A Serverless Notification System using AWS SNS and SES

*A Project Based Learning Report Submitted in partial fulfilment of the requirements for the award of the degree*

*of*

**Bachelor of Technology**

**in the Department of Computer Science & Engineering**

**Cloud Based AI/ML Speciality (22SDCS07A)**

Submitted by

**2210030093 : A. Venkat Reddy**

Under the guidance of

**Ms. P. Sree Lakshmi**



Department of Computer Science and Engineering

Koneru Lakshmaiah Education Foundation, Aziz Nagar

Aziz Nagar – 500075

March - 2025.

1. **Introduction**

* **Overview of the Project**

A Serverless Notification System using AWS Simple Notification Service (SNS) and Simple Email Service (SES) enables real-time, scalable messaging across multiple channels, including SMS, email, and push notifications. The system eliminates the need for server management, reducing operational complexity and cost. SNS ensures message distribution to various endpoints, while SES handles bulk email sending for transactional and marketing purposes. By leveraging AWS serverless services, organizations can automate notifications for diverse use cases such as system alerts, customer engagement, and user authentication, providing a reliable and cost-effective communication framework.

* **AWS Services Being Used and Justification**
* The core AWS services used in this system include:
* **AWS SNS:** Facilitates multi-channel message distribution via SMS, push notifications, and email, ensuring real-time delivery.
* **AWS SES:** Handles high-volume transactional and marketing emails with advanced deliverability features.
* **AWS Lambda:** Processes event-driven notifications without the need for server management.
* **Amazon DynamoDB:** Stores message logs and metadata for audit and tracking.
* **AWS IAM:** Manages security policies and access controls. These services ensure a scalable, cost-effective, and reliable notification system with seamless AWS integration.
* **Project Purpose and Expected Outcome**

The purpose of this project is to build an automated, real-time notification system that scales efficiently without infrastructure overhead. The expected outcome includes improved communication reliability, reduced operational costs, and enhanced user engagement through a seamless serverless messaging solution, benefiting businesses with automated alerts and notifications.

1. **Methodology**

* **Architecture and Workflow**

The system follows an event-driven architecture. AWS Lambda triggers notifications based on predefined events, pushing messages to SNS. SNS then distributes the notifications across multiple endpoints, including SMS, mobile push, and email. For email notifications, SES is used to send high-volume messages efficiently. DynamoDB stores notification logs for tracking, while IAM policies ensure secure access control.

* **Explanation of AWS Services Interaction**

When an event occurs, AWS Lambda processes the event and triggers SNS. SNS then routes the message to subscribed endpoints such as SMS or mobile devices. If an email notification is required, SNS integrates with SES to send bulk or transactional emails. DynamoDB logs each notification for audit purposes. IAM roles restrict access, ensuring only authorized components interact with SNS and SES.

* **Justification for AWS Service Selection**

AWS SNS and SES are chosen for their seamless integration, scalability, and cost efficiency. SNS enables multi-channel distribution, while SES specializes in email handling. Lambda ensures serverless execution, reducing management overhead. DynamoDB provides efficient storage for logs, and IAM enforces security policies, making AWS the ideal choice for this notification system.

1. Implementation Steps

* AWS Infrastructure Setup

1. Create an SNS Topic – Set up a topic to manage and distribute notifications.
2. Configure SES – Verify domains and set up email sending policies.
3. Deploy AWS Lambda – Write functions to process event triggers and integrate with SNS.
4. Set Up DynamoDB – Configure tables to store notification logs.
5. Define IAM Policies – Restrict access to SNS, SES, and DynamoDB to authorized users and services.
6. Test the System – Send test notifications to verify end-to-end functionality.
7. Security Policies, IAM Roles, and Access Controls

To secure the notification system:

* IAM Policies: Restrict access based on roles (e.g., Lambda access to SNS, SES, and DynamoDB).
* Encryption: Enable AWS Key Management Service (KMS) for encrypting sensitive data.
* Authentication & Authorization: Use AWS Cognito for managing user permissions.
* Logging & Monitoring: Integrate AWS CloudWatch to monitor logs and detect anomalies.

1. Automation and CI/CD Pipeline

For efficient deployment, a CI/CD pipeline is implemented using AWS CodePipeline:

1. CodeCommit: Stores Lambda functions and configurations.
2. CodeBuild: Automates testing and validation.
3. CodeDeploy: Deploys updates to AWS Lambda, SNS, and SES.
4. CloudFormation: Automates infrastructure provisioning for consistency. This pipeline ensures smooth deployment, reducing downtime and human errors in system updates.