Vadakathi Muhammed Suhaib
Technical Apprentice
Emp ID: X48GRSTML
muhammed.suhaib@cprime.com

Proof of Concept

Serverless Blog Platform with Content Moderation

Serverless Blog Platform with Content Moderation

1. Overview

The Serverless Blog Platform enables users to post and manage blog content. The platform will automatically moderate uploaded images to ensure compliance with content guidelines. This Proof of Concept (PoC) aims to validate the feasibility of using AWS serverless architecture to build this solution efficiently.

2. Introduction

In modern digital platforms, content moderation is essential to maintain the integrity and compliance of user-generated content. The goal of this project is to leverage AWS services to create a fully serverless blog platform that provides content hosting, automated moderation, and a seamless user experience. This PoC will demonstrate how AWS services such as S3, CloudFront, Lambda, API Gateway, DynamoDB, and Rekognition can be integrated to build a scalable, secure, and cost-efficient blog system.

3. Problem Statement

- Traditional blog platforms require manual moderation of uploaded content, which is inefficient and prone to errors.
- Managing infrastructure for scalability, availability, and security is complex and resource-intensive.
- A need exists for an automated, serverless approach that moderates images and ensures compliance with policies while minimizing operational overhead.

4. PoC Expectations

This PoC aims to validate the following:

- The feasibility of using AWS serverless services for hosting and managing a blog platform.
- The effectiveness of Amazon Rekognition in moderating uploaded images.
- The integration of a serverless backend using AWS Lambda, API Gateway, and DynamoDB for content management.
- The ability to automate deployments and infrastructure management using CodePipeline and CloudFormation.

5. Services Used

- **Frontend**: React.js hosted on **S3 + CloudFront**
- **Backend**: API Gateway + Lambda (Node.js)
- **Database**: DynamoDB for storing blog posts
- **Image Storage**: S3 bucket (uploads)
- **Image Moderation**: Amazon Rekognition
- **Infrastructure as Code**: AWS CloudFormation
- **CI/CD Pipeline**: AWS CodePipeline
- **Notifications**: SNS + SQS
- **Domain Setup**: Route 53

6. Execution Plan

The project execution will follow these key steps:

Step 1: Frontend Hosting

- **Service Used:** Amazon S3, CloudFront, Route 53
- The static frontend (React/HTML/JS) will be hosted on **S3**.
- **CloudFront** will act as a Content Delivery Network (CDN) for faster global access.
- **Route 53** will be used to configure a custom domain.

Step 2: API Development

- **Service Used:** API Gateway, AWS Lambda
- API Gateway will handle HTTP requests from the frontend.
- AWS Lambda functions will process user requests (e.g., creating blog posts, retrieving content).

Step 3: Content Storage

- **Service Used:** DynamoDB, Amazon S3
- Blog posts and comments will be stored in **DynamoDB** for fast and scalable access.
- Images will be uploaded to **Amazon S3**.

Step 4: Image Moderation

- **Service Used:** Amazon Rekognition, SNS
- Images uploaded to S3 will trigger an **AWS Lambda function** to call **Amazon Rekognition** for moderation.
- If the image contains inappropriate content, an alert will be sent via **SNS** to notify the user.

Step 5: Notifications & Messaging

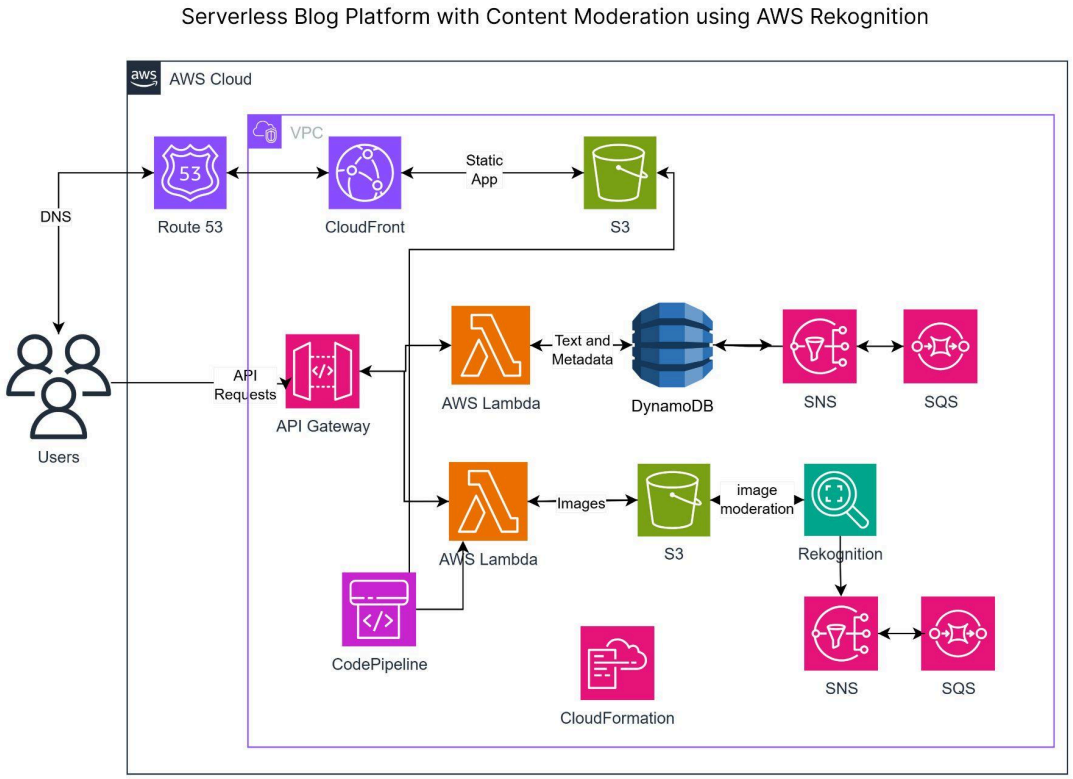
- **Service Used:** SNS, SQS
- SNS will be used to send notifications regarding content moderation results.
- SQS can be used for queuing messages to handle moderation asynchronously.

Step 6: CI/CD and Infrastructure Management

- **Service Used:** AWS CodePipeline, CloudFormation
- **CodePipeline** will automate the build and deployment process for frontend and backend updates.
- **CloudFormation** will provision infrastructure, ensuring consistency across deployments.

7. Architectural Diagram

1. Users access the **frontend** via Route 53 and CloudFront (Static files stored in S3).
2. API requests are sent via **API Gateway** to **AWS Lambda** for backend processing.
3. Blog content is stored in **DynamoDB**, and images are stored in **S3**.
4. Uploaded images trigger **Lambda**, which invokes **Amazon Rekognition** for moderation.
5. If flagged, **SNS** notifies the user/admin about inappropriate content.
6. **CodePipeline** ensures continuous integration and deployment.



8. Step-by-Step Implementation

Step 1: Setup AWS Infrastructure

- **Create an S3 bucket** to host the static frontend.
- **Enable Static Website Hosting** on S3.
- **Create another S3 bucket** for storing uploaded images.
- **Set bucket policies** to manage access control.
- **Create a CloudFront Distribution** for caching and global delivery.

Step 2: Develop and Deploy the Frontend

- **Create a frontend application** using React.js (or another framework).
- **Integrate API calls** for user interactions such as uploading blog posts and images.
- **Build the project** for deployment.
- **Upload the build files to the S3 bucket** and make them publicly accessible.

Step 3: Setup API Gateway and AWS Lambda

- **Create an API Gateway** to manage backend API requests.
- **Define API endpoints** for:
    - Fetching pre-signed S3 upload URLs
    - Creating a new blog post
    - Fetching all blog posts
- **Develop AWS Lambda functions** to handle API requests.
- **Deploy the Lambda functions** and connect them to API Gateway.

Step 4: Setup DynamoDB for Blog Posts

- **Create a DynamoDB table** to store blog posts.
- **Define attributes** such as `postId`, `title`, `content`, `imageUrl`, and `createdAt`.
- **Modify Lambda functions** to interact with DynamoDB (store and retrieve posts).

Step 5: Integrate Amazon Rekognition for Image Moderation

- **Create a Lambda function** to process images using Rekognition.
- **Configure the function** to analyze uploaded images for inappropriate content.
- **Set up an S3 trigger** to invoke the function when a new image is uploaded.

Step 6: Automate Deployments using AWS CodePipeline

- **Set up AWS CodePipeline** for continuous deployment.
- **Connect CodePipeline to GitHub** for automatic updates.
- **Create a build stage using AWS CodeBuild** to compile and prepare the frontend.

- **Deploy the build to the S3 bucket** for frontend hosting.

Step 7: Add Notifications with SNS & SQS

- **Create an SNS topic** to send notifications when a blog post is published.
- **Set up SNS subscribers** (such as email notifications for admins).
- **Use SQS to queue moderation requests** if needed for scalability.

Step 8: Setup Domain with Route 53

- **Register a domain name** in Route 53.
- **Create a CNAME record** to point the domain to the CloudFront distribution.

9. Conclusion

This PoC will validate the efficiency and effectiveness of a serverless architecture for a blog platform with automated content moderation. If successful, this model can be extended to production with enhanced features such as user authentication, role-based access, and more advanced moderation techniques.