Architecture Design for Notification Platform

Table of contents

Chapter 1: Introduction

1.1. Problem Statement

Design a "Notification Platform" that enables multiple services to send notifications to end-users through various channels such as Email, SMS, and Push Notifications. The system should be designed primarily for deployment on AWS Cloud.

1.2. Scope and Objectives

This project focus on design and implementation of a highly scalable and secure notification platform, aimed at efficiently receiving API requests and sending notifications to millions of users via several delivery channels. The solution handles automatic transition between the delivery channels, seamless storage in a database, and secure message transmission to end users.

Chapter 2: Software Requirements

2.1. Functional Requirements

Request Routing: The system should allow for secure reception of API request into Amazon API Gateway via an endpoint secured with an API key.

Automatic Processing: Upon sending the request, a Lambda function should be automatically triggered which collects and parses the request to get the data from the payload, determine the intended channel for that notification, forwards the message to be sent via the appropriate channel and prepare it for storage in DynamoDB.

Notification Publishing: The appropriate SNS topic for each notification channel publishes the received message to all its subscribers.

Data Storage: Payload information from the API request would be stored in DynamoDB with a counter field in place for the number of times each delivery channel has been used to send notifications.

Query and Retrieval: Authorized users should be able to query data stored in DynamoDB. Implement queries for retrieving records based on ID or other relevant criteria.

Security and Access Control: Ensure data security at all storage levels and access control. Use IAM roles and policies to restrict access to DynamoDB tables based on the principle of least privilege.

2.2. Non-functional requirements

Scalability: The solution should scale seamlessly as number of end users increase. Use AWS services like API Gateway, SNS, DynamoDB that support automatic scaling to accommodate growing data needs.

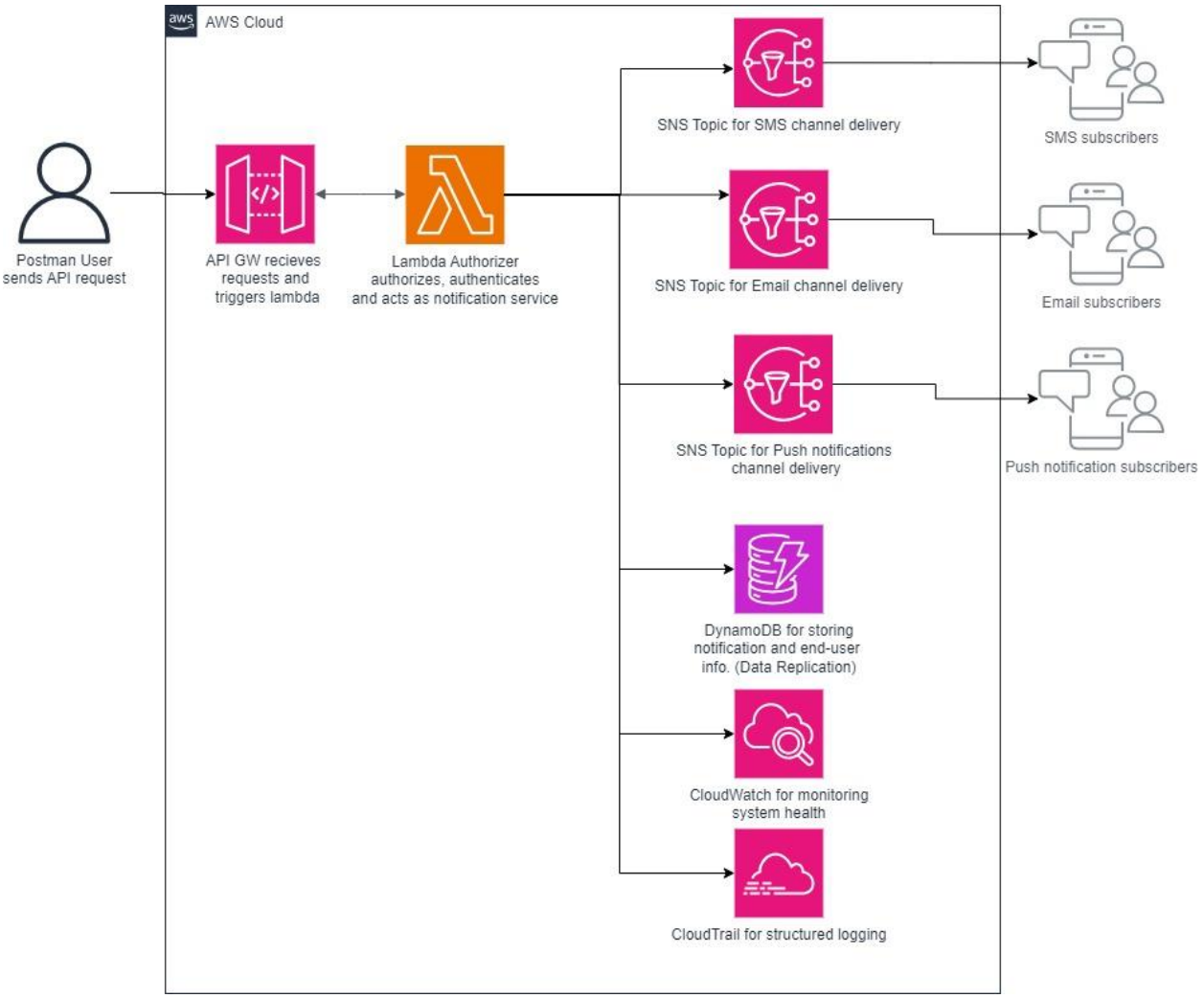
Reliability: The system should be highly available and reliable. Implement fault-tolerant architecture using AWS services with built-in redundancy and disaster recovery mechanisms.

Security: Ensure data integrity and confidentiality. Implement encryption (in transit and at rest), secure API calls, and access controls to protect data from unauthorized access or breaches.

Cost-effectiveness: Optimize costs associated with AWS services. Utilize serverless architecture to minimize infrastructure management overhead and pay only for actual usage (e.g., Lambda function invocations, DynamoDB read/write capacity).

Chapter 3 Architecture Overview

The solution architecture is designed to be serverless, utilizing AWS services for data storage, processing, and access. Key components include AWS Lambda for serverless compute, Amazon API Gateway for API calls, Amazon DynamoDB for NoSQL database storage, and AWS IAM for access management and SNS for publishing messages to end users.



The diagram shows the overall architecture of the "Notification Platform," including major components and their interactions, illustrating how services will interface with the platform to send notifications.

Design Overview:

A request is made in Postman using the invocation URL for an API created in API gateway. This triggers a lambda function which upon receiving the request, parses it and determines the delivery channel to use based on a notification type element in the request body using if-else statements. Once the channel is selected it forwards the message

in the notification through the appropriate SNS topic as a channel to the end user as an email, an SMS, a push notification. It also stores the data in DynamoDB.

Scalability & High Availability:

To handle increasing number of end users and ensure high availability, more subscribers would be added to the various SNS topics and given that the very high scalability of SNS, the system would be highly available.

Security:

For security, at the level of the API request, an API key is added to the head before sending the request ensuring the integrity of the data.

Data at every point in this system is automatically encrypted at rest and is done at SNS by enabling Server-Side Encryption (SSE) using AWS KMS keys.

IAM roles give the authorization to the lambda function to perform operations in API gateway, SNS and DynamoDB.

Monitoring and Logging:

Monitor system performance and log activities. Utilize AWS CloudWatch for monitoring Lambda function executions, API Gateway access, DynamoDB operations, and CloudTrail for logging purposes.

Multi-Channel Handling:

A set of if-else statements in the lambda function handles sending notifications through the different notification channels (Email, SMS, Push Notifications) based off a notification type element specified in the API request.

Extensibility:

If new notification channels need to be added in the future, a new SNS topic would be created for that channel, and it would be added as a choice in the if else loop.

4. Conclusion

In all, the implementation of the Notification Platform for deployment on AWS following this architecture provides a scalable, secure and cost-effective approach to sending notifications to end users by leveraging AWS services such as Lambda, DynamoDB, API Gateway and SNS.

5. Recommendations

To ensure the secure, resilient and highly available operation of this system, the following should be done;

Conduct regular performance monitoring and optimization of Lambda functions and DynamoDB to ensure scalability.

Implement automated backups and disaster recovery strategies to safeguard against data loss.

Implement loop back mechanism from notifications stored in DynamoDB to retry sending notifications which did not go.

6. Appendix

API Gateway invocation URL for making requests securely:

https://rnoy78a3lf.execute-api.us-east-1.amazonaws.com/dev/messages