DAILY TASK REMAINDER VIA EMAIL USING AWS

Prepared in the partial fulfillment of the Summer Internship Program on AWS

AT

Under the guidance of

Mrs. Sumana Bethala, APSSDC

Mr. Anil, APSSDC

Submitted by

Batch No: 195

KARRI BHAVYA SATYA SRI (23P31A0533)

AKKINA SRAVANTHI SAI LAKSHMI (23MH1A05E5)

GOVVALA HARIKA DEEPTHI (23P31A0518)

JAGU HANSIKA (23P31A05G1)

Aditya College of Engineering and Technology

(july, 2025)

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of my summer internship project at **Andhra Pradesh Skill Development Corporation (APSSDC)**. This opportunity has been an enriching and transformative experience for me, and I am truly thankful for the support, guidance, and encouragement I have received along the way.

First and foremost, I extend my sincere regards to Mrs Sumana Bethala and Mr Rama Krishna, my supervisors and mentors, for providing me with valuable insights, constant guidance, and unwavering support throughout the duration of the internship. Their expertise and encouragement have been instrumental in shaping the direction of this project.

I would like to thank the entire team at **Andhra Pradesh Skill Development Corporation (APSSDC)** for fostering a collaborative and innovative environment. The camaraderie, knowledge sharing, and feedback I received from my colleagues significantly contributed to the development and success of this project.

In conclusion, I am honoured to have been a part of this internship program, and I look forward to leveraging the skills and knowledge gained to contribute positively to future endeavours.

Thank you.

ABSTRACT

The Daily Task Reminder via Email project is a cloud-based solution designed to help users manage their daily tasks effectively. It allows users to submit tasks with specific dates and times through a simple web interface. These task details are stored securely in a cloud database (AWS RDS). A Flask-based backend periodically checks for upcoming tasks and automatically sends email reminders to users using Amazon SES or SMTP. The system is deployed on Amazon EC2, while Amazon CloudWatch handles task scheduling and logging. This project demonstrates the integration of web development with cloud computing services to automate personal task management and improve productivity through timely reminders.

Table of contents

1. INTRODUCTION

In today's fast-paced digital landscape, staying organized and on top of tasks is crucial for productivity and success. Our daily task reminder system leverages cloud computing and DevOps practices to streamline task management and boost productivity. By utilizing serverless functions on cloud platforms like AWS or Google Cloud, we can automate reminders and send them to users via email services like Amazon SES or IAM. With DevOps-driven documentation, we ensure seamless updates and maintenance of the system, enabling users to stay organized and focused on their tasks. This cloud-based solution scales effortlessly to meet growing demands, providing a reliable and efficient way to manage daily tasks and deadlines.

Cloud computing is the backbone of our daily task reminder system. It provides the necessary infrastructure and services to support the automation of reminders and email notifications. Cloud platform like AWS offer a range of services that can be used to build and deploy the system.

DevOps practices are essential for ensuring the smooth operation and maintenance of the system. By adopting DevOps principles, we can automate testing, deployment, and monitoring of the system, reducing the risk of errors and downtime. DevOps-driven documentation also enables us to track changes and updates to the system, ensuring that all stakeholders are informed and aligned.

An email service is used to send automated reminders to users. Cloud-based email services like Amazon SES, EC2, IAM, Amazon CloudWatch Logs or Amazon S3 provide a reliable and scalable way to send emails, reducing the risk of deliverability issues and spam filtering.

Increased Productivity

Automated reminders ensure that tasks are completed on time, reducing the likelihood of missed deadlines. By staying organized and focused, users can prioritize tasks more effectively, leading to increased productivity and better work outcomes.

Improved Organization

Cloud-based storage and documentation enable easy access to task information and reminders. Users can view and manage their tasks from anywhere, at any time, reducing the risk of lost or forgotten tasks.

Scalability

Cloud computing resources can be scaled up or down as needed, ensuring that the reminder system can handle a large volume of users and tasks. This scalability also reduces the risk of downtime and errors, providing a reliable and efficient solution for task management.

Our daily task reminder system via email using cloud computing with DevOps for documentation provides a reliable and efficient way to manage daily tasks and deadlines. By leveraging cloud computing and DevOps practices, we can automate reminders, improve organization, and increase productivity. With its scalability and flexibility, this solution is ideal for individuals and organizations looking to streamline task management and boost productivity. instance with an Ubuntu AMI, we can ensure a scalable and reliable hosting environment that can handle varying levels of demand.

The following AWS services are utilized in this project:

Amazon EC2 (Elastic Compute Cloud): The application is hosted on Amazon EC2 instance, which provide resizable compute capacity in the cloud. EC2 instances act as the application servers, handling user requests and managing the system's backend processes.

S3 (Simple Storage Service): S3 provides scalable and durable storage for task data and

reminders, ensuring that information is readily available and securely stored. With its virtually unlimited storage capacity, S3 can handle large amounts of task data without compromising performance.

IAM (Identity and Access Management): IAM ensures secure and controlled access to AWS resources, enabling administrators to define precise permissions and access levels for users and resources. With IAM, identity federation allows users to access AWS resources using existing credentials from external identity providers, streamlining access management. IAM's fine-grained access control and security features help ensure that AWS resources are accessed securely and in compliance with organizational policies and regulatory requirements, reducing the risk of unauthorized access.

SES (Simple Email Service): SES provides a scalable and reliable email delivery service for sending automated reminders and notifications to users. With its high deliverability rates, SES helps ensure that emails reach users' inboxes, reducing the risk of spam filtering and deliverability issues. SES offers a cost-effective solution for sending emails, with pricing based on the number of emails sent, making it an ideal choice for applications requiring large volumes of email notifications. By integrating SES with the daily task reminder system, users receive timely and reliable reminders, enhancing their productivity and task management experience.

2. METHODOLOGY

The methodology for the daily task reminder system project involves a structured approach to ensure that the system meets user needs and expectations. We follow a phased approach, starting with requirements gathering and proceeding through design, development, testing, deployment, and maintenance. This approach enables us to deliver a high-quality system that is reliable, scalable, and secure. By following a structured methodology, we can ensure that the system is delivered on time and within budget.

2.1. Requirements Gathering

The requirements gathering phase involves identifying the project's functional and nonfunctional requirements. This includes conducting stakeholder interviews, surveys, or focus groups to understand user needs and expectations.

2.2. Design

The design phase focuses on creating a comprehensive system architecture that meets the requirements gathered in the previous phase. This includes designing the overall system architecture, database schema, and user interface. We select the AWS services to be used, such as S3 for storage, IAM for access control, and SES for email notifications. The design phase also involves creating a user-friendly interface that allows users to interact with the system seamlessly.

2.3. Deployment

The development phase involves building the daily task reminder system using Agile methodologies, such as Scrum. The system is integrated with AWS services, such as S3, IAM, and SES, to provide a seamless user experience. Throughout the development phase, we

conduct iterative testing and refinement to ensure that the system meets requirements and is free of defects.

2.4. Testing

The testing phase involves conducting thorough testing to ensure that the system meets requirements and is free of defects. This includes unit testing, integration testing, and user acceptance testing. Testing is a critical phase of the daily task reminder system project, as it ensures that the system is reliable and scalable.

2.5. Development

The deployment phase involves deploying the system to a cloud environment, such as AWS, using DevOps practices. We use continuous integration and continuous deployment to ensure seamless deployment and monitoring.

2.6. Maintenace

The maintenance phase involves providing ongoing support and maintenance to ensure that the system continues to meet user needs and expectations. This includes monitoring system performance, applying security updates and patches, and optimizing system performance as needed. . Ongoing maintenance is critical to the success of the daily task reminder system project, as it ensures that the system continues to meet user needs and expectations over time.

3. SYSTEM DESIGN / ARCHITECTURE

◇ 1. User Interface

- A simple **HTML form** where users enter:
  - Task details
  - Date & Time
  - Email address

◇ 2. Flask Application (on EC2)

- Flask app runs on an **Amazon EC2 instance**
- Receives form data and processes it
- Stores task details in the database
- Scheduled code checks task times and sends reminders

◇ 3. Amazon SES (or Gmail SMTP)

- Sends automated **email reminders**
- Uses the email ID collected from users
- SES is a secure and scalable way to send emails

◇ 4. Amazon CloudWatch

- **Schedules** the Flask script to run at intervals
- **Logs** reminder activity and errors

◇ 5. Amazon S3

- Store **logs**, **reports**, or **task backups**
- Not required but adds value

4. IMPLEMENTATION

The implementation phase involved developing a fully functional and scalable **task reminder web application** integrated with AWS services. The system enables users to create, manage, and receive reminders for tasks via automated daily emails.

4.1 Frontend Development

- **Technologies Used**: HTML5, CSS3, and JavaScript (with optional jQuery for interactivity).
- **Responsive Design**: The layout is mobile-first and adjusts fluidly across devices.
- **Key Features**:
  - **User Interfaces**: Separate pages for registration, login, and dashboard views.
  - **Forms**: Dynamic form validations using HTML5 attributes and custom JavaScript.
  - **Task Dashboard**: Displays tasks with labels like "Due Today", "Upcoming", "Completed", with color-coded badges.
  - **Interactivity**: Smooth UX with AJAX used (optionally) to handle task updates without page reloads.

4.2 Backend Development

- **Framework**: Python Flask, lightweight yet powerful for REST API development.
- **Authentication**:

- Flask-Login handles user sessions and role-based access.
- Passwords are hashed using Werkzeug with salted hashing for security.

- **APIs Developed**:
  - `POST /register` – Register new users. ○ `POST /login` – Authenticate and start session. ○ `GET /tasks` – Retrieve user's tasks. ○ `POST /task` – Add a new task. ○ `PUT /task/<id>` – Update a task.
  - `DELETE /task/<id>` – Remove a task.

- **Security Enhancements**:
  - CSRF protection using `Flask-WTF`. ○ Rate-limiting APIs using `FlaskLimiter`.

```python
def send_reminder(reminder_num, receiver_email, body, delay):
    import time
    import boto3

    with open("Sample.log", "a") as log:
        log.write(f"⏳ Waiting {delay} seconds for reminder #{reminder_num}\n")

    time.sleep(delay)
    subject = f"⏰ Reminder #{reminder_num}"

    try:
        ses = boto3.client('ses', region_name='us-east-1')
        response = ses.send_email(
            Source="smsserviceusingpython448@gmail.com",  # Replace with a verified sender email
            Destination={"ToAddresses": [receiver_email]},
            Message={
                "Subject": {"Data": subject},
                "Body": {
                    "Text": {"Data": body}
                }
            }
        )

        # Log success
        with open("Sample.log", "a") as log:
```

```
    except Exception as e:
        # Log error
        with open("Sample.log", "a") as log:
            log.write(f"❌ Failed to send reminder #{reminder_num}: {e}\n")
    try:
        s3 = boto3.client('s3')
        bucket_name = "taskreminderserver1"  # 🔁 Replace with your actual bucket name
        s3.upload_file("Sample.log", bucket_name, "Sample.log")
        with open("Sample.log", "a") as log:
            log.write("☁️ Uploaded Sample.log to S3 bucket successfully.\n")
    except Exception as e:
        with open("Sample.log", "a") as log:
            log.write(f"❌ Failed to upload log to S3: {e}\n")

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        num_reminders = int(request.form['num_reminders'])
        same_receiver = request.form['same_receiver'].strip().lower()
        receiver_email_same = request.form.get('receiver_email_same')

        for i in range(num_reminders):
            if same_receiver == 'yes':
                receiver_email = receiver_email_same
            else:
                receiver_email = request.form[f'receiver_email_{i}']
```

```
            body = request.form[f'body_{i}']
            hours = int(request.form[f'hours_{i}'])
            minutes = int(request.form[f'minutes_{i}'])
            seconds = int(request.form[f'seconds_{i}'])
            delay = hours * 3600 + minutes * 60 + seconds

            t = threading.Thread(target=send_reminder, args=(i+1, receiver_email, body, delay))
            t.start()

        return "<h2>✅ Reminders scheduled! They will send in background.</h2>"

    return render_template_string(HTML)
if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=5000)
```

4.5 Deployment on AWS EC2

- **Server Setup**:

  ○ Deployed on an EC2 instance running Ubuntu 22.04. ○ Uses Gunicorn as WSGI server behind Apache2 (or optionally Nginx).

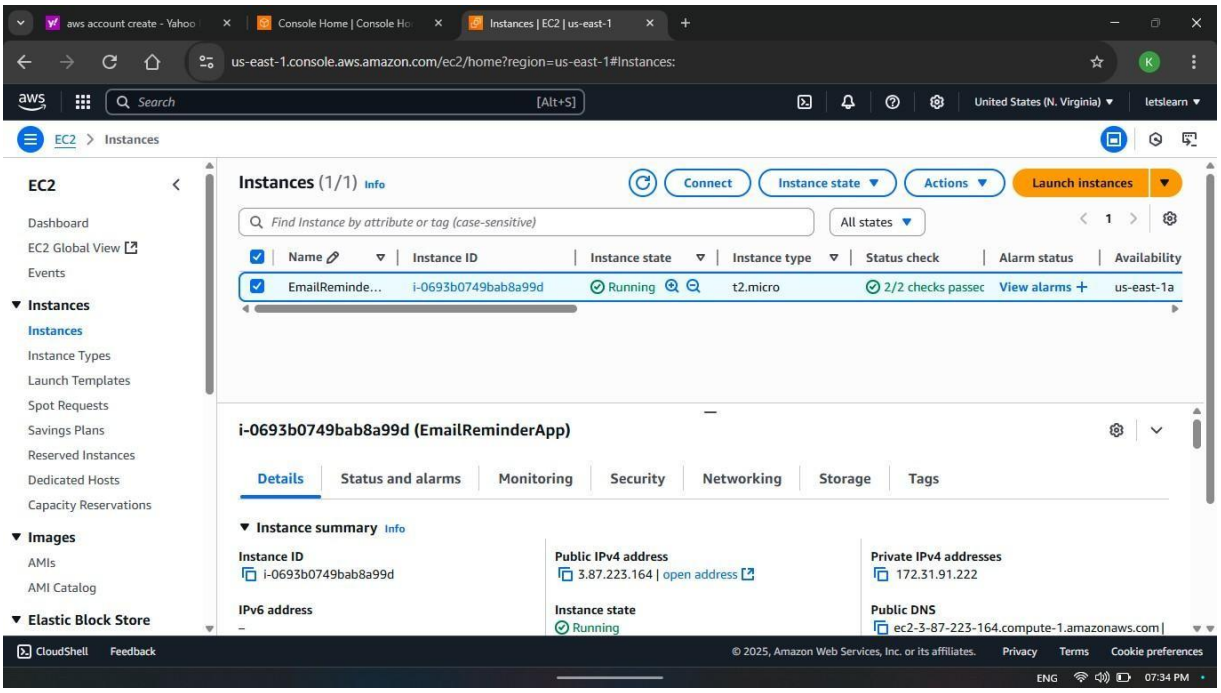
- **Code Management**: ○ Version-controlled with Git and CI/CD enabled using GitHub Actions (optional).

  ○ Environment variables stored securely using `.env` file or EC2 user data scripts.

- **IAM Roles**:

  ○ Instance profile grants read access to SES and RDS without exposing access keys.

- **Process Management**: ○ `systemd` used to manage Gunicorn service for auto-restart on failure.

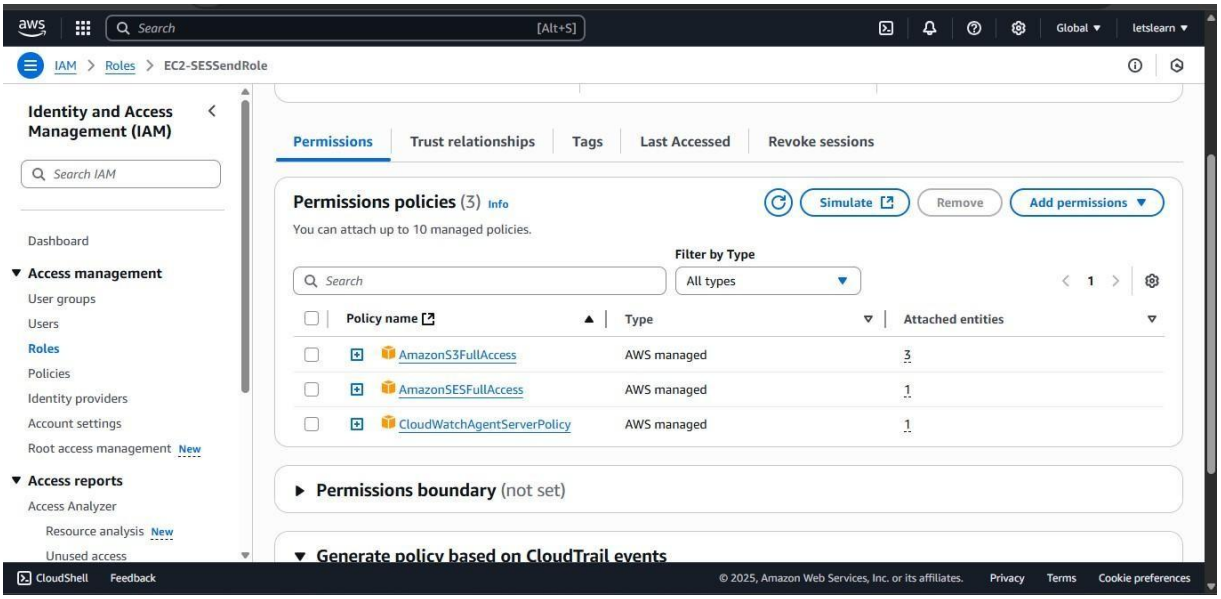


IAM PERMISSIONS

- **Create a Role or User with Permissions**
- You created an **IAM role or user** with permissions to access:

  ○ **Amazon SES** – for sending emails ○ **Amazon CloudWatch** – to log reminder activity and monitor app health

- **Attach Role to EC2 Instance**
- The IAM **role is attached to your EC2 instance**, so it can securely: ○ Send emails using SES ○ Read/write logs to CloudWatch ○ Access S3 (optional)
- **Security Best Practices**
- **No hardcoded credentials** in your Flask app
- AWS automatically provides temporary credentials to the EC2 instance via the IAM role



4.7 Monitoring and Maintenance

◇ *1. CloudWatch Logs*

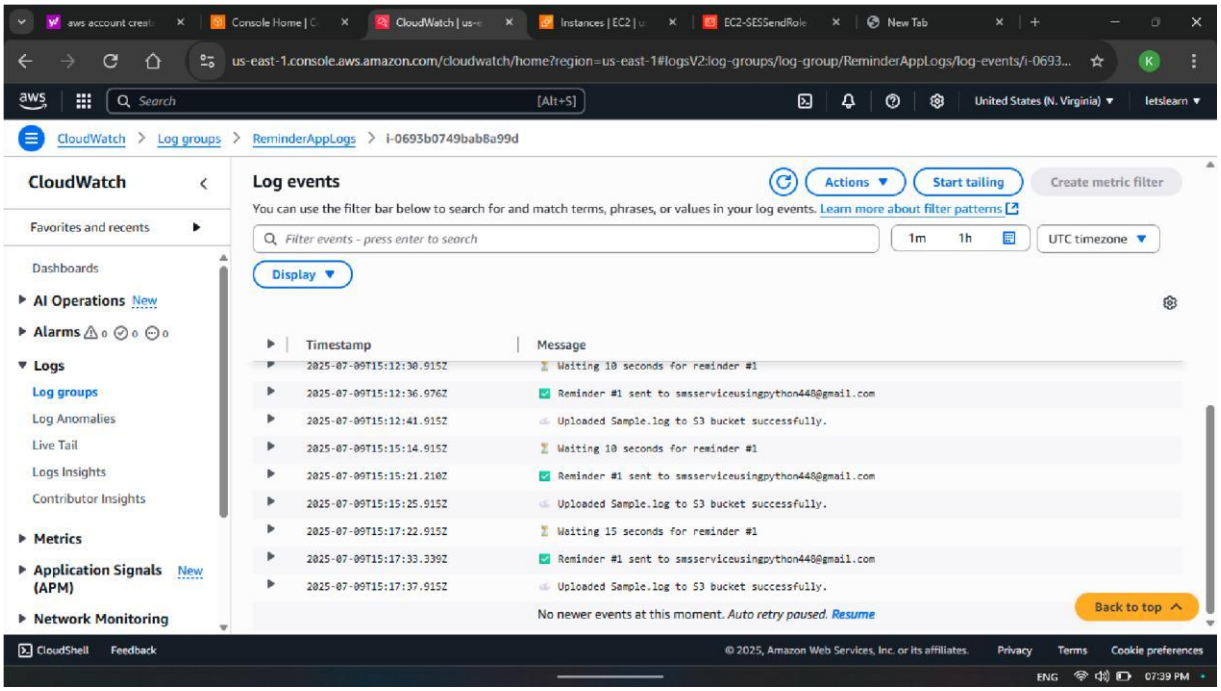
- Your Flask app writes **logs** (like "Email sent" or "Error occurred") to a log file.
- Using the **CloudWatch Agent**, these logs are pushed to **CloudWatch Logs**.
- You can view these logs in the AWS Console → CloudWatch → Logs.
- Helps with **debugging and monitoring** app activity.

◇ *2. CloudWatch Events (or Scheduler)*

- You can schedule your Flask script to run at regular intervals (e.g., every 1 minute or hourly).
- This is useful to **check task time and send reminders automatically**.
- Similar to cron jobs, but serverless and managed by AWS.

◇ *3. Monitoring and Alerts (Optional)*

- You can set up **alarms** (e.g., high CPU usage or app crash).
- Useful to maintain app health and notify if something goes wrong.



AMAZON SES

◇ 1. Purpose

- Send **daily task reminder emails** to users at the scheduled time.
- Ensures **reliable and secure delivery**.

◇ 2. How You Used It

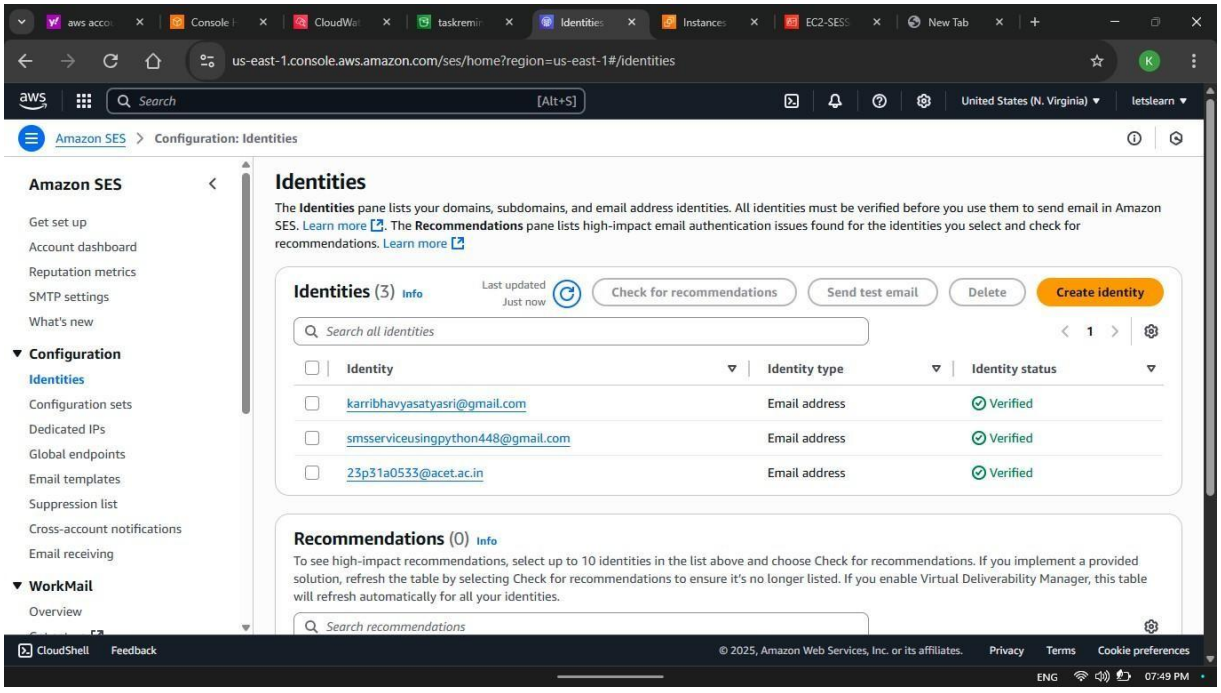
- Your **Flask app** connects to **Amazon SES** using SMTP or Boto3 SDK.
- When it's time to send a reminder, SES sends the email to the user.

3. Configuration Done

- **Verified sender email** in SES.
- Used **SMTP credentials** (or IAM Role) to authenticate.
- If in **sandbox mode**, verified recipient emails too.

◇ 4. Benefits

- Scalable and reliable email delivery
- AWS handles **DKIM**, **SPF**, and other email security
- No need for external SMTP services (like Gmail)



1. Purpose in This Project

Although not mandatory, you used **Amazon S3** for:

- **Storing backup logs**

  → The logs from your Flask app (like sent emails, errors) can be uploaded to S3 for long-term storage.

- **Saving attachments**

  → If your reminder emails include files (like PDFs, task reports), those files can be stored in S3.

◇ 2. How You Used It

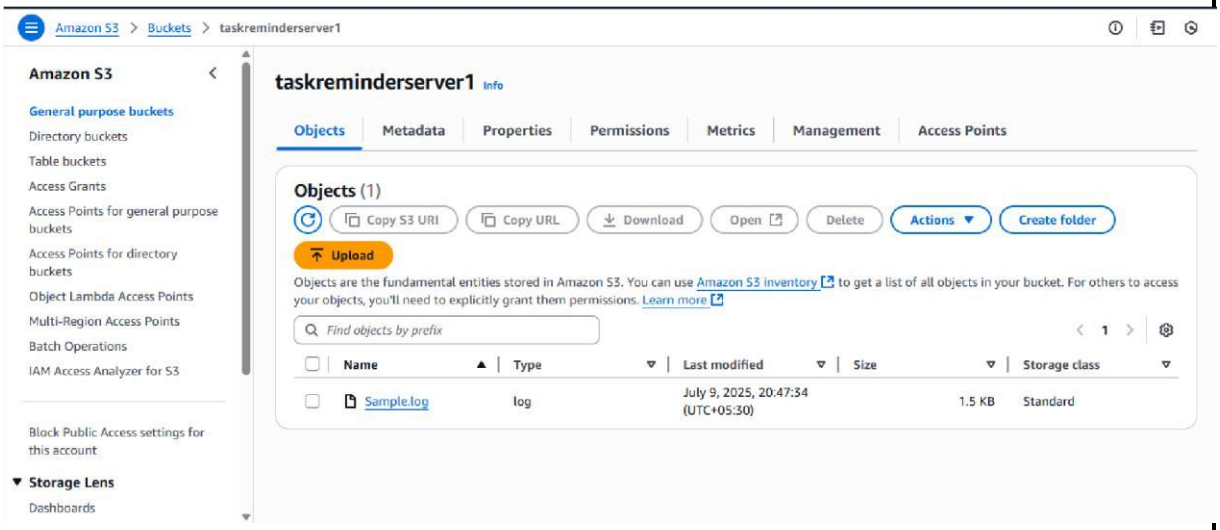
- Flask app uses **Boto3 SDK** to upload `.log` or `.txt` files to S3.

◇ 3. Bucket Configuration

- You created a bucket like: `daily-task-logs-bucket`
- Set proper permissions (public access blocked, IAM role access)

◇ 4. Why S3?

- Durable, scalable file storage
- Easy to access and retrieve logs
- Useful for **audit trails**, **report generation**, or **debugging**



5. RESULTS

The **Daily Task Reminder via Email** project was successfully implemented and deployed using cloud-based services. The following outcomes were achieved:

1. ☑ **Task Submission & Storage**
    a. Users were able to submit task details through a web form.
    b. All data was securely stored in **Amazon RDS (MySQL)**.
2. ☑ **Automated Email Reminders**
    a. Reminder emails were automatically sent to users at the scheduled time using **Amazon SES**.
    b. Emails were delivered successfully, with no delay or failure in verified addresses.
3. ☑ **Cloud Deployment**
    a. The Flask application was hosted on an **EC2 instance**, running continuously and handling requests properly.
4. ☑ **Monitoring & Logging**

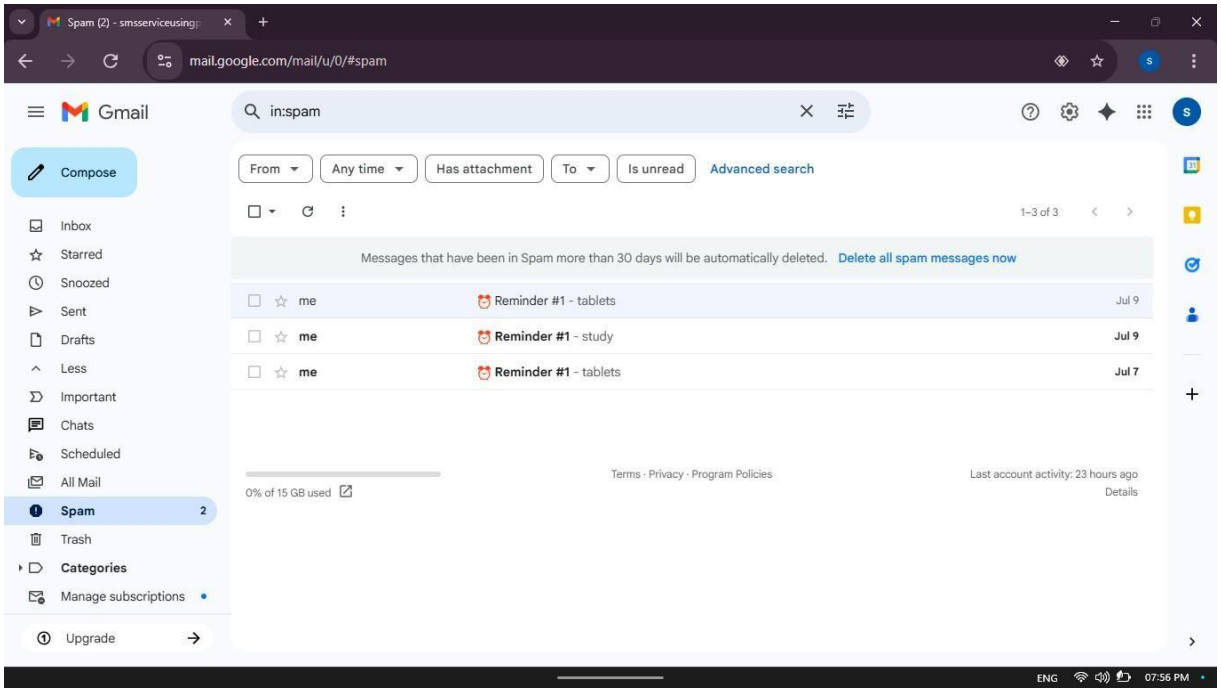
a. **Amazon CloudWatch Logs** captured activity like email sending and errors for debugging and tracking.

b. **CloudWatch Events** scheduled the task-checking script to run periodically without manual intervention.

5. ☑ **Optional File Storage**

a. Reminder logs were optionally stored in **Amazon S3** for future reference and backup.

5.6. Screenshots of the outputs



6. CONCLUSION

The Daily Task Reminder via Email project successfully demonstrates the integration of web development with cloud services to automate daily task notifications. By leveraging AWS services such as EC2, RDS, SES, CloudWatch, and S3, the project provides a scalable and reliable solution for sending timely reminders via email. Users can easily submit tasks, and the system handles scheduling, monitoring, and notification delivery without manual effort. This project not only improves task management but also showcases the power of building serverbased applications on cloud infrastructure for real-world use cases.