HTTP API, body may be JSON string
    try:
        body = event.get('body')
        if isinstance(body, str):
            data = json.loads(body)
        else:
            data = body or {}
    except Exception:
        return {"statusCode":400, "body":"Invalid request body"}

    name = data.get('name', 'Anonymous')
    email = data.get('email', '')
    message = data.get('message', '')

    if not message or not email or not validate_email(email):
        return {"statusCode":400, "body":"Missing or invalid fields"}

    subject = f"Website Contact Form: {name}"
    body_text = f"Name: {name}\nEmail: {email}\n\nMessage:\n{message}"
    body_html = f"""
    <html>
    <body>
      <h2>Contact Form Submission</h2>
      <p><strong>Name:</strong> {name}</p>
      <p><strong>Email:</strong> {email}</p>
      <p><strong>Message:</strong><br/>{message.replace('\\n','<br/>')}</p>
    </body>
    </html>
    """

    try:
        resp = ses.send_email(
            Source=SENDER,
            Destination={'ToAddresses':[RECIPIENT]},
            Message={
                'Subject': {'Data': subject},
                'Body': {
                    'Text': {'Data': body_text},
                    'Html': {'Data': body_html}
                }
```

```
            }
        )
        return {"statusCode":200, "body": json.dumps({"message":"Email
sent","id":resp.get('MessageId')})}
    except Exception as e:
        return {"statusCode":500, "body": f"Error sending email: {str(e)}"}
```

If you want attachments or richer formatting, use `SendRawEmail` with a MIME message.

D. API Gateway

1. Create a new API (HTTP API recommended for simplicity).
2. Create a POST route `/contact` and integrate with Lambda function.
3. **Enable CORS** for the origin that hosts your HTML (e.g., `https://your-bucket.s3.amazonaws.com` or `https://yourdomain.com`). Allow `Content-Type`, `Origin`, `Accept`.
4. Deploy the API and note the endpoint URL, e.g. `https://{id}.execute-api.{region}.amazonaws.com/contact`.

Security notes:

- If this form is public, implement at least rate-limiting, AWS WAF, or simple spam protection (reCAPTCHA) to prevent abuse and SES sending limits exhaustion.
- For production, prefer API Gateway Authorizers (JWT/Cognito) or CAPTCHA to deter spam.

E. Host HTML form on S3

1. Create S3 bucket (unique name).
2. Enable static website hosting or serve through CloudFront.
3. Upload `index.html` (form) and set appropriate bucket policy for public read (or restrict and use CloudFront with origin access).
4. Example minimal `index.html` using Fetch to call API Gateway:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Contact Us</title>
</head>
<body>
  <h1>Contact Us</h1>
  <form id="contactForm">
    <input name="name" placeholder="Your name"/><br/>
    <input name="email" placeholder="Your email"/><br/>
    <textarea name="message" placeholder="Your message"></textarea><br/>
    <button type="submit">Send</button>
  </form>

  <script>
    const apiEndpoint = "https://{your-api-id}.execute-
api.{region}.amazonaws.com/contact"; // replace

    document.getElementById('contactForm').addEventListener('submit', async
(e) => {
      e.preventDefault();
      const form = e.target;
      const data = {
```

```
        name: form.name.value,
        email: form.email.value,
        message: form.message.value
      };
      try {
        const res = await fetch(apiEndpoint, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify(data)
        });
        const result = await res.json();
        if (res.ok) alert('Message sent!');
        else alert('Error: ' + JSON.stringify(result));
      } catch (err) {
        alert('Network error');
      }
    });
  </script>
</body>
</html>
```

F. Enable CORS & Deploy API

- In API Gateway, enable CORS for `POST /contact` or from the HTTP API console enable CORS with allowed origins and methods.
- Deploy/Publish the API.

G. Test: Send an email

1. Open the S3-hosted form URL.
2. Submit the form.
3. Verify that:
    - API returns HTTP 200 and JSON success.
    - Lambda logs show the invocation (CloudWatch).
    - SES delivered the email to the recipient inbox (check spam folder).
4. If SES is in **sandbox**, ensure recipient is verified OR move out of sandbox.

---

8. Conclusion

This serverless contact form provides a straightforward, maintainable, and scalable way to collect messages from users and deliver them via SES. It requires minimal infrastructure, and can be hardened with authentication, reCAPTCHA, DDoS/WAF protection, and improved monitoring.

---

9. GitHub Link

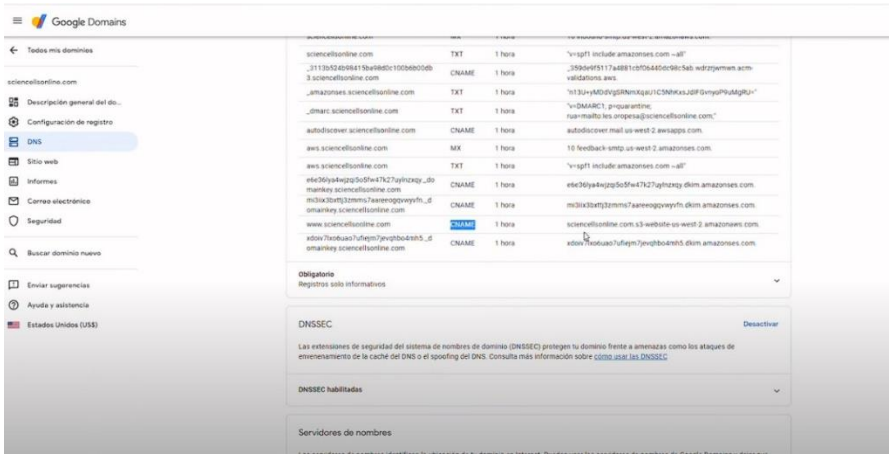
https://github.com/rsr605/ServelessContactFormwithAWS.git

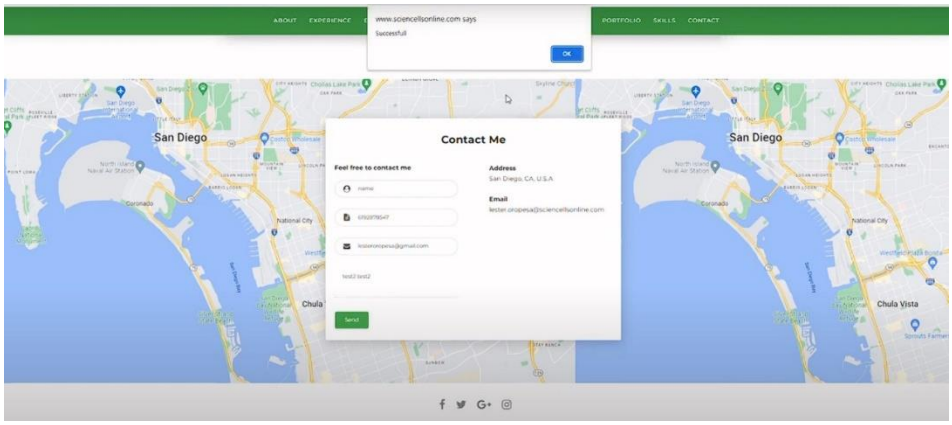
---

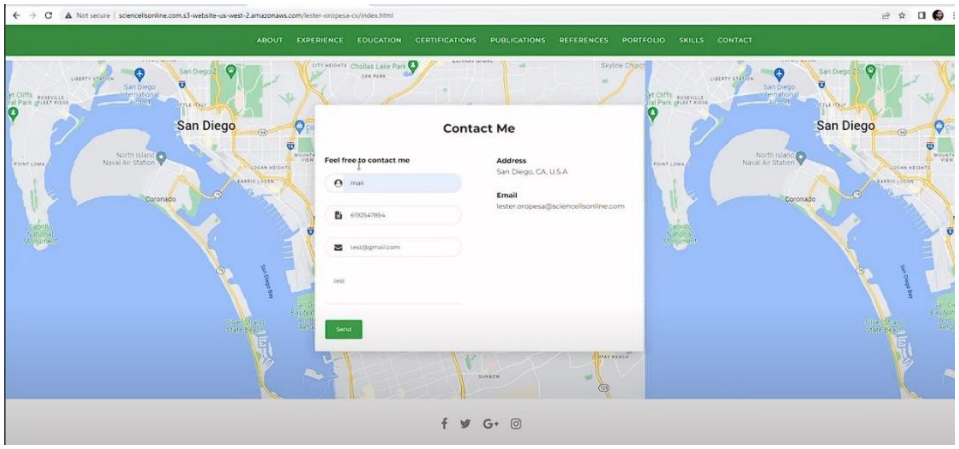
10. References

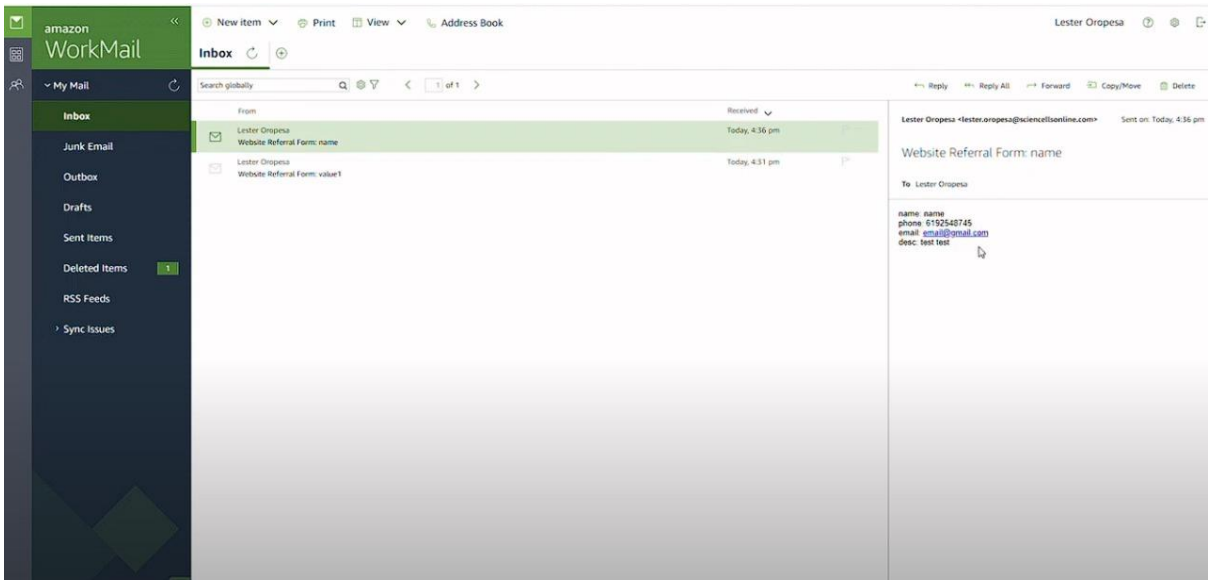
- AWS Lambda docs — https://docs.aws.amazon.com/lambda/
- Amazon SES docs — https://docs.aws.amazon.com/ses/
- Amazon API Gateway docs — https://docs.aws.amazon.com/apigateway/
- AWS S3 static hosting — https://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteHosting.html

11. Screenshots









12. Video

Untitled.mp4