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

AWS Capstone Project

Serverless Blog Platform with Content Moderation

GitHub Repository: https://github.com/suhaib-md/Serverless_Blog_Platform_with_Content_Moderation_AWS

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. Services Used

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

5. 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

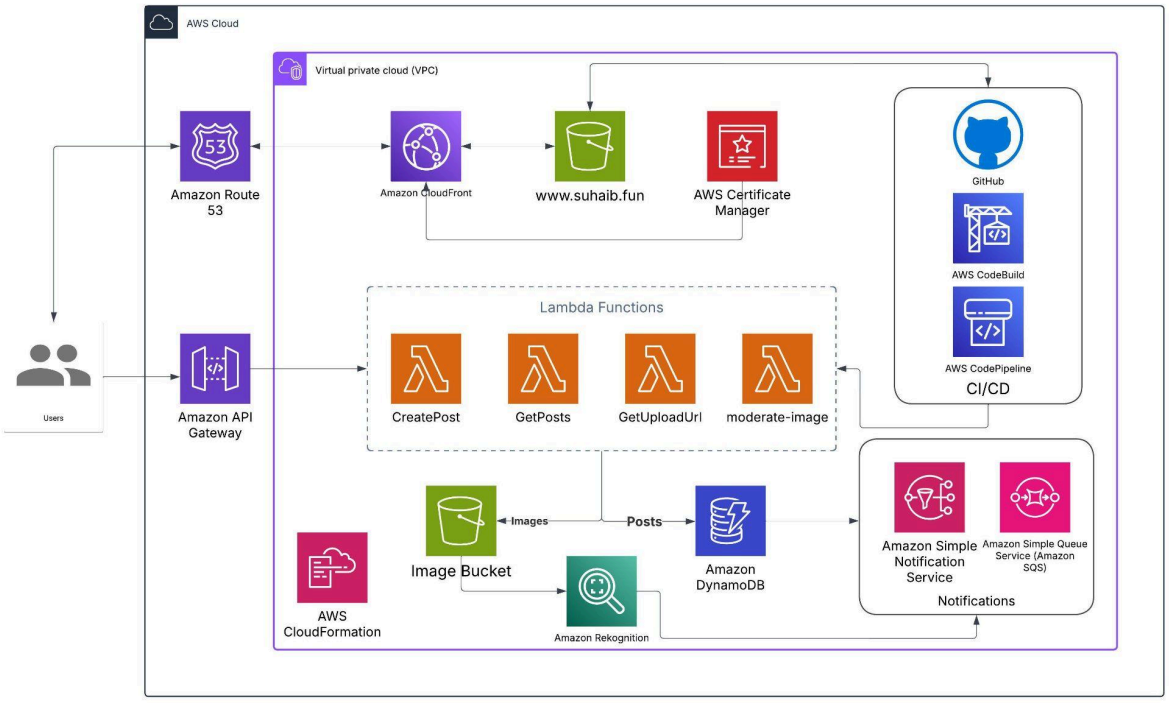
- **Service Used:** AWS CodePipeline, CloudFormation
- **CodePipeline** will automate the build and deployment process for frontend and backend updates.
- **CloudFormation** will provision infrastructure, ensuring consistency across deployments.

6. Architectural Diagram

1️⃣ **User requests the blog frontend** → CloudFront serves from S3.
2️⃣ **User submits a blog post** → API Gateway routes the request to Lambda.
3️⃣ **Lambda stores the post** → DynamoDB saves blog data.
4️⃣ **User uploads an image** → S3 triggers Lambda for moderation.
5️⃣ **Amazon Rekognition** analyzes the image.

- If safe → Stored in S3
- If flagged → SNS alerts admin, S3 adds a warning label.
  6️⃣ **CI/CD (CodePipeline + CodeBuild)** automatically deploys:
- **Frontend** to S3 & CloudFront.
- **Backend (Lambda functions)** via CodeBuild.
  7️⃣ **SNS notifications** alert users/admins on flagged content.

Serverless Blog Platform with Content Moderation

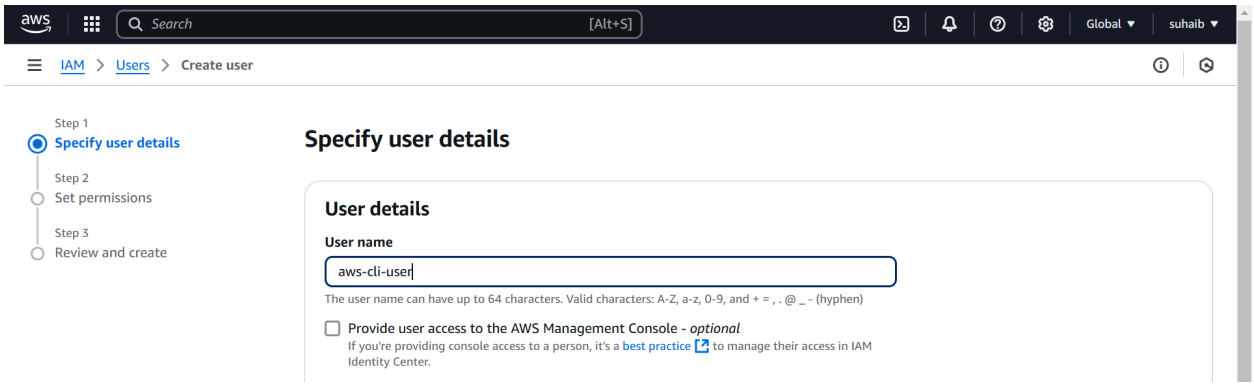


7. Project Implementation:

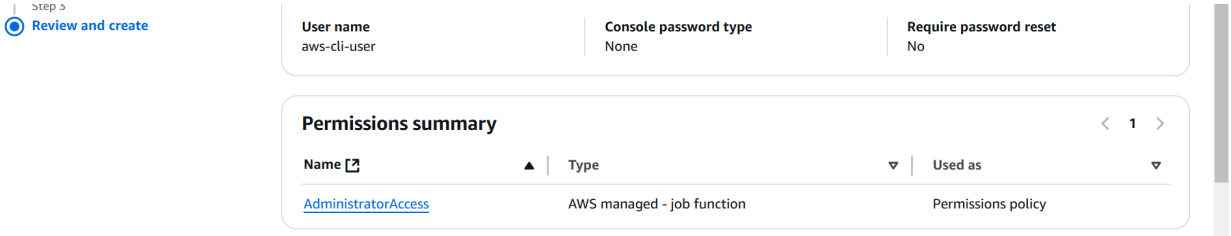
✅ Step 1: Create an IAM User for AWS CLI

1️⃣ Create an IAM User via AWS Console

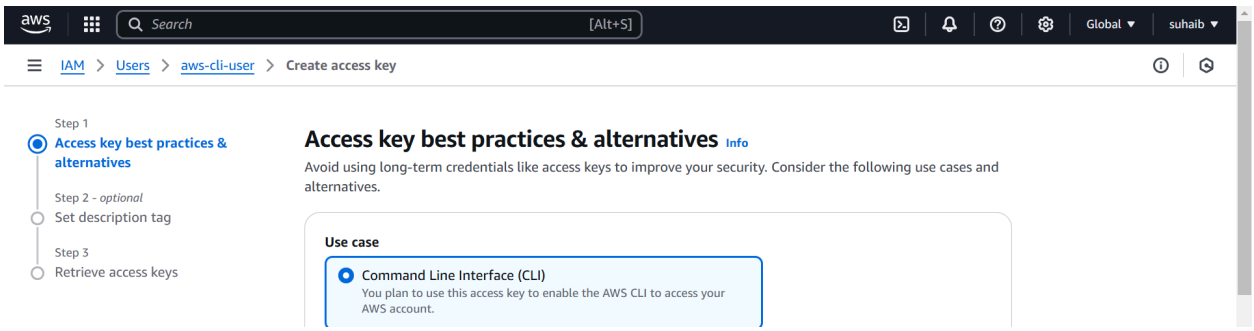
1. Go to the **AWS IAM Console** → **Users** → Click **Add User**.
2. **User Name:** aws-cli-user

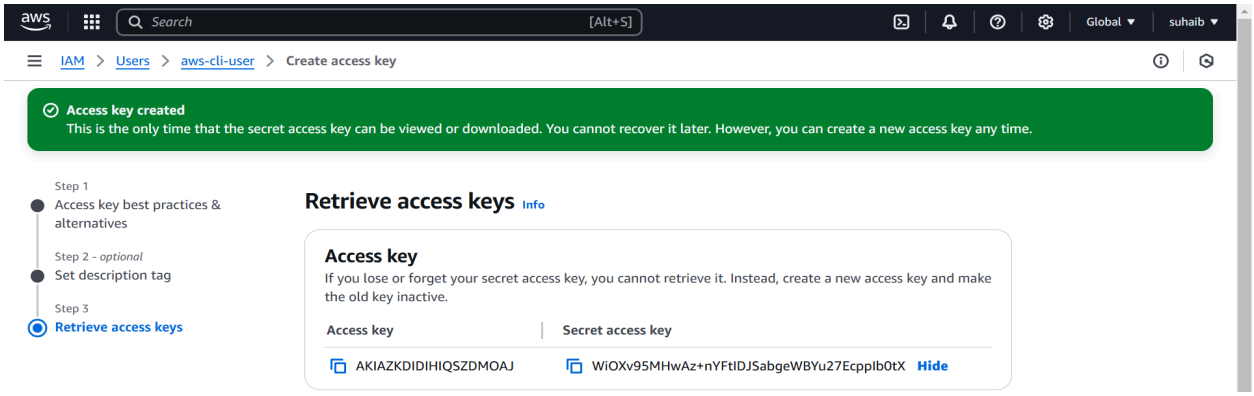


3. **Access Type:** Select **Programmatic Access**.
4. Click **Next** and attach the following policies:
    - **AdministratorAccess** (or use custom policies for security).



5. Click **Create User** and **Download the access key & secret key**.





2️⃣ Configure AWS CLI with IAM Credentials

Now, configure the AWS CLI on your local machine using the IAM user credentials:

```
aws configure
```

You'll be prompted to enter:

- **AWS Access Key ID**
- **AWS Secret Access Key**
- **Default Region (e.g., us-east-1)**
- **Output format (JSON recommended)**

```
C:\Users\Vadakathi Suhaib>aws configure
AWS Access Key ID [****************6Q5I]: AKIAZKDIDIHIQSZDMOAJ
AWS Secret Access Key [****************ZIFp]: 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
Default region name [us-east-2]: us-east-1
Default output format [json]: json

C:\Users\Vadakathi Suhaib>
```

Test the setup:

```
aws s3 ls
```

```
C:\Users\Vadakathi Suhaib>aws s3 ls
2025-02-24 15:15:17 cdcp-sampleapp-suhaib
```

If no errors appear, the AWS CLI is correctly configured! ✅

✅ Step 2: Create a Custom VPC using CloudFormation

We will now create a **VPC** with **public & private subnets**, an **Internet Gateway**, and **security groups**.

1️⃣ Create a CloudFormation Template (vpc.yml)

Create a new file called vpc.yml and paste the following YAML:

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: "CloudFormation Template for a Custom VPC"

Resources:
  MyVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: "10.0.0.0/16"
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: MyCustomVPC

  PublicSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: "10.0.1.0/24"
      MapPublicIpOnLaunch: true
      AvailabilityZone: "us-east-1a"
      Tags:
        - Key: Name
          Value: PublicSubnet

  PrivateSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: "10.0.2.0/24"
      AvailabilityZone: "us-east-1b"
```

```
      Tags:
        - Key: Name
          Value: PrivateSubnet

  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: MyInternetGateway

  GatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref MyVPC
      InternetGatewayId: !Ref InternetGateway

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref MyVPC
      Tags:
        - Key: Name
          Value: PublicRouteTable

  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: "0.0.0.0/0"
      GatewayId: !Ref InternetGateway

  PublicSubnetRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnet
      RouteTableId: !Ref PublicRouteTable

Outputs:
  VPCId:
    Description: "VPC ID"
    Value: !Ref MyVPC
```

```
  PublicSubnetId:
    Description: "Public Subnet ID"
    Value: !Ref PublicSubnet
  PrivateSubnetId:
    Description: "Private Subnet ID"
    Value: !Ref PrivateSubnet
```

2️⃣ Deploy the VPC via AWS CLI

Run the following command to create the VPC:

```
aws cloudformation create-stack --stack-name MyVPCStack --template-body "file://C:/Users/Vadakathi Suhaib/Documents/Training @ Cprime/AWS/AWS Capstone Project/CloudFormation/vpc.yml" --capabilities CAPABILITY_NAMED_IAM
```

```
C:\Users\Vadakathi Suhaib>aws cloudformation create-stack --stack-name MyVPCStack --template-body "file://C:/Users/Vadak
athi Suhaib/Documents/Training @ Cprime/AWS/AWS Capstone Project/CloudFormation/vpc.yml" --capabilities CAPABILITY_NAMED
_IAM
{
    "StackId": "arn:aws:cloudformation:us-east-1:640168444369:stack/MyVPCStack/7055bbe0-f4cb-11ef-8ead-12ec7a6f0bf3"
}


C:\Users\Vadakathi Suhaib>
```

Monitor the deployment:

```
aws cloudformation describe-stacks --stack-name MyVPCStack --query "Stacks[0].StackStatus"
```

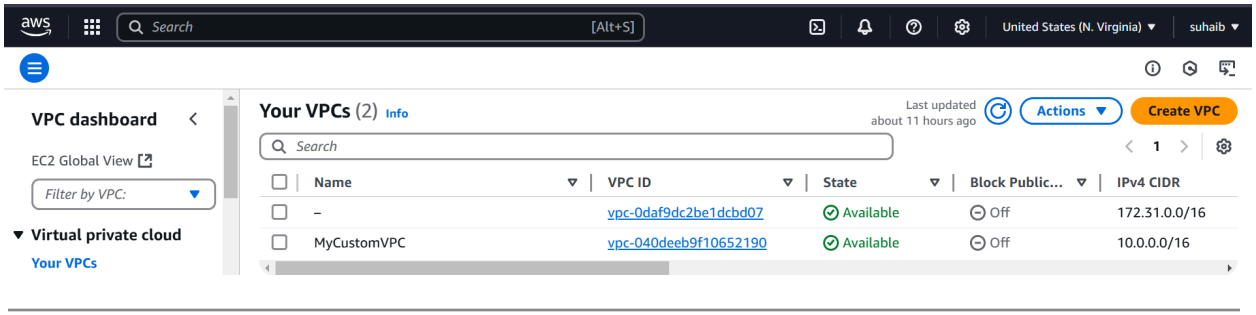
```
C:\Users\Vadakathi Suhaib>aws cloudformation describe-stacks --stack-name MyVPCStack --query "Stacks[0].StackStatus"
"CREATE_IN_PROGRESS"


C:\Users\Vadakathi Suhaib>aws cloudformation describe-stacks --stack-name MyVPCStack --query "Stacks[0].StackStatus"
"CREATE_COMPLETE"


C:\Users\Vadakathi Suhaib>
```

Get VPC details after deployment:

```
aws cloudformation describe-stacks --stack-name MyVPCStack --query "Stacks[0].Outputs"
```

```
C:\Users\Vadakathi Suhaib>aws cloudformation describe-stacks --stack-name MyVPCStack --query "Stacks[0].Outputs"
[
    {
        "OutputKey": "VPCId",
        "OutputValue": "vpc-09de69d3662e247b2",
        "Description": "VPC ID"
    },
    {
        "OutputKey": "PublicSubnetId",
        "OutputValue": "subnet-072f1b8dff65bdbc6",
        "Description": "Public Subnet ID"
    },
    {
        "OutputKey": "PrivateSubnetId",
        "OutputValue": "subnet-0b05c6e268f5a74b2",
        "Description": "Private Subnet ID"
    }
]
```


```
[
    {
        "OutputKey": "VPCId",
        "OutputValue": "vpc-09de69d3662e247b2",
        "Description": "VPC ID"
    },
    {
        "OutputKey": "PublicSubnetId",
        "OutputValue": "subnet-072f1b8dff65bdbc6",
        "Description": "Public Subnet ID"
    },
    {
        "OutputKey": "PrivateSubnetId",
        "OutputValue": "subnet-0b05c6e268f5a74b2",
        "Description": "Private Subnet ID"
    }
]
```

✅ Step 3: Setup Frontend for the application using S3, CloudFront and Route 53

This section outlines the step-by-step process to deploy a **React application** using **AWS CloudFront** with an **S3 static website hosting architecture**. The deployment includes domain management with **Amazon Route 53**, security via **AWS Certificate Manager (ACM)**, and content delivery using **CloudFront distributions**.

1️⃣ Architecture Overview

This process follows this structure:

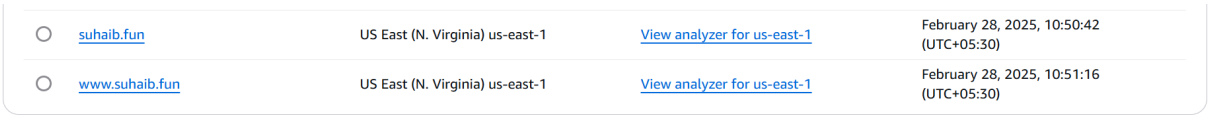
- **Amazon S3**:
  - **www.suhaib.fun** (Public) – Hosts the static React application.
  - **suhaib.fun** (Private) – Redirects traffic to **www.suhaib.fun**.
- **Amazon Route 53**:
  - Manages the domain **suhaib.fun** and its DNS records.
- **AWS Certificate Manager (ACM)**:
  - Issues an SSL certificate for **HTTPS support**.
- **AWS CloudFront**:
  - Distributes content globally with **HTTPS enforcement** and **caching**.

2️⃣ Implementation Steps

2.1 Setting Up S3 Buckets

Step 1: Create Two S3 Buckets

Bucket Name	Type	Access Control	Purpose
suhaib.fun	Private	Block Public Access ✅	Redirects to www.suhaib.fun
www.suhaib.fun	Public	Block Public Access ❌	Hosts the static website



Alternately, We can deploy S3 buckets through CloudFormation:

Create an S3 CloudFormation Template (s3-website.yml):

- Create a new file s3-website.yml and paste this:

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: www.suhaib.fun
      OwnershipControls:
        Rules:
          - ObjectOwnership: BucketOwnerEnforced  # Enforces bucket
ownership
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: false
        IgnorePublicAcls: true
        RestrictPublicBuckets: false
  S3BucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref S3Bucket
      PolicyDocument:
        Statement:
          - Effect: Allow
            Principal: "*"
            Action: "s3:GetObject"
            Resource: !Sub "arn:aws:s3:::${S3Bucket}/*"
```

Deploy S3 Bucket using AWS CLI

Run the following command:

```
aws cloudformation create-stack --stack-name S3WebsiteStack --template-body "file://C:/Users/Vadakathi Suhaib/Documents/Training @ Cprime/AWS/AWS Capstone Project/CloudFormation/s3-website.yml" --capabilities CAPABILITY_NAMED_IAM
```

```
C:\Users\Vadakathi Suhaib>aws cloudformation create-stack --stack-name S3WebsiteStack --template-body "file://C:/Users/Vadakathi Suhaib/Documents/Training @ Cprime/AWS/AWS Capstone Project/CloudFormation/s3-website.yml" --capabilities CAPABILITY_NAMED_IAM
{
    "StackId": "arn:aws:cloudformation:us-east-1:640168444369:stack/S3WebsiteStack/4c6bd3b0-f4ce-11ef-a1da-0eb45e5c7d2d"
}
```

Once deployed, check status:

```
aws cloudformation describe-stacks --stack-name S3WebsiteStack --query "Stacks[0].StackStatus"
```

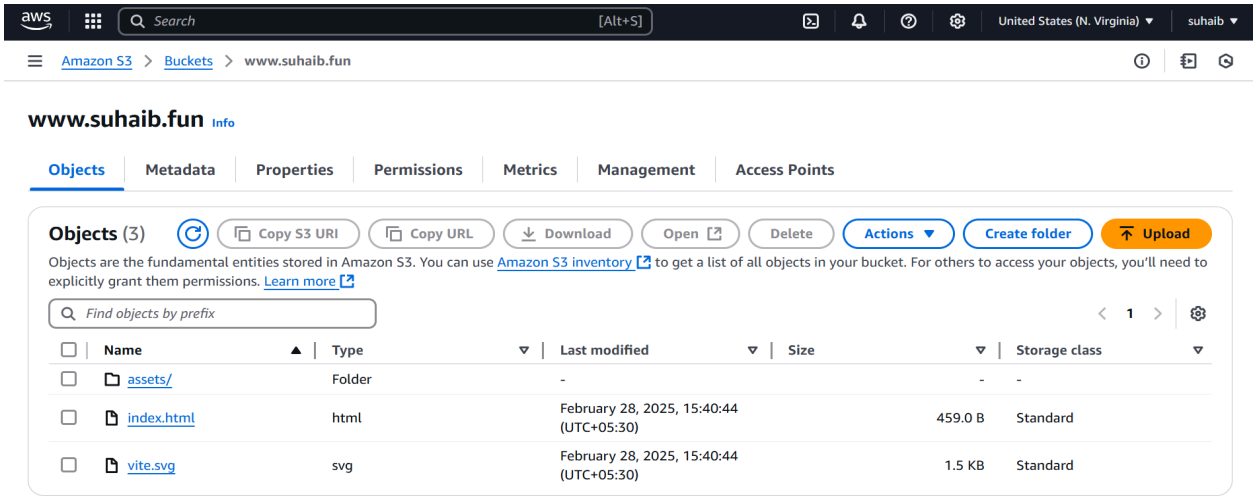
```
C:\Users\Vadakathi Suhaib>aws cloudformation describe-stacks --stack-name S3WebsiteStack --query "Stacks[0].StackStatus"
"CREATE_COMPLETE"
```

Step 2: Upload React Application

1. Build the React application using:

```
npm run build
```

2. Navigate to the dist/ directory.
3. Upload all files to the www.suhaib.fun S3 bucket.



Alternately, we can upload using AWS CLI:

Upload React Build Files to S3

Now, deploy your frontend:

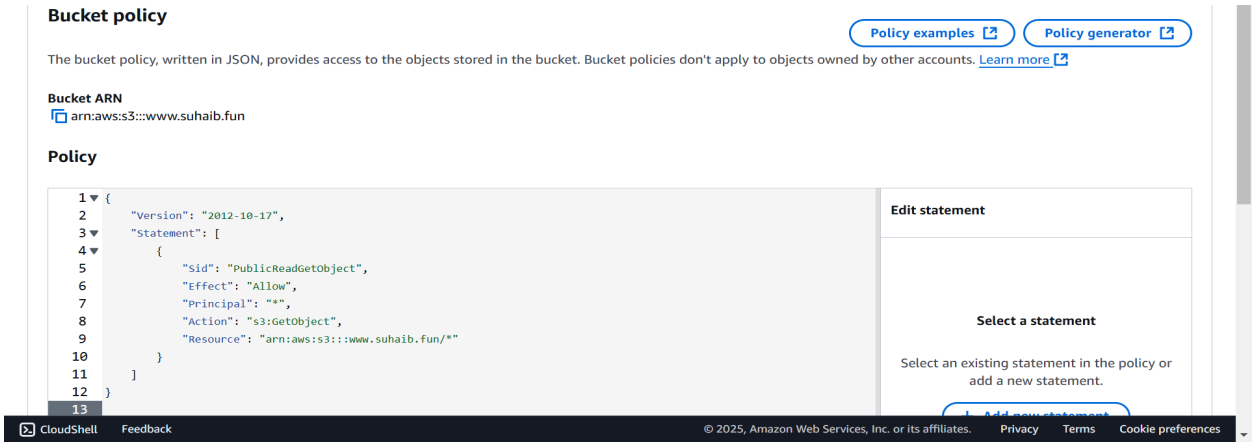
```
npm run build
aws s3 sync "C:/Users/Vadakathi Suhaib/Documents/Training @ Cprime/AWS/AWS Capstone Project/serverlessblog/dist" s3://my-serverless-blog-frontend
```

```
C:\Users\Vadakathi Suhaib>aws s3 sync "C:/Users/Vadakathi Suhaib/Documents/Training @ Cprime/AWS/AWS Capstone Project/serverlessblog/dist" s3://my-serverless-blog-frontend
upload: Documents\Training @ Cprime\AWS\AWS Capstone Project\serverlessblog\dist\vite.svg to s3://my-serverless-blog-frontend/vite.svg
upload: Documents\Training @ Cprime\AWS\AWS Capstone Project\serverlessblog\dist\index.html to s3://my-serverless-blog-frontend/index.html
upload: Documents\Training @ Cprime\AWS\AWS Capstone Project\serverlessblog\dist\assets\index-WDPIioA3.css to s3://my-serverless-blog-frontend/assets/index-WDPIioA3.css
upload: Documents\Training @ Cprime\AWS\AWS Capstone Project\serverlessblog\dist\assets\index-CJw4L29H.js to s3://my-serverless-blog-frontend/assets/index-CJw4L29H.js

C:\Users\Vadakathi Suhaib>
```

Step 3: Configure Bucket Policy

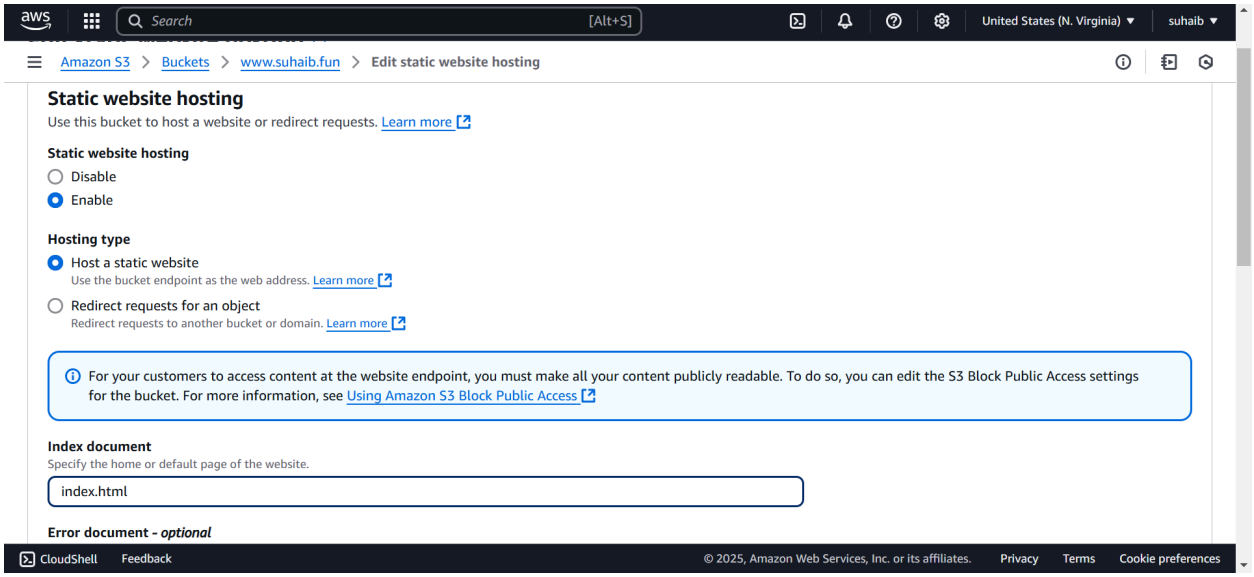
1. Navigate to **Bucket Policy** under **Permissions** for www.suhaib.fun.
2. Add the following policy (update domain name as needed):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::www.suhaib.fun/*"
        }
    ]
}
```



Step 4: Enable Static Website Hosting

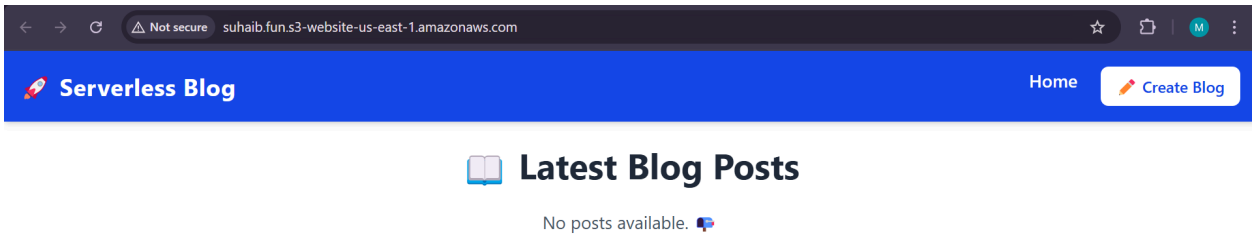
- **For www.suhaib.fun (Public)**:
    - Enable static website hosting.
    - Set **Index Document** to index.html.

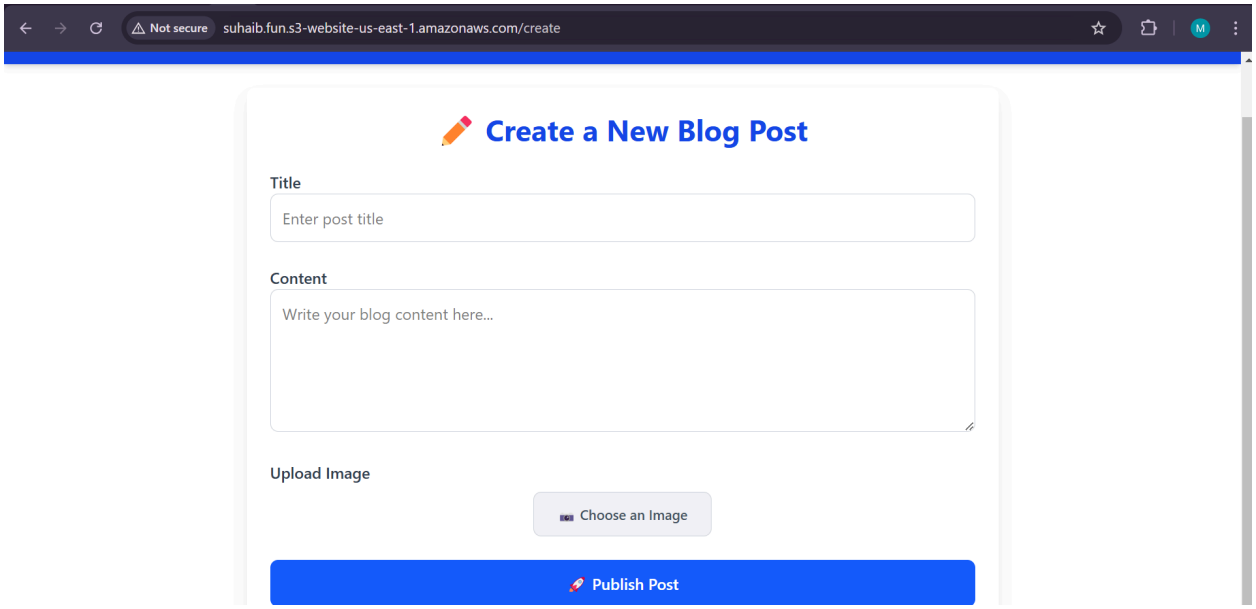


- **For suhaib.fun (Private)**:
    - Enable static website hosting.
    - Redirect all requests to www.suhaib.fun (Protocol: HTTP).

Verification Tests

✅ Open http://www.suhaib.fun.s3-website-us-east-1.amazonaws.com to check if the app loads.



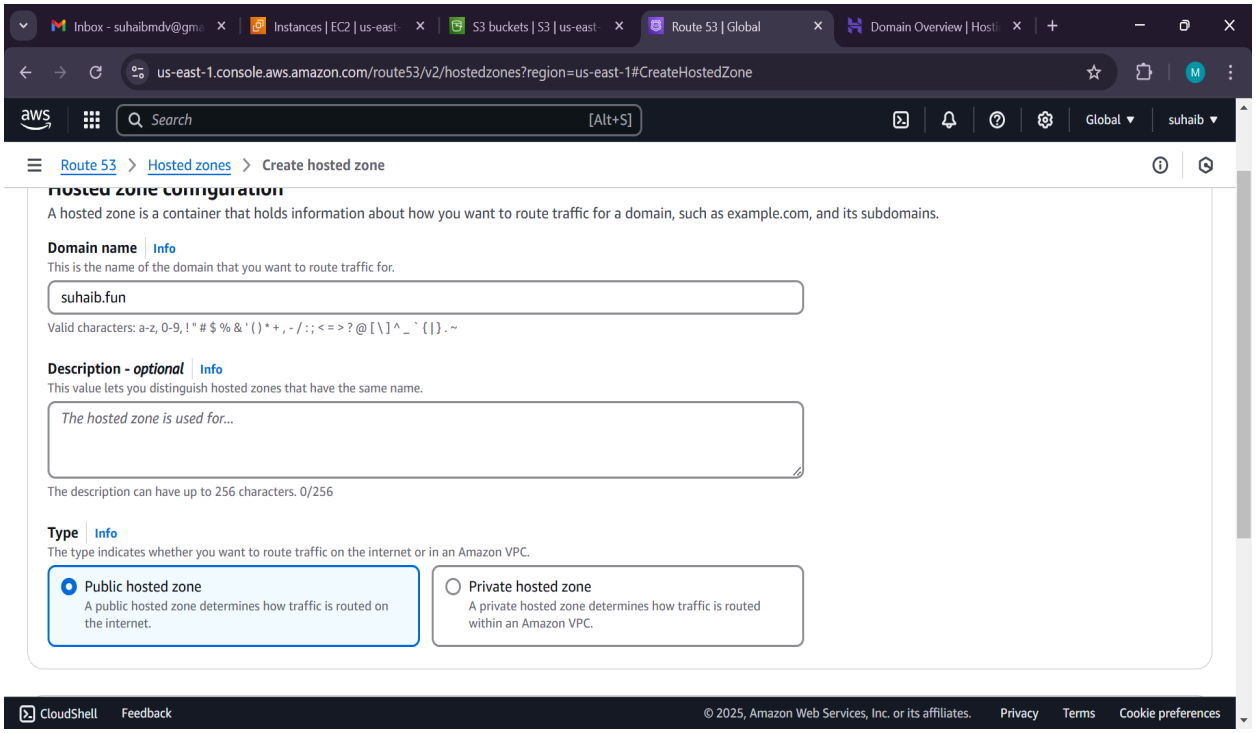


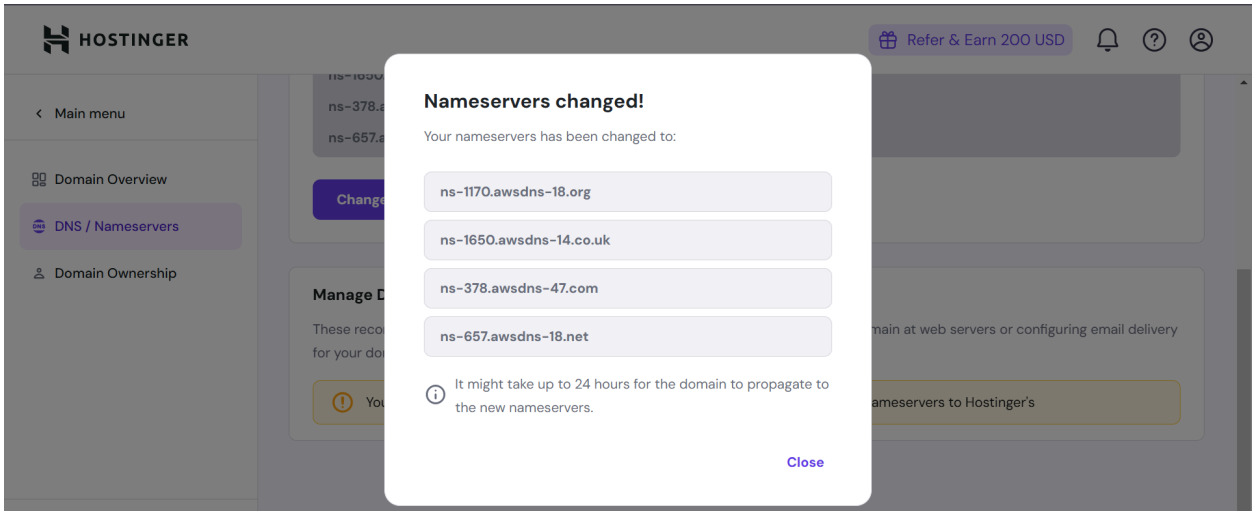
✅ Open http://suhaib.fun.s3-website-us-east-1.amazonaws.com to check if it redirects correctly.

2.2 Configuring Amazon Route 53

Step 1: Create Hosted Zone

1. Navigate to **Amazon Route 53**.
2. Create a **Public Hosted Zone** with the domain **suhaib.fun**.
3. Copy the **Name Servers (NS)** and update them in **Hostinger (or another domain registrar)**.

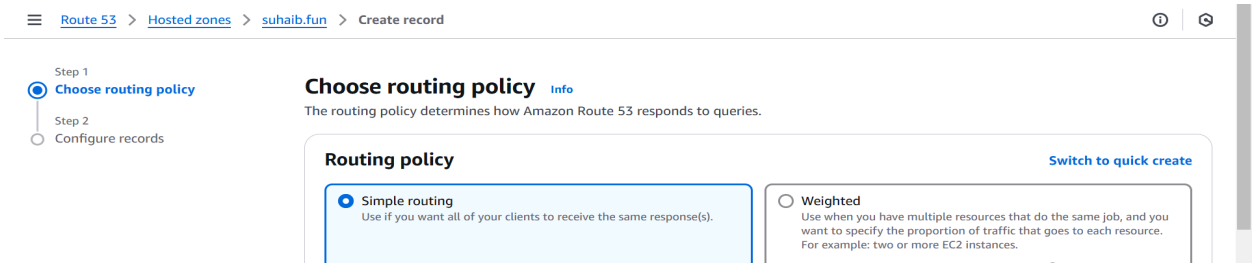


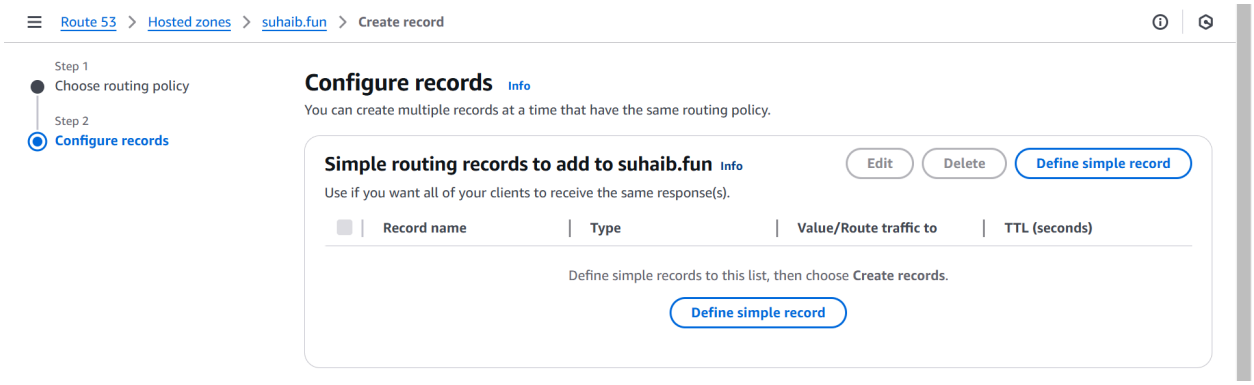


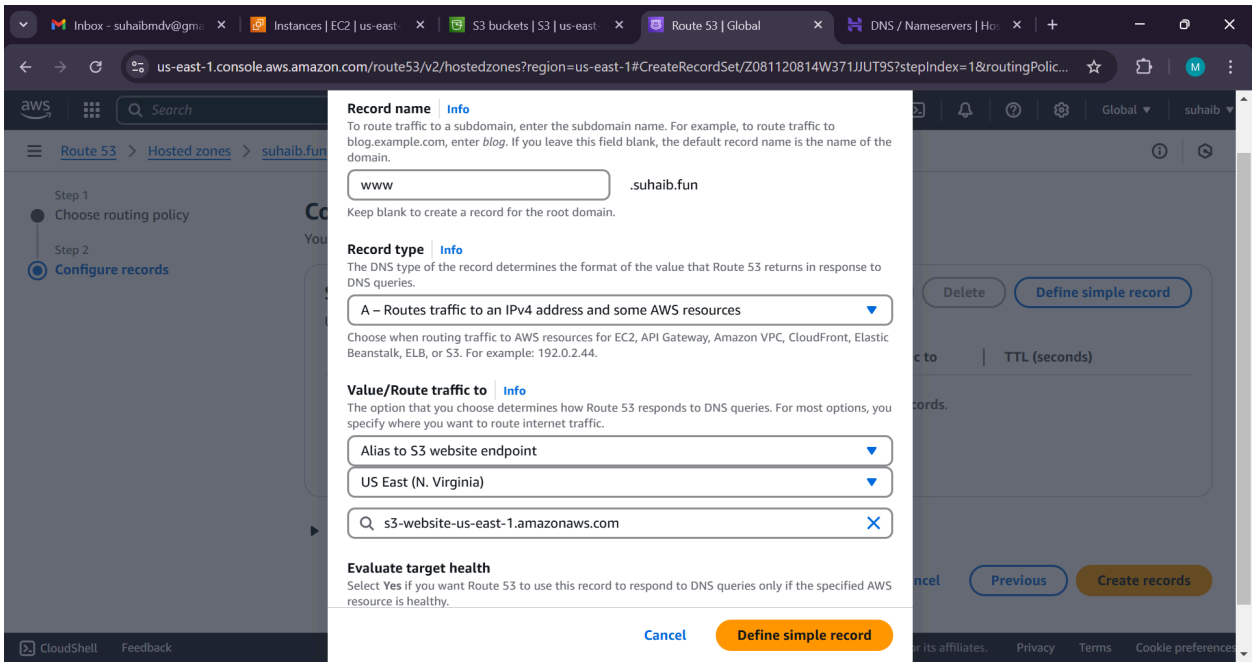
Step 2: Configure DNS Records

- **Create an A Record for www.suhaib.fun**
  - Type: **A**

- Name: www
- Target: **S3 Bucket (www.suhaib.fun)**
- Region: **Northern Virginia (us-east-1)**
- Health Check: **No**

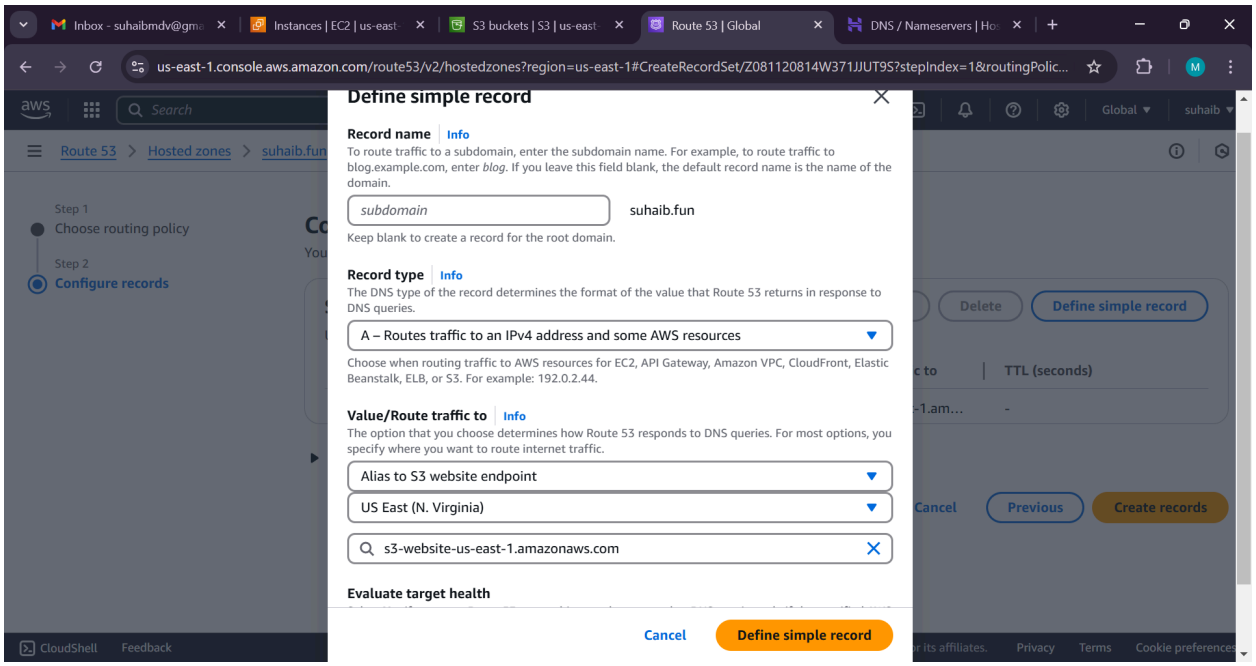


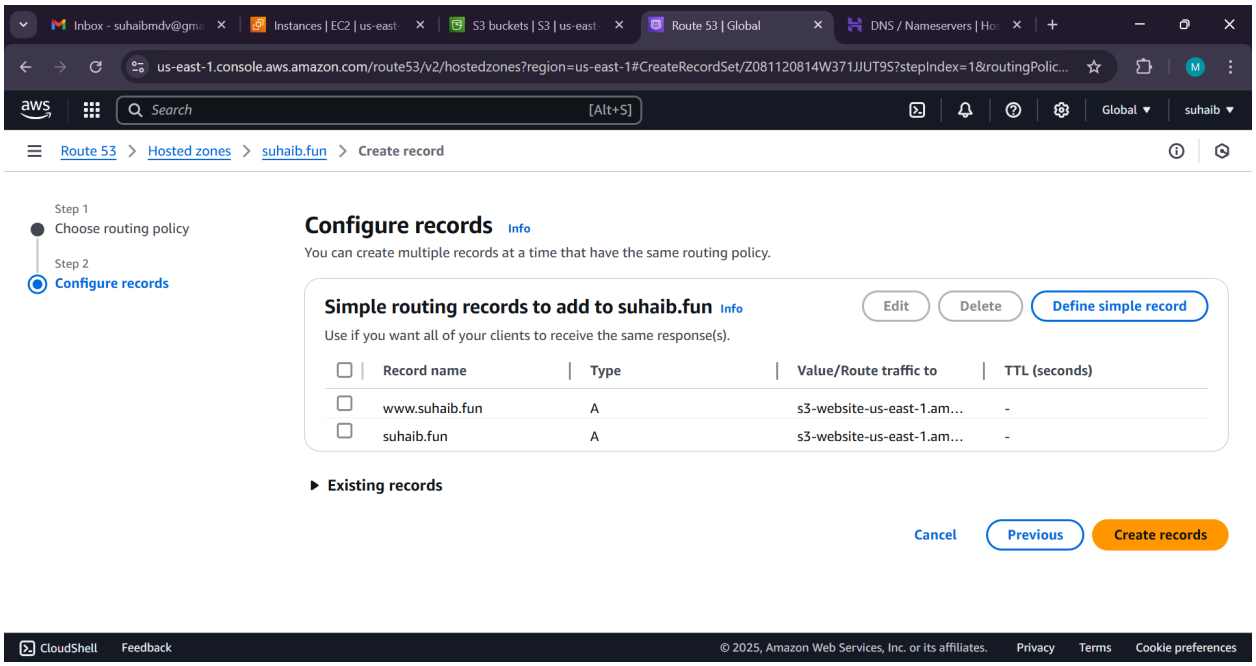




- **Create an A Record for suhaib.fun**

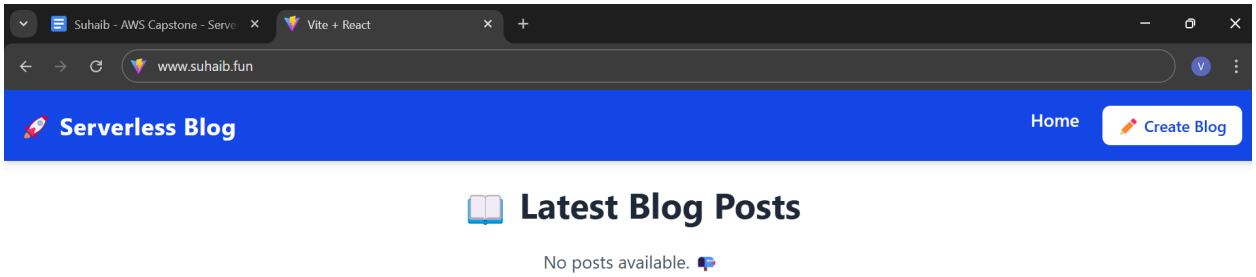
- Type: **A**
- Name: *(leave empty)*
- Target: **S3 Bucket (suhaib.fun)**
- Region: **Northern Virginia (us-east-1)**
- Health Check: **Yes**





Verification Tests

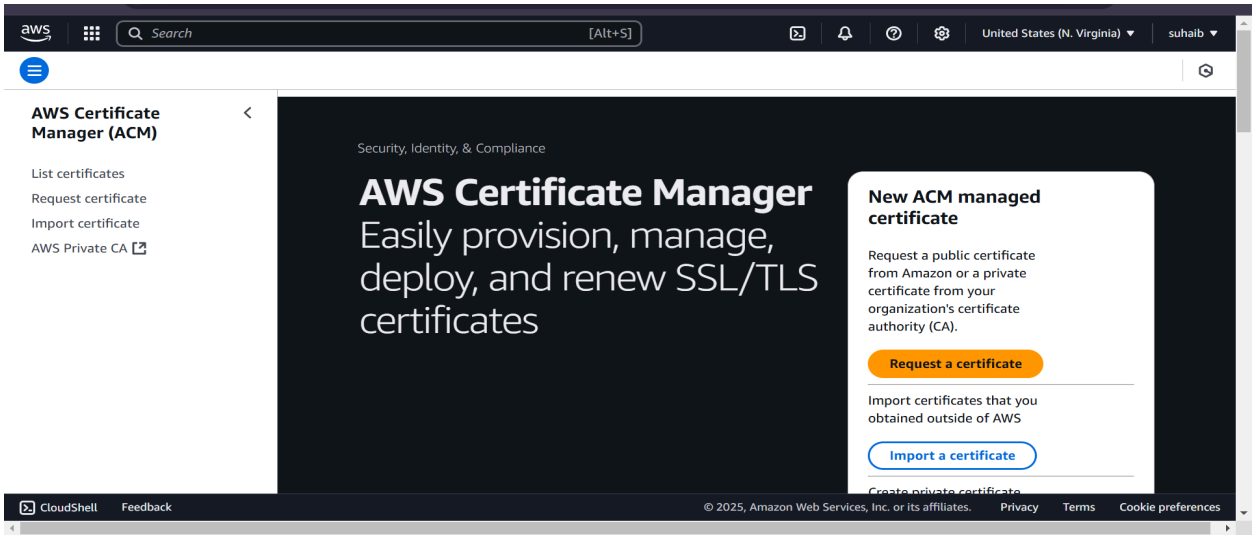
✅ Open www.suhaib.fun → React app should load.



✅ Open suhaib.fun → It should redirect to www.suhaib.fun.

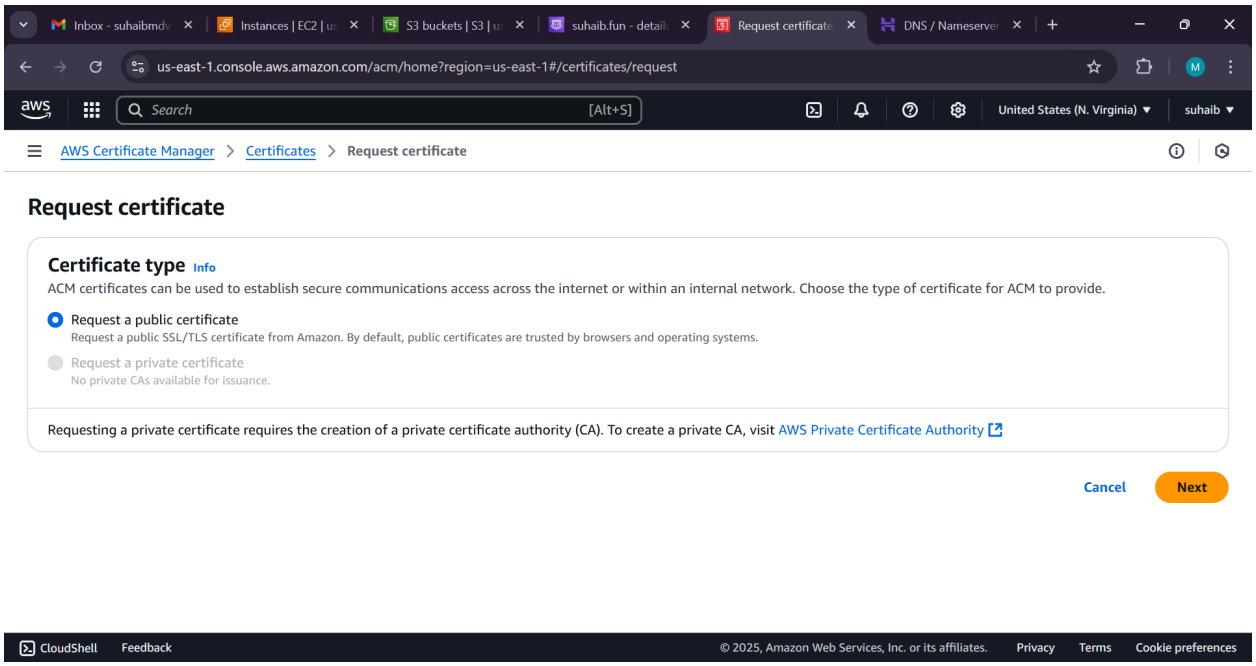
2.3 Requesting an SSL Certificate

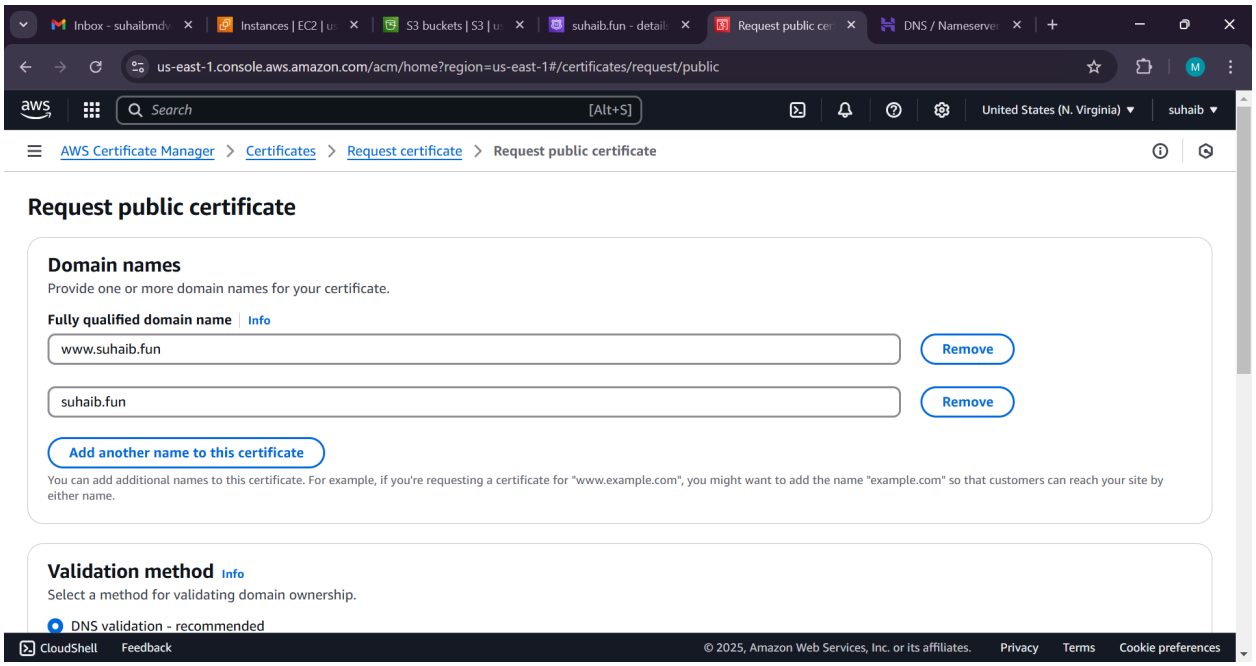
1. Navigate to **AWS Certificate Manager (ACM)**.

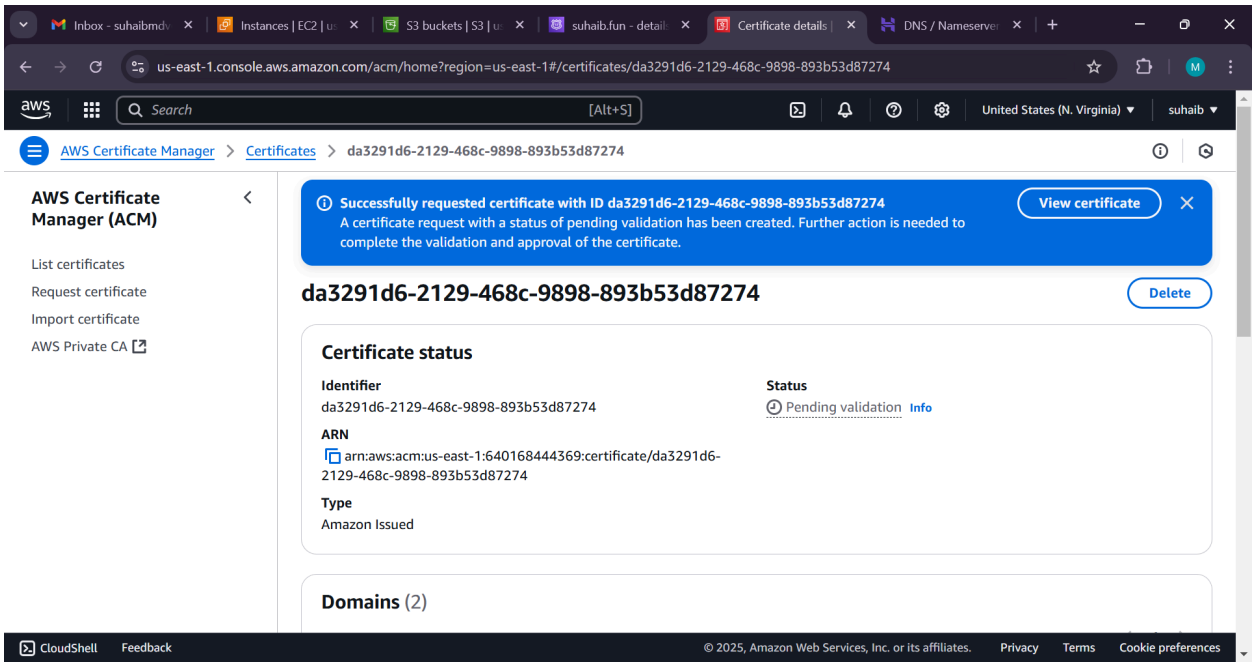


2. Request a **Public SSL Certificate** for:

- suhaib.fun
- www.suhaib.fun



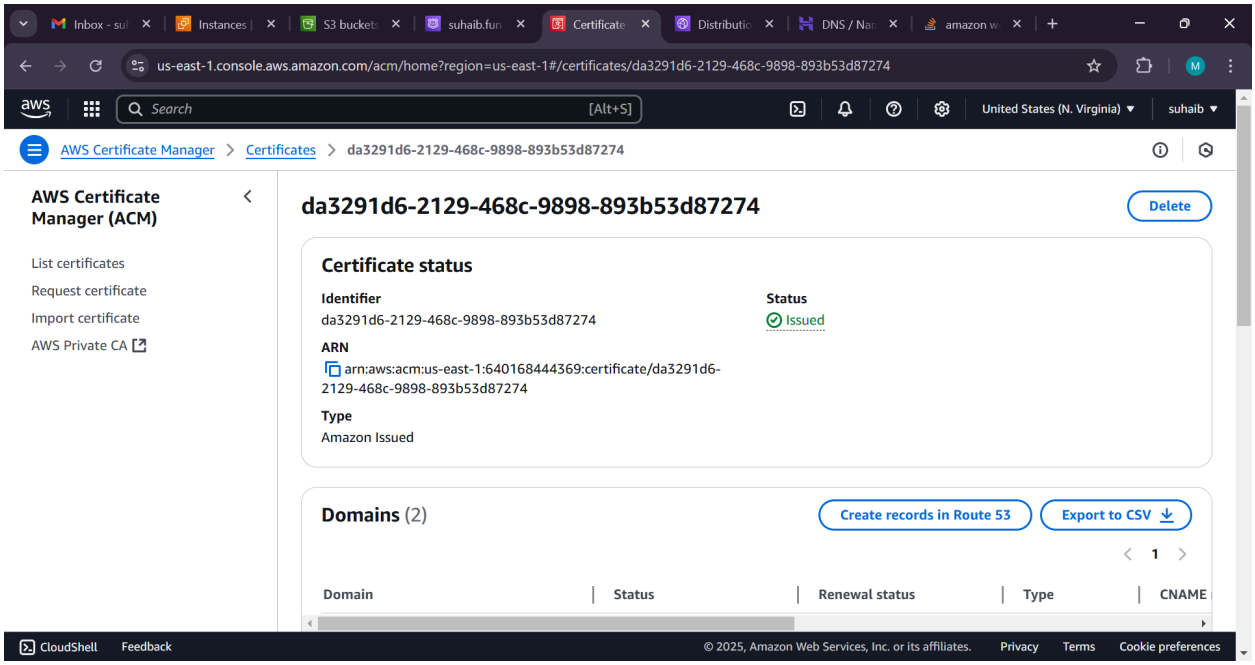




3. Choose **DNS validation**.
4. Create the required **DNS records in Route 53**.

Verification Tests

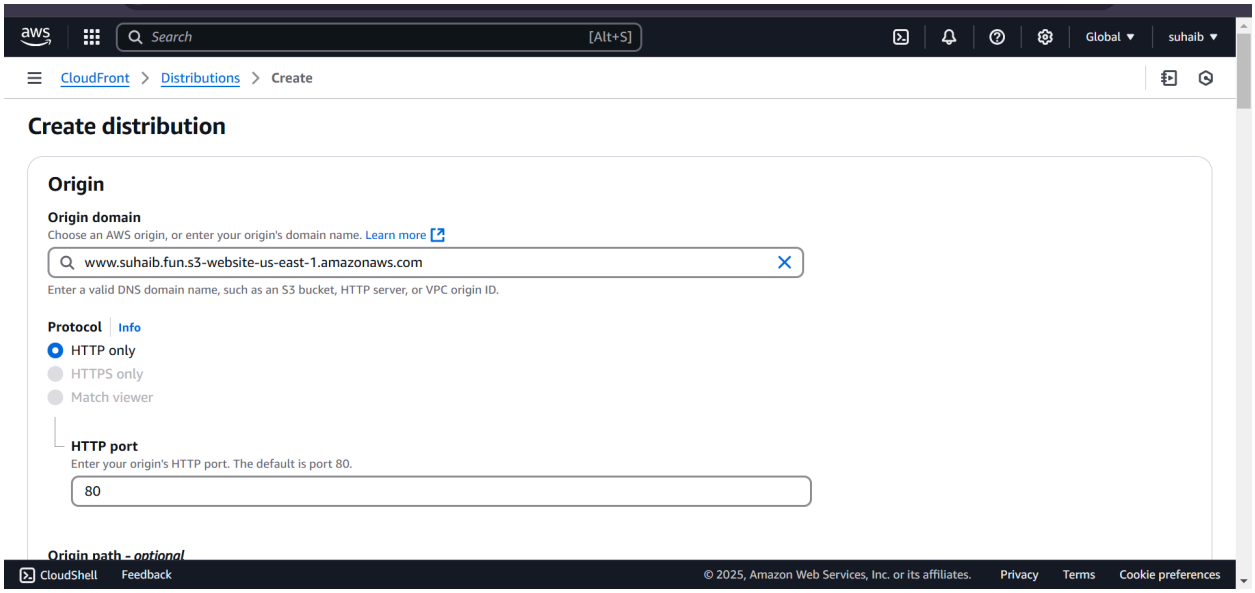
✅ Check **Certificate Manager** to ensure the certificate is **Issued** before proceeding.



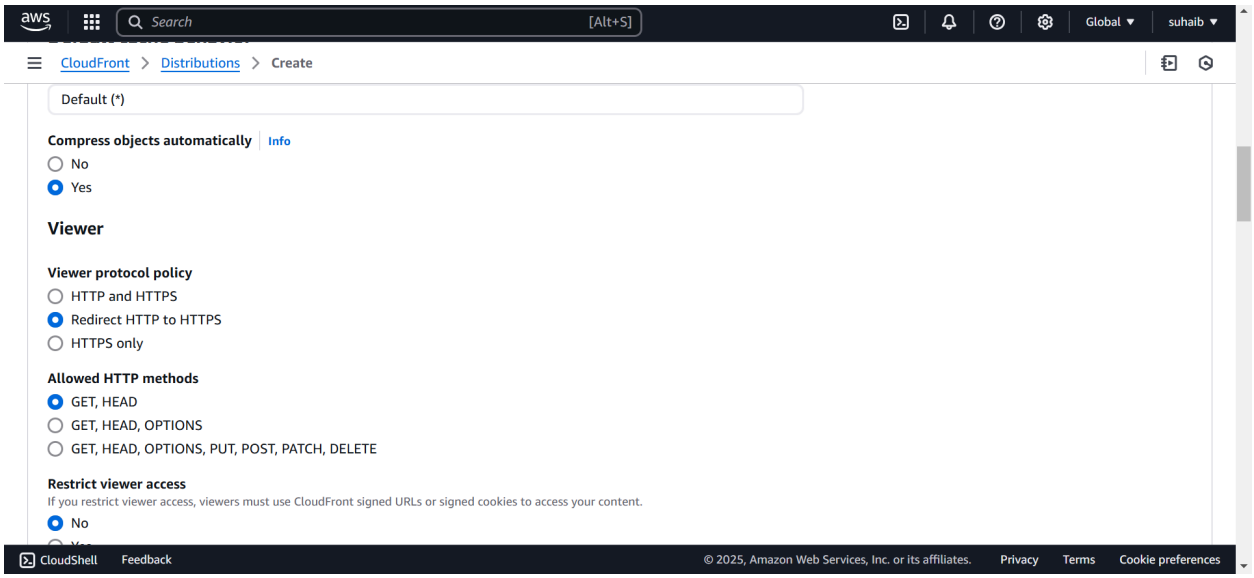
2.4 Configuring CloudFront

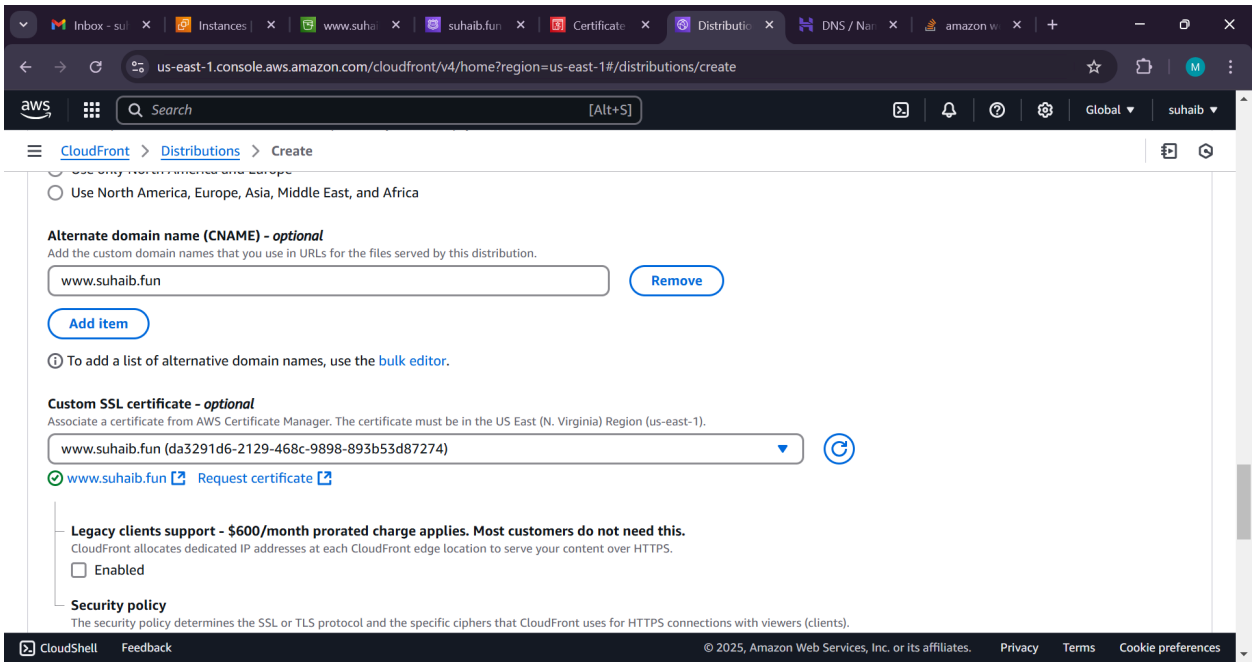
Step 1: Create CloudFront Distribution for WWW

- **Origin Domain**: Use the **Static Website Hosting URL** from the **www.suhaib.fun** S3 bucket.



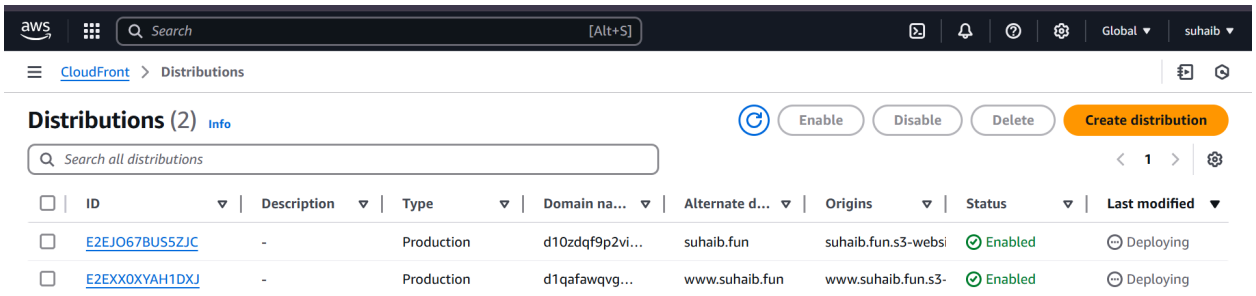
- **Settings**:
  - Redirect **HTTP to HTTPS**
  - Alternate Domain Name (CNAME): www.suhaib.fun
  - Attach the **SSL Certificate**





Step 2: Create CloudFront Distribution for Non-WWW

- **Origin Domain**: Use the **Static Website Hosting URL** from the **suhaib.fun** S3 bucket.
- **Settings**:
  - Redirect **HTTP to HTTPS**
  - Alternate Domain Name (CNAME): suhaib.fun
  - Attach the **SSL Certificate**



Verification Tests

✅ Open the CloudFront **domain URL** for www.suhaib.fun → The site should load over **HTTPS**.

✅ Open the CloudFront **domain URL** for suhaib.fun → It should redirect correctly.

Alternately we can use CloudFormation to setup CloudFront:

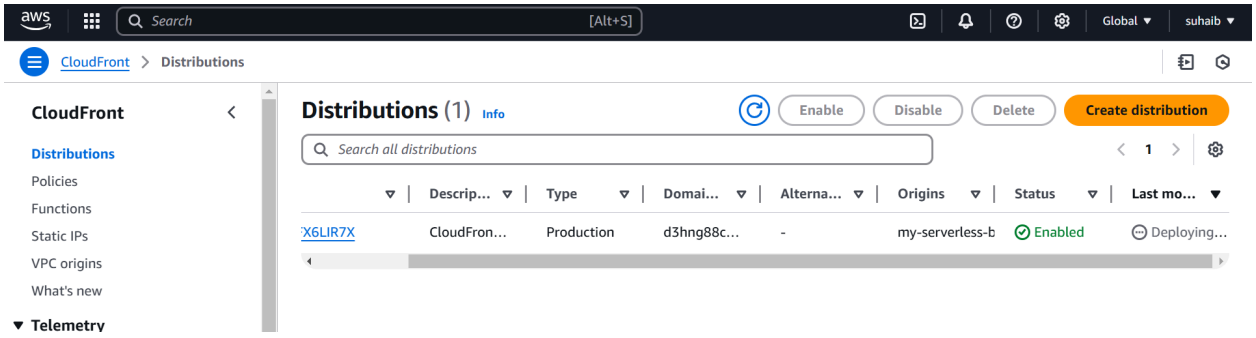
Create a **CloudFormation YAML file** (cloudfront.yml) with the following content:

```
AWSTemplateFormatVersion: '2010-09-09'
Parameters:
  ExistingS3Bucket:
    Type: String
    Description: "Name of the existing S3 bucket hosting the static site"

Resources:
  MyCloudFrontOAC:
    Type: AWS::CloudFront::OriginAccessControl
    Properties:
      OriginAccessControlConfig:
        Name: MyS3OAC
        Description: OAC for S3 static website
        SigningProtocol: sigv4
        SigningBehavior: always
        OriginAccessControlOriginType: s3

  MyCloudFrontDistribution:
    Type: AWS::CloudFront::Distribution
    Properties:
```

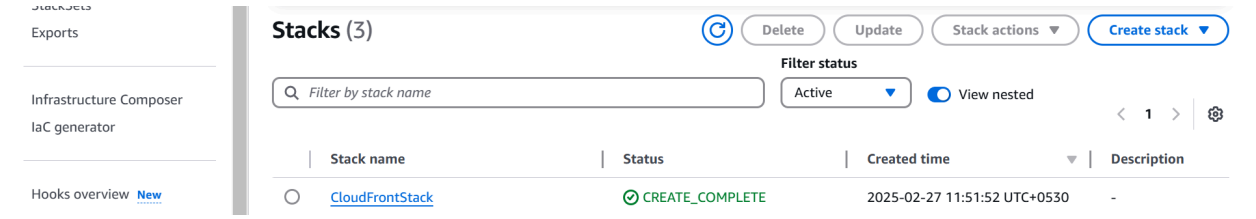
```
      DistributionConfig:
        Enabled: true
        Comment: "CloudFront Distribution for S3 Static Website"
        DefaultRootObject: "index.html"
        Origins:
          - Id: S3Origin
            DomainName: !Sub "${ExistingS3Bucket}.s3.amazonaws.com"
            S3OriginConfig:
              OriginAccessIdentity: ""
            OriginAccessControlId: !Ref MyCloudFrontOAC
        DefaultCacheBehavior:
          TargetOriginId: S3Origin
          ViewerProtocolPolicy: "redirect-to-https"
          AllowedMethods: ["GET", "HEAD"]
          CachedMethods: ["GET", "HEAD"]
          Compress: true
          CachePolicyId: "658327ea-f89d-4fab-a63d-7e88639e58f6"  # AWS
managed caching policy
        PriceClass: "PriceClass_100"
        ViewerCertificate:
          CloudFrontDefaultCertificate: true
        CustomErrorResponses:
          - ErrorCode: 403
            ResponsePagePath: "/index.html"
            ResponseCode: 200
          - ErrorCode: 404
            ResponsePagePath: "/index.html"
            ResponseCode: 200

Outputs:
  CloudFrontDomain:
    Description: "CloudFront Distribution Domain Name"
    Value: !GetAtt MyCloudFrontDistribution.DomainName
```

Run the following **AWS CLI command** to deploy the CloudFront stack:

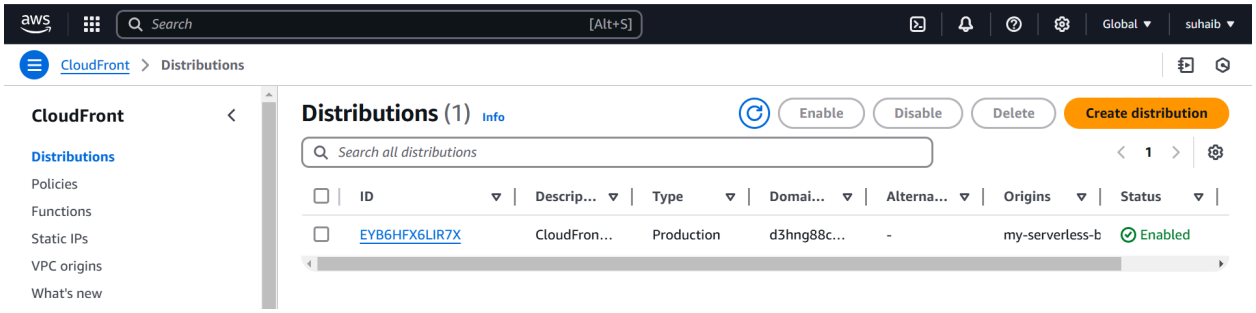
```
aws cloudformation create-stack --stack-name CloudFrontStack
--template-body "file://C:/Users/Vadakathi Suhaib/Documents/Training @
Cprime/AWS/AWS Capstone Project/CloudFormation/cloudfront.yml" --parameters
ParameterKey=ExistingS3Bucket,ParameterValue=my-serverless-blog-frontend
--capabilities CAPABILITY_NAMED_IAM
```




```
C:\Users\Vadakathi Suhaib>aws cloudformation create-stack --stack-name CloudFrontStack --template-body "file://C:/Users/
Vadakathi Suhaib/Documents/Training @ Cprime/AWS/AWS Capstone Project/CloudFormation/cloudfront.yml" --parameters Parame
terKey=ExistingS3Bucket,ParameterValue=my-serverless-blog-frontend --capabilities CAPABILITY_NAMED_IAM
{
    "StackId": "arn:aws:cloudformation:us-east-1:640168444369:stack/CloudFrontStack/c7b62fd0-f4d2-11ef-ac9f-0affc335abeb
"
}
```



- **Wait for the stack to be created** (~10 minutes).



- Once completed, retrieve the **CloudFront URL**:

```
aws cloudformation describe-stacks --stack-name CloudFrontStack --query
"Stacks[0].Outputs"
```

```
C:\Users\Vadakathi Suhaib>aws cloudformation describe-stacks --stack-name CloudFrontStack --query "Stacks[0].Outputs"
[
    {
        "OutputKey": "CloudFrontDomain",
        "OutputValue": "d3hng88c8fe6kp.cloudfront.net",
        "Description": "CloudFront Distribution Domain Name"
    }
]
```

```
[
    {
```

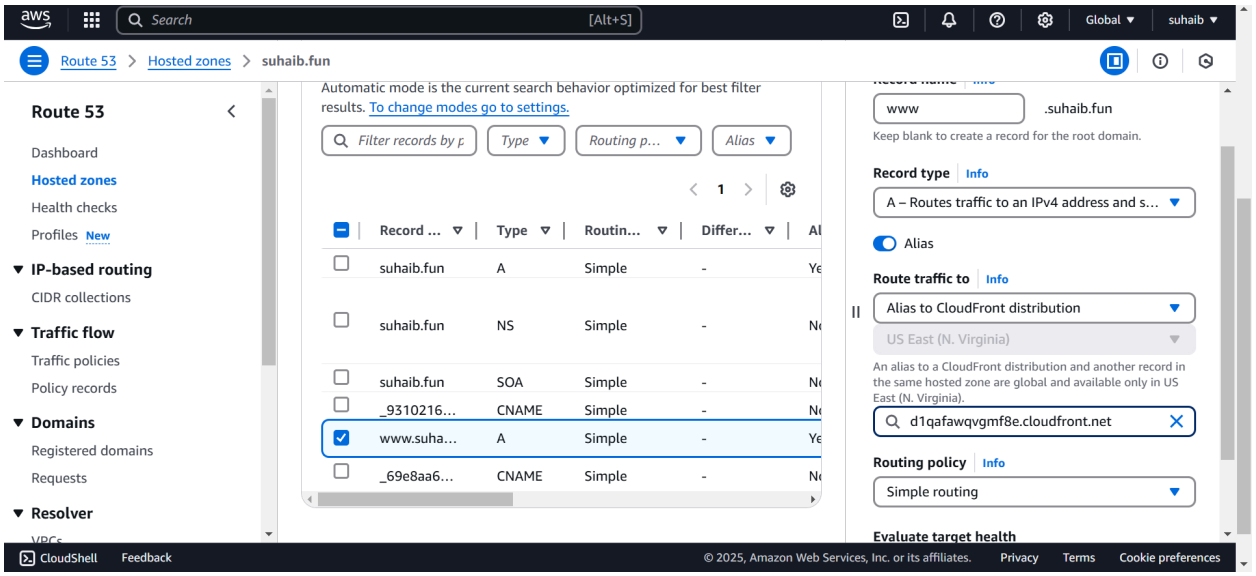
```
        "OutputKey": "CloudFrontDomain",
        "OutputValue": "d3hng88c8fe6kp.cloudfront.net",
        "Description": "CloudFront Distribution Domain Name"
    }
]
```

✅ **Copy the CloudFront domain name** and access your website using the new URL!

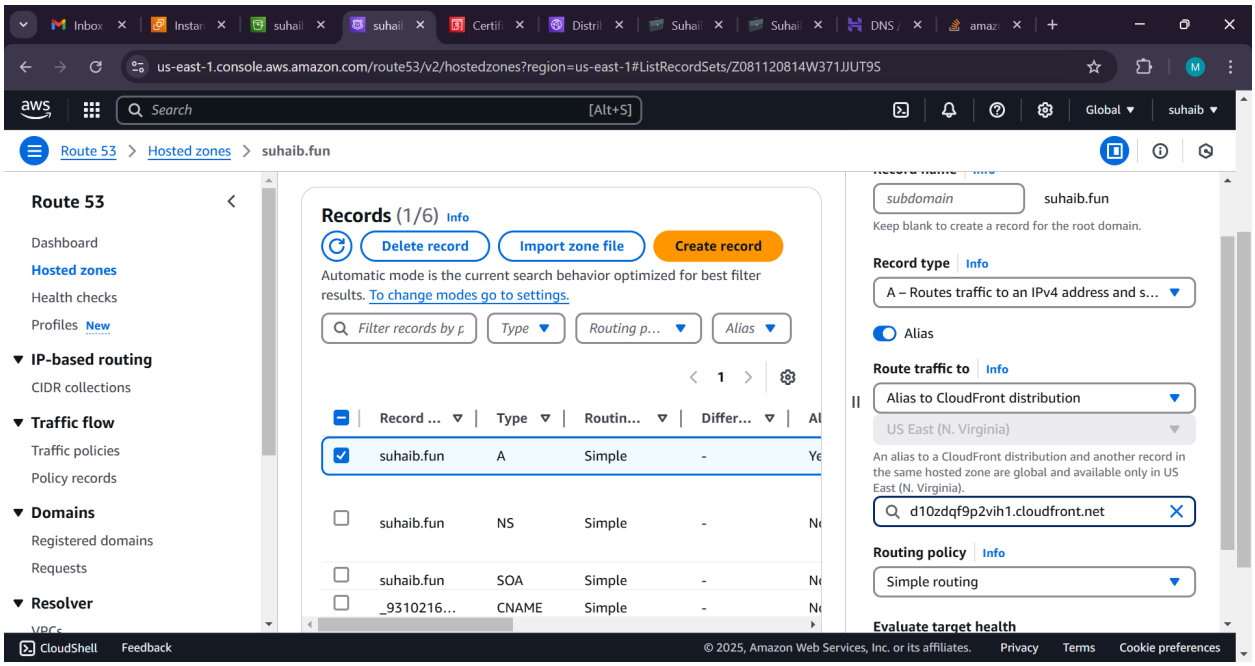
2.5 Updating Route 53 for CloudFront

Step 1: Update A Records to CloudFront

1. Edit the existing **A Record for WWW**:
    - Target: **CloudFront distribution for www.suhaib.fun**

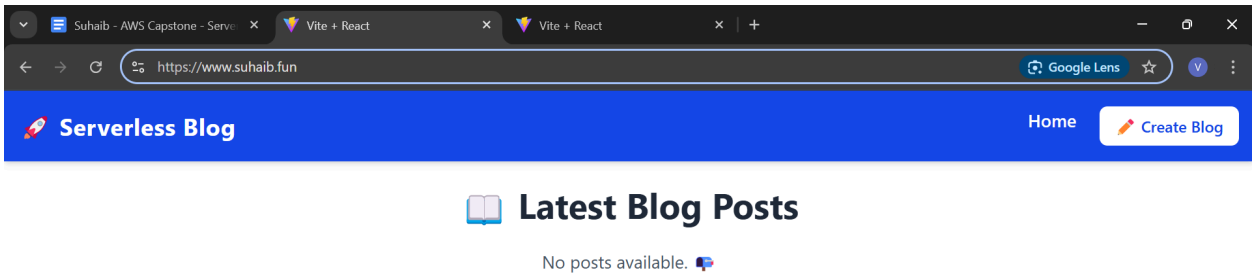


2. Edit the existing **A Record for Non-WWW**:
    - Target: **CloudFront distribution for suhaib.fun**

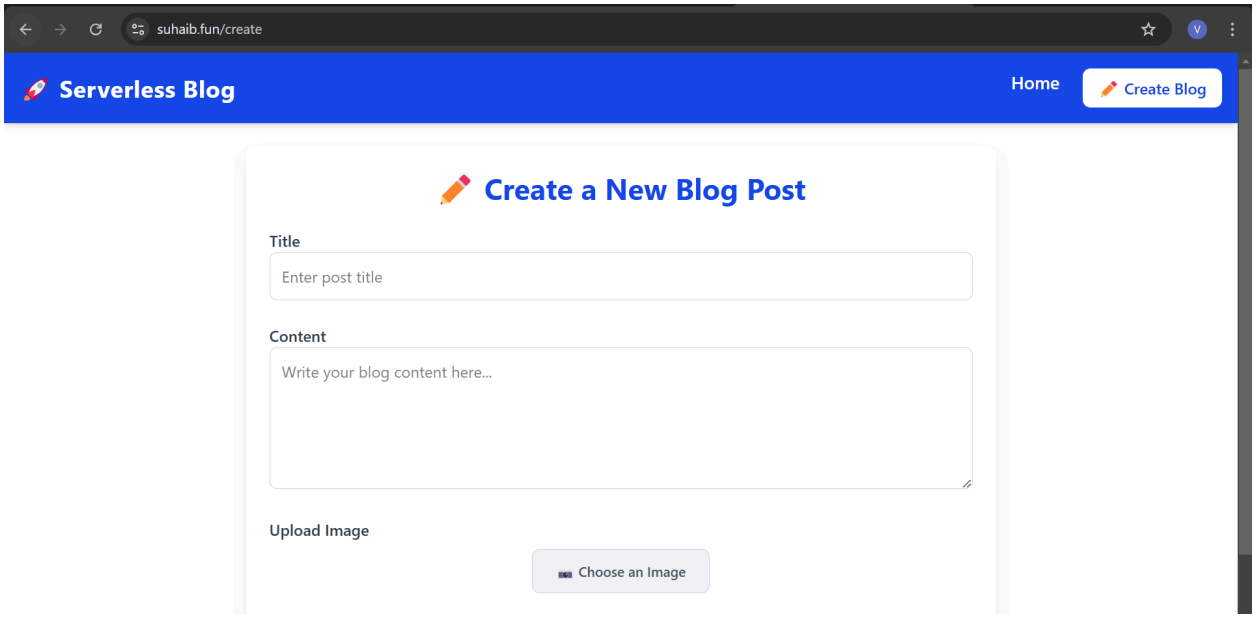


Final Verification Tests

✅ Open https://www.suhaib.fun → The React application should load securely.



✅ Open https://suhaib.fun → It should redirect to www.suhaib.fun with HTTPS.



✅ **Cache Verification**: Even if the S3 bucket is deleted, CloudFront should still serve the site from cache.

Amazon Rekognition: An Overview

1. Introduction

Amazon Rekognition is a fully managed AI service that provides image and video analysis capabilities, including object detection, facial recognition, text detection, and content moderation. It uses deep learning models to analyze and extract meaningful insights from images and videos, enabling developers to build intelligent applications with minimal effort.

2. Key Features

- **Object & Scene Detection** – Identifies objects, people, activities, and scenes in images and videos.
- **Facial Analysis & Recognition** – Detects faces, recognizes individuals, and analyzes facial expressions.
- **Content Moderation** – Scans images and videos for explicit or inappropriate content.
- **Text Detection (OCR)** – Extracts and recognizes text from images.
- **Celebrity & PPE Detection** – Identifies famous individuals and detects protective equipment (e.g., masks, helmets).

3. How Amazon Rekognition Works

Amazon Rekognition processes images and videos by utilizing AWS's deep learning models. The workflow typically involves:

1. **Image/Video Input** – Users upload an image or video to Amazon Rekognition via an API call.
2. **Analysis** – Rekognition processes the media and extracts insights based on selected features (e.g., detecting objects, faces, or text).
3. **Response** – The service returns structured JSON responses containing detected objects, confidence scores, bounding box locations, and other metadata.

4. Integration with AWS Services

Amazon Rekognition seamlessly integrates with:

- **Amazon S3** (for storing images/videos)
- **AWS Lambda** (for automated processing)
- **Amazon DynamoDB** (for metadata storage)
- **Amazon SNS & SQS** (for notifications & message queuing)

5. Use Cases

- **Security & Surveillance** – Identifying unauthorized individuals or suspicious activities.
- **Social Media Moderation** – Filtering inappropriate content automatically.
- **E-commerce** – Product tagging and visual search capabilities.
- **Healthcare** – Detecting protective equipment in medical environments.

6. Pricing

Amazon Rekognition follows a pay-as-you-go model, charging based on the number of images/videos analyzed. Free tier options are available for basic usage.

7. Conclusion

Amazon Rekognition enables businesses to add powerful image and video analysis to their applications without requiring expertise in machine learning. With its seamless AWS integration and broad feature set, it is widely used for security, media moderation, and automation purposes.

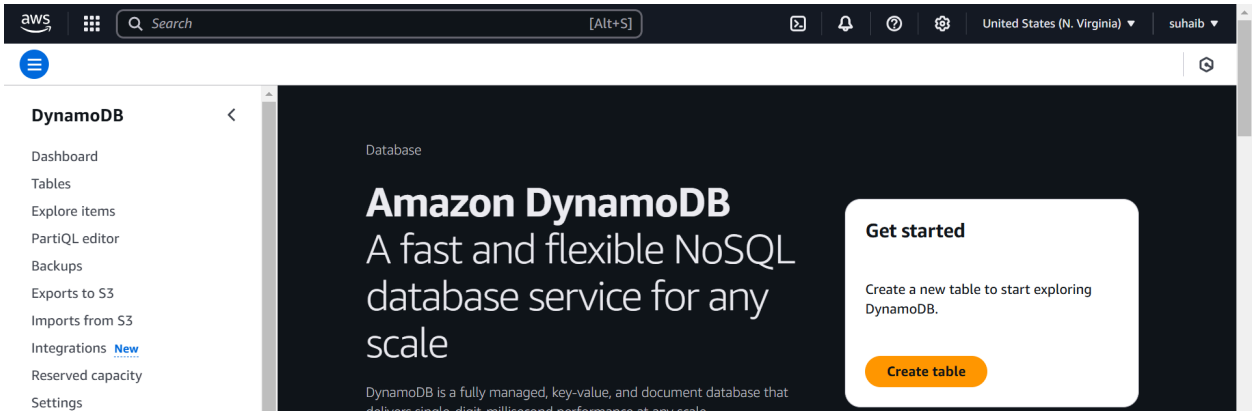
✅ Step 4: Setup Backend for the application using Lambda, API Gateway and DynamoDB

Step 1: Set Up DynamoDB for Storing Blog Posts

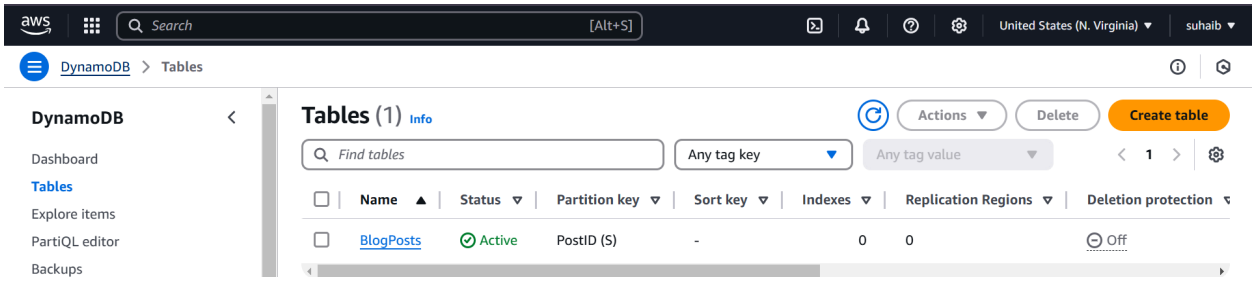
We need a database to store blog posts. You'll create a **DynamoDB table** where each blog post will have a PostID, Title, Content, ImageURL, ModerationStatus, and Timestamp.

1.1 Create the DynamoDB Table

1. Navigate to **DynamoDB > Tables**.
2. Click **Create Table**.



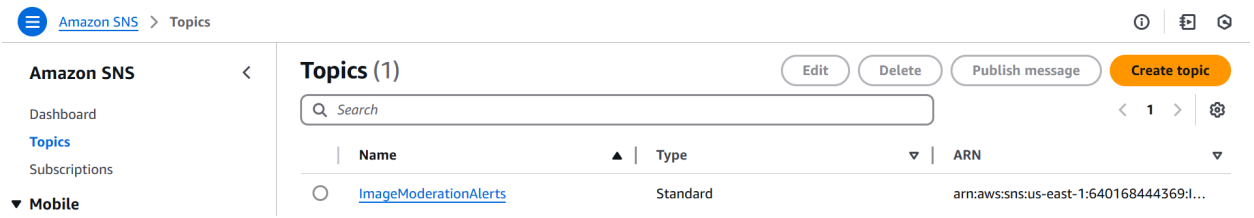
3. Set **Table name**: BlogPosts.
4. Set **Partition key**: PostID (String).
5. Choose **Pay-per-request** billing mode.
6. Click **Create Table**.



Step 2: Create an SNS Topic and Subscribe to receive Email and SQS Notifications

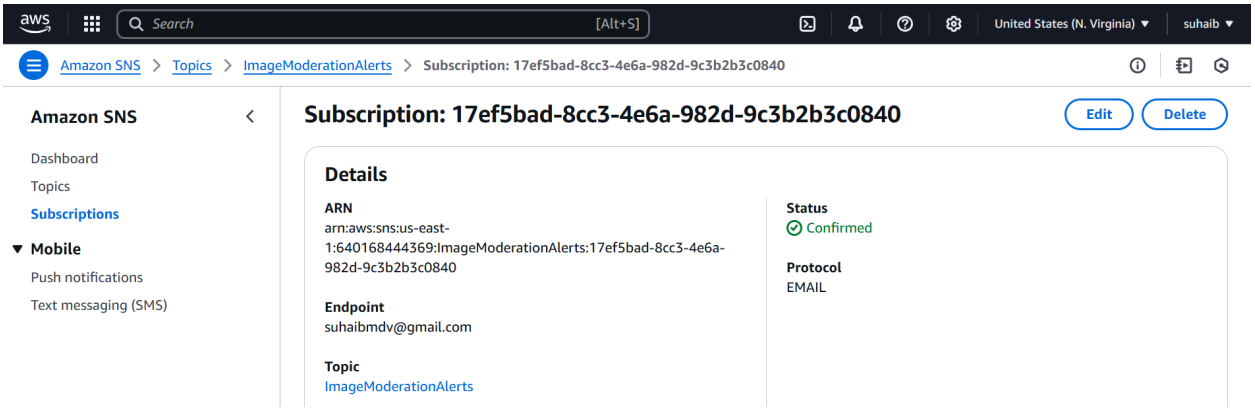
2.1 Create the SNS Topic

1. Sign in to the AWS Management Console.
2. Navigate to **Amazon SNS** by searching for "SNS" in the AWS Services search bar.
3. Click on **Topics** in the left panel.
4. Click **Create topic**.
5. Choose **Standard** as the topic type.
6. Enter a **Topic name** (e.g., `ImageModerationAlerts`).
7. Click **Create topic**.



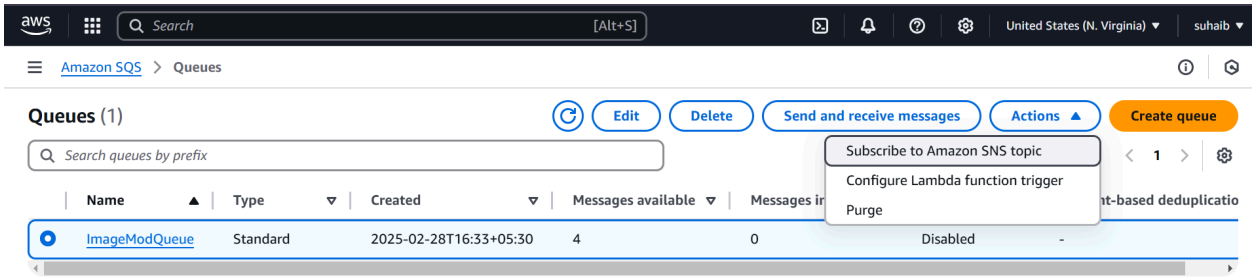
2.2 Subscribe an Email Address to the SNS Topic

1. In the **SNS Topics** section, select the topic you just created.
2. Click **Create subscription**.
3. In the **Protocol** dropdown, select **Email**.
4. In the **Endpoint** field, enter the email address where you want to receive notifications.
5. Click **Create subscription**.
6. Go to your email inbox and look for a confirmation email from AWS SNS.
7. Click the **Confirm subscription** link in the email.

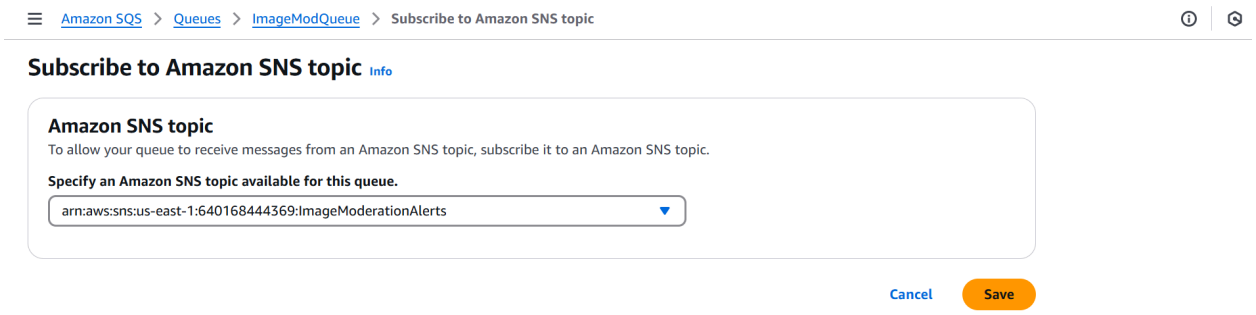


2.3 Subscribe an SQS Queue to the SNS Topic

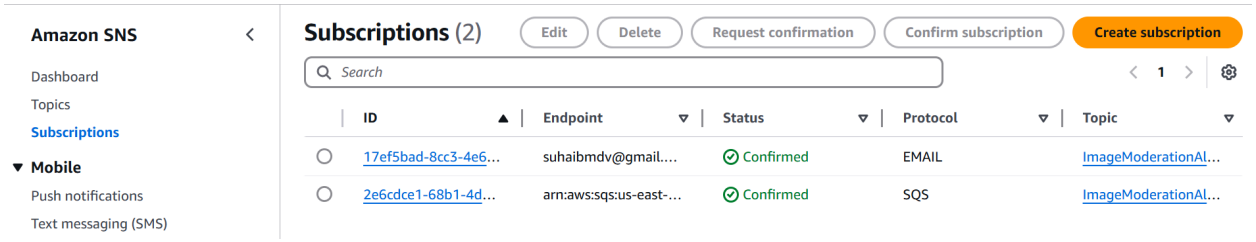
1. Navigate to **Amazon SQS** in the AWS Console.
2. Click **Create queue** and choose either **Standard** or **FIFO**.
3. Enter a **Queue name** (e.g., `BlogModerationQueue`).
4. Configure additional settings if needed, then click **Create queue**.
5. Once created, click on Actions then subscribe to SNS topic.



6. Choose the SNS topic and click save.



7. Verify SQS subscription.



Step 3: Create an S3 Bucket for Image Uploads

Since images will be uploaded to S3 before being moderated, we need a dedicated bucket.

3.1 Create the S3 Bucket

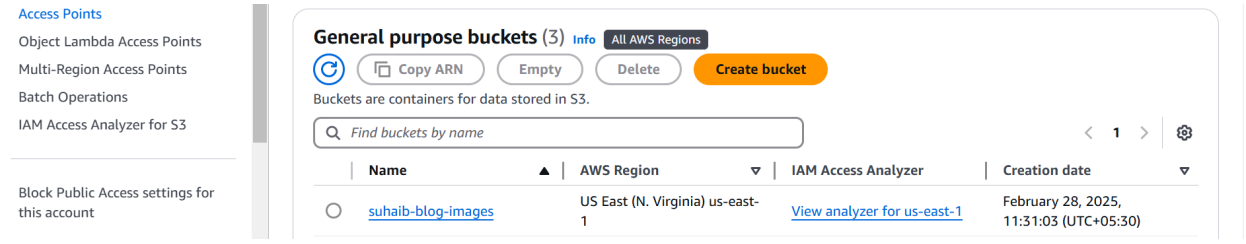
Run the following CLI command:

```
aws s3api create-bucket --bucket suhaib-blog-images --region us-east-1
```

```
C:\Users\Vadakathi Suhaib>aws s3api create-bucket --bucket suhaib-blog-images --region us-east-1
{
    "Location": "/suhaib-blog-images"
}
```

Or in the AWS console:

1. Go to **S3**.
2. Click **Create bucket**.
3. Set **Bucket name**: suhaib-blog-images.
4. Choose **Region**: us-east-1.
5. **Block Public Access**: Keep it **enabled**.
6. Click **Create Bucket**.



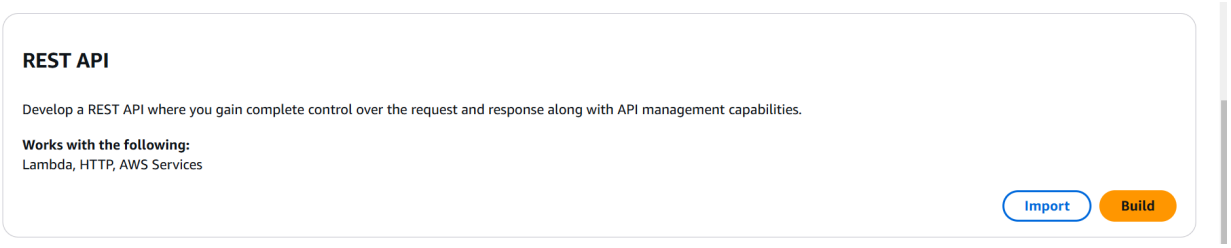
Step 4: Create the API Gateway

Now, we set up **API Gateway** to expose endpoints for:

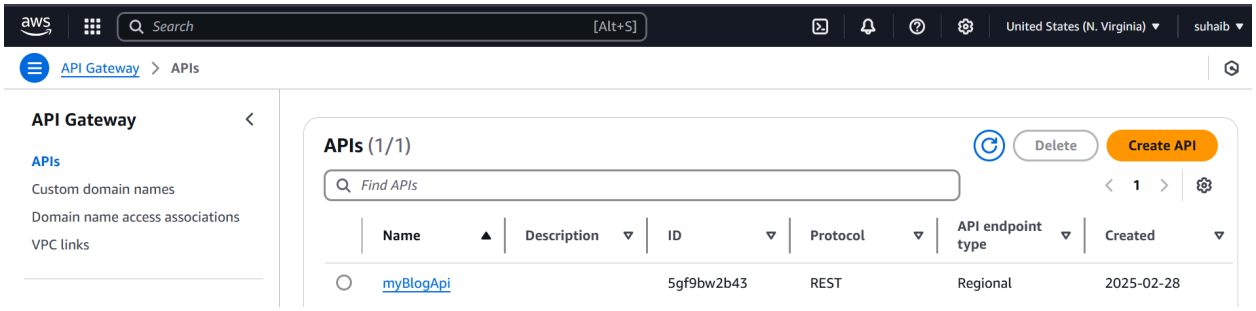
1. Creating a blog post (POST /create-post).
2. Fetching blog posts (GET /get-posts).
3. Getting a presigned URL for image uploads (GET /get-upload-url).

4.1 Create a REST API

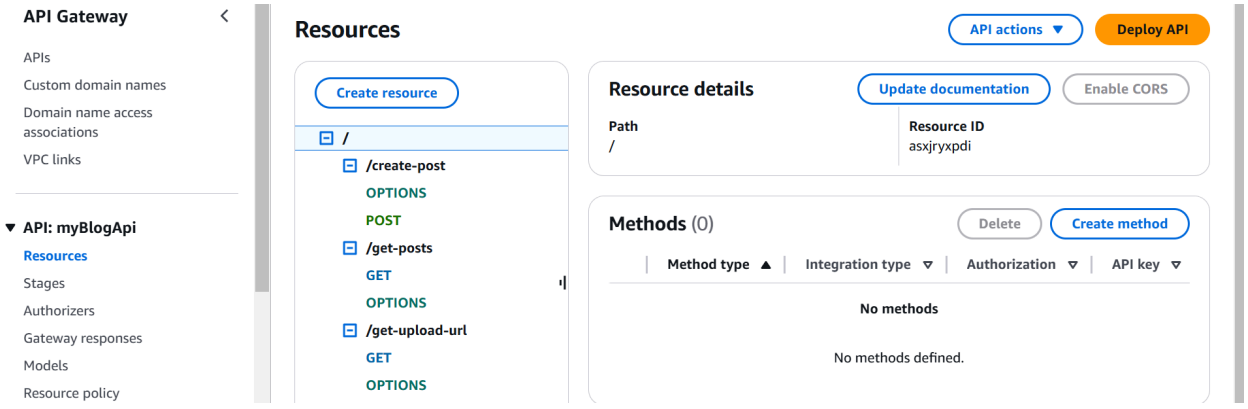
1. Go to **API Gateway** in the AWS Console.
2. Click **Create API > REST API**.



3. Choose **Regional** and click **Create API**.



4. Click **Actions > Create Resource** and create resources:
   - /create-post
   - /get-posts
   - /get-upload-url
5. Click **Actions > Create Method** under each resource:
   - POST for /create-post.
   - GET for /get-posts.
   - GET for /get-upload-url.



We'll integrate these methods with Lambda functions.

4.2 Enable CORS for API Gateway

If your API doesn't have **CORS (Cross-Origin Resource Sharing)** enabled, browsers **block** requests from your React app.

Fix: Enable CORS in API Gateway

1. Go to **API Gateway** in AWS Console.
2. Select your API and **go to "Gateway Responses"**.
3. Enable **CORS** by adding this in the response headers:

```
{
  "Access-Control-Allow-Origin": "*",
  "Access-Control-Allow-Methods": "GET, POST, PUT, DELETE, OPTIONS",
  "Access-Control-Allow-Headers": "Content-Type"
}
```

Step 5: Create Lambda Functions

Create an IAM Role for Lambda

Your Lambda function needs permissions to:

- Read images from your S3 bucket.
- Call Rekognition's moderation APIs.
- Write records to DynamoDB.
- Publish notifications via SNS.

Steps:

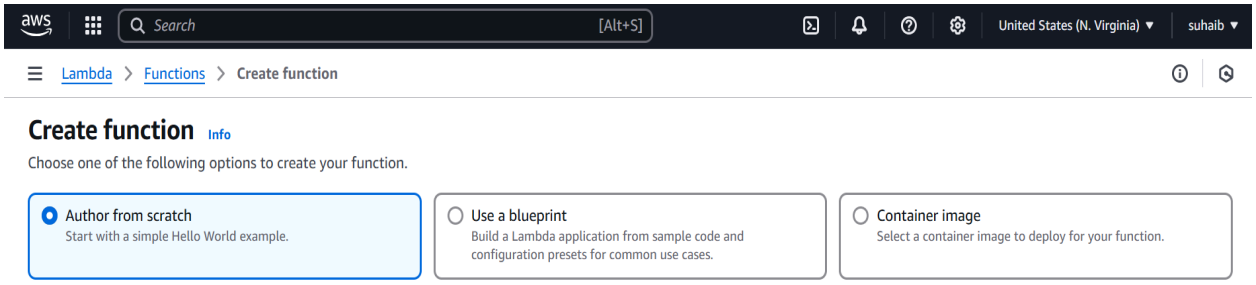
- **Navigate to IAM in the AWS Console.**
- **Create a new role** with "AWS service" as the trusted entity and select **Lambda**.
- **Attach policies** (or create a custom policy) with permissions for:
  - **S3:** Allow s3:GetObject on your image bucket.
  - **Rekognition:** Allow rekognition:DetectModerationLabels.
  - **DynamoDB:** Allow dynamodb:PutItem (and any other needed actions) on your table.
  - **SNS:** Allow sns:Publish on your SNS topic.
- **Review and create the role.**
  Tip: For tighter security, restrict resource ARNs to your specific bucket, table, and topic.

5.1 Create the Blog Post Lambda (CreatePost)

This Lambda function handles post creation by storing the post in DynamoDB.

Create a new Lambda function:

1. Go to **AWS Lambda**.
2. Click **Create function**.
3. Choose **Author from scratch**.



4. Set:
   - **Function name**: CreatePost
   - **Runtime**: Python 3.9
   - **Execution Role**: Create a new role with **DynamoDB full access**.
5. Click **Create function**.

5.1.1 Add the following Python code:

```
import json
import boto3
import uuid
import time
import os
from decimal import Decimal
from urllib.parse import urlparse
from botocore.exceptions import ClientError

# Initialize AWS services
dynamodb = boto3.resource("dynamodb")
rekognition = boto3.client("rekognition")
s3 = boto3.client("s3")
sns = boto3.client("sns")
codepipeline = boto3.client("codepipeline")

# Environment variables
```

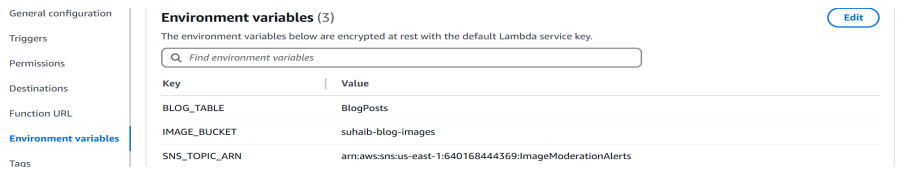
```
table_name = os.environ['BLOG_TABLE']
table = dynamodb.Table(table_name)
sns_topic_arn = os.environ["SNS_TOPIC_ARN"]  # Ensure this is set in Lambda
environment variables

def report_pipeline_success(event):
    """ Reports success to CodePipeline """
    if "CodePipeline.job" in event:
        job_id = event["CodePipeline.job"]["id"]
        codepipeline.put_job_success_result(jobId=job_id)

# Custom JSON Encoder for handling Decimal values
class DecimalEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Decimal):
            return float(obj)
        return super(DecimalEncoder, self).default(obj)

def lambda_handler(event, context):
    print("Received event:", json.dumps(event, indent=2))

    try:
        # Parse the request body
        if "body" in event:
            body = json.loads(event["body"])
        else:
            body = event

        title = body.get("title")
        content = body.get("content")
        image_url = body.get("imageUrl")

        if not title or not content or not image_url:
            return {
                "statusCode": 400,
                "body": json.dumps({"error": "Missing required fields"},
cls=DecimalEncoder)
            }

        # 🔹 Extract bucket name and object key from image_url (More Reliable)
        parsed_url = urlparse(image_url)
        bucket_name = parsed_url.netloc.split('.')[0]  # Extract bucket from
domain
        object_key = parsed_url.path.lstrip("/")  # Remove leading slash
```

```
        # 🔹 Moderate the image using Rekognition
        try:
            response = rekognition.detect_moderation_labels(
                Image={"S3Object": {"Bucket": bucket_name, "Name":
object_key}},
                MinConfidence=75  # Confidence threshold for moderation
            )
        except ClientError as e:
            print(f"Rekognition error: {e}")
            return {
                "statusCode": 500,
                "body": json.dumps({"error": "Image moderation failed"},
cls=DecimalEncoder)
            }

        if response["ModerationLabels"]:  # If image is flagged
            labels = [label["Name"] for label in response["ModerationLabels"]]
            print("Image flagged for:", labels)

            # Send an SNS warning
            sns.publish(
                TopicArn=sns_topic_arn,
                Message=f"⚠️ Warning: Image flagged for {',
'.join(labels)}.\nImage URL: {image_url}",
                Subject="⚠️ Image Moderation Alert"
            )

            # 🔹 Delete the flagged image from S3
            s3.delete_object(Bucket=bucket_name, Key=object_key)
            print(f"Flagged image {object_key} deleted from S3")

            return {
                "statusCode": 400,
                "body": json.dumps({
                    "error": f"Your image was flagged for {', '.join(labels)}.
The post was rejected.",
                    "warning": "Do not upload inappropriate content!"
                })
            }

        # ✅ Image is safe → Save post in DynamoDB
        post_id = str(uuid.uuid4())
        table.put_item(
            Item={
                "PostID": post_id,
```

```
                "title": title,
                "content": content,
                "imageUrl": image_url,
                "createdAt": int(time.time())
            }
        )

        return {
            "statusCode": 200,
            "body": json.dumps({"message": "Post created successfully!",
"postId": post_id}, cls=DecimalEncoder)
        }

    except ClientError as e:
        print("AWS Error:", str(e))
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)}, cls=DecimalEncoder)
        }

    except Exception as e:
        print("Error:", str(e))
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)}, cls=DecimalEncoder)
        }

    finally:
        # ✅ Report success to CodePipeline (even if an error occurs)
        report_pipeline_success(event)
```

5.1.2 Set Environment Variables

- Click **Configuration > Environment Variables**.
- Add:
    - BLOG_TABLE = BlogPosts
    - SNS_TOPIC_ARN = arn:aws:sns:us-east-1:640168444369:ImageModerationAlerts
    - IMAGE_BUCKET = suhaib-blog-images



5.1.3 Deploy and Connect to API Gateway

1. Go to **API Gateway**.
2. Select /create-post > Integration type: **Lambda Function**.
3. Select CreatePost.
4. Deploy API: **Actions > Deploy API** (Create a new stage like "prod").
5. Copy the API URL and replace "https://your-api-url.com/create-post" in your React app.

5.2 Create the Get Posts Lambda (GetPosts)

5.2.1 Create a new Lambda function in AWS Lambda:

- **Function name**: GetPosts
- **Runtime**: Python 3.9
- **Execution Role**: Use the same role as CreatePost.

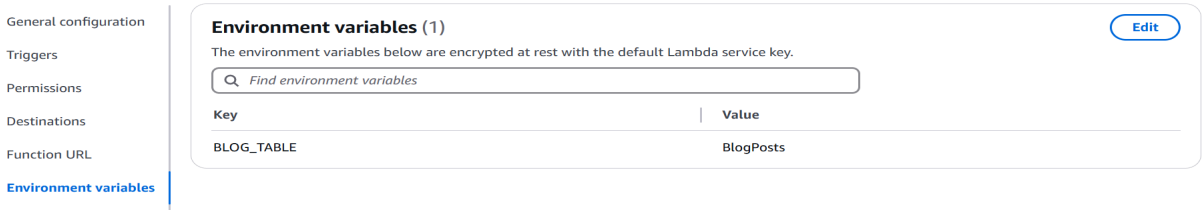
5.2.2 Add the following Python code:

```
import json
import boto3
import os
from decimal import Decimal

# Initialize DynamoDB
dynamodb = boto3.resource("dynamodb")
table_name = os.environ["BLOG_TABLE"]
table = dynamodb.Table(table_name)
codepipeline = boto3.client("codepipeline")

def report_pipeline_success(event):
    """ Reports success to CodePipeline """
    if "CodePipeline.job" in event:
        job_id = event["CodePipeline.job"]["id"]
        codepipeline.put_job_success_result(jobId=job_id)

#  🔹 Custom JSON Encoder for Decimal values
class DecimalEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Decimal):
            return float(obj)  # ✅ Convert Decimal to float
        return super(DecimalEncoder, self).default(obj)
```

```
def lambda_handler(event, context):
    try:
        posts = []
        response = table.scan()

        # Handle pagination in case there are more items
        while "LastEvaluatedKey" in response:
            posts.extend(response.get("Items", []))  # Add current items to
list
            response =
table.scan(ExclusiveStartKey=response["LastEvaluatedKey"])

        posts.extend(response.get("Items", []))  # Add the last batch of items

        report_pipeline_success(event)

        return {
            "statusCode": 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",  # Allow CORS for all
domains
                "Access-Control-Allow-Methods": "GET",
                "Access-Control-Allow-Headers": "Content-Type",
            },
            "body": json.dumps(posts, cls=DecimalEncoder),  # ✅ Use custom
encoder
        }

    except Exception as e:
        return {
            "statusCode": 500,
            "headers": {
                "Access-Control-Allow-Origin": "*"
            },
            "body": json.dumps({"error": str(e)}, cls=DecimalEncoder),  # ✅
Use encoder for error response
        }
```

5.2.3 Set Environment Variables

- BLOG_TABLE = BlogPosts



5.2.4 Connect to API Gateway

- Add **GET /get-posts** in API Gateway.
- Integrate it with the GetPosts Lambda.
- Deploy API.
- Update "https://your-api-url.com/get-posts" in your React app.

5.3 Create the Presigned URL Lambda (GetUploadUrl)

This Lambda generates a presigned URL so users can upload images directly to S3.

5.3.1 Create a new Lambda function:

- **Function name**: GetUploadUrl
- **Runtime**: Python 3.9
- **Execution Role**: Give it **S3 write access**.

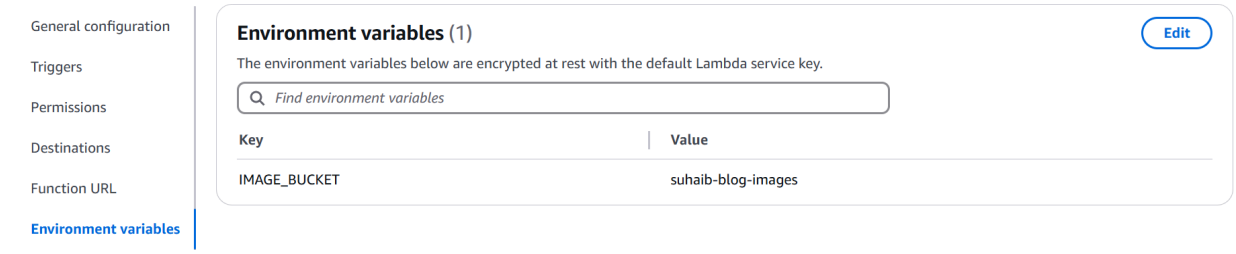
5.3.2 Add the following Python code:

```
import json
import boto3
import os
import time

s3_client = boto3.client('s3')
bucket_name = os.environ['IMAGE_BUCKET']
codepipeline = boto3.client("codepipeline")

def report_pipeline_success(event):
    """ Reports success to CodePipeline """
    if "CodePipeline.job" in event:
        job_id = event["CodePipeline.job"]["id"]
        codepipeline.put_job_success_result(jobId=job_id)
```

```
def lambda_handler(event, context):
    try:
        query_params = event.get('queryStringParameters', {})
        file_name = query_params.get('fileName',
f"image_{int(time.time())}.jpg")

        presigned_url = s3_client.generate_presigned_url(
            'put_object',
            Params={'Bucket': bucket_name, 'Key': f"uploads/{file_name}",
'ContentType': 'image/jpeg'},
            ExpiresIn=3600
        )

        report_pipeline_success(event)

        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Methods': 'GET, OPTIONS',
                'Access-Control-Allow-Headers': 'Content-Type'
            },
            'body': json.dumps({'uploadUrl': presigned_url})
        }

    except Exception as e:
        return {
            'statusCode': 500,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Methods': 'GET, OPTIONS',
                'Access-Control-Allow-Headers': 'Content-Type'
            },
            'body': json.dumps({'error': str(e)})
        }
```

5.3.3 Set Environment Variables

- IMAGE_BUCKET = suhaib-blog-images



5.3.4 Connect to API Gateway

- Add **GET /get-upload-url**.
- Integrate it with the GetUploadUrl Lambda.
- Deploy API.
- Update "https://your-api-url.com/get-upload-url" in your React app.

5.4 Create the Moderate Image Lambda (moderate_image)

This Lambda ensures that manually uploaded images are also moderated using rekognition.

5.4.1 Create a new Lambda function:

- **Function name**: moderate_image
- **Runtime**: Python 3.9
- **Execution Role**: Give it **Rekognition, SNS, S3 access.**

5.4.2 Add the following Python code:

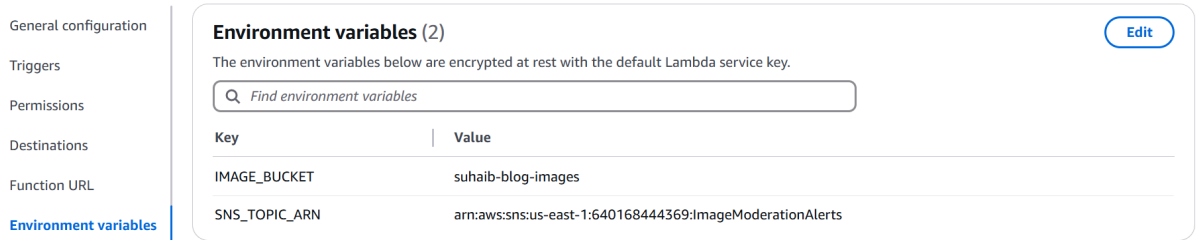
```
import json
import boto3
import os

# Initialize AWS clients
rekognition = boto3.client("rekognition")
s3 = boto3.client("s3")
sns = boto3.client("sns")
codepipeline = boto3.client("codepipeline")

# Environment variables
sns_topic_arn = os.environ.get("SNS_TOPIC_ARN")  # Use `.get()` to prevent
```

```
KeyError

def report_pipeline_success(event):
    """ Reports success to CodePipeline """
    if "CodePipeline.job" in event:
        job_id = event["CodePipeline.job"]["id"]
        codepipeline.put_job_success_result(jobId=job_id)

def lambda_handler(event, context):
    print("Received event:", json.dumps(event, indent=2))

    # ✅ Validate event structure
    if "Records" not in event:
        print("⚠️ Warning: No 'Records' key found in event. Possibly
triggered manually.")
        return {
            "statusCode": 400,
            "body": json.dumps({"error": "Invalid event format. No
'Records' found."})
        }

    for record in event["Records"]:
        try:
            bucket_name = record["s3"]["bucket"]["name"]
            object_key = record["s3"]["object"]["key"]

            print(f"Processing image: s3://{bucket_name}/{object_key}")

            #  🔹 Run Rekognition moderation
            response = rekognition.detect_moderation_labels(
                Image={"S3Object": {"Bucket": bucket_name, "Name":
object_key}},
                MinConfidence=75
            )

            if response.get("ModerationLabels"):  # If flagged
                labels = [label["Name"] for label in
response["ModerationLabels"]]
                print("Image flagged for:", labels)

                # Send SNS Alert
                if sns_topic_arn:
                    sns.publish(
```

```
                        TopicArn=sns_topic_arn,
                        Message=f"⚠️ Manually uploaded image flagged for
{', '.join(labels)}.\nImage: s3://{bucket_name}/{object_key}",
                        Subject="⚠️ Manual Upload Image Moderation Alert"
                    )

                # 🔹 Delete the flagged image from S3
                s3.delete_object(Bucket=bucket_name, Key=object_key)
                print(f"Deleted flagged image:
s3://{bucket_name}/{object_key}")

        except KeyError as ke:
            print(f"❌ KeyError: Missing expected key in event record:
{ke}")
        except Exception as e:
            print(f"❌ Error processing image: {e}")

    # ✅ Move outside the loop (to be executed once per event)
    report_pipeline_success(event)

    return {"statusCode": 200, "body": json.dumps({"message": "Moderation
process completed"})}
```

5.4.3 Set Environment Variables

- IMAGE_BUCKET = suhaib-blog-images
- SNS_TOPIC_ARN = arn:aws:sns:us-east-1:640168444369:ImageModerationAlerts



5.4.4 Set up trigger in S3 bucket

- Open your **S3 bucket** (`suhaib-blog-images`).
- Go to **Properties → Event notifications → Create event notification**.
- Name it **ImageUploadTrigger**.
- **Event type:** PUT (object created).

- **Destination:** Choose `Lambda Function` and select `moderate_image`.

Now, whenever an image is uploaded, this Lambda function will be triggered.

ISSUES:

CORS ISSUES:

Add CORS to API Gateway for GetPosts and others

📌 In API Gateway Console:

1. **Go to API Gateway** (`5gf9bw2b43`).
2. Select your API, **go to `GET /getposts/get-posts`**.
3. Click **"Method Response"** (right panel).
4. **Add a response header:**
   - `Access-Control-Allow-Origin` → `*`
5. Click **"Integration Response"** (below Method Response).
6. Expand **"200" response**, scroll to "Header Mapping".
7. **Add a response header mapping:**
   - `Access-Control-Allow-Origin` → `' * '`
8. **Deploy API changes** (`Actions` → `Deploy API`).

Enable CORS for OPTIONS Requests

1. In **API Gateway**, go to `getposts/get-posts`.
2. Click **"Actions"** → **"Create Method"**.
3. Select **`OPTIONS`**, then click the checkmark.
4. **Set "Mock" as Integration Type** (No backend needed for CORS).
5. Click **"Method Response"**:
   - Add a `200` response.
   - Under "Response Headers," add:
     - `Access-Control-Allow-Origin`
     - `Access-Control-Allow-Methods`
     - `Access-Control-Allow-Headers`
6. Click **"Integration Response"**:
   - Expand `200` response, go to "Header Mapping".
   - Set:
     - `Access-Control-Allow-Origin` → `' * '`
     - `Access-Control-Allow-Methods` → `'GET, OPTIONS'`

        - `Access-Control-Allow-Headers` → `'Content-Type'`
7. Click **"Deploy API"**.

Clear CloudFront Cache (If Using)

If your frontend is behind **CloudFront**, the old CORS settings might be cached.

- **Go to AWS CloudFront → Your Distribution → Invalidation**
- Enter: `/*` → Click "Invalidate."

CORS Configuration Missing in S3

Your S3 bucket needs a proper CORS policy to allow browser-based uploads.
Go to **AWS S3 Console → Your Bucket → Permissions → CORS policy**
Make sure you have this **CORS configuration**:

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["GET", "PUT", "POST", "DELETE"],
    "AllowedOrigins": ["https://www.suhaib.fun"],
    "ExposeHeaders": []
  }
]
```

- Ensure `"AllowedOrigins"` matches your website's domain (`https://www.suhaib.fun`).
- Include `"PUT"` in `"AllowedMethods"` to allow image uploads.
- Use `"*"` in `"AllowedHeaders"` to avoid restrictions.

CloudFront Configuration

If you are serving the S3 bucket via **CloudFront**, you need to allow CORS in its behavior settings.

- In **CloudFront → Behaviors → Edit the Behavior** for your S3 origin.
- Under **Cache & origin request settings**, set:
    - **Allowed HTTP Methods:** `GET, HEAD, OPTIONS, PUT, POST`
    - **Forward Headers:** `All`

✅ **Save, invalidate cache, and test again.**

Recommended Settings for suhaib-blog-images (image S3 bucket):

Setting	Enable or Disable?	Reason
Block all public access	❌ (Disable)	You need public access for images (via s3:GetObject).
Block public access granted through new ACLs	✅ (Enable)	Prevents accidental public ACL settings.
Block public access granted through any ACLs	✅ (Enable)	Uses bucket policies instead of ACLs for security.
Block public access granted through new public bucket policies	❌ (Disable)	Your bucket policy allows s3:GetObject, so you must disable this.
Block public and cross-account access through public bucket policies	❌ (Disable)	You need a public policy for images, so disable this.

Best Practice for API Gateway Stages

You should **deploy all resources (all endpoints: `/getuploadurl`, `/getposts`, etc.) in the same stage**.

- **Single Stage (e.g., `prod`)**
  - `/prod/get-upload-url`
  - `/prod/get-posts`
  - `/prod/create-post`

This way, your frontend **always calls the same base URL**, like:

```
https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-posts
https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-upload-url
```

The image is uploading successfully but the post isn't appearing in DynamoDB or on the home page, let's debug systematically:

Possible Issues & Fixes:

✅ 1. Check if the Lambda function is triggered

- Go to **AWS Lambda** → Find the function handling the post request.
- Open **"Monitor" → "Logs" (CloudWatch)**
- See if there are any errors.

✅ 2. Check API Gateway Execution Logs

- Go to **API Gateway** → Select your API.
- Click **"Stages"** → Select **the deployed stage**.
- Under **"Logs/Tracing"**, check if logging is enabled.
- Look for any error responses in **CloudWatch**.

If API Gateway isn't forwarding the request to Lambda, it could be:

1. **Wrong integration settings** (Check if Lambda is correctly linked).
2. **Missing permissions** (Lambda needs `dynamodb:PutItem` access).

✅ 3. Check if DynamoDB has the data

- Go to **DynamoDB** → Open the **Table**.
- Click **"Explore Table Items"**.
- Search for recent records.

✅ 4. Check the Frontend API Call

- Open the browser console (**F12 → Console/Network Tab**).
- See if the frontend **actually sends** the request to the API.
- If it fails:
    - Look for **CORS errors**.
    - Ensure the API URL is correct.

Other issues:

- **Error: `Object of type Decimal is not JSON serializable` - happens because DynamoDB stores numbers as Decimal type, but json.dumps() in Python does not support Decimal by default**

Step 6: Testing

6.1 Test the API

Run the following to test the API manually:

Get a Presigned URL

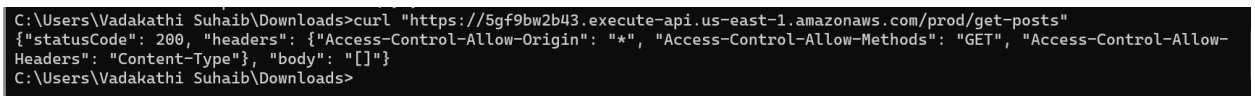
```
curl -X GET
"https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-upload-url
?fileName=test.jpg"
```

Create a Post

```
curl -X POST
"https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/create-post"
-H "Content-Type: application/json" --data-raw "{\"title\":\"Test Post\",
\"content\":\"This is a test.\",
\"imageUrl\":\"https://suhaib-blog-images.s3.amazonaws.com/uploads/test.jpg
\"}"
```

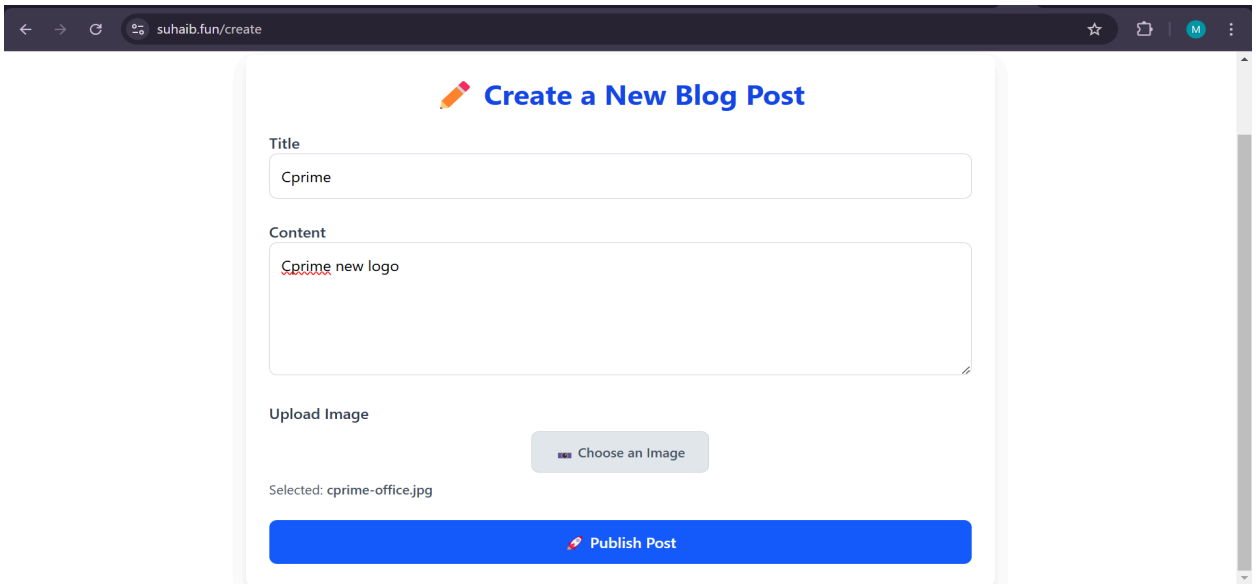
Fetch Posts

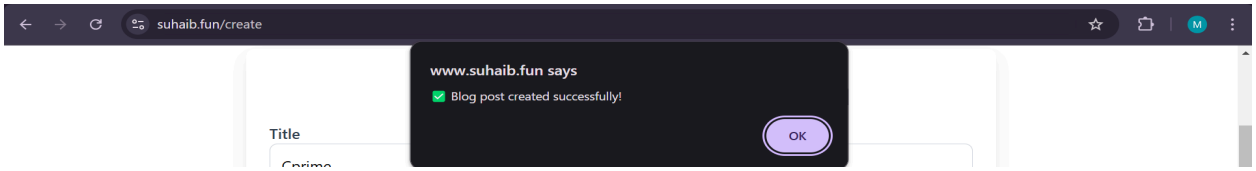
```
curl
"https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-posts"
```



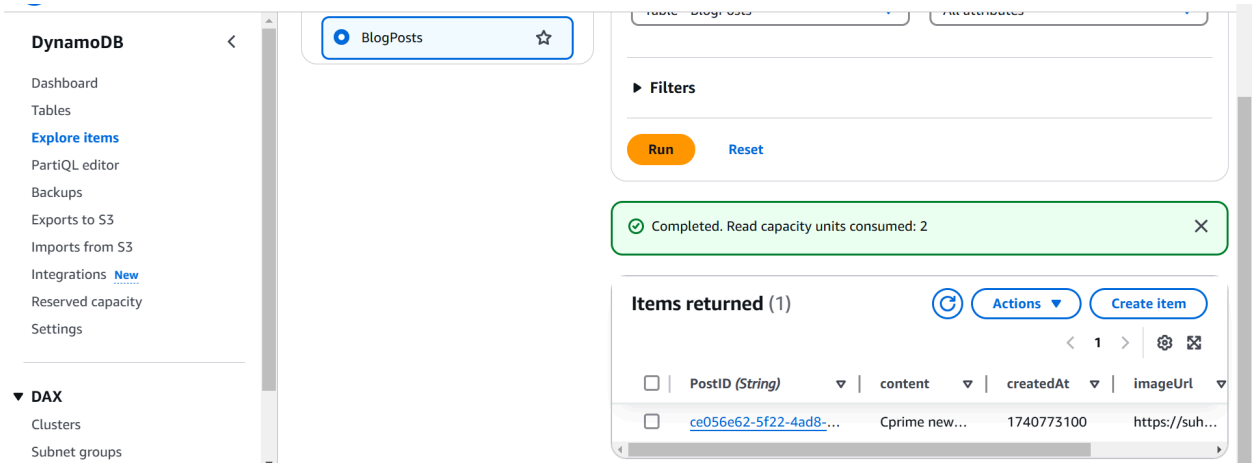
6.2 Upload a Normal Image

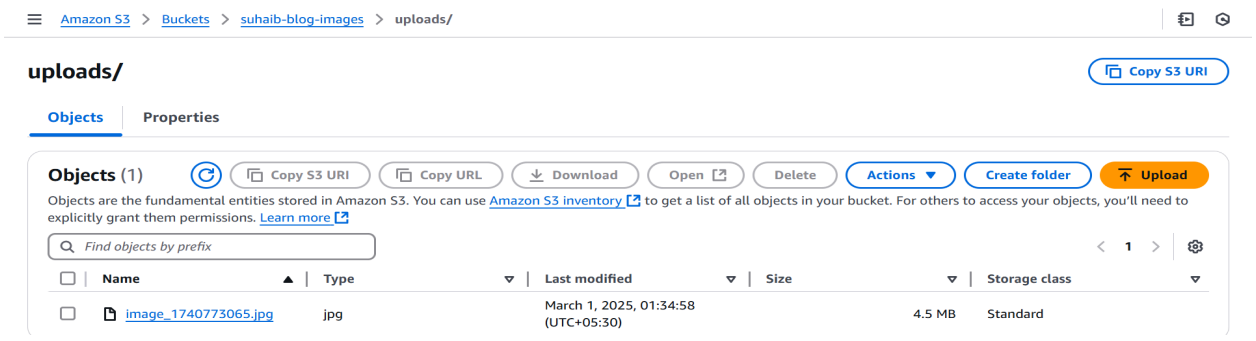
- Try uploading a **safe** image.





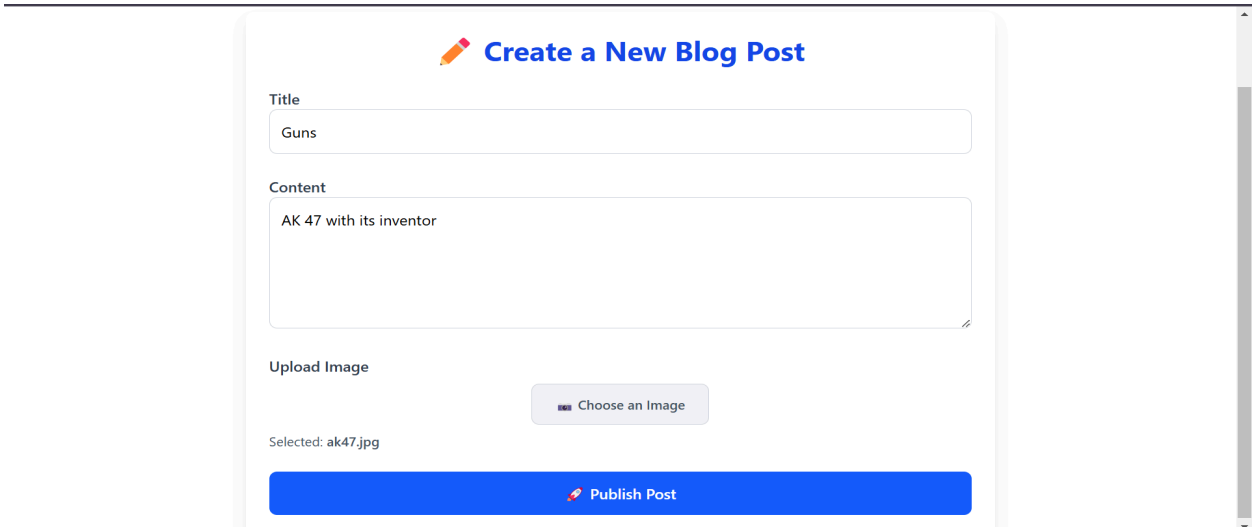
- It should pass moderation and be stored in **DynamoDB**.



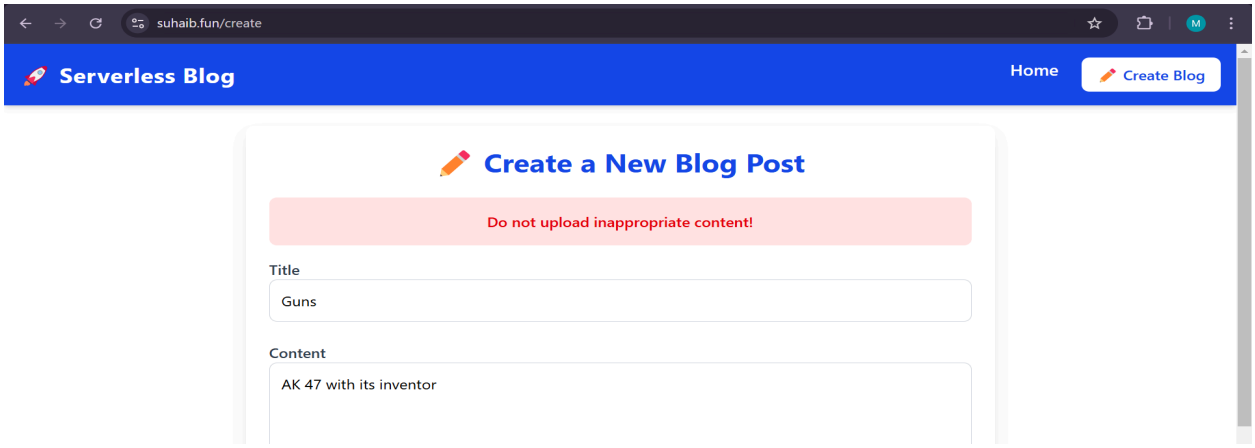


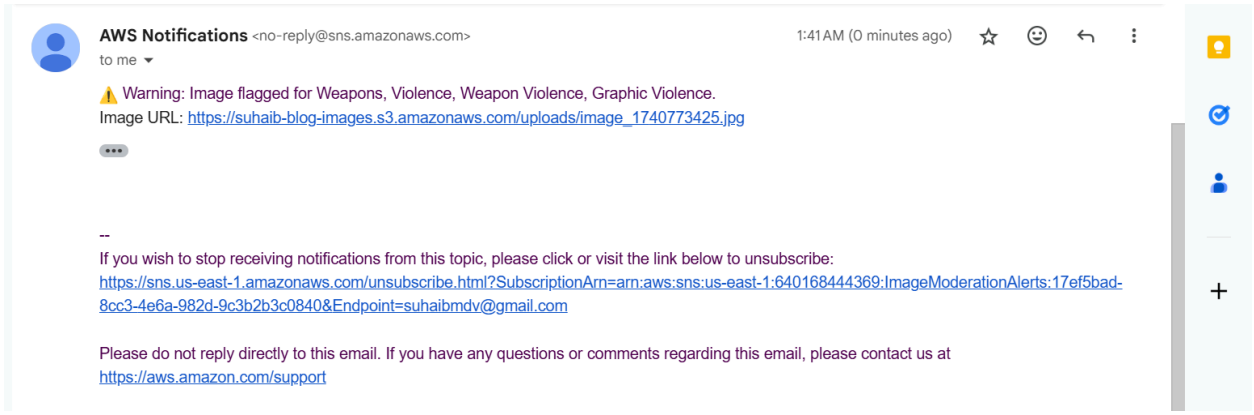
6.3 Upload an Inappropriate Image

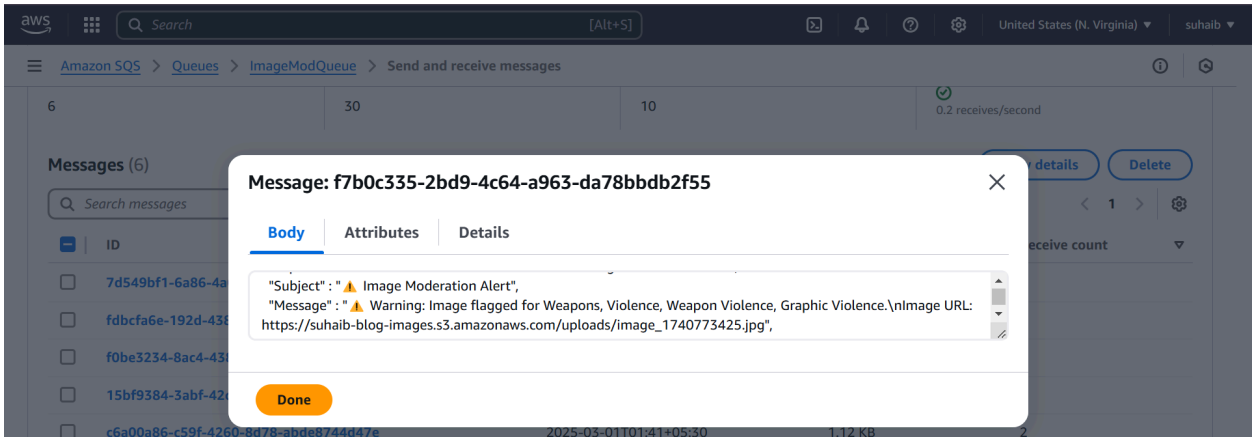
- Try uploading an image that Rekognition might flag (e.g., nudity, violence).



- The image should be **flagged**, and an **SNS notification** should be sent.


suhaib.fun/create
Serverless Blog
Home
Create Blog
Create a New Blog Post
Do not upload inappropriate content!
Title
Guns
Content
AK 47 with its inventor

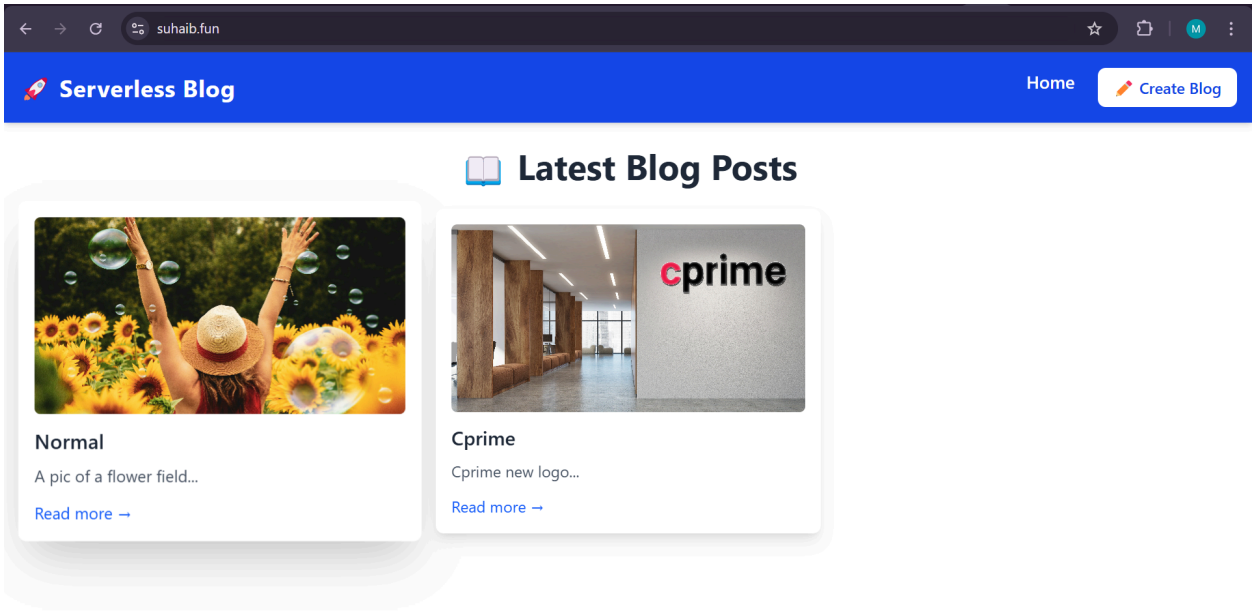

AWS Notifications <no-reply@sns.amazonaws.com>
1:41 AM (0 minutes ago)
to me
⚠️ Warning: Image flagged for Weapons, Violence, Weapon Violence, Graphic Violence.
Image URL: https://suhaib-blog-images.s3.amazonaws.com/uploads/image_1740773425.jpg
--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:640168444369:ImageModerationAlerts:17ef5bad-8cc3-4e6a-982d-9c3b2b3c0840&Endpoint=suhaibmdv@gmail.com
Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support


Amazon SQS
Queues
ImageModQueue
Send and receive messages
Messages (6)
Message: f7b0c335-2bd9-4c64-a963-da78bbdb2f55
Body
Attributes
Details
"Subject" : "⚠️ Image Moderation Alert",
"Message" : "⚠️ Warning: Image flagged for Weapons, Violence, Weapon Violence, Graphic Violence.\nImage URL: https://suhaib-blog-images.s3.amazonaws.com/uploads/image_1740773425.jpg",
Done

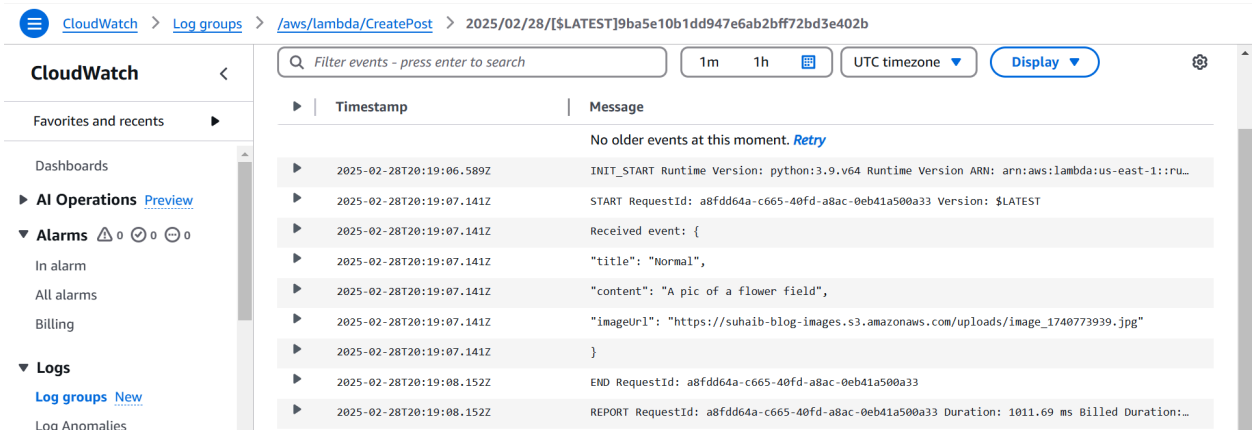
Step 7. Validate the System and Check Logs

7.1 Testing the Lambda Function

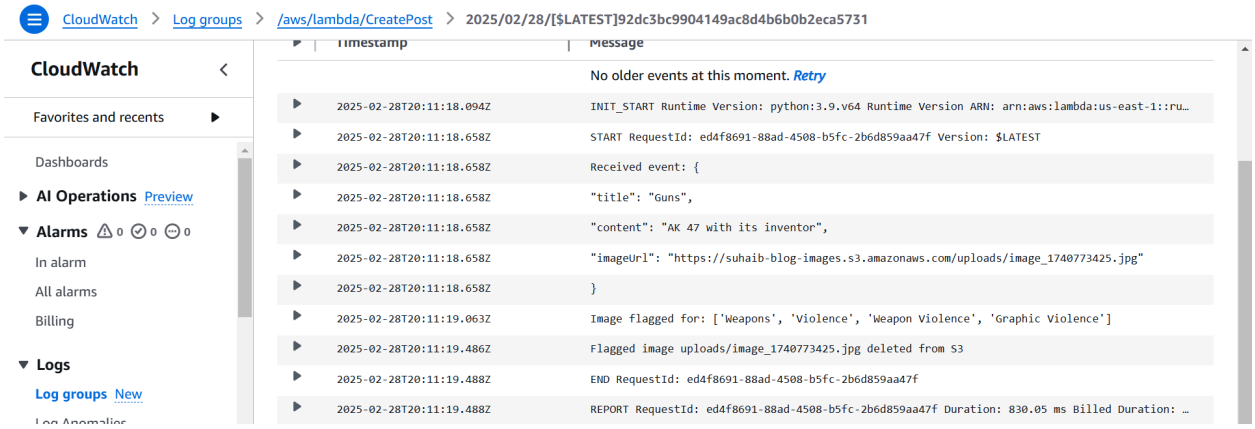
- **Upload test images** to your S3 bucket.



- Monitor the **CloudWatch Logs** for your Lambda function to see the processing details and any errors.

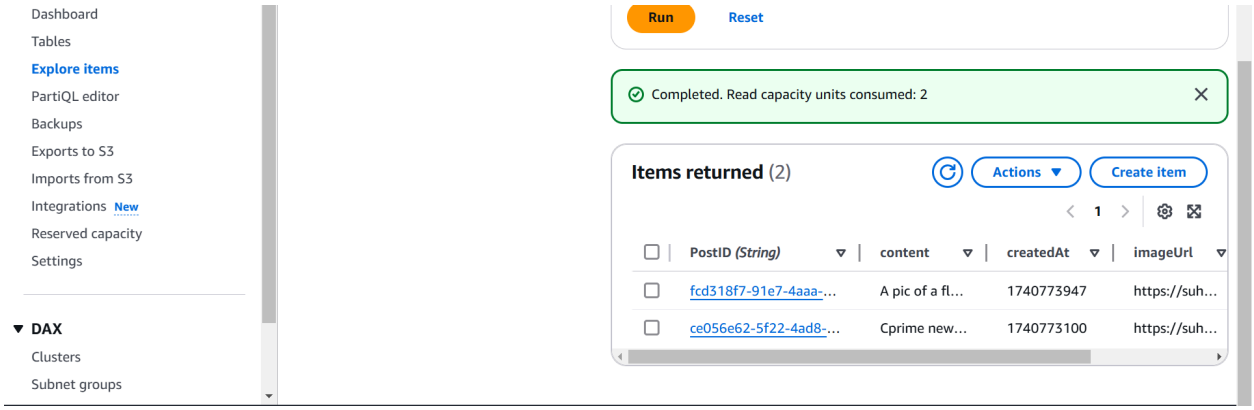


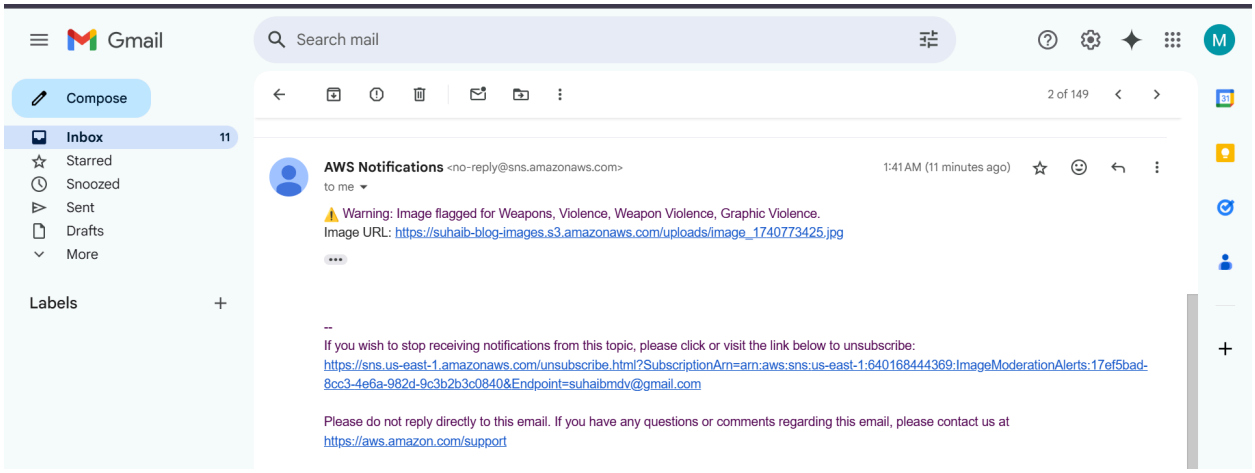
- Verify that the function calls Rekognition, writes to DynamoDB, and (if applicable) sends SNS notifications.

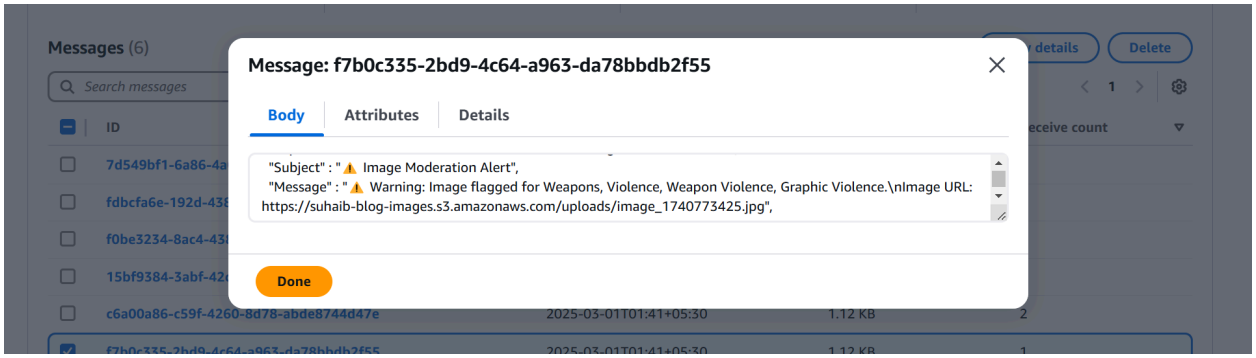


7.2 Confirm DynamoDB and SNS

- Check your DynamoDB table for new entries corresponding to the uploaded images.
- Verify that you receive notifications







✅ Step 5: Setup CI/CD with AWS CodePipeline

We will set up **two CodeBuild projects** inside a single **CodePipeline** to handle both frontend and backend builds.

---

1. Store Your Code in GitHub

- Create a **GitHub repository** and push your project structure.

📂 GitHub Repository Structure for Your Serverless Blog Platform

Here's how you can structure your GitHub repository for better organization and maintainability:

```
serverless-blog/
├── frontend/
│   ├── src/
│   ├── public/
│   ├── dist/              # ✅ Vite build output
│   ├── .env               # ✅ Frontend environment variables
│   ├── buildspec.yml      # ✅ For frontend CodeBuild
│   ├── package.json       # ✅ Frontend dependencies
│   ├── vite.config.js     # ✅ Vite config file
├── backend/
│   ├── lambdas/
│   │   ├── CreatePost.py
│   │   ├── GetPosts.py
```

```
│   │   ├── GetPostUrl.py
│   │   ├── moderate_image.py
│   ├── buildspec.yml      # ✅ For backend CodeBuild
├── cloudformation/
│   ├── template.yaml      # ✅ CloudFormation Template
├── .gitignore
├── README.md
```

2. Configure files:

Frontend (frontend/.gitignore)

```
node_modules/
dist/
.env
.vite
```

Backend (backend/.gitignore)

```
__pycache__/
*.pyc
node_modules/
.env
```

Global .gitignore

```
.DS_Store
*.log
*.swp
```

---

3. Set Up .env File for Frontend

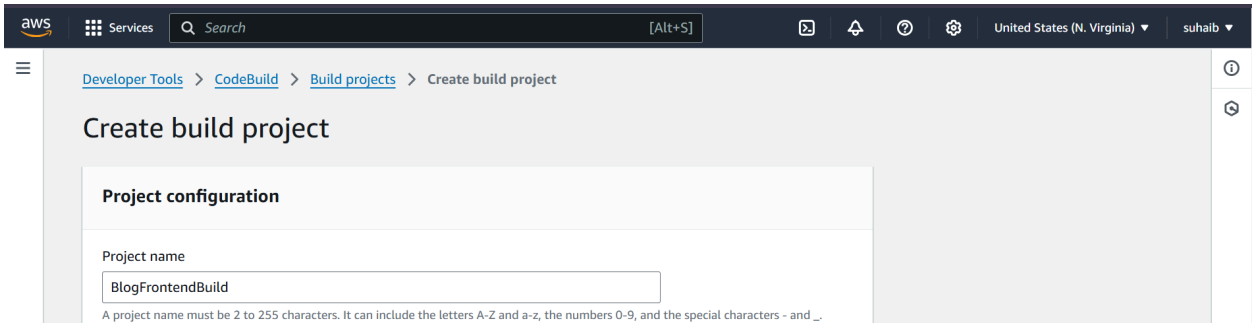
Create a .env file inside **frontend/**:

```
VITE_API_URL=https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod
```

4. Create a CodeBuild Project for Frontend

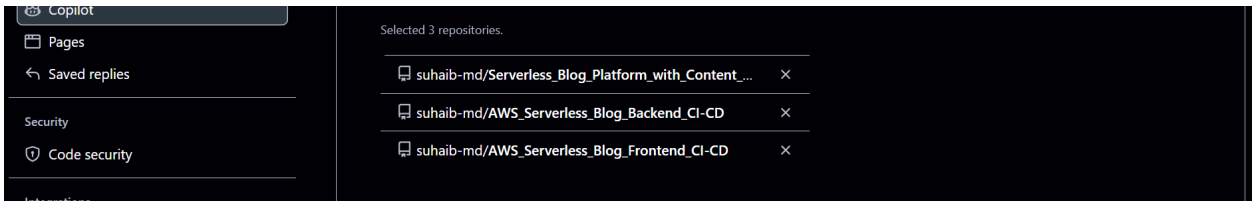
1. **Go to AWS CodeBuild Console**
   - Click **Create Build Project**.
2. **Enter Project Details:**

- **Project name**: BlogFrontendBuild

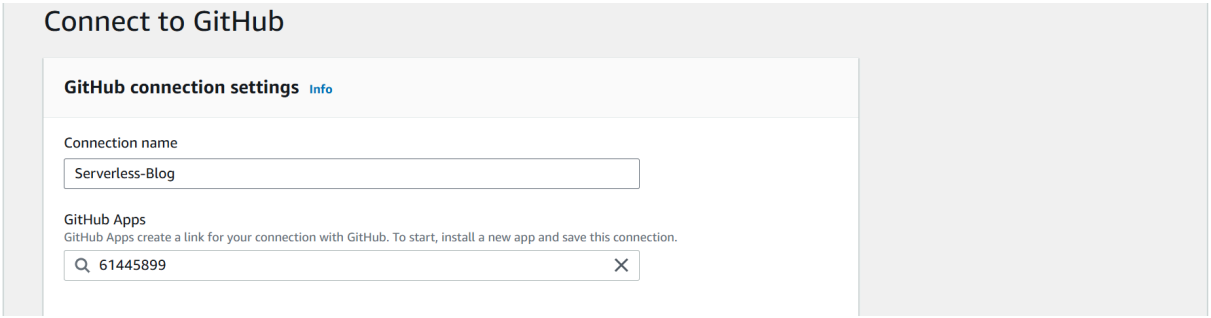


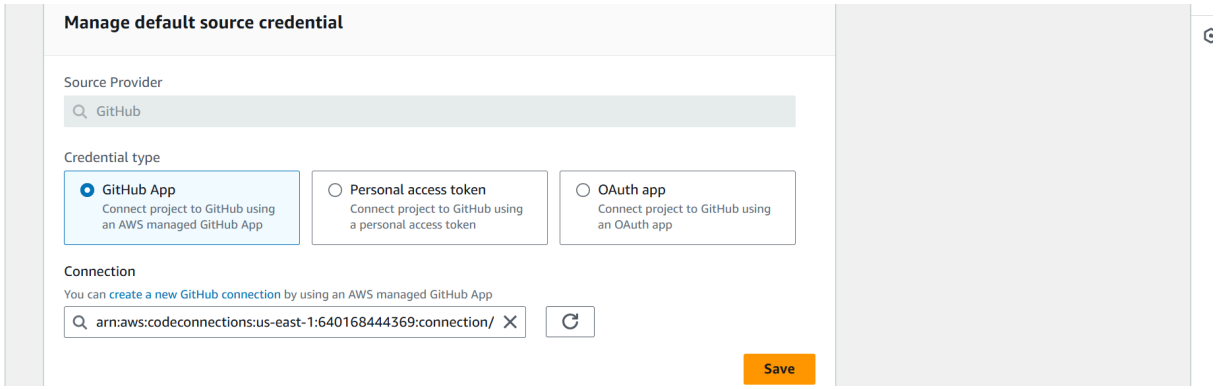
- **Description**: "Build and deploy the frontend"
- **Environment**:
  - Select **Managed image**
  - Operating system: **Amazon Linux 2**
  - Runtime: **Standard**
  - Image: aws/codebuild/standard:latest
  - Environment type: **Linux**
- **Service role**: Create a new role or use an existing one.

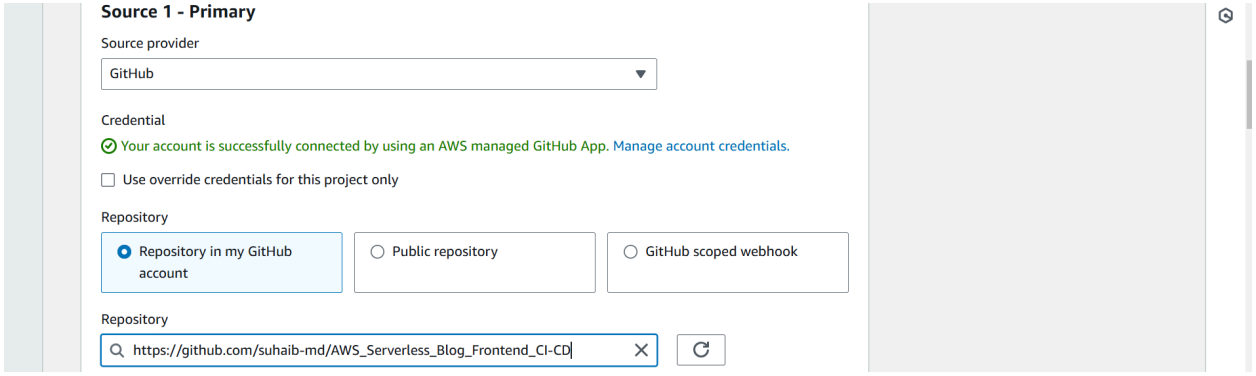
3. **Specify Source:**
   - Select **GitHub**.
   - Connect your GitHub account.

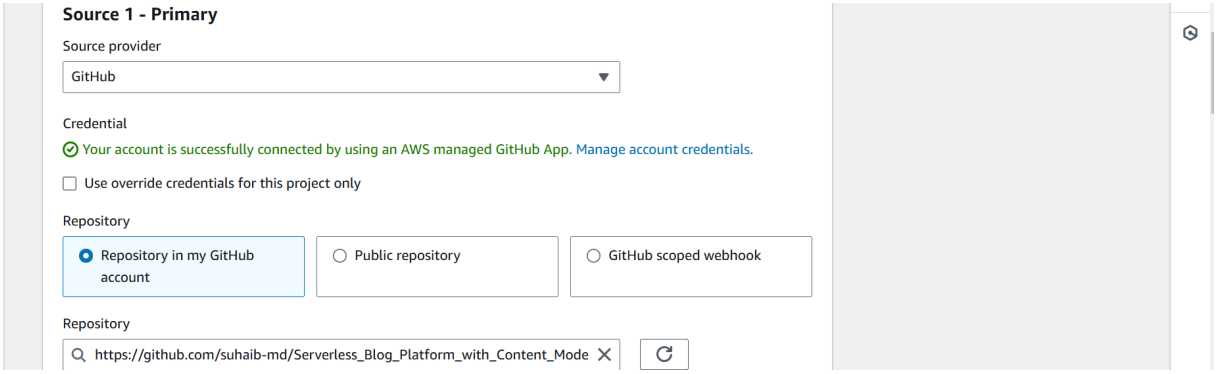


- Choose your **frontend repository**.
- Select the **main branch**.





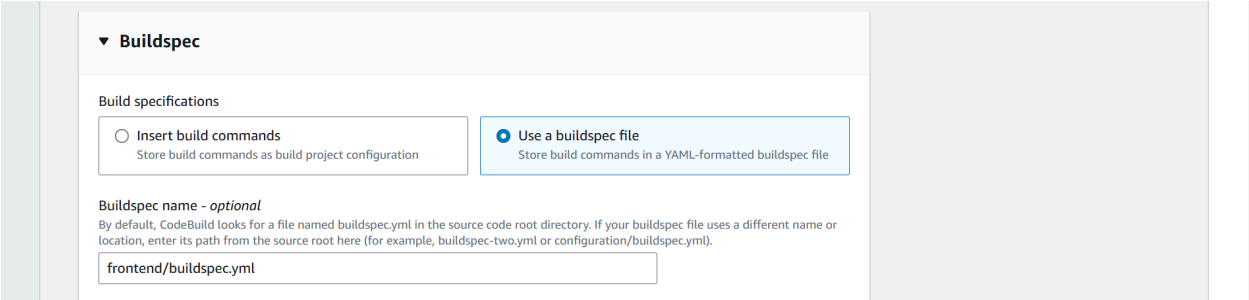




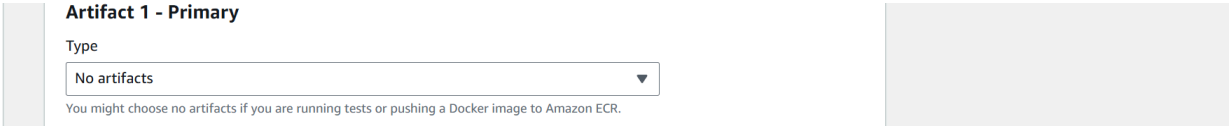
4. **Configure Build Commands:**
   - Choose **Use a buildspec file**.
   - Ensure your **buildspec.yml** exists in your repository.

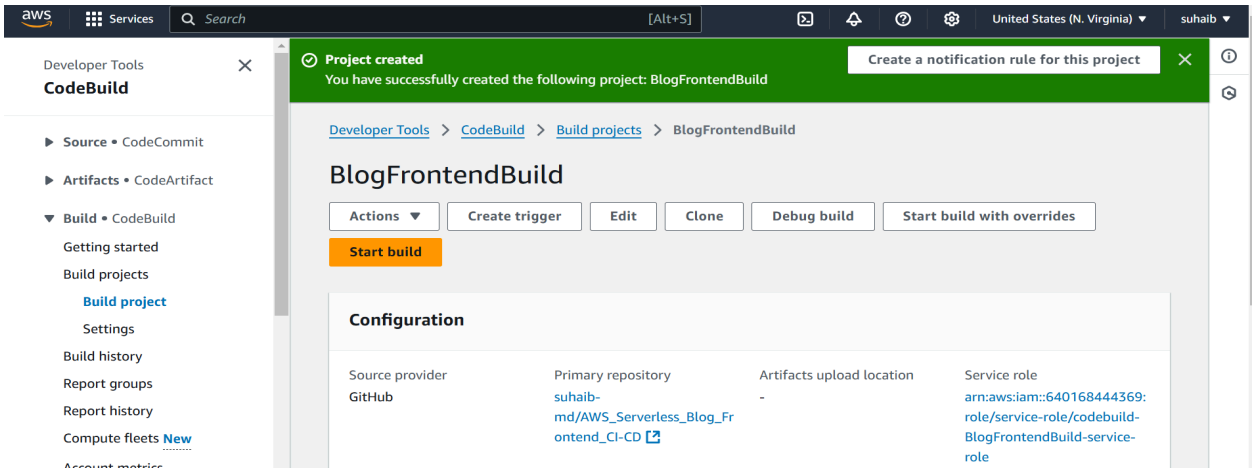
```
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 18
    commands:
```

```
      - cd frontend  # Move into the frontend directory
      - npm install
  build:
    commands:
      - npm run build
  post_build:
    commands:
      - aws s3 sync dist/ s3://www.suhaib.fun/ --delete

artifacts:
  files:
    - frontend/dist/**/*
```

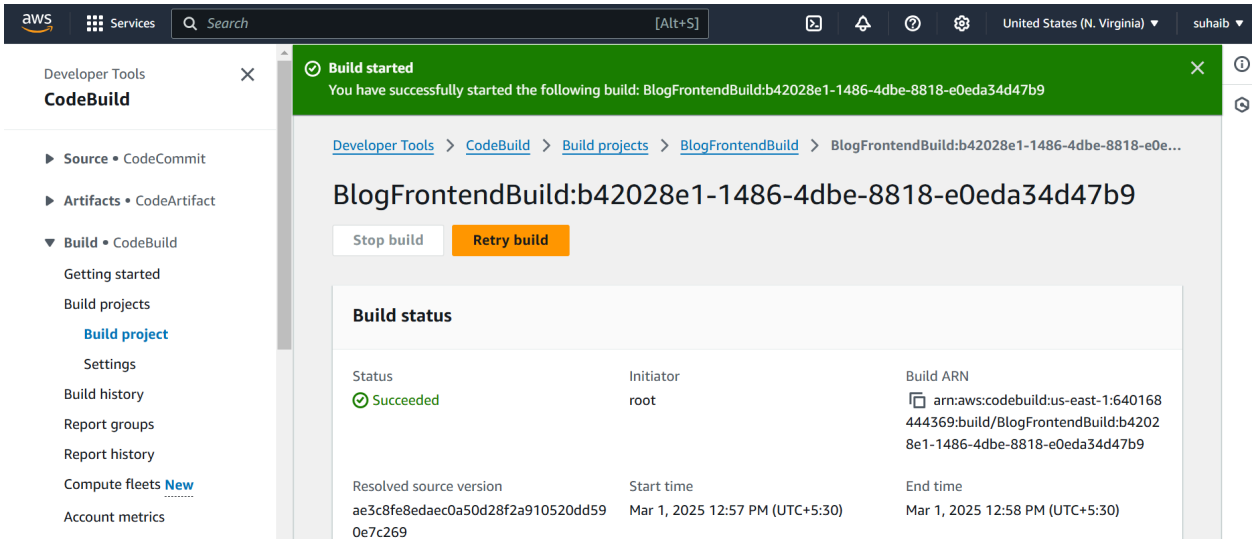


5. **Artifacts:**
    - Choose **No Artifacts**.
    - Click **Create Build Project**.





6. Run build:

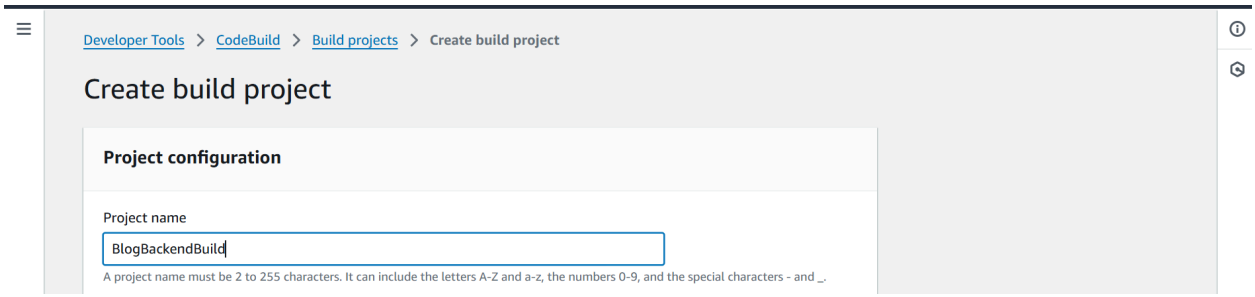


ISSUES:

- **Attach Permissions to access S3 so build can sync updates with S3**

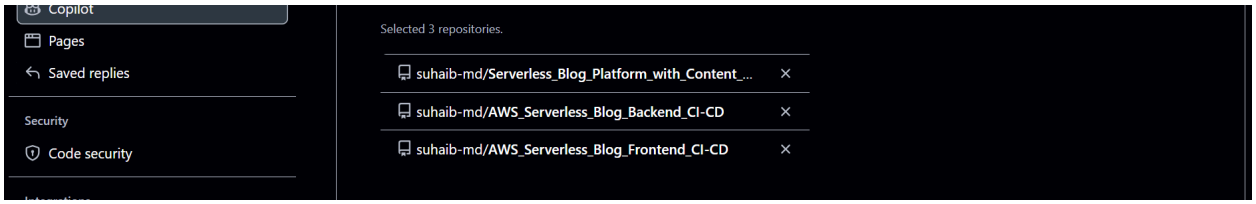
5. Create a CodeBuild Project for Backend

- **Go to AWS CodeBuild Console**
  - Click **Create Build Project**.
- **Enter Project Details:**
  - **Project name**: BlogBackendBuild

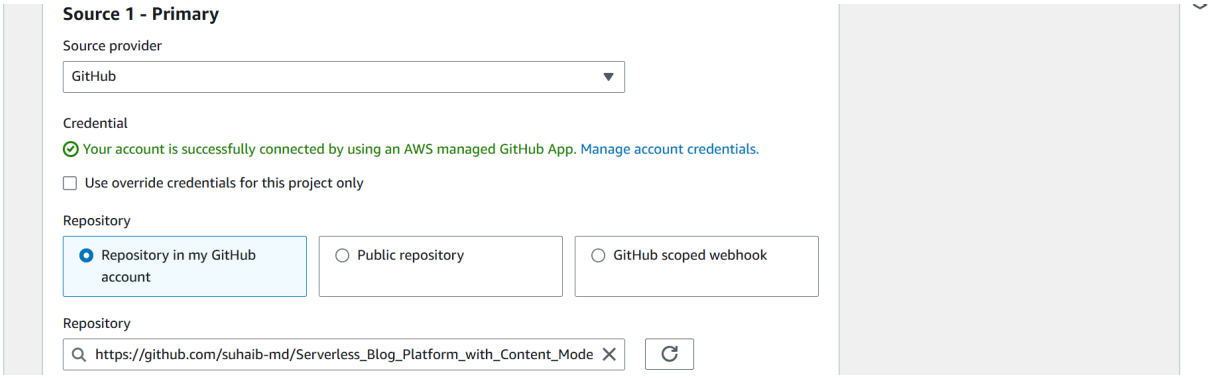


  - **Description**: "Build and deploy the backend"
  - **Environment**:
    - Select **Managed image**
    - Operating system: **Amazon Linux 2**
    - Runtime: **Standard**
    - Image: aws/codebuild/standard:latest
    - Environment type: **Linux**

    - **Service role**: Create a new role or use an existing one.
- **Specify Source:**
    - Select **GitHub**.
    - Connect your GitHub account.



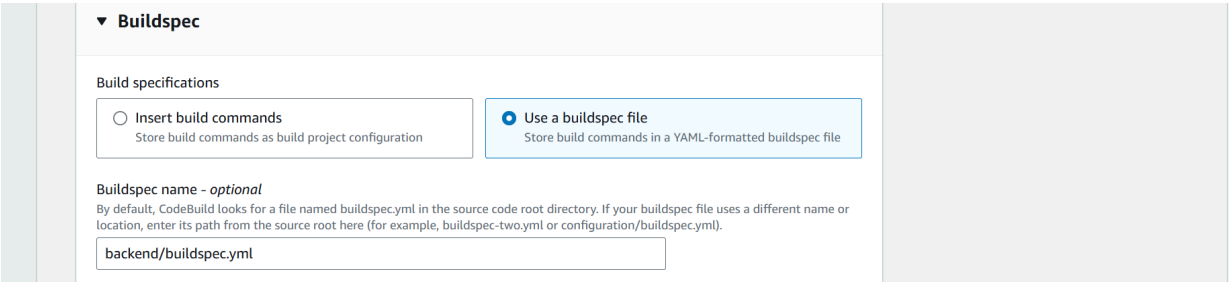
    - Choose your back**end repository**.
    - Select the **main branch**.



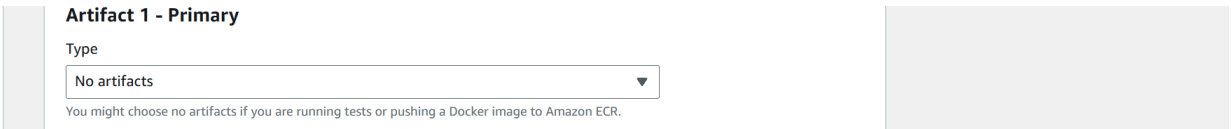
- **Configure Build Commands:**
    - Choose **Use a buildspec file**.
    - Ensure your **buildspec.yml** exists in your repository.

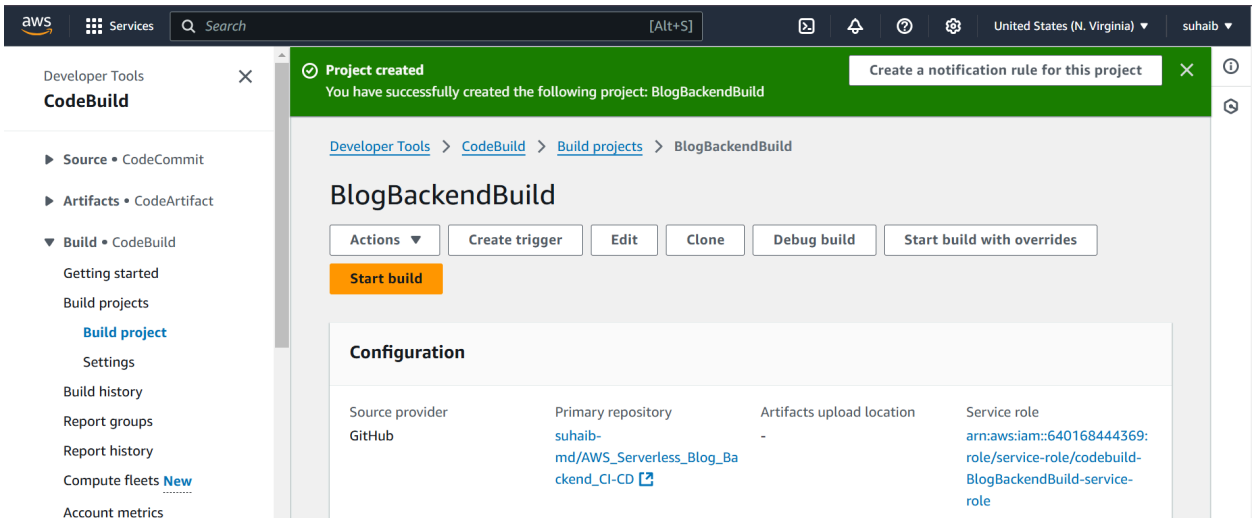
```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      python: 3.9
    commands:
      - cd backend  # Move to backend directory
      - pip install --upgrade pip
      - pip install -r requirements.txt -t .

  build:
    commands:
      - zip -r deploy.zip *
      - mv deploy.zip $CODEBUILD_SRC_DIR/deploy.zip  # Move ZIP file to root
```

```
directory

  post_build:
    commands:
      - ls -lah $CODEBUILD_SRC_DIR  # List files to confirm deploy.zip exists
      - test -f $CODEBUILD_SRC_DIR/deploy.zip && echo "deploy.zip found" ||
echo "deploy.zip MISSING"
      - aws lambda update-function-code --function-name CreatePost --zip-file
fileb://$CODEBUILD_SRC_DIR/deploy.zip
      - aws lambda update-function-code --function-name GetPosts --zip-file
fileb://$CODEBUILD_SRC_DIR/deploy.zip
      - aws lambda update-function-code --function-name GetUploadUrl --zip-file
fileb://$CODEBUILD_SRC_DIR/deploy.zip
      - aws lambda update-function-code --function-name moderate_image
--zip-file fileb://$CODEBUILD_SRC_DIR/deploy.zip

artifacts:
  files:
    - deploy.zip  # Ensure deploy.zip is available after build
```

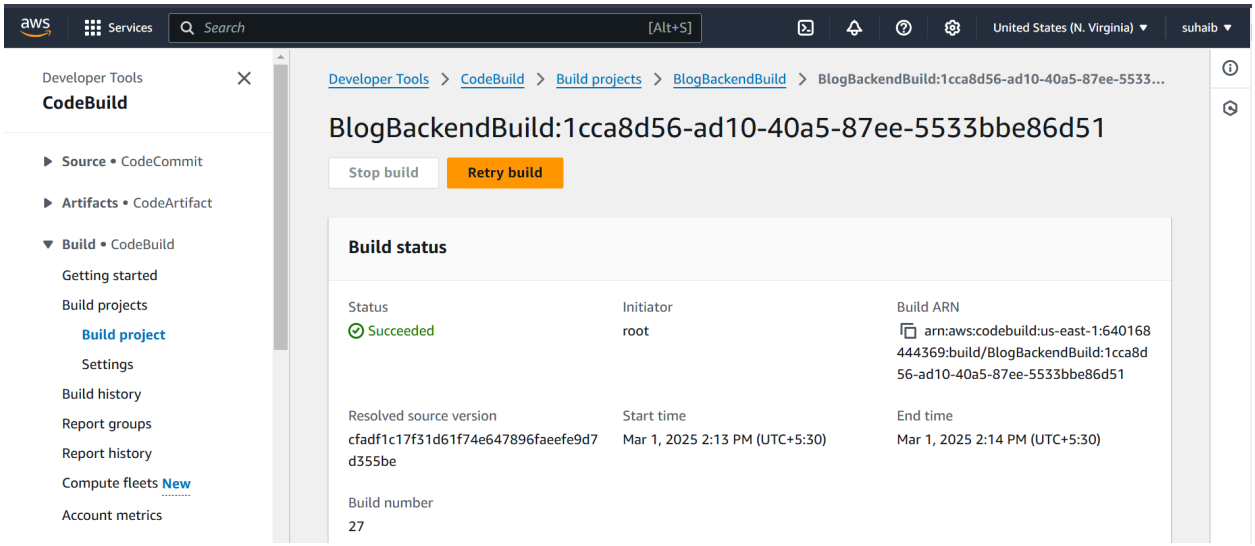


- **Artifacts:**
  - Choose **No Artifacts**.
  - Click **Create Build Project**.





- Click Start Build:



ISSUES:

- Permission for updating lambda functions

---

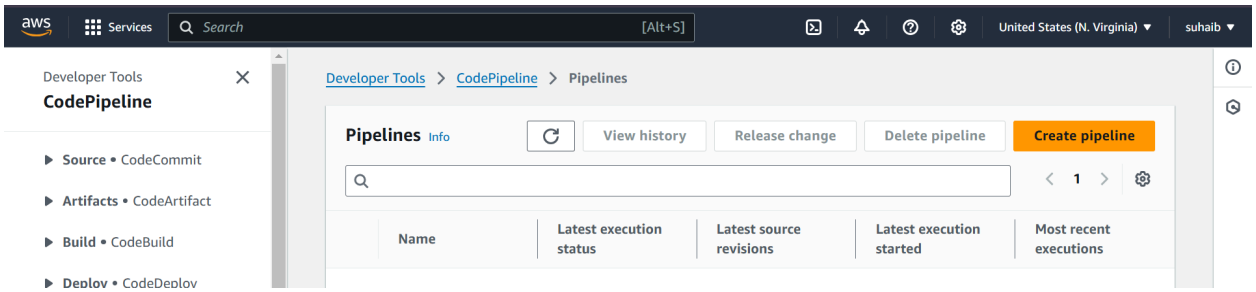
6. Create the CodePipeline

You will create a **single AWS CodePipeline** that integrates **two CodeBuild projects**:

1. **Frontend Build** → Builds & Deploys to S3 & CloudFront
2. **Backend Build** → Builds & Updates Lambda Functions

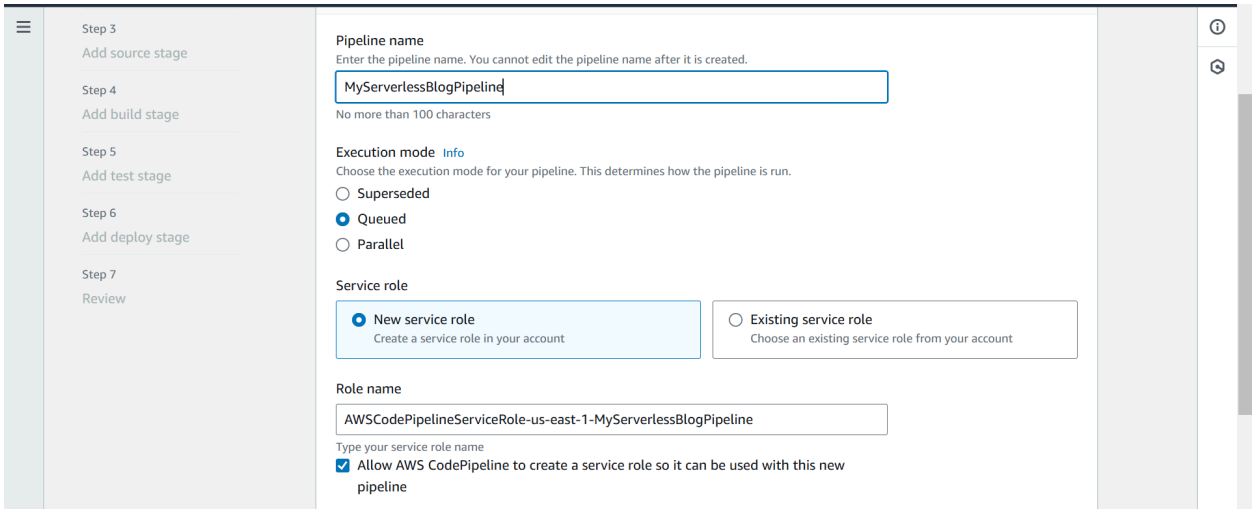
🔹 Step 1: Open AWS CodePipeline

1. Sign in to the **AWS Management Console**.
2. Navigate to **AWS CodePipeline**.
3. Click **Create pipeline**.



🔹 Step 2: Configure Pipeline Settings

1. **Pipeline name** → MyServerlessBlogPipeline
2. **Service role** → Select New service role
    - AWS will automatically create the necessary permissions

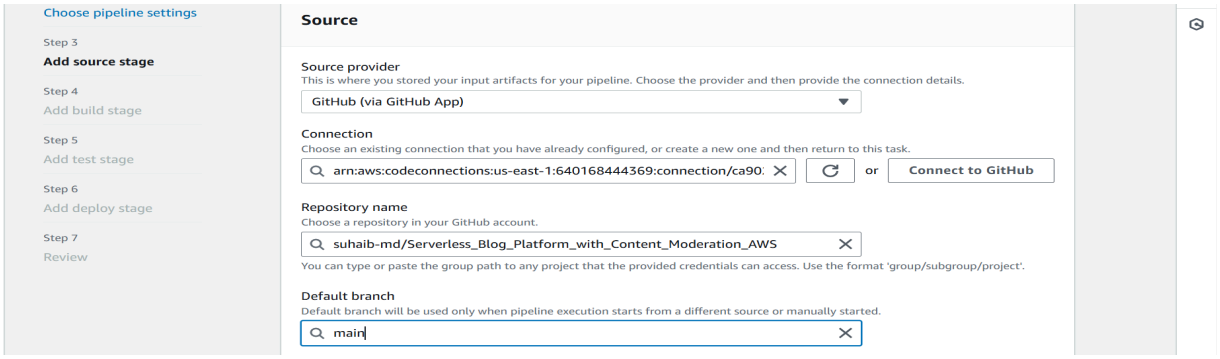


3. **Artifact store** → Choose **Amazon S3**
    - Select an existing S3 bucket (or let AWS create one)
4. Click **Next**

🔹 Step 3: Add Source Stage

1. **Source provider** → Select **GitHub (Version 2)**
2. **Repository** → Select your GitHub repo
3. **Branch name** → Choose main (or your deployment branch)
4. **Output artifact** → Name it SourceArtifact

5. Click **Next**

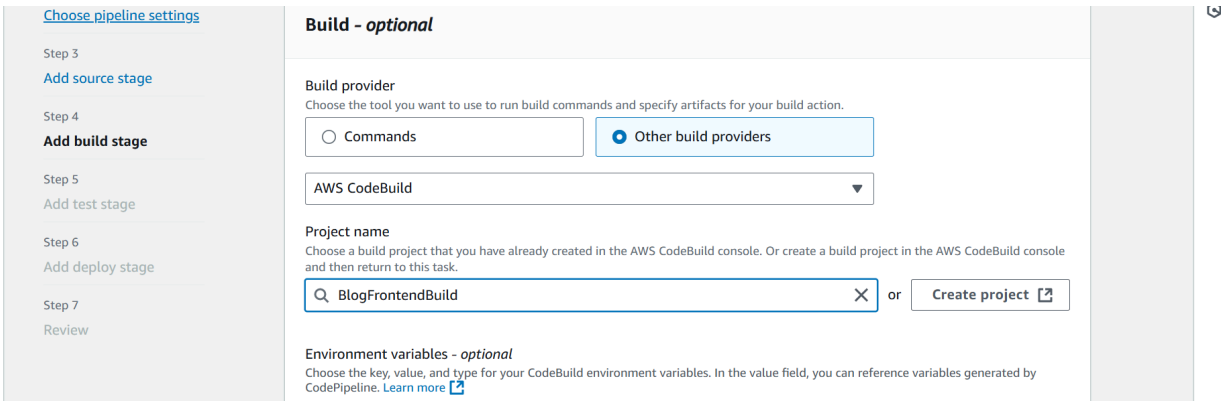


🔹 Step 4: Add Build Stages

You'll add **two CodeBuild projects** – one for the frontend and one for the backend.

➤ Add Frontend Build Stage

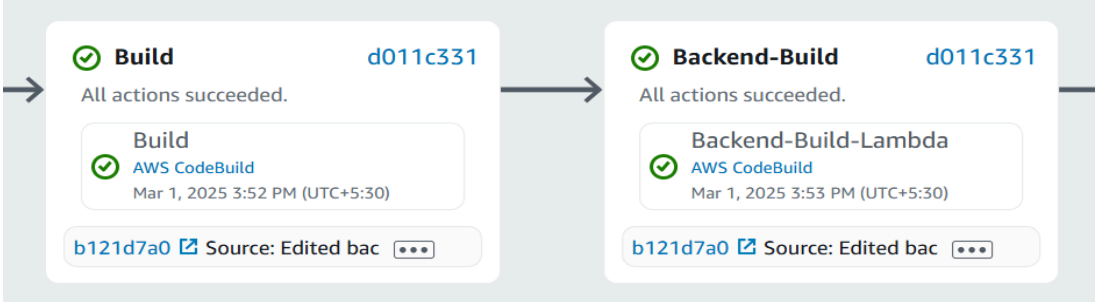
1. **Stage name** → Build-Frontend
2. **Action name** → FrontendBuild
3. **Action provider** → Select **AWS CodeBuild**
4. **Region** → Choose your AWS region
5. **Input artifact** → Select SourceArtifact
6. **Project name** → Select your **frontend CodeBuild project**
7. **Output artifact** → Name it FrontendBuildArtifact
8. Click **Add action**



➤ Add Backend Build Stage

1. Click **Add stage** → Name it Build-Backend
2. Click **Add action**
3. **Action name** → BackendBuild

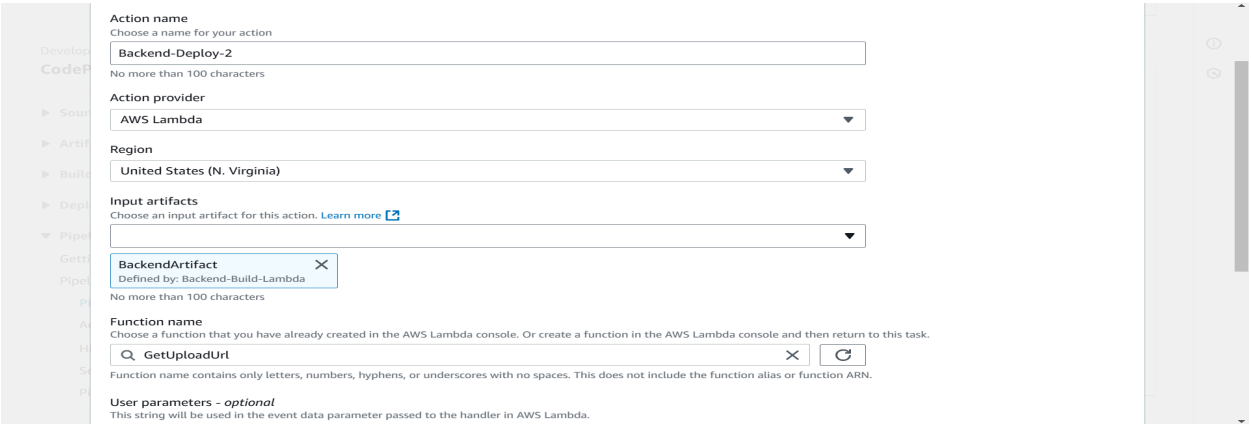
4. **Action provider** → Select **AWS CodeBuild**
5. **Region** → Choose your AWS region
6. **Input artifact** → Select SourceArtifact
7. **Project name** → Select your **backend CodeBuild project**
8. **Output artifact** → Name it BackendBuildArtifact
9. Click **Next**



🔹 Step 5: Add Deploy Stage

This stage will update your **AWS Lambda functions**.

1. Click **Add stage** → Name it Deploy
2. Click **Add action**
3. **Action name** → DeployLambda
4. **Action provider** → Select **AWS Lambda**
5. **Function name** → Select your **Lambda function** (e.g., CreatePost)
6. Click **Add action** for each Lambda function you need to update
7. Click **Next**



Action name
Choose a name for your action
Backend-Deploy-3
No more than 100 characters

Action provider
AWS Lambda

Region
United States (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more

BackendArtifact
Defined by: Backend-Build-Lambda
No more than 100 characters

Function name
Choose a function that you have already created in the AWS Lambda console. Or create a function in the AWS Lambda console and then return to this task.
moderate_image
Function name contains only letters, numbers, hyphens, or underscores with no spaces. This does not include the function alias or function ARN.

Action name
Choose a name for your action
Backend-Deploy-4
No more than 100 characters

Action provider
AWS Lambda

Region
United States (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more

BackendArtifact
Defined by: Backend-Build-Lambda
No more than 100 characters

Function name
Choose a function that you have already created in the AWS Lambda console. Or create a function in the AWS Lambda console and then return to this task.
CreatePost
Function name contains only letters, numbers, hyphens, or underscores with no spaces. This does not include the function alias or function ARN.

Edit: Deploy

Edit stage

Conditions Entry: Not configured Success: Not configured Failure: Not configured View details

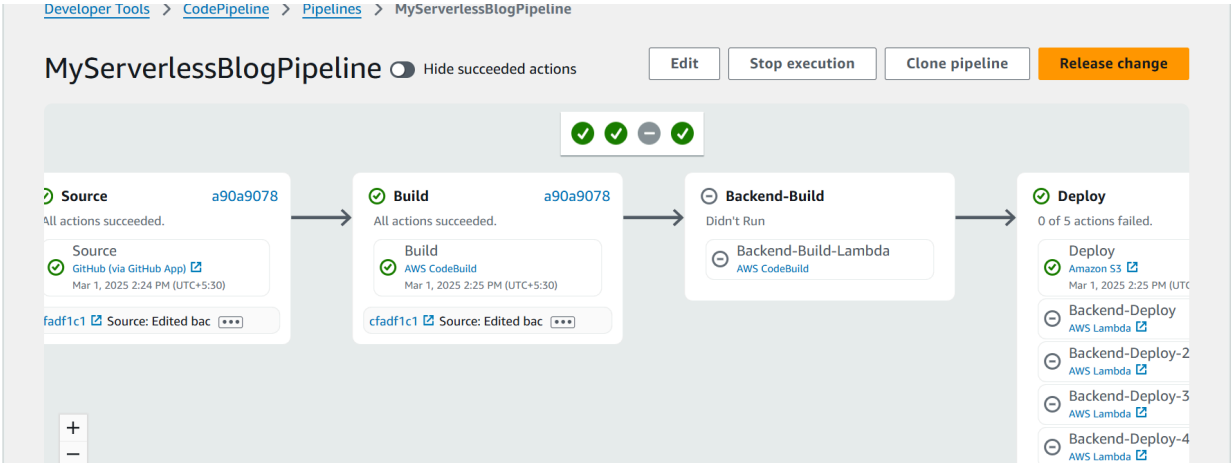
Deploy Amazon S3	Backend-Deploy AWS Lambda	Backend-Deploy-2 AWS Lambda
Backend-Deploy-3 AWS Lambda	Backend-Deploy-4 AWS Lambda	

Automated stage configuration: Enable automatic rollback on stage failure

🔹 Step 6: Review & Create Pipeline

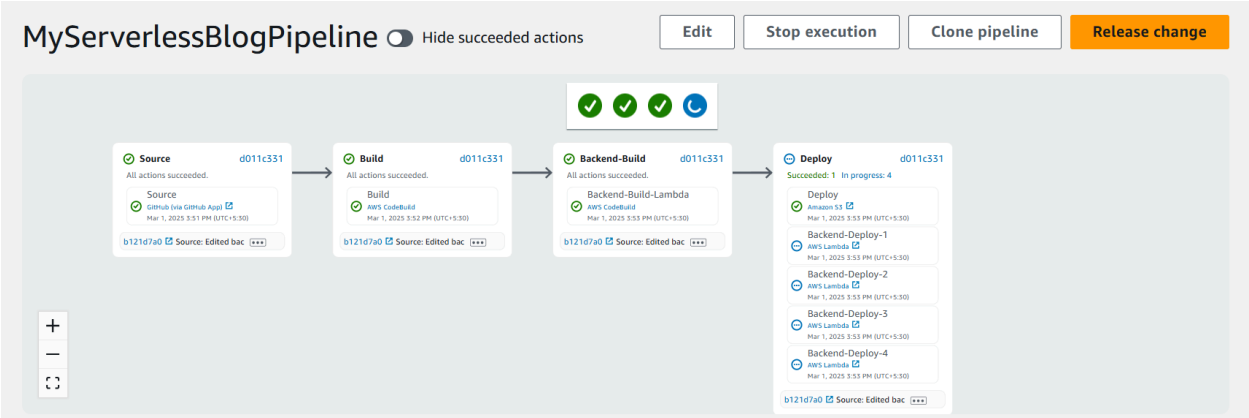
1. Review all stages
2. Click **Create pipeline**

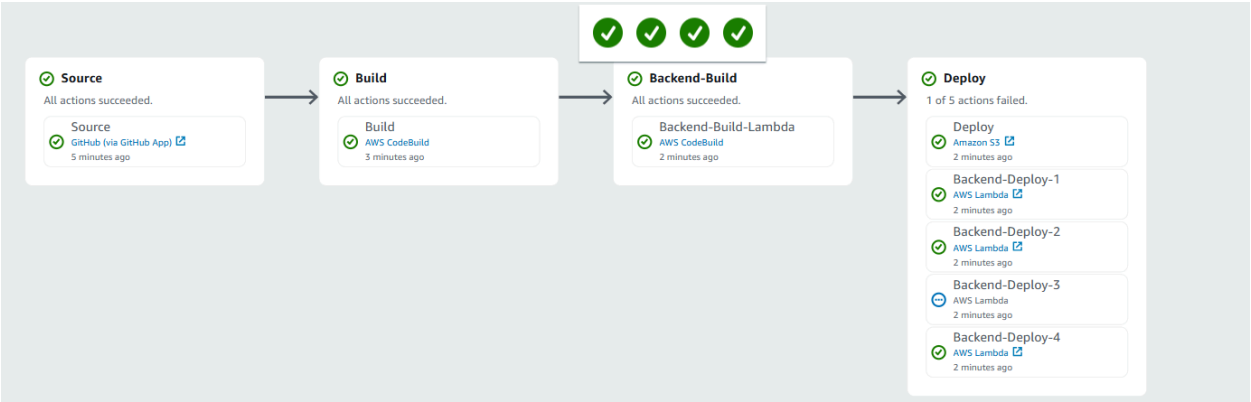
Your pipeline will automatically start running!



🔹 Step 7: Test Your Pipeline

1. In **AWS CodePipeline**, select MyServerlessBlogPipeline.
2. Click **Release change** to trigger a manual deployment.
3. Watch the progress through **Source → Build → Deploy** stages.





ISSUES:

- **Permission issues**: ensure the role has access to codepipeline, lambda and S3 so it knows the build is over and when to succeed.
- **The AWS Lambda function GetPostsExternal link failed to return a result. Check the function to verify that it has permission to call the PutJobSuccessResult action and that it made a call to PutJobSuccessResult:** The error message suggests that **CodePipeline is waiting for a response from your Lambda function**, but it's not getting one. Add a report_pipeline_success event to each of the lambda functions.

What Happens Now?

1. Whenever you push to GitHub, CodePipeline **automatically runs**.
2. **Frontend:**
    - CodeBuild runs Vite build → uploads `dist/` to S3.
    - CloudFront automatically serves updated files.
3. **Backend:**
    - CodeBuild packages and deploys Lambda functions.
    - API Gateway serves new backend changes.
4. Rekognition automatically **moderates images** and sends SNS notifications.

Conclusion:

This project successfully demonstrates the implementation of a **serverless blog platform** using AWS services, leveraging **S3 and CloudFront** for static website hosting, **API Gateway and Lambda** for backend processing, **DynamoDB** for data storage, and **Amazon Rekognition** for automated image moderation. The integration of **SNS and SQS** ensures efficient notification and message handling. By utilizing a fully serverless architecture, this solution achieves **high scalability, cost efficiency, and minimal operational overhead**.

React Code:

Home.jsx:

```
import { useEffect, useState } from "react";

const Home = () => {
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(true); // 🔥 Loading state

  useEffect(() => {

fetch("https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-posts")
      .then((res) => res.json())
      .then((data) => {
        try {
          const parsedData = JSON.parse(data.body);
          if (Array.isArray(parsedData)) {
            setPosts(parsedData);
          } else {
            console.error("Unexpected API response:", parsedData);
          }
        } catch (error) {
          console.error("Error parsing response:", error);
        }
      })
      .catch((err) => console.error("Error fetching posts:", err))
      .finally(() => setLoading(false)); // ✅ Stop loading
  }, []);

  return (
    <div className="container mx-auto p-6">
      <h2 className="text-4xl font-bold text-gray-800 text-center mb-6">📖 Latest Blog Posts</h2>

      {/* 🔥 Loading Skeleton */}
      {loading ? (
        <div className="grid md:grid-cols-3 gap-6 animate-pulse">
```

```
          {[...Array(6)].map((_, index) => (
            <div key={index} className="bg-gray-200 p-4 rounded-lg
h-56"></div>
          ))}
        </div>
      ) : posts.length > 0 ? (
        <div className="grid md:grid-cols-3 gap-6">
          {posts.map((post) => (
            <div
              key={post.PostID}
              className="bg-white p-4 shadow-lg rounded-lg transform
transition hover:scale-105 hover:shadow-2xl"
            >
              {post.imageUrl && (
                <img
                  src={post.imageUrl}
                  alt={post.title}
                  className="w-full h-48 object-cover rounded-md"
                />
              )}
              <h3 className="text-xl font-semibold mt-3
text-gray-800">{post.title}</h3>
              <p className="text-gray-600
mt-2">{post.content.substring(0, 100)}...</p>
              <button className="mt-3 text-blue-600
hover:text-blue-800">Read more →</button>
            </div>
          ))}
        </div>
      ) : (
        <p className="text-gray-600 text-center text-lg mt-6">No
posts available. 📭</p>
      )}
    </div>
  );
};

export default Home;
```

CreatePost.jsx:

```
import { useState } from "react";

const CreatePost = () => {
  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");
  const [image, setImage] = useState(null);
  const [uploadUrl, setUploadUrl] = useState("");
  const [isUploading, setIsUploading] = useState(false);
  const [errorMessage, setErrorMessage] = useState(""); // 🔹 Store
error messages

  const getPresignedUrl = async (file) => {
    setIsUploading(true);
    setErrorMessage(""); // Clear previous errors
    try {
      const response = await fetch(

`https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-uplo
ad-url?fileName=${file.name}`
      );

      if (!response.ok) {
        throw new Error(`Failed to fetch pre-signed URL:
${response.statusText}`);
      }

      const data = await response.json();
      const parsedData = JSON.parse(data.body);
      if (!parsedData.uploadUrl) throw new Error("No pre-signed URL
received");

      setUploadUrl(parsedData.uploadUrl);
    } catch (error) {
      setErrorMessage("⚠️ Error getting pre-signed URL. Please try
again.");
    } finally {
```

```
      setIsUploading(false);
    }
  };

  const handleImageChange = async (e) => {
    const file = e.target.files[0];
    if (!file) return;
    setImage(file);
    await getPresignedUrl(file);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!title || !content || !image) {
      alert("All fields are required.");
      return;
    }

    if (!uploadUrl) {
      alert("Please wait for the image to finish uploading.");
      return;
    }

    try {
      // Upload image to S3
      const uploadResponse = await fetch(uploadUrl, {
        method: "PUT",
        body: image,
        headers: { "Content-Type": image.type },
      });

      if (!uploadResponse.ok) {
        throw new Error(`Image upload failed:
${uploadResponse.statusText}`);
      }

      const imageUrl = uploadUrl.split("?")[0];
```

```
      // Submit blog post
      const response = await fetch(

"https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/create-post",
        {
          method: "POST",
          body: JSON.stringify({ title, content, imageUrl }),
          headers: { "Content-Type": "application/json" },
        }
      );

      const responseData = await response.json();
      const parsedData = JSON.parse(responseData.body);

      if (!response.ok) {
        throw new Error(parsedData.error || "Failed to create post");
      }

      //  🔹 Handle Moderation Warning from API
      if (parsedData.warning) {
        setErrorMessage(parsedData.warning);
        return;
      }

      alert("✅ Blog post created successfully!");
      setTitle("");
      setContent("");
      setImage(null);
      setUploadUrl("");
    } catch (error) {
      setErrorMessage(`⚠️ ${error.message}`);
    }
  };

  return (
    <div className="max-w-3xl mx-auto p-6 bg-white shadow-lg
rounded-lg mt-6">
      <h2 className="text-3xl font-bold mb-6 text-center
```

```
text-blue-700">
        ✏️ Create a New Blog Post
      </h2>

      {/*  🔹 Display Error Messages */}
      {errorMessage && (
        <div className="mb-4 p-4 bg-red-100 text-red-600
font-semibold rounded-lg text-center">
          {errorMessage}
        </div>
      )}

      <form onSubmit={handleSubmit} className="space-y-6">
        {/* Title Input */}
        <div>
          <label className="block text-gray-700
font-semibold">Title</label>
          <input
            type="text"
            className="w-full border border-gray-300 p-3 rounded-lg
focus:ring-2 focus:ring-blue-400 focus:outline-none"
            value={title}
            onChange={(e) => setTitle(e.target.value)}
            placeholder="Enter post title"
            required
          />
        </div>

        {/* Content Input */}
        <div>
          <label className="block text-gray-700
font-semibold">Content</label>
          <textarea
            className="w-full border border-gray-300 p-3 rounded-lg
focus:ring-2 focus:ring-blue-400 focus:outline-none"
            rows="5"
            value={content}
            onChange={(e) => setContent(e.target.value)}
            placeholder="Write your blog content here..."
```

```
            required
          ></textarea>
        </div>

        {/* Image Upload */}
        <div>
          <label className="block text-gray-700 font-semibold mb-2">
            Upload Image
          </label>
          <div className="flex items-center justify-center w-full">
            <label className="cursor-pointer bg-gray-100
hover:bg-gray-200 border border-gray-300 rounded-lg px-6 py-3
text-gray-600 text-sm font-medium">
              📷 Choose an Image
              <input
                type="file"
                accept="image/*"
                onChange={handleImageChange}
                className="hidden"
                required
              />
            </label>
          </div>
          {image && (
            <p className="mt-2 text-sm text-gray-600">
              Selected: <span
className="font-semibold">{image.name}</span>
            </p>
          )}
          {isUploading && (
            <p className="text-blue-600 text-sm mt-1">Uploading
image, please wait...</p>
          )}
        </div>

        {/* Submit Button */}
        <button
          type="submit"
          className={`w-full py-3 rounded-lg text-white font-semibold
```

```
transition ${
            isUploading
              ? "bg-gray-400 cursor-not-allowed"
              : "bg-blue-600 hover:bg-blue-700"
          }`}
          disabled={isUploading}
        >
          {isUploading ? "Uploading..." : "🚀 Publish Post"}
        </button>
      </form>
    </div>
  );
};

export default CreatePost;
```

components/Navbar.jsx:

```
import { Link } from "react-router-dom";
import { useState } from "react";

const Navbar = () => {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <nav className="bg-blue-700 text-white p-4 shadow-md">
      <div className="container mx-auto flex justify-between
items-center">
        {/* Logo / Title */}
        <Link to="/" className="text-2xl font-bold tracking-wide
hover:text-gray-200 transition">
          🚀 Serverless Blog
        </Link>

        {/* Hamburger Menu (Mobile) */}
        <button
          onClick={() => setIsOpen(!isOpen)}
```

```
          className="md:hidden focus:outline-none"
        >
          <svg
            className="w-8 h-8"
            fill="none"
            stroke="currentColor"
            viewBox="0 0 24 24"
            xmlns="http://www.w3.org/2000/svg"
          >
            <path
              strokeLinecap="round"
              strokeLinejoin="round"
              strokeWidth={2}
              d="M4 6h16M4 12h16m-7 6h7"
            />
          </svg>
        </button>

        {/* Navigation Links (Desktop) */}
        <div className="hidden md:flex space-x-6">
          <Link to="/" className="hover:text-gray-200 transition
text-lg font-medium">
            Home
          </Link>
          <Link
            to="/create"
            className="bg-white text-blue-700 px-4 py-2 rounded-lg
font-medium hover:bg-gray-100 transition"
          >
            ✏️ Create Blog
          </Link>
        </div>
      </div>

      {/* Mobile Menu */}
      {isOpen && (
        <div className="md:hidden mt-4 bg-blue-800 p-4 rounded-lg
space-y-2">
          <Link
```

```
            to="/"
            className="block text-lg font-medium hover:text-gray-200
transition"
            onClick={() => setIsOpen(false)}
          >
            Home
          </Link>
          <Link
            to="/create"
            className="block bg-white text-blue-700 px-4 py-2
rounded-lg font-medium hover:bg-gray-100 transition w-full
text-center"
            onClick={() => setIsOpen(false)}
          >
            ✏️ Create Blog
          </Link>
        </div>
      )}
    </nav>
  );
};

export default Navbar;
```