



**VIT**<sup>®</sup>  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

## **AWS SOLUTION ARCHITECT**

### **GITHUB LINK FOR PROJECT:**

<https://github.com/VISH-creater11/plant-watering-remainder.git>

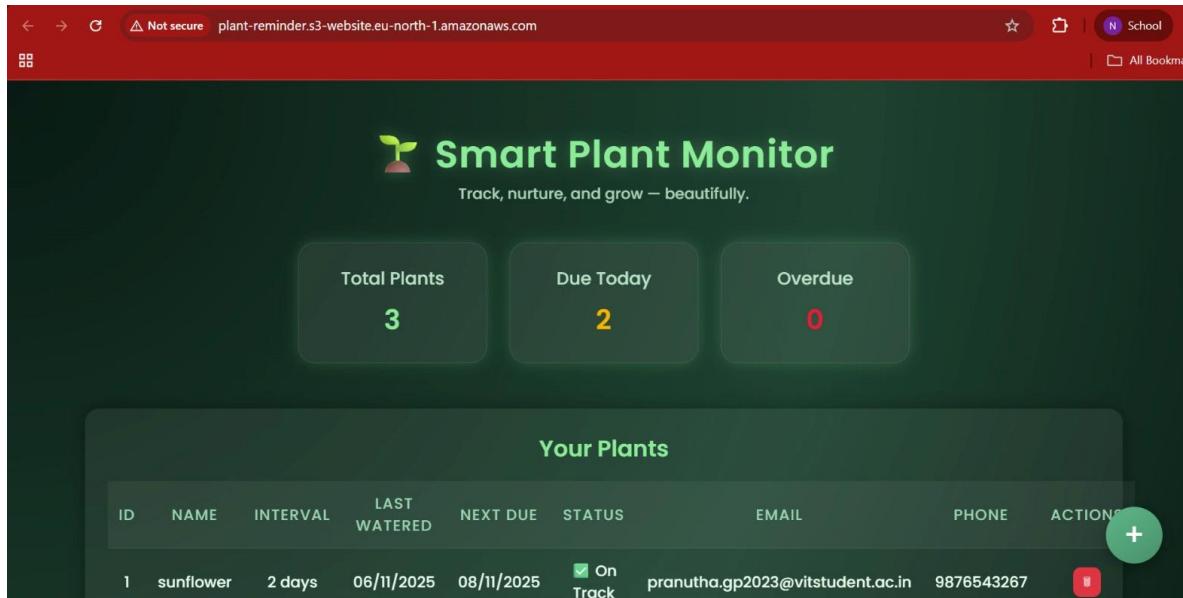
### **TEAM MEMBERS:**

VISHNUPRIYA S

NIKITHA MARIMUTHU

RUKKSHANA

**TITLE: PLANT WATERING REMINDER  
SYSTEM USING AWS**



## DESCRIPTION:

The Plant Watering Reminder System is a cloud-based application designed to help users manage their plant-watering schedules efficiently. It uses AWS Lambda functions to process plant details such as name, watering intervals, and reminder timing. The frontend interface, hosted on Amazon S3, enables users to easily add, view, and manage their plants. EventBridge triggers scheduled checks to determine when a plant requires watering. Amazon SNS sends timely email or SMS alerts to users to ensure plants are watered on time. This solution eliminates the need for manual tracking, making plant care more reliable. The project demonstrates a fully serverless approach using integrated AWS services for automated plant maintenance.

## OUTCOME:

The Plant Watering Reminder System was successfully built using AWS cloud services to simplify plant maintenance. The application allows users to add and track plant details through an interactive web interface hosted on Amazon S3. Automated reminders are generated using EventBridge and sent via SNS to notify users when a plant is due for watering. The system efficiently calculates overdue plants and highlights them on the dashboard, ensuring timely care. AWS Lambda handles

backend processing, enabling a fully serverless, low-cost, and scalable design. Overall, the system provides an easy-to-use solution that automates plant monitoring and reminders, improving healthy plant management.

## **KEY TECHNOLOGIES:**

- **AWS Lambda** – To execute backend logic for fetching and updating plant information.
- **Amazon SNS (Simple Notification Service)** – For sending automated watering reminders.
- **Amazon EventBridge** – To schedule and trigger reminder events automatically.
- **Amazon S3** – For hosting the static web frontend (HTML, CSS, JavaScript).
- **JavaScript, HTML, CSS** – For building an interactive and responsive web interface.

## **OVERVIEW:**

1. The Plant Watering Reminder System is a cloud-based application designed to simplify plant maintenance.
2. Users can add plant details such as name, watering interval, and last watered date through an interactive web interface hosted on Amazon S3.
3. The system automatically calculates upcoming and overdue watering schedules to help users track plant care easily.
4. AWS Lambda functions handle backend processing, including determining which plants are due or overdue for watering.
5. AWS EventBridge triggers periodic checks to evaluate plant watering status at scheduled intervals.
6. Amazon SNS sends timely email notifications to remind users when watering is overdue.
7. The dashboard displays overdue plant status, enabling quick and convenient monitoring.

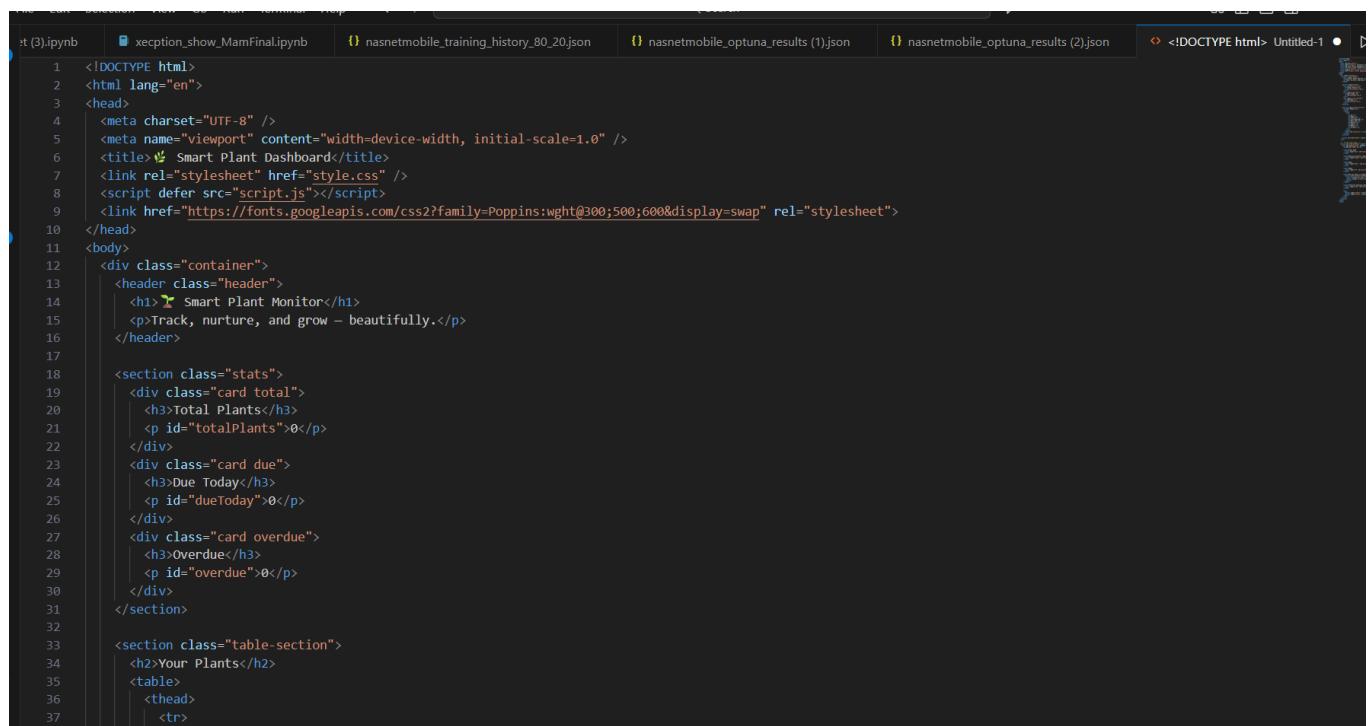
8. The system follows a fully serverless architecture, ensuring automation, scalability, and low operational cost.

## **PROCEDURE:**

### **Step 1: Create a Static Website (Frontend)**

1. Create a project folder containing: index.html, script.js, styles.css, and images.
2. The website allows users to add plant details such as name, last-watered date, and watering interval.
3. The JavaScript code handles capturing user input and sending it to AWS Lambda via a trigger (HTTPS link or custom integration).
4. Plant status (due / overdue) is calculated and visually displayed on the webpage.
5. The frontend prepares data to be sent and shows reminders dynamically.

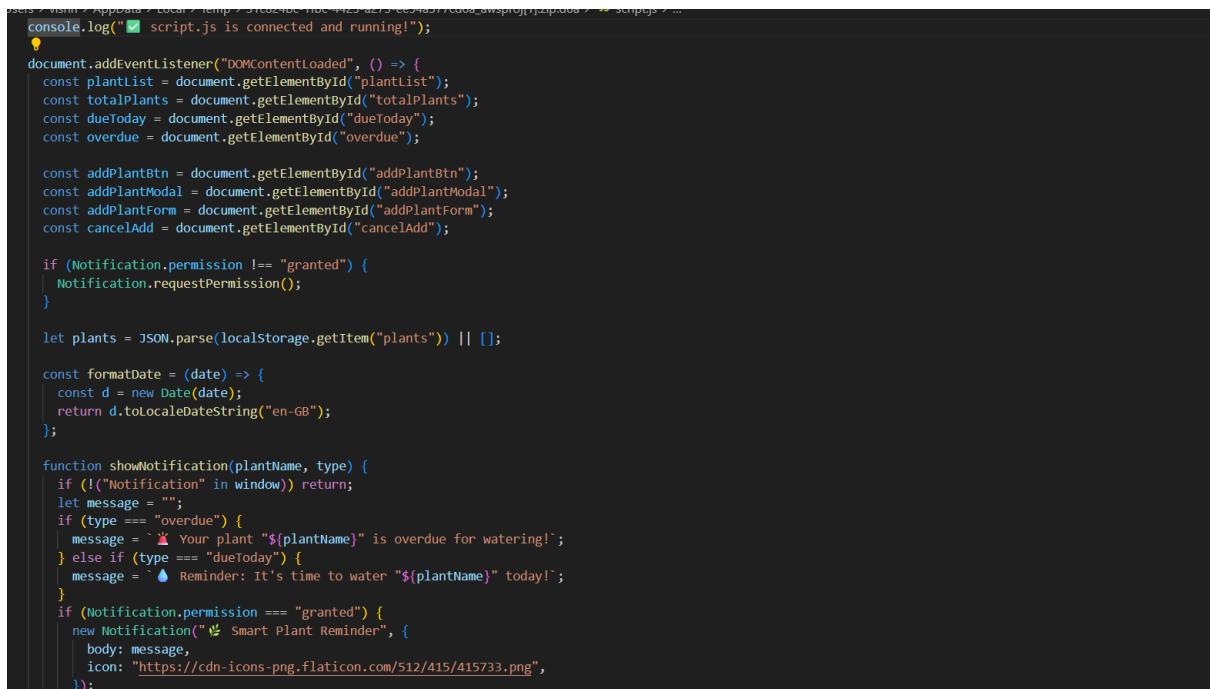
Figure 1.1: HTML file



The screenshot shows a Jupyter Notebook interface with multiple tabs open. The active tab is 'index.html', which contains the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title> Smart Plant Dashboard</title>
7     <link rel="stylesheet" href="style.css" />
8     <script defer src="script.js"></script>
9     <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;500;600&display=swap" rel="stylesheet">
10   </head>
11   <body>
12     <div class="container">
13       <header class="header">
14         <h1> Smart Plant Monitor</h1>
15         <p>Track, nurture, and grow - beautifully.</p>
16       </header>
17
18       <section class="stats">
19         <div class="card total">
20           <h3>Total Plants</h3>
21           <p id="totalPlants">0</p>
22         </div>
23         <div class="card due">
24           <h3>Due Today</h3>
25           <p id="dueToday">0</p>
26         </div>
27         <div class="card overdue">
28           <h3>Overdue</h3>
29           <p id="overdue">0</p>
30         </div>
31       </section>
32
33       <section class="table-section">
34         <h2>Your Plants</h2>
35         <table>
36           <thead>
37             <tr>
```

Figure 1.2 script file



```
users > vishn > AppData > Local > Temp > 4b7dcbf4-685a-4cdf-913f-452e7c4562af.awsproj[1].zip.2af > script.js > ...
console.log("script.js is connected and running!");

document.addEventListener("DOMContentLoaded", () => {
  const plantList = document.getElementById("plantlist");
  const totalPlants = document.getElementById("totalPlants");
  const dueToday = document.getElementById("dueToday");
  const overdue = document.getElementById("overdue");

  const addPlantBtn = document.getElementById("addPlantBtn");
  const addPlantModal = document.getElementById("addPlantModal");
  const addPlantForm = document.getElementById("addPlantForm");
  const cancelAdd = document.getElementById("cancelAdd");

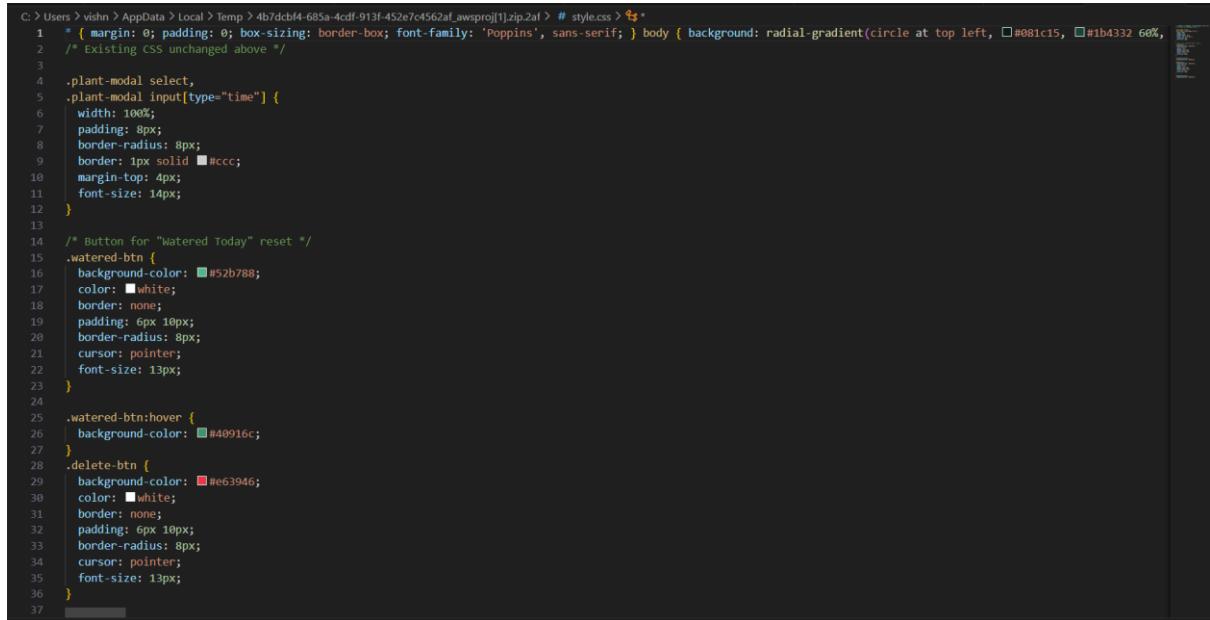
  if (Notification.permission !== "granted") {
    | Notification.requestPermission();
  }

  let plants = JSON.parse(localStorage.getItem("plants")) || [];

  const formatDate = (date) => {
    const d = new Date(date);
    return d.toLocaleDateString("en-GB");
  };

  function showNotification(plantName, type) {
    if (!("Notification" in window)) return;
    let message = "";
    if (type === "overdue") {
      message = "⚠ Your plant "${plantName}" is overdue for watering!";
    } else if (type === "dueToday") {
      message = "⚠ Reminder: It's time to water "${plantName}" today!";
    }
    if (Notification.permission === "granted") {
      new Notification("Smart Plant Reminder", {
        body: message,
        icon: "https://cdn-icons-png.flaticon.com/512/415/415733.png",
      });
    }
  }
});
```

Figure 1.3: css file:



```
C:\> Users > vishn > AppData > Local > Temp > 4b7dcbf4-685a-4cdf-913f-452e7c4562af.awsproj[1].zip.2af > style.css > 44
1 * { margin: 0; padding: 0; box-sizing: border-box; font-family: 'Poppins', sans-serif; } body { background: radial-gradient(circle at top left, #081c15, #1b4332 60%, #2e3436 100%); }
2 /* Existing CSS unchanged above */
3
4 .plant-modal select,
5 .plant-modal input[type="time"] {
6   width: 100%;
7   padding: 8px;
8   border-radius: 8px;
9   border: 1px solid #ccc;
10  margin-top: 4px;
11  font-size: 14px;
12 }
13
14 /* Button for "Watered Today" reset */
15 .watered-btn {
16   background-color: #52b788;
17   color: white;
18   border: none;
19   padding: 6px 10px;
20   border-radius: 8px;
21   cursor: pointer;
22   font-size: 13px;
23 }
24
25 .watered-btn:hover {
26   background-color: #40916c;
27 }
28 .delete-btn {
29   background-color: #e63946;
30   color: white;
31   border: none;
32   padding: 6px 10px;
33   border-radius: 8px;
34   cursor: pointer;
35   font-size: 13px;
36 }
37
```

## Step 2: Deploy Frontend Using Amazon S3 (with commands/actions)

1. Search for Amazon S3 in AWS Console
  - o Open AWS Console → search S3 → click Amazon S3.
2. Create an S3 Bucket
  - o Click Create Bucket

- Enter a unique bucket name → example:
- plant-reminder-frontend
- Choose a region → EX: us-east-1
- Click Create bucket

### 3. Enable Static Website Hosting

- Open your created bucket → Go to Properties
- Scroll to Static website hosting
- Click Edit → Enable
- Enter:
  - index.html
  - error.html
- Click Save changes

### 4. Configure Public Access Permissions

- Go to Permissions → Block Public Access
- Turn OFF → “Block all public access”
- Confirm → Save
- Now open Bucket policy → paste policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicRead",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::plant-reminder-frontend/*"
    }
  ]
}
```

- Click Save

## 5. Upload Website Files

- Go to Objects → Upload
- Upload your project files (HTML, CSS, JS, images)
- index.html
- styles.css
- script.js
- sapling.jpg
- Click Upload

## 6. Get Website Link

- Go to Properties → Static website hosting
- Copy Bucket Website Endpoint, e.g.:
- <http://plant-reminder-frontend.s3-website-us-east-1.amazonaws.com>
- Open it in browser → Website Live

Figure 2.1. creation

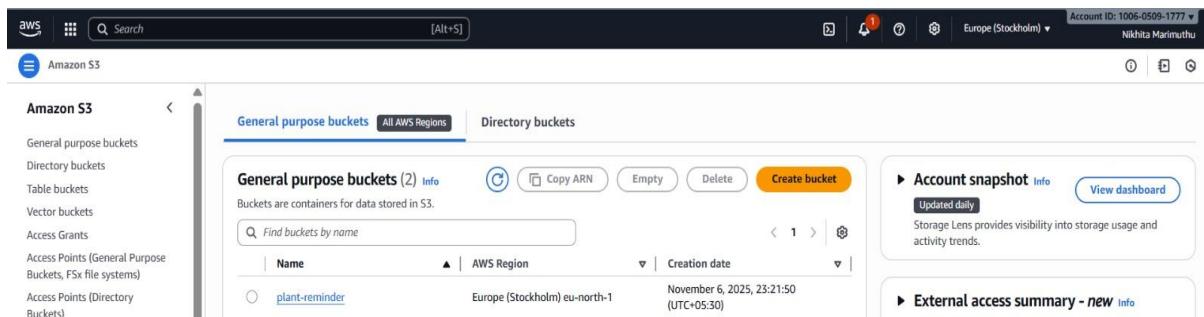


Figure 2.2: configuration

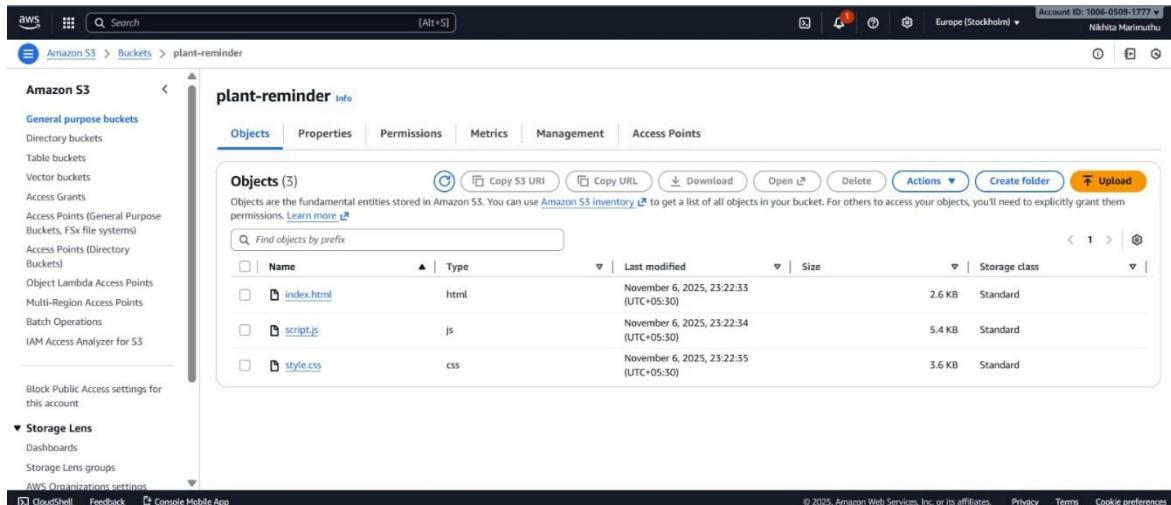
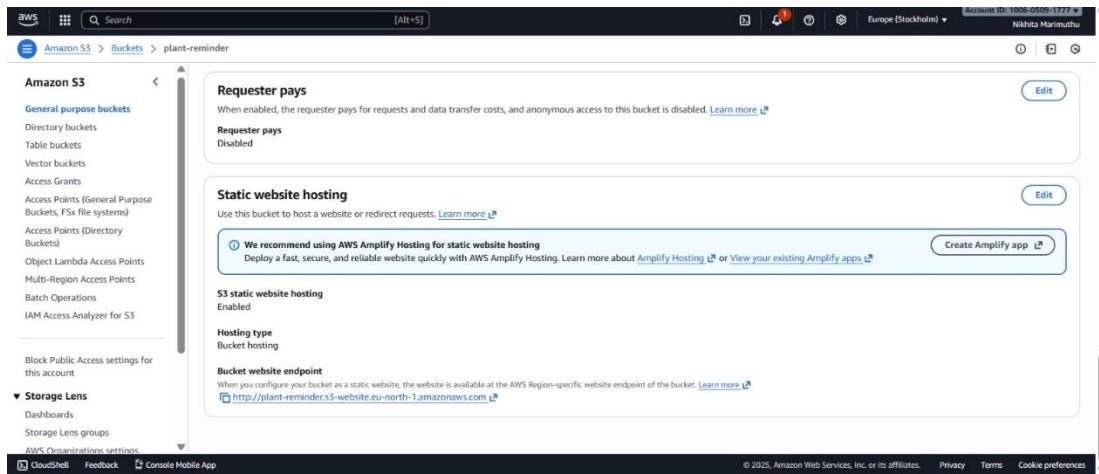


Figure 2.3.creating connection



## Step 3: Create AWS Lambda Function (Plant Management + Reminder Logic)

### 1. Open Lambda → Create Function

- Go to AWS Console → search Lambda
- Click Create function

### 2. Configure Function

- Choose → Author from scratch
- Enter function name: *PlantReminderManager*
- Select runtime: Python 3.x OR Node.js
- Click Change default execution role → Create a new role with basic Lambda permissions
- Click Create Function

### 3. Add Lambda Code

- Scroll to Code → Code editor
- Paste your Lambda logic to:
  - a). Temporarily store plant data (in memory or S3)
  - b). Calculate due/overdue based on date + interval
  - c). Send SNS notifications

## Example (Python)

```
import json
import boto3
from datetime import datetime, timedelta

sns = boto3.client('sns')
TOPIC_ARN = "arn:aws:sns:REGION:ACCOUNT_ID:PlantReminderTopic"

def lambda_handler(event, context):
    plants = event.get("plants", [])
    today = datetime.now().date()

    due_plants = []

    for p in plants:
        last = datetime.strptime(p['lastWatered'], "%Y-%m-%d").date()
        next_due = last + timedelta(days=p['intervalDays'])

        if next_due <= today:
            due_plants.append(p)

    if due_plants:
        message = "Plants due for watering:\n" + "\n".join([p['name'] for p in due_plants])
        sns.publish(
            TopicArn=TOPIC_ARN,
            Message=message,
            Subject="Watering Reminder!"
        )

    return {
        'statusCode': 200,
        'body': json.dumps({"duePlants": due_plants})
    }
```

## 4. Add SNS Permission to Lambda

Go to Configuration → Permissions → Execution Role → Add Permissions

- Attach policy: *AmazonSNSFullAccess*

## 5. Test Lambda

- Click Test
- Create test event

Example event:

```
{  
  "plants": [  
    {
```

```

        "name": "Tulsi",
        "lastWatered": "2025-01-01",
        "intervalDays": 3
    }
]
}

```

- Click Test → View output

## 6. Deploy

Click: Deploy

Lambda Handles

- ✓ Processing plant input data
- ✓ Checking watering due/overdue logic
- ✓ Triggering SNS reminders

Figure3.1 .SendPlantRemainder creation

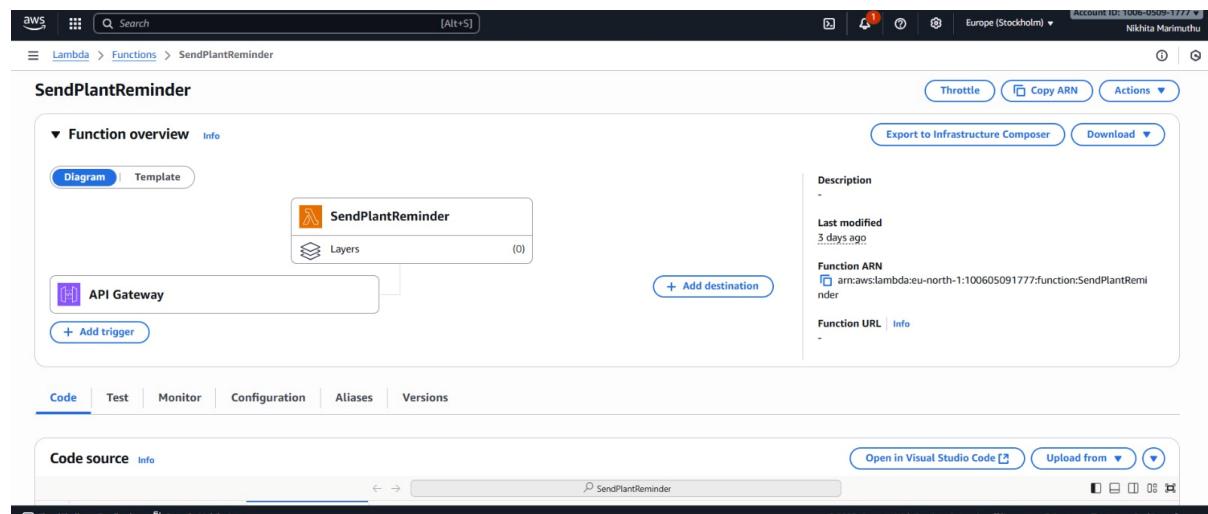
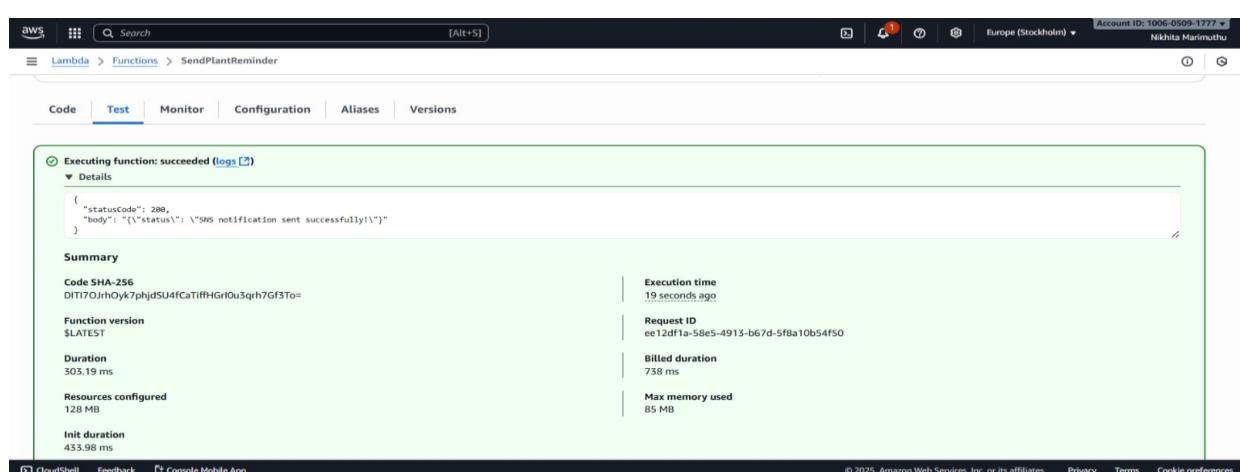


Figure 3.2.lambda function creation



## Step 4: Configure Amazon EventBridge (Schedule Processing)

1. Search for EventBridge → Click Create rule.
2. Rule name example: PlantScheduleChecker.
3. In Schedule pattern, choose:
  - Rate: 1 day (or custom interval)
4. Choose target → Select your Lambda function PlantReminderManager.
5. Save the rule.

EventBridge will now automatically trigger the Lambda function regularly.

Figure 4.1.set\_Rules

The screenshot shows the AWS EventBridge Rules interface. On the left, there's a navigation sidebar with options like Dashboard, Developer resources, Buses, Pipes, Scheduler, and Integration. The main area is titled 'Rules' and contains a section for 'Select event bus' where 'default' is chosen. Below this is a table titled 'Rules on default event bus (1)'. The table has columns for Name, Status, Type, Event bus, ARN, and Description. It lists one rule: 'dailyPlantReminder' (Status: Enabled, Type: Scheduled Standard, Event bus: default, ARN: arn:aws:events:eu-north-1:10060509177:rule/dailyPlantReminder, Description: Schedule expression). There are buttons for Find rules, Delete, Enable, Edit, CloudFormation Template, and Create rule.

Figure 4.2. event schedule

This screenshot shows the detailed view of the 'dailyPlantReminder' rule. At the top, it says 'Edit', 'Disable', 'Delete', and 'CloudFormation Template'. The 'Rule details' section includes fields for Rule name (dailyPlantReminder), Status (Enabled), Description (Schedule expression), Rule ARN (arn:aws:events:eu-north-1:10060509177:rule/dailyPlantReminder), Event bus name (default), Event bus ARN (arn:aws:events:eu-north-1:10060509177:7:event-bus/default), and Type (Scheduled Standard). Below this is the 'Event schedule' section, which shows a Cron expression '0 1 \* \* ? \*' and a list of the next 10 trigger dates from Friday, Nov 07, 2025, to Thursday, Nov 13, 2025. There's also an 'Edit' button for the event schedule.

## Step 5: Configure Amazon SNS for Notifications

1. Open Amazon SNS → Create topic.
2. Choose Standard, name it PlantReminderTopic.
3. Open created topic → Click Create subscription.
4. Choose Protocol: Email, then enter your email.
5. Confirm using the link sent to your inbox.

SNS will now send email reminders when Lambda produces an alert.

Figure 5.1. configuration step

The screenshot shows the AWS SNS console with the following details:

- Left sidebar:** Shows 'Amazon SNS' selected under 'Topics'. Other options include 'Dashboard', 'Subscriptions', and 'Mobile' (Push notifications, Text messaging (SMS)).
- Top navigation:** Includes 'Search' bar, account information ('Account ID: 1006-0509-1777'), and location ('Europe (Stockholm)').
- Topic Details:** The topic is named 'PlantReminderTopic'. It is a 'Standard' type topic. The ARN is listed as 'arn:aws:sns:eu-north-1:100605091777:PlantReminderTopic'. The 'Topic owner' is listed as '100605091777'. The 'Display name' is left blank.
- Subscriptions:** There are two confirmed subscriptions listed:

ID	Endpoint	Status	Protocol
55f4720d-e37b-4471-bd22-1da564e7532e	marimuthunikhitaj@gmail.com	Confirmed	EMAIL
c825ddd8-686a-407e-b73f-9c76e8a13b02	nrukshana89@gmail.com	Confirmed	EMAIL
- Bottom footer:** Includes links for 'cloudShell', 'Feedback', 'Console Mobile App', and copyright information ('© 2025, Amazon Web Services, Inc. or its affiliates.'), along with 'Privacy', 'Terms', and 'Cookie preferences'.

## Step 6: Connect Webpage to AWS Services

1. In your script.js, store Lambda invocation endpoint / logic.
2. When a plant is added, the webpage triggers Lambda.
3. Lambda processes plant info, checks schedule, and (if due/overdue) sends notifications via SNS.
4. Frontend shows updated plant status visually (Due Today / Overdue / Healthy).
5. Direct calling of Lambda is used → no API Gateway required.

## Step 7: Test Entire System

1. Open your website from the S3 public endpoint.

2. Add plant details and verify:
  - a). Plant is displayed on UI
  - b). Due/overdue calculation works
  - c). Email reminder arrives when plant watering is due
3. Confirm EventBridge triggers Lambda daily.

## WEBSITE VIEW:

Figure 6.1 Website UI

The screenshot shows the homepage of the Smart Plant Monitor website. At the top, there's a navigation bar with icons for back, forward, refresh, and a 'Not secure' warning. The URL is 'plant-reminder.s3-website.eu-north-1.amazonaws.com'. On the right of the bar are bookmarks and a 'School' button. Below the bar, the title 'Smart Plant Monitor' is displayed with a small plant icon, followed by the tagline 'Track, nurture, and grow – beautifully.' Three summary boxes are shown: 'Total Plants' (3), 'Due Today' (2), and 'Overdue' (0). A large section titled 'Your Plants' lists three plants: 1. sunflower (LAST WATERED: 06/11/2025, NEXT DUE: 08/11/2025, STATUS: On Track, EMAIL: pranutha(gp)2023@vitstudent.ac.in, PHONE: 9876543267). There's a green '+' button in the 'ACTIONS' column for this row. The rest of the page has a dark background with light-colored text and buttons.

Figure 6.2. Add new Plant details

This screenshot shows the 'Add New Plant' form on the left and the 'Your Plants' list on the right. The form fields include 'Plant Name' (empty), 'Watering Interval (days)' (empty), 'Email' (you@example.com), 'Phone' (+91xxxxxxxx), 'Did you water it today?' (Yes), and 'Preferred watering reminder time' (dropdown menu). The main area shows the 'Your Plants' table with three rows. The first row (sunflower) has a status of 'On Track'. The second row (marimuthu(nikhita)) has a status of 'Due Today' and a green 'Watered Today' button. The third row (nrukshana89) also has a status of 'Due Today' and a green 'Watered Today' button. A green '+' button is located at the bottom right of the plant list table.

Figure 6.3 .working of website

ID	NAME	INTERVAL	LAST WATERED	NEXT DUE	STATUS	EMAIL	PHONE	ACTIONS
1	sunflower	2 days	06/11/2025	08/11/2025	<input checked="" type="checkbox"/> On Track	pranutha(gp2023@vitstudent.ac.in)	9876543267	<span>Edit</span>
2	rose	1 days	05/11/2025	06/11/2025	<span>Watered Today</span>	marimuthunikhita@gmail.com	9042131671	<span>Edit</span>
3	hibiscus	1 days	05/11/2025	06/11/2025	<span>Watered Today</span>	nrukshana89@gmail.com	9535056270	<span>Edit</span>

## WORKING OF WEBSITE (STEP-BY-STEP)

### 1. User Opens the Website

- The user accesses the web application via the S3 bucket hosting link.

### 2. Add Plant Details

- The user clicks “Add Plant” and enters:
  - Plant name
  - Watering interval (days)
  - Last watered date
  - Email & phone (optional)

### 3. Store & Process Plant Data

- The details are passed to the backend Lambda function.
- Lambda automatically processes the plant data.

### 4. Calculate Next Watering / Overdue Status

- The system compares today’s date with last-watered date + interval.
- It then marks plants as:
  - ✓ Healthy
  - Due
  - Overdue

### 5. Visual Display on Dashboard

- The dashboard shows:
  - Total plants
  - Due today
  - Overdue count
  - Current plant list with status badges

## 6. Automated Daily Check via EventBridge

- EventBridge triggers Lambda daily.
- System scans all plants to find which ones need watering.

## 7. Send Notification via SNS

- For plants marked **due or overdue**,  
SNS sends an **email reminder** to the user automatically.

## 8. User Waters Plant & Updates the Record

- The user can update last-watered date.
- Status becomes **healthy** again.

## 9. Continuous Monitoring

- The entire cycle repeats daily automatically—NO manual tracking needed.