



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

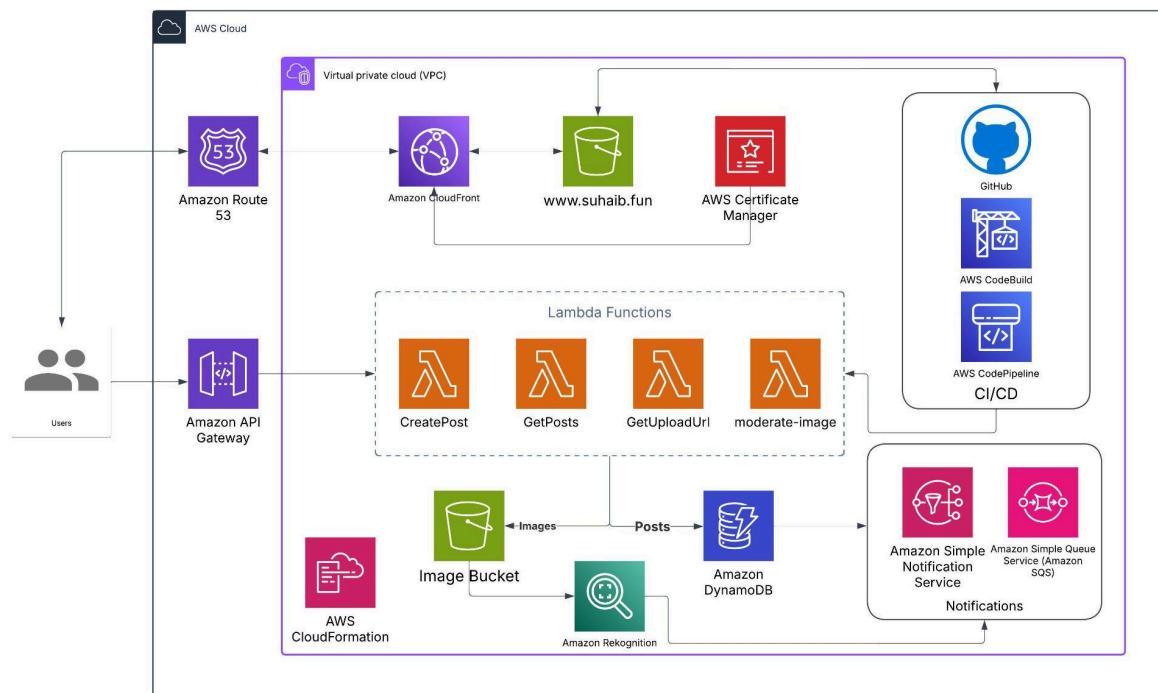
- **Service Used:** AWS CodePipeline, CloudFormation
- **CodePipeline** will automate the build and deployment process for frontend and backend updates.
- **CloudFormation** will provision infrastructure, ensuring consistency across deployments.

6. Architectural Diagram

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

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

1 Create an IAM User via AWS Console

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

The screenshot shows the 'Specify user details' step of the 'Create user' wizard. On the left, a sidebar lists three steps: Step 1 (selected), Step 2 (Set permissions), and Step 3 (Review and create). The main area is titled 'User details' and contains a 'User name' field with the value 'aws-cli-user'. Below the field is a note: 'The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = . @ _ - (hyphen)'. There is also an optional checkbox for 'Provide user access to the AWS Management Console'.

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

The screenshot shows the 'Review and create' step of the wizard. It displays the user details: 'User name' is 'aws-cli-user', 'Console password type' is 'None', and 'Require password reset' is 'No'. Below this is the 'Permissions summary' section, which shows a single policy named 'AdministratorAccess' attached to the user. The policy is described as 'AWS managed - job function' and is listed under 'Permissions policy'.

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

The screenshot shows the 'Access key best practices & alternatives' step of the wizard. The sidebar lists three steps: Step 1 (selected), Step 2 (optional), and Step 3. The main area is titled 'Access key best practices & alternatives' and includes a note: 'Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.' Below this is a 'Use case' section with a radio button selected for 'Command Line Interface (CLI)', with a note: 'You plan to use this access key to enable the AWS CLI to access your AWS account.'

The screenshot shows the AWS IAM 'Create access key' page. At the top, there's a green banner with the text 'Access key created' and a note: 'This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.' Below the banner, a sidebar on the left lists three steps: Step 1 (Access key best practices & alternatives), Step 2 - optional (Set description tag), and Step 3 (Retrieve access keys, which is currently selected). The main area is titled 'Retrieve access keys' and contains two sections: 'Access key' and 'Secret access key'. Under 'Access key', the value 'AKIAZKIDIDIHIQSZDMOAJ' is shown. Under 'Secret access key', the value 'WiOXv95MHwAz+nYFtIDJSabgeWBYu27EcppIb0tX' is shown with a 'Hide' link.

2 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]: AKIAZKIDIDIHIQSZDMOAJ
AWS Secret Access Key [*****ZIFp]: WiOXv95MHwAz+nYFtIDJSabgeWBYu27EcppIb0tX
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**.

1 Create a CloudFormation Template (`vpc.yml`)

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

2 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"
}
]
```

The screenshot shows the AWS VPC dashboard. On the left, there's a sidebar with 'VPC dashboard' selected, along with 'EC2 Global View' and a dropdown for 'Filter by VPC'. Below that is a section for 'Virtual private cloud' with a link to 'Your VPCs'. The main content area is titled 'Your VPCs (2)' and shows a table with the following data:

	Name	VPC ID	State	Block Public...	IPv4 CIDR
<input type="checkbox"/>	-	vpc-0daf9dc2be1dc0d07	Available	Off	172.31.0.0/16
<input type="checkbox"/>	MyCustomVPC	vpc-040deeb9f10652190	Available	Off	10.0.0.0/16

At the top of the dashboard, there are several navigation and search tools, including a search bar, a 'Create VPC' button, and account information for 'United States (N. Virginia)' and 'suhaib'.

✓ 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**.

1 Architecture Overview

This process follows this structure:

- **Amazon S3:**
 - www.suhaib.fun (Public) – Hosts the static React application.
 - **suhaiib.fun** (Private) – Redirects traffic to www.suhaib.fun.
- **Amazon Route 53:**
 - Manages the domain **suhaiib.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**.

2 Implementation Steps

2.1 Setting Up S3 Buckets

Step 1: Create Two S3 Buckets

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

<input type="radio"/>	suhai.fun	US East (N. Virginia) us-east-1	View analyzer for us-east-1	February 28, 2025, 10:50:42 (UTC+05:30)
<input type="radio"/>	www.suhai.fun	US East (N. Virginia) us-east-1	View analyzer for us-east-1	February 28, 2025, 10:51:16 (UTC+05:30)

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.suhai.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.

The screenshot shows the AWS S3 console interface. The top navigation bar includes the AWS logo, a search bar, and a dropdown for 'United States (N. Virginia)'. Below the navigation is a breadcrumb trail: 'Amazon S3 > Buckets > www.suhaib.fun'. The main content area is titled 'www.suhaib.fun' with a 'Info' link. A navigation bar below the title includes tabs for 'Objects', 'Metadata', 'Properties', 'Permissions', 'Metrics', 'Management', and 'Access Points'. The 'Objects' tab is active. The object list table has the following data:

Name	Type	Last modified	Size	Storage class
assets/	Folder	-	-	-
index.html	html	February 28, 2025, 15:40:44 (UTC+05:30)	459.0 B	Standard
vite.svg	svg	February 28, 2025, 15:40:44 (UTC+05:30)	1.5 KB	Standard

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/*"
        }
    ]
}
```

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)

Bucket ARN
arn:aws:s3:::www.suhaib.fun

Policy

```
1 ▼ {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Sid": "PublicReadGetObject",
6             "Effect": "Allow",
7             "Principal": "*",
8             "Action": "s3:GetObject",
9             "Resource": "arn:aws:s3:::www.suhaib.fun/*"
10        }
11    ]
12}
13
```

Edit statement

Select a statement

Select an existing statement in the policy or add a new statement.

[Add new statement](#)

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 4: Enable Static Website Hosting

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

The screenshot shows the AWS S3 console interface. The top navigation bar includes the AWS logo, search bar, and region selector (United States (N. Virginia)). The main navigation path is: Amazon S3 > Buckets > www.suhaiib.fun > Edit static website hosting. The page title is "Static website hosting". It displays the configuration for the bucket "www.suhaiib.fun". Under "Static website hosting", "Enable" is selected. Under "Hosting type", "Host a static website" is selected. The "Index document" field contains "index.html". A note at the bottom states: "For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see [Using Amazon S3 Block Public Access](#)". The footer includes CloudShell, Feedback, and links to Privacy, Terms, and Cookie preferences.

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

The screenshot shows the AWS S3 console interface. The top navigation bar includes the AWS logo, search bar, and region selector (United States (N. Virginia)). The main navigation path is: Amazon S3 > Buckets > suhaiib.fun > Edit static website hosting. The page title is "Static website hosting". It displays the configuration for the bucket "suhaiib.fun". Under "Static website hosting", "Enable" is selected. Under "Hosting type", "Redirect requests for an object" is selected. The "Host name" field contains "www.suhaiib.fun". The "Protocol" field is set to "http". The footer includes CloudShell, Feedback, and links to Privacy, Terms, and Cookie preferences.

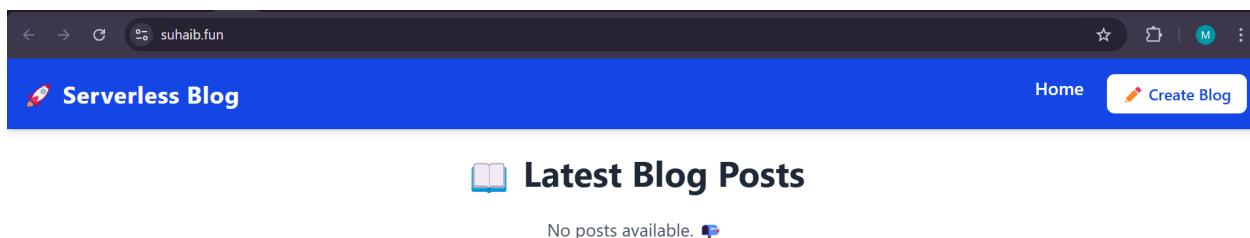
Verification Tests

- ✓ Open <http://www.suhaiib.fun.s3-website-us-east-1.amazonaws.com> to check if the app loads.



The screenshot shows a web browser window with a blue header bar. The URL in the address bar is <http://suhaiib.fun.s3-website-us-east-1.amazonaws.com/create>. The main content area has a white background and features a form titled "Create a New Blog Post". The form includes a "Title" field with a placeholder "Enter post title", a "Content" field with a placeholder "Write your blog content here...", and an "Upload Image" section with a "Choose an Image" button. At the bottom of the form is a large blue button with the text "Publish Post" and a rocket icon.

- ✓ Open <http://suhaiib.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 **suhaiib.fun**.
3. Copy the **Name Servers (NS)** and update them in **Hostinger (or another domain registrar)**.

The screenshot shows the 'Create hosted zone' configuration page in the AWS Route 53 console. The 'Domain name' field is set to 'suhaiib.fun'. The 'Description - optional' field contains 'The hosted zone is used for...'. The 'Type' section shows 'Public hosted zone' selected. The page includes standard AWS navigation and status bars.

The screenshot shows a modal dialog from Hostinger's DNS management interface. The title is 'Nameservers changed!' and it states: 'Your nameservers has been changed to:' followed by a list of four new nameservers: ns-1170.awsdns-18.org, ns-1650.awsdns-14.co.uk, ns-378.awsdns-47.com, and ns-657.awsdns-18.net. A note at the bottom says: 'It might take up to 24 hours for the domain to propagate to the new nameservers.' There is a 'Close' button at the bottom right.

Step 2: Configure DNS Records

- Create an A Record for www.suhaiib.fun
 - Type: A

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

[Route 53](#) > [Hosted zones](#) > [suhaiib.fun](#) > Create record

Step 1 Choose routing policy
Step 2 Configure records

Choose routing policy Info

The routing policy determines how Amazon Route 53 responds to queries.

Routing policy

Simple routing
Use if you want all of your clients to receive the same response(s).

Weighted
Use when you have multiple resources that do the same job, and you want to specify the proportion of traffic that goes to each resource. For example: two or more EC2 instances.

[Switch to quick create](#)

[Route 53](#) > [Hosted zones](#) > [suhaiib.fun](#) > Create record

Step 1 Choose routing policy
Step 2 Configure records

Configure records Info

You can create multiple records at a time that have the same routing policy.

Simple routing records to add to suhaiib.fun Info

Use if you want all of your clients to receive the same response(s).

Record name	Type	Value/Route traffic to	TTL (seconds)

Define simple records to this list, then choose **Create records**.

[Define simple record](#)

[Inbox - suhaiibmdv@gmail.com](#) | [Instances | EC2 | us-east-1](#) | [S3 buckets | S3 | us-east-1](#) | [Route 53 | Global](#) | [DNS / Nameservers | Hosted zones](#) | +

[us-east-1.console.aws.amazon.com/route53/v2/hostedzones?region=us-east-1#CreateRecordSet/Z081120814W371JJUT9S?stepIndex=1&routingPolicyId=1&hostedZoneId=Z081120814W371JJUT9S&recordSetName=www&recordType=A&value=s3-website-us-east-1.amazonaws.com&tTLSeconds=3600&evaluateTargetHealth=false&weight=1&order=1&aliasTargetName=suhaiib.fun&aliasTargetType=A&aliasTargetValue=s3-website-us-east-1.amazonaws.com&aliasTargetRegion=US East \(N. Virginia\)&aliasTargetType=](#)

Route 53 > Hosted zones > suhaiib.fun

Step 1 Choose routing policy
Step 2 Configure records

Record name Info
To route traffic to a subdomain, enter the subdomain name. For example, to route traffic to blog.example.com, enter blog. If you leave this field blank, the default record name is the name of the domain.
 www .suhaiib.fun

Keep blank to create a record for the root domain.

Record type Info
The DNS type of the record determines the format of the value that Route 53 returns in response to DNS queries.
 A – Routes traffic to an IPv4 address and some AWS resources
Choose when routing traffic to AWS resources for EC2, API Gateway, Amazon VPC, CloudFront, Elastic Beanstalk, ELB, or S3. For example: 192.0.2.44.

Value/Route traffic to Info
The option that you choose determines how Route 53 responds to DNS queries. For most options, you specify where you want to route internet traffic.
 Alias to S3 website endpoint
 US East (N. Virginia)
 s3-website-us-east-1.amazonaws.com

Evaluate target health
Select Yes if you want Route 53 to use this record to respond to DNS queries only if the specified AWS resource is healthy.

[Cancel](#) [Define simple record](#) [Previous](#) [Create records](#)

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

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

Record name suhaiib.fun

Record type A – Routes traffic to an IPv4 address and some AWS resources

Value/Route traffic to Alias to S3 website endpoint
US East (N. Virginia)
s3-website-us-east-1.amazonaws.com

Create records

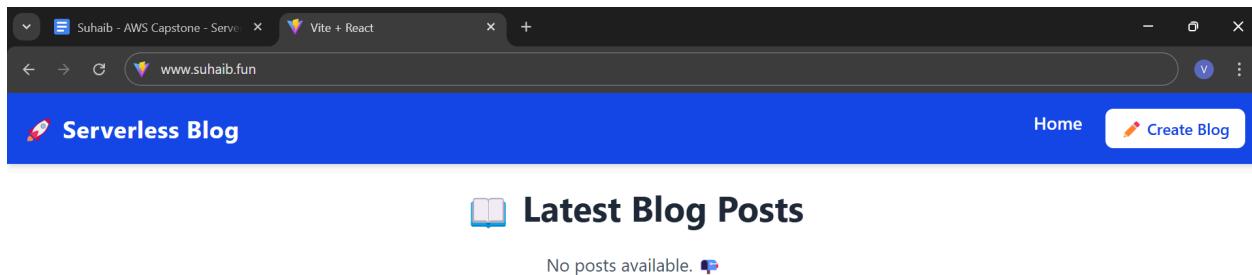
Simple routing records to add to suhaiib.fun

Record name	Type	Value/Route traffic to	TTL (seconds)
www.suhaiib.fun	A	s3-website-us-east-1.amazonaws.com	-
suhaiib.fun	A	s3-website-us-east-1.amazonaws.com	-

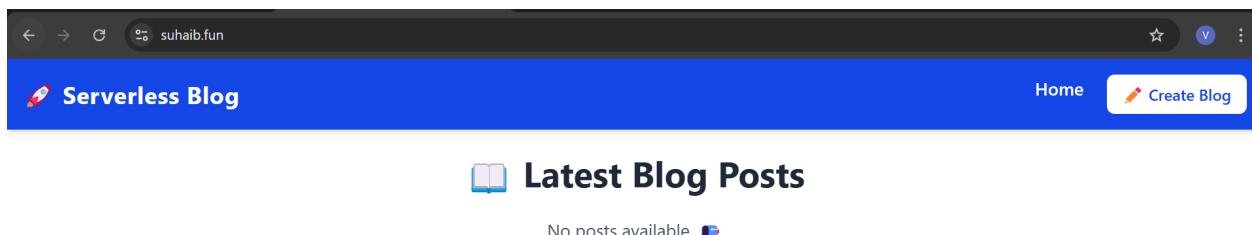
Create records

Verification Tests

✓ Open www.suhaiib.fun → React app should load.

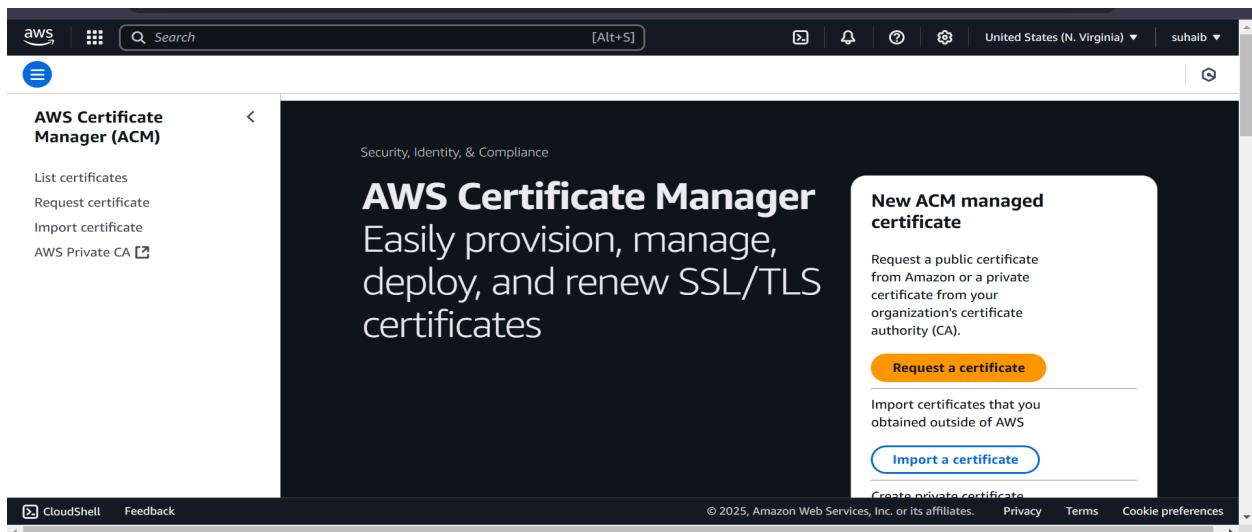


✓ Open suhaiib.fun → It should redirect to www.suhaiib.fun.



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

The screenshot shows the 'Request certificate' step in the AWS Certificate Manager. The 'Certificate type' section is selected, showing two options: 'Request a public certificate' (selected) and 'Request a private certificate'. A note below states that requesting a private certificate requires creating a private CA. At the bottom right are 'Cancel' and 'Next' buttons.

The screenshot shows the 'Request public certificate' step. In the 'Domain names' section, two entries are listed: 'www.suhaib.fun' and 'suhaib.fun', each with a 'Remove' button. Below this is a link to 'Add another name to this certificate'. In the 'Validation method' section, 'DNS validation - recommended' is selected. The bottom navigation bar includes 'CloudShell', 'Feedback', 'Privacy', 'Terms', and 'Cookie preferences'.

The screenshot shows the AWS Certificate Manager console with a certificate named 'da3291d6-2129-468c-9898-893b53d87274'. A prominent blue notification bar at the top indicates: 'Successfully requested certificate with ID da3291d6-2129-468c-9898-893b53d87274. A certificate request with a status of pending validation has been created. Further action is needed to complete the validation and approval of the certificate.' Below this, the certificate details are listed: Identifier (da3291d6-2129-468c-9898-893b53d87274), ARN (arn:aws:acm:us-east-1:640168444369:certificate/da3291d6-2129-468c-9898-893b53d87274), and Type (Amazon Issued). The status is shown as 'Pending validation' with a link to 'Info'. The 'Domains' section lists two domains: suhaib.fun and suhaib.funny. There are 'View certificate' and 'Delete' buttons at the top right of the main card.

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.

The screenshot shows the AWS Certificate Manager console with the same certificate 'da3291d6-2129-468c-9898-893b53d87274'. The status has now changed to 'Issued' (indicated by a green checkmark icon) from 'Pending validation'. The rest of the certificate details remain the same: Identifier (da3291d6-2129-468c-9898-893b53d87274), ARN (arn:aws:acm:us-east-1:640168444369:certificate/da3291d6-2129-468c-9898-893b53d87274), and Type (Amazon Issued). The 'Domains' section still lists 'suhaib.fun' and 'suhaib.funny'. At the bottom of the 'Domains' table, there are buttons for 'Create records in Route 53' and 'Export to CSV'.

2.4 Configuring CloudFront

Step 1: Create CloudFront Distribution for WWW

- **Origin Domain:** Use the **Static Website Hosting URL** from the [www.suhail.fun](#) S3 bucket.

The screenshot shows the 'Create distribution' wizard in the AWS CloudFront console. The 'Origin' configuration step is selected. In the 'Origin domain' section, the URL 'www.suhail.fun.s3-website-us-east-1.amazonaws.com' is entered into the search bar. Under 'Protocol', 'HTTP only' is selected. In the 'HTTP port' section, the port number '80' is specified. The 'Origin path - optional' field is empty. The bottom navigation bar includes links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

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

The screenshot shows the 'Create distribution' wizard in the AWS CloudFront console. The 'Settings' configuration step is selected. Under 'Compress objects automatically', 'Yes' is selected. In the 'Viewer' section, 'Redirect HTTP to HTTPS' is selected under 'Viewer protocol policy'. Under 'Allowed HTTP methods', 'GET, HEAD' is selected. Under 'Restrict viewer access', 'No' is selected. The bottom navigation bar includes links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

Alternate domain name (CNAME) - optional
Add the custom domain names that you use in URLs for the files served by this distribution.

www.suhai.fun Remove

[Add item](#)

To add a list of alternative domain names, use the [bulk editor](#).

Custom SSL certificate - optional
Associate a certificate from AWS Certificate Manager. The certificate must be in the US East (N. Virginia) Region (us-east-1).

www.suhai.fun (da3291d6-2129-468c-9898-893b53d87274) @ Request certificate

www.suhai.fun [Request certificate](#)

Legacy clients support - \$600/month prorated charge applies. Most customers do not need this.
CloudFront allocates dedicated IP addresses at each CloudFront edge location to serve your content over HTTPS.

Enabled

Security policy
The security policy determines the SSL or TLS protocol and the specific ciphers that CloudFront uses for HTTPS connections with viewers (clients).

Step 2: Create CloudFront Distribution for Non-WWW

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

ID	Description	Type	Domain name	Alternate domain	Origins	Status	Last modified
E2EJO67BUS5ZJC	-	Production	d10zdf9p2vi...	suhai.fun	suhai.fun.s3-website.com	Enabled	Deploying
E2EXX0XYAH1DXJ	-	Production	d1qafawqvg...	www.suhai.fun	www.suhai.fun.s3-website.com	Enabled	Deploying

Verification Tests

- ✓ Open the CloudFront **domain URL** for **www.suhai.fun** → The site should load over **HTTPS**.



✓ Open the CloudFront **domain URL** for suhai.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

```

Distributions (1) [Info](#)

ID	Type	Domain Name	Alternate Domains	Origins	Status	Last modified
X6LIR7X	CloudFront	d3hng88c...	-	my-serverless-b	Enabled	Deploying...

Stacks (3)

Stack name	Status	Created time	Description
CloudFrontStack	CREATE_COMPLETE	2025-02-27 11:51:52 UTC+0530	-

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

Distributions (1) [Info](#)

ID	Type	Domain Name	Alternate Domains	Origins	Status
EYB6HFX6LIR7X	CloudFront	d3hng88c...	-	my-serverless-b	Enabled

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

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

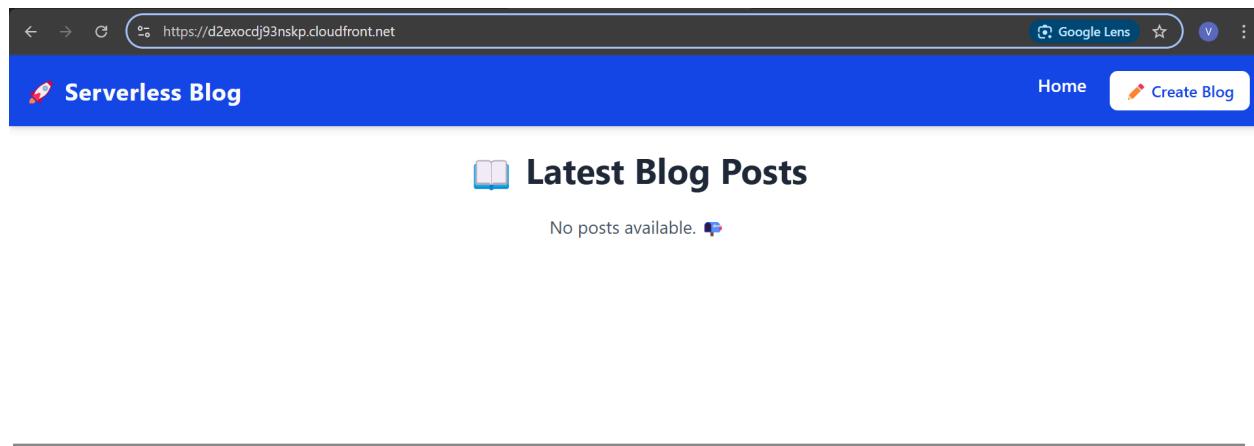
```
C:\Users\Vadakathi Suhaib>aws cloudformation describe-stacks --stack-name CloudFrontStack --query "Stacks[0].Outputs"
[
    {
        "OutputKey": "CloudFrontDomain",
        "OutputValue": "d3hng88c8fe6kp.cloudfront.net",
        "Description": "CloudFront Distribution Domain Name"
    }
]
```

```

        "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.suhaiib.fun](#)**

Record Name	Type	Routing Policy	Target
www.suhaiib.fun	A	Simple	d1qafawqvqgmf8e.cloudfront.net
suhaiib.fun	NS	Simple	
suhaiib.fun	SOA	Simple	
_9310216...	CNAME	Simple	
www.suhaiib.fun	A	Simple	d1qafawqvqgmf8e.cloudfront.net
_69e8aa6...	CNAME	Simple	

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

The screenshot shows the AWS Route 53 console with the URL <https://us-east-1.console.aws.amazon.com/route53/v2/hostedzones?region=us-east-1#ListRecordSets/Z081120814W371JJUT9S>. The left sidebar shows navigation options like Dashboard, Hosted zones, and Traffic flow. The main area shows a table of records for the hosted zone suhaib.fun. One record is selected, showing it points to a CloudFront distribution.

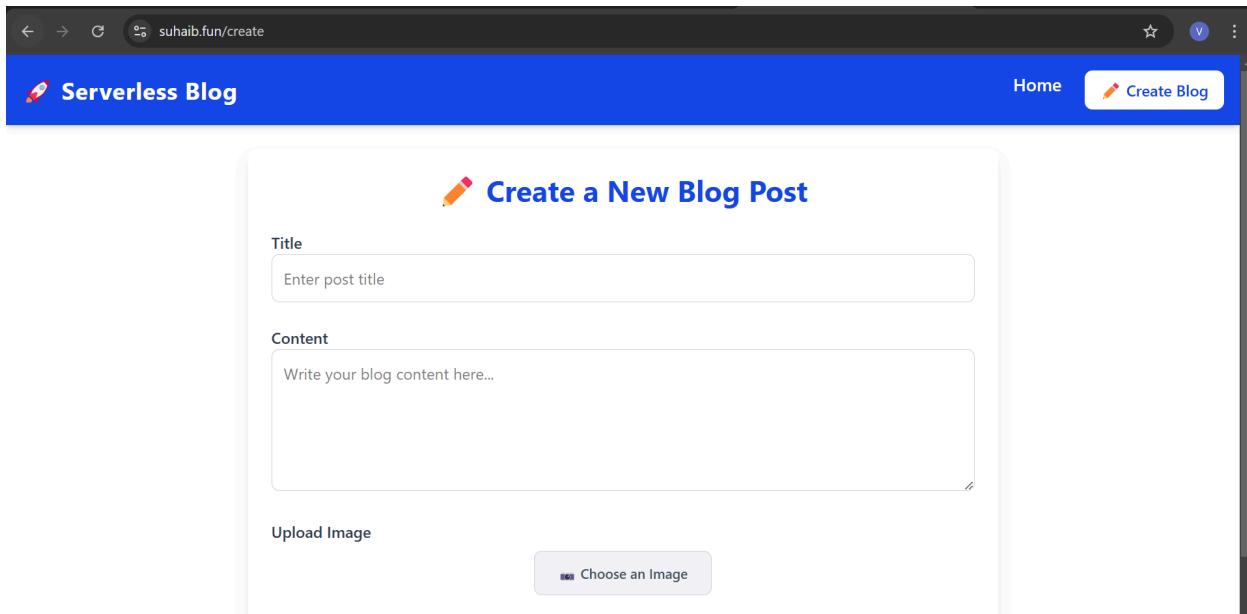
Record Name	Type	Routing Policy	Alias
<input checked="" type="checkbox"/> suhaib.fun	A	Simple	Yes
<input type="checkbox"/> suhaib.fun	NS	Simple	No
<input type="checkbox"/> suhaib.fun	SOA	Simple	No
<input type="checkbox"/> _9310216...	CNAME	Simple	No

Final Verification Tests

- ✓ Open <https://www.suhaib.fun> → The React application should load securely.

The screenshot shows a browser window with two tabs: "Suhaib - AWS Capstone - Server" and "Vite + React". The active tab shows the URL <https://www.suhaib.fun>. The page content includes a header "Serverless Blog" and a section titled "Latest Blog Posts" with the message "No posts available."

- ✓ 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.

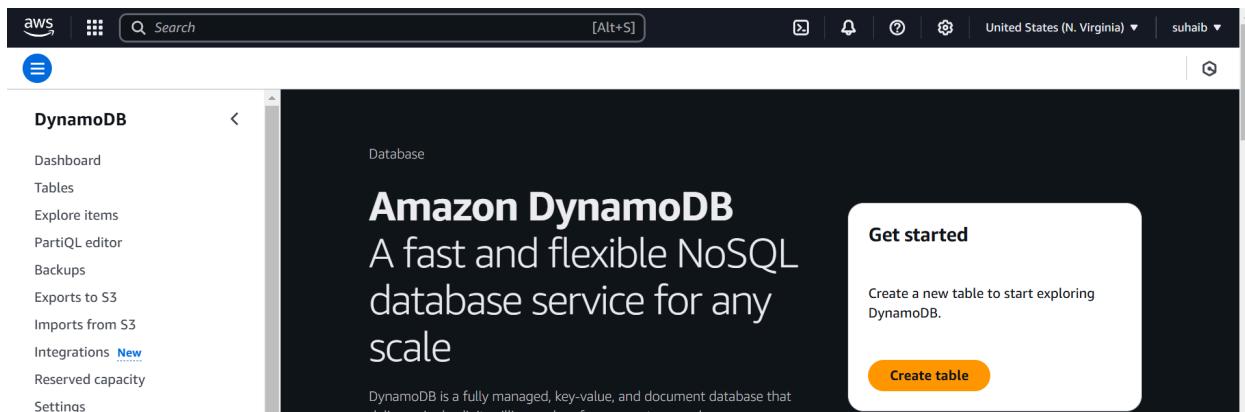
✓ Step 4: Setup Backend for the application using Lambda, API Gateway and DynamoDB

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**.

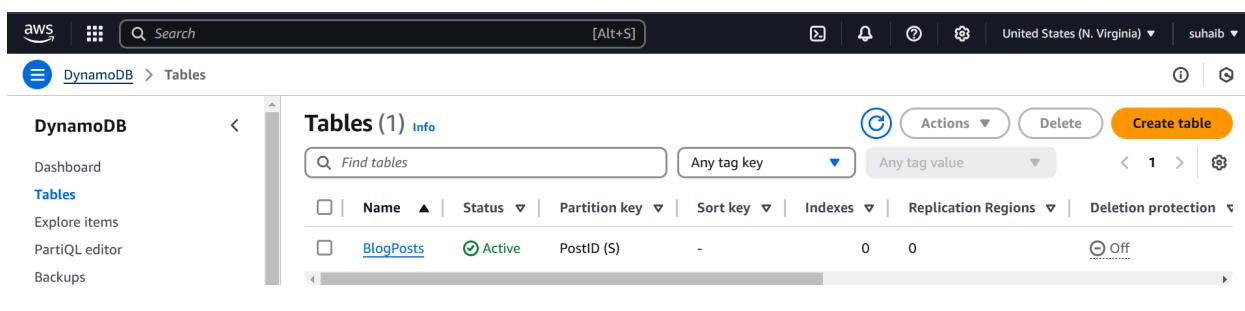
1.1 Create the DynamoDB Table

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



The screenshot shows the Amazon DynamoDB 'Create table' wizard. On the left, there's a sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area has a dark background with white text. It says 'Amazon DynamoDB' and 'A fast and flexible NoSQL database service for any scale'. Below that, it says 'DynamoDB is a fully managed, key-value, and document database that delivers single-digit millisecond response times at any scale'. A 'Get started' box contains the text 'Create a new table to start exploring DynamoDB.' with a 'Create table' button.

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



The screenshot shows the 'Tables' list in the Amazon DynamoDB console. The sidebar is identical to the previous screenshot. The main area shows a table with one item: 'BlogPosts'. The table has columns for Name (BlogPosts), Status (Active), Partition key (PostID (\$)), Sort key (-), Indexes (0), Replication Regions (0), and Deletion protection (Off). There are buttons for Actions, Delete, and Create table.

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
BlogPosts	Active	PostID (\$)	-	0	0	Off

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**.

The screenshot shows the 'Topics' page in the Amazon SNS section of the AWS Management Console. On the left, there's a navigation sidebar with 'Amazon SNS' selected, along with links for 'Dashboard', 'Topics', 'Subscriptions', and 'Mobile'. The main area has a header 'Topics (1)' with buttons for 'Edit', 'Delete', 'Publish message', and 'Create topic'. Below is a search bar and a table with columns 'Name' and 'Type'. One row is shown, labeled 'ImageModerationAlerts' with 'Standard' under 'Type'. At the bottom right of the table, the ARN is partially visible: 'arn:aws:sns:us-east-1:640168444369:I...'. The status bar at the bottom indicates 'United States (N. Virginia)' and the user 'suhail'.

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.

The screenshot shows the 'Subscription' details for the 'ImageModerationAlerts' topic. The left sidebar shows 'Amazon SNS' selected, along with 'Topics' and 'Subscriptions'. The main area shows a 'Subscription' card for 'Subscription: 17ef5bad-8cc3-4e6a-982d-9c3b2b3c0840'. The card has sections for 'Details', 'Status' (Confirmed), and 'Protocol' (EMAIL). It lists the ARN, Endpoint (suhaimdvv@gmail.com), and Topic (ImageModerationAlerts). The status bar at the bottom indicates 'United States (N. Virginia)' and the user 'suhail'.

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.

The screenshot shows the AWS SQS Queues page. At the top, there's a search bar and navigation links for 'Amazon SQS' and 'Queues'. Below the header, a table lists one queue: 'ImageModQueue' (Standard type, created on 2025-02-28T16:33+05:30). A context menu is open over this queue, with 'Subscribe to Amazon SNS topic' highlighted. Other options in the menu include 'Configure Lambda function trigger' and 'Purge'.

6. Choose the SNS topic and click save.

The screenshot shows the 'Subscribe to Amazon SNS topic' dialog. It includes a section for 'Amazon SNS topic' with a note about allowing the queue to receive messages from an SNS topic. A dropdown menu shows 'arn:aws:sns:us-east-1:640168444369:ImageModerationAlerts' selected. At the bottom are 'Cancel' and 'Save' buttons.

7. Verify SQS subscription.

The screenshot shows the AWS SNS Subscriptions page. The left sidebar has 'Amazon SNS' selected, with 'Subscriptions' highlighted. The main area displays a table of two subscriptions:

ID	Endpoint	Status	Protocol	Topic
17ef5bad-8cc3-4e6...	suhaiibmdv@gmail.com	Confirmed	EMAIL	ImageModerationAI...
2e6cdce1-68b1-4d...	arn:aws:sqs:us-east-...	Confirmed	SQS	ImageModerationAI...

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 Suhail>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**: **suhaiib-blog-images**.
4. Choose **Region**: **us-east-1**.
5. **Block Public Access**: Keep it **enabled**.
6. Click **Create Bucket**.

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with 'Access Points' and links for Object Lambda Access Points, Multi-Region Access Points, Batch Operations, and IAM Access Analyzer for S3. Below that, it says 'Block Public Access settings for this account'. The main area is titled 'General purpose buckets (3)' with a 'Create bucket' button. It shows a table with columns: Name, AWS Region, IAM Access Analyzer, and Creation date. The table contains one row for the newly created bucket:

Name	AWS Region	IAM Access Analyzer	Creation date
suhaiib-blog-images	US East (N. Virginia) us-east-1	View analyzer for us-east-1	February 28, 2025, 11:31:03 (UTC+05:30)

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**.

REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Works with the following:
Lambda, HTTP, AWS Services

[Import](#) [Build](#)

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

The screenshot shows the AWS API Gateway interface. On the left, there's a sidebar with 'API Gateway' and 'APIs' selected. The main area is titled 'APIs (1/1)' and shows a table with one row. The row contains the following information:

Name	Description	ID	Protocol	API endpoint type	Created
myBlogApi		5gf9bw2b43	REST	Regional	2025-02-28

At the top right of the table are buttons for 'Delete' and '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](#).

The screenshot shows the AWS API Gateway interface. On the left, there's a sidebar with 'API Gateway' and 'APIs' selected. Under 'APIs', 'myBlogApi' is expanded, showing 'Resources'. The main area is titled 'Resources' and shows a list of resources:

- [Create resource](#)
- [/](#) (selected)
- [/create-post](#)
 - OPTIONS
 - POST
- [/get-posts](#)
 - GET
 - OPTIONS
- [/get-upload-url](#)
 - GET
 - OPTIONS

To the right of the resources, there are two panels: 'Resource details' and 'Methods (0)'. The 'Resource details' panel shows the path '/' and a resource ID 'asxjryxpdi'. The 'Methods (0)' panel shows a table with columns for 'Method type', 'Integration type', 'Authorization', and 'API key'. Both panels have buttons for 'Update documentation', 'Enable CORS', 'Delete', and 'Create method'.

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:

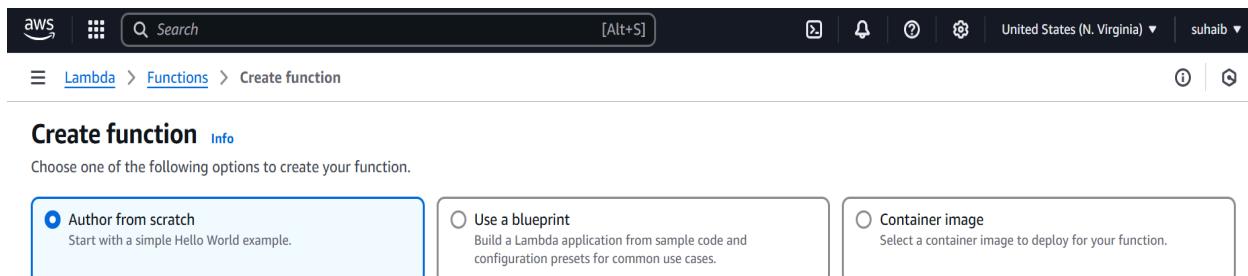
- **Navigate to IAM in the AWS Console.**
- **Create a new role** with "AWS service" as the trusted entity and select **Lambda**.
- **Attach policies** (or create a custom policy) with permissions for:
 - **S3**: Allow `s3:GetObject` on your image bucket.
 - **Rekognition**: Allow `rekognition:DetectModerationLabels`.
 - **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:
 - o **Function name:** [CreatePost](#)
 - o **Runtime:** [Python 3.9](#)
 - o **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
```

```



```

```

# ♦ 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)}.\nThe 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": imageUrl,
        "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

Environment variables (3)	
The environment variables below are encrypted at rest with the default Lambda service key.	
<input type="text"/> Find environment variables	
Key	Value
BLOG_TABLE	BlogPosts
IMAGE_BUCKET	suhaiib-blog-images
SNS_TOPIC_ARN	arn:aws:sns:us-east-1:640168444369:ImageModerationAlerts

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

The screenshot shows the 'Environment variables' section of a Lambda function configuration. On the left, a sidebar lists 'General configuration', 'Triggers', 'Permissions', 'Destinations', 'Function URL', and 'Environment variables'. The 'Environment variables' option is selected and highlighted in blue. The main panel title is 'Environment variables (1)'. A note states: 'The environment variables below are encrypted at rest with the default Lambda service key.' Below this is a search bar labeled 'Find environment variables'. A table lists one variable: 'Key' is 'BLOG_TABLE' and 'Value' is 'BlogPosts'. There is also an 'Edit' button in the top right corner.

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

Environment variables (1)

The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
IMAGE_BUCKET	suhaiib-blog-images

Edit

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

The screenshot shows the 'Environment variables' section of a Lambda function configuration. On the left, there is a sidebar with tabs for General configuration, Triggers, Permissions, Destinations, Function URL, and Environment variables. The Environment variables tab is selected. On the right, there is a table with two rows. The first row has 'Key' as 'IMAGE_BUCKET' and 'Value' as 'suhaiib-blog-images'. The second row has 'Key' as 'SNS_TOPIC_ARN' and 'Value' as 'arn:aws:sns:us-east-1:640168444369:ImageModerationAlerts'. There is also a small 'Edit' button in the top right corner of the environment variables panel.

Environment variables (2)	
The environment variables below are encrypted at rest with the default Lambda service key.	
<input type="text" value="Find environment variables"/>	
Key	Value
IMAGE_BUCKET	suhaiib-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** (`suhaiib-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:**
 - o `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:**
 - o `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**":
 - o Add a **200** response.
 - o Under "Response Headers," add:
 - `Access-Control-Allow-Origin`
 - `Access-Control-Allow-Methods`
 - `Access-Control-Allow-Headers`
6. Click "**Integration Response**":
 - o Expand **200** response, go to "Header Mapping".
 - o 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://suhail-blog-images.s3.amazonaws.com/uploads/test.jpg  
\"}"
```

Fetch Posts

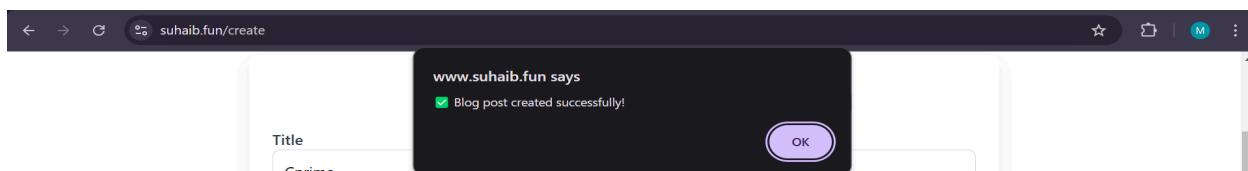
```
curl "https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-posts"
```

```
C:\Users\Vadakathi Suhaib\Downloads>curl "https://5gf9bw2b43.execute-api.us-east-1.amazonaws.com/prod/get-posts"
{"statusCode": 200, "headers": {"Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "GET", "Access-Control-Allow-Headers": "Content-Type"}, "body": ["[]"]}
C:\Users\Vadakathi Suhaib\Downloads>
```

6.2 Upload a Normal Image

- Try uploading a **safe** image.

The screenshot shows a web browser window with the URL `suhai.fun/create`. The page title is "Create a New Blog Post". The form has fields for "Title" (containing "Cprime") and "Content" (containing "Cprime new logo"). Below the content area is an "Upload Image" section with a "Choose an Image" button. A message below the button says "Selected: cprime-office.jpg". At the bottom is a blue "Publish Post" button with a rocket icon.



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

The screenshot shows the AWS DynamoDB console. The left sidebar shows "DynamoDB" with options like "Dashboard", "Tables", "Explore items", "PartiQL editor", "Backups", "Exports to S3", "Imports from S3", "Integrations", "Reserved capacity", and "Settings". Under "Tables", "BlogPosts" is selected. The main area shows a "Filters" section with "Run" and "Reset" buttons. A green notification bar at the bottom says "Completed. Read capacity units consumed: 2". Below it, a table titled "Items returned (1)" shows one item with columns: "PostID (String)", "content", "createdAt", and "imageUrl". The item data is: PostID: "ce056e62-5f22-4ad8...", content: "Cprime new...", createdAt: "1740773100", imageUrl: "https://suh...".

The screenshot shows a web browser window with the URL 'suhai.fun'. The page title is 'Serverless Blog'. On the right, there are 'Home' and 'Create Blog' buttons. Below the title, a section titled 'Latest Blog Posts' is displayed. The first post is titled 'Cprime' and features a thumbnail image of a modern office hallway with wooden walls and a grey wall with the 'cprime' logo. The post content includes the text 'Cprime new logo...' and a 'Read more →' link.

Latest Blog Posts



Cprime

Cprime new logo...

[Read more →](#)

The screenshot shows the AWS S3 console. The path in the top navigation bar is 'Amazon S3 > Buckets > suhai-blog-images > uploads/'. The main view shows a single object named 'image_1740773065.jpg' which is a jpg file uploaded on March 1, 2025, at 01:34:58 (UTC+05:30). The file size is 4.5 MB and it is stored in the Standard storage class. There are buttons for 'Copy S3 URI', 'Actions', 'Create folder', and 'Upload'.

6.3 Upload an Inappropriate Image

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

The screenshot shows a 'Create a New Blog Post' form. The 'Title' field contains 'Guns' and the 'Content' field contains 'AK 47 with its inventor'. Under 'Upload Image', there is a file input field with 'Selected: ak47.jpg' and a 'Choose an Image' button. A large blue 'Publish Post' button is at the bottom.

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

← → ⌛ suhaib.fun/create

 Serverless Blog

Home 

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>
to me ▾ 1:41AM (0 minutes ago) ⭐ 😊 ← ⋮

⚠ Warning: Image flagged for Weapons, Violence, Weapon Violence, Graphic Violence.
Image URL: https://suhaiib-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=suhaiibmdv@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>

aws Search [Alt+S] United States (N. Virginia) suhaiib ▾

Amazon SQS > Queues > ImageModQueue > Send and receive messages

6 30 10 0.2 receives/second

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://suhaiib-blog-images.s3.amazonaws.com/uploads/image_1740773425.jpg",

Done

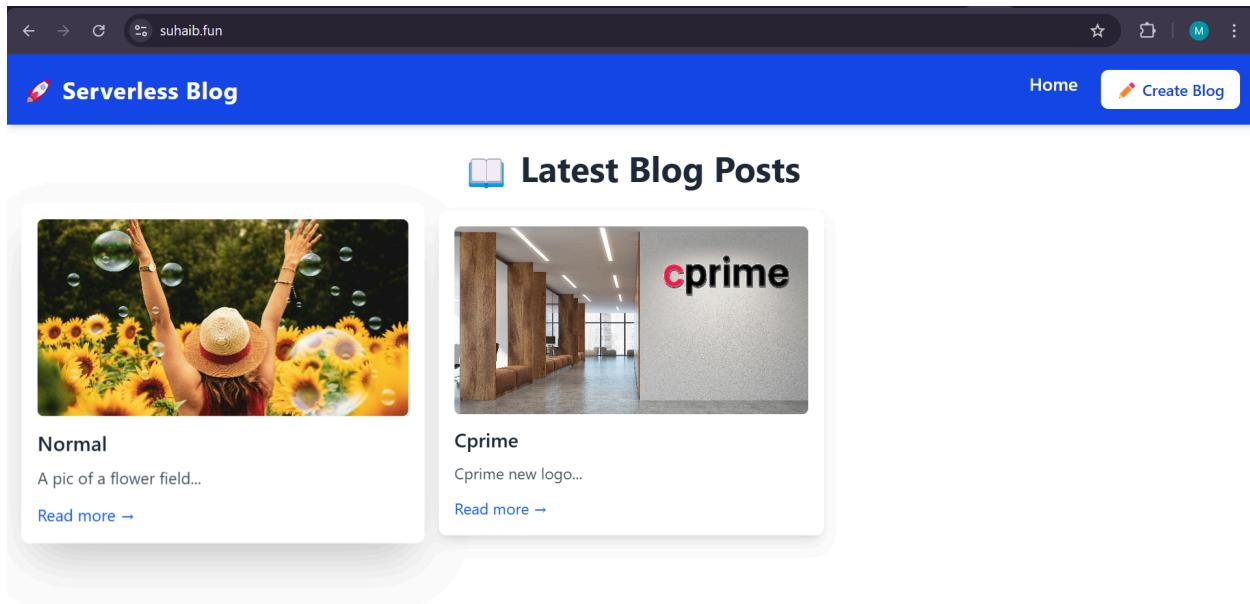
2025-05-01 01:41:05.50 1.12 KB



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.

A screenshot of the AWS CloudWatch Log Groups interface. The left sidebar shows navigation options like "CloudWatch", "Log groups", and "Logs". The main area displays log entries for a specific log group: "/aws/lambda/CreatePost" on "2025/02/28/[...LATEST]9ba5e10b1dd947e6ab2bff72bd3e402b".

Timestamp	Message
2025-02-28T20:19:06.589Z	INIT_START Runtime Version: python:3.9.v64 Runtime Version ARN: arn:aws:lambda:us-east-1::ru...
2025-02-28T20:19:07.141Z	START RequestId: a8fdd64a-c665-40fd-a8ac-0eb41a500a33 Version: \$LATEST
2025-02-28T20:19:07.141Z	Received event: {
2025-02-28T20:19:07.141Z	"title": "Normal",
2025-02-28T20:19:07.141Z	"content": "A pic of a flower field",
2025-02-28T20:19:07.141Z	"imageUrl": "https://suhaib-blog-images.s3.amazonaws.com/uploads/image_1740773939.jpg"
2025-02-28T20:19:07.141Z	}
2025-02-28T20:19:08.152Z	END RequestId: a8fdd64a-c665-40fd-a8ac-0eb41a500a33
2025-02-28T20:19:08.152Z	REPORT RequestId: a8fdd64a-c665-40fd-a8ac-0eb41a500a33 Duration: 1011.69 ms Billed Duration:...

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

CloudWatch > Log groups > /aws/lambda/CreatePost > 2025/02/28/[...LATEST]92dc3bc9904149ac8d4b6b0b2eca5731

Timestamp	Message
2025-02-28T20:11:18.094Z	No older events at this moment. Retry
2025-02-28T20:11:18.658Z	INIT_START Runtime Version: python:3.9.v64 Runtime Version ARN: arn:aws:lambda:us-east-1::...
2025-02-28T20:11:18.658Z	START RequestId: ed4f8691-88ad-4508-b5fc-2b6d859aa47f Version: \$LATEST
2025-02-28T20:11:18.658Z	Received event: {
2025-02-28T20:11:18.658Z	"title": "Guns",
2025-02-28T20:11:18.658Z	"content": "AK 47 with its inventor",
2025-02-28T20:11:18.658Z	"imageUrl": "https://suhail-blog-images.s3.amazonaws.com/uploads/image_1740773425.jpg"
2025-02-28T20:11:19.063Z	}
2025-02-28T20:11:19.063Z	Image flagged for: ['Weapons', 'Violence', 'Weapon Violence', 'Graphic Violence']
2025-02-28T20:11:19.486Z	Flagged image uploads/image_1740773425.jpg deleted from S3
2025-02-28T20:11:19.488Z	END RequestId: ed4f8691-88ad-4508-b5fc-2b6d859aa47f
2025-02-28T20:11:19.488Z	REPORT RequestId: ed4f8691-88ad-4508-b5fc-2b6d859aa47f Duration: 830.05 ms Billed Duration: ...

7.2 Confirm DynamoDB and SNS

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

Completed. Read capacity units consumed: 2

Items returned (2)							
	Actions	Create item					
<input type="checkbox"/> PostID (String)	▼	content	▼	createdAt	▼	imageUrl	▼
<input type="checkbox"/> fcd318f7-91e7-4aaa-...	A pic of a fl...	1740773947	https://suh...				
<input type="checkbox"/> ce056e62-5f22-4ad8-...	Cprime new...	1740773100	https://suh...				

Completed. Read capacity units consumed: 2 of 149

Items returned (2)

	Actions	Create item					
<input type="checkbox"/> PostID (String)	▼	content	▼	createdAt	▼	imageUrl	▼
<input type="checkbox"/> fcd318f7-91e7-4aaa-...	A pic of a fl...	1740773947	https://suh...				
<input type="checkbox"/> ce056e62-5f22-4ad8-...	Cprime new...	1740773100	https://suh...				

Gmail

Inbox 11

Compose

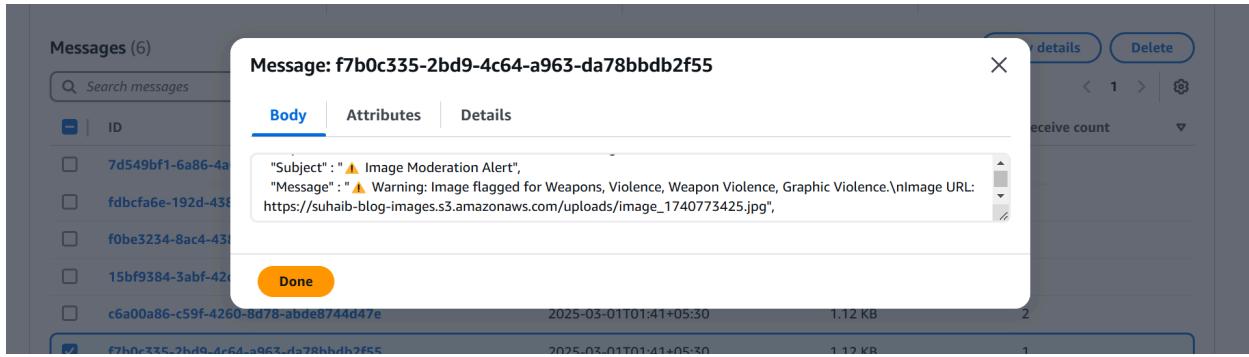
Search mail

AWS Notifications <no-reply@sns.amazonaws.com> to me 1:41AM (11 minutes ago)

Warning: Image flagged for Weapons, Violence, Weapon Violence, Graphic Violence.
Image URL: https://suhail-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-8cc3-4eba-982d-9c3b2b3c0840&Endpoint=suhailbmdv@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>



✓ 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
clouformation/
└── 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**
 - o Click **Create Build Project**.
2. **Enter Project Details:**

- **Project name:** BlogFrontendBuild

Project configuration

Project name

BlogFrontendBuild

A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and _.

- **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.

Copilot

Selected 3 repositories.

- suhaiib-md/Serverless_Blog_Platform_with_Content...
- suhaiib-md/AWS_Serverless_Blog_Backend_CI-CD
- suhaiib-md/AWS_Serverless_Blog_Frontend_CI-CD

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

Connect to GitHub

GitHub connection settings [Info](#)

Connection name

Serverless-Blog

GitHub Apps

GitHub Apps create a link for your connection with GitHub. To start, install a new app and save this connection.

61445899

Manage default source credential

Source Provider

Credential type
 GitHub App
 Connect project to GitHub using an AWS managed GitHub App
 Personal access token
 Connect project to GitHub using a personal access token
 OAuth app
 Connect project to GitHub using an OAuth app

Connection
 You can create a new GitHub connection by using an AWS managed GitHub App

Source 1 - Primary

Source provider

Credential
 Your account is successfully connected by using an AWS managed GitHub App. [Manage account credentials](#).

Use override credentials for this project only

Repository
 Repository in my GitHub account
 Public repository
 GitHub scoped webhook

Repository

Source 1 - Primary

Source provider

Credential
 Your account is successfully connected by using an AWS managed GitHub App. [Manage account credentials](#).

Use override credentials for this project only

Repository
 Repository in my GitHub account
 Public repository
 GitHub scoped webhook

Repository

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.suhail.fun/ --delete

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

```

Buildspec

Build specifications

Insert build commands
Store build commands as build project configuration

Use a buildspec file
Store build commands in a YAML-formatted buildspec file

Buildspec name - optional
By default, CodeBuild looks for a file named buildspec.yml in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, buildspec-two.yml or configuration/buildspec.yml).

frontend/buildspec.yml

5. Artifacts:

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

Artifact 1 - Primary

Type

No artifacts

You might choose no artifacts if you are running tests or pushing a Docker image to Amazon ECR.

Project created
You have successfully created the following project: BlogFrontendBuild

Developer Tools > CodeBuild > Build projects > BlogFrontendBuild

BlogFrontendBuild

Actions ▾ Create trigger Edit Clone Debug build Start build with overrides

Configuration

Source provider	Primary repository	Artifacts upload location	Service role
GitHub	suhai...md/AWS_Serverless_Blog_Frontend_CI-CD	-	arn:aws:iam::640168444369:role/service-role/codebuild-BlogFrontendBuild-service-role

6. Run build:

The screenshot shows the AWS CodeBuild console. On the left, there's a sidebar with options like Source, Artifacts, Build, Build project, Settings, Build history, Report groups, Report history, Compute fleets, and Account metrics. The main area displays a green banner at the top stating "Build started" and "You have successfully started the following build: BlogFrontendBuild:b42028e1-1486-4dbe-8818-e0eda34d47b9". Below this, the build name is shown again. There are two buttons: "Stop build" and "Retry build". Under the "Build status" section, it shows the Status as "Succeeded", Initiator as "root", and Build ARN as "arn:aws:codebuild:us-east-1:640168444369:build/BlogFrontendBuild:b42028e1-1486-4dbe-8818-e0eda34d47b9". It also lists the Resolved source version as "ae3c8fe8edaec0a50d28f2a910520dd59 0e7c269", Start time as "Mar 1, 2025 12:57 PM (UTC+5:30)", and End time as "Mar 1, 2025 12:58 PM (UTC+5:30)".

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`

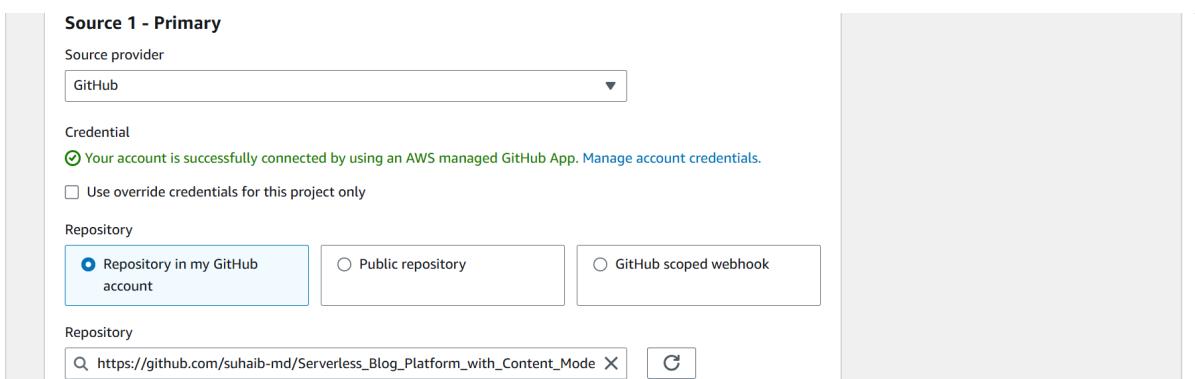
The screenshot shows the "Create build project" page. At the top, there's a breadcrumb navigation: Developer Tools > CodeBuild > Build projects > Create build project. The main area has a title "Create build project" and a "Project configuration" section. In the "Project name" field, the value "BlogBackendBuild" is entered. A note below the field states: "A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and _." There are tabs for "Project configuration", "Source", "Artifacts", "Environment", and "Lifecycle rules".

- **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 **backend repository**.
- Select the **main branch**.



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

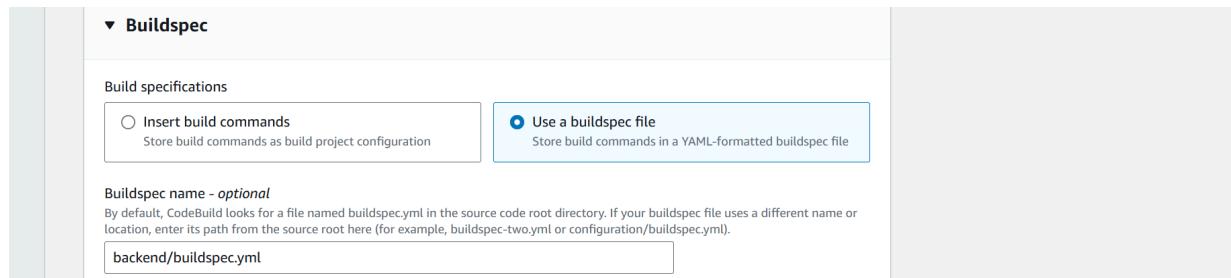
```
version: 0.2

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
```

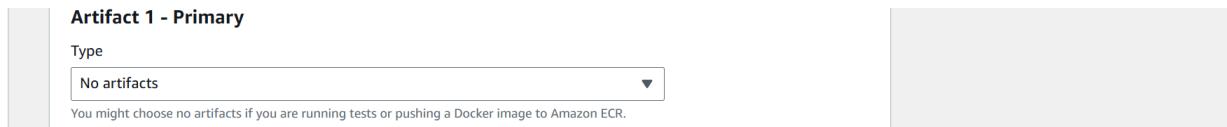
```
directory
```

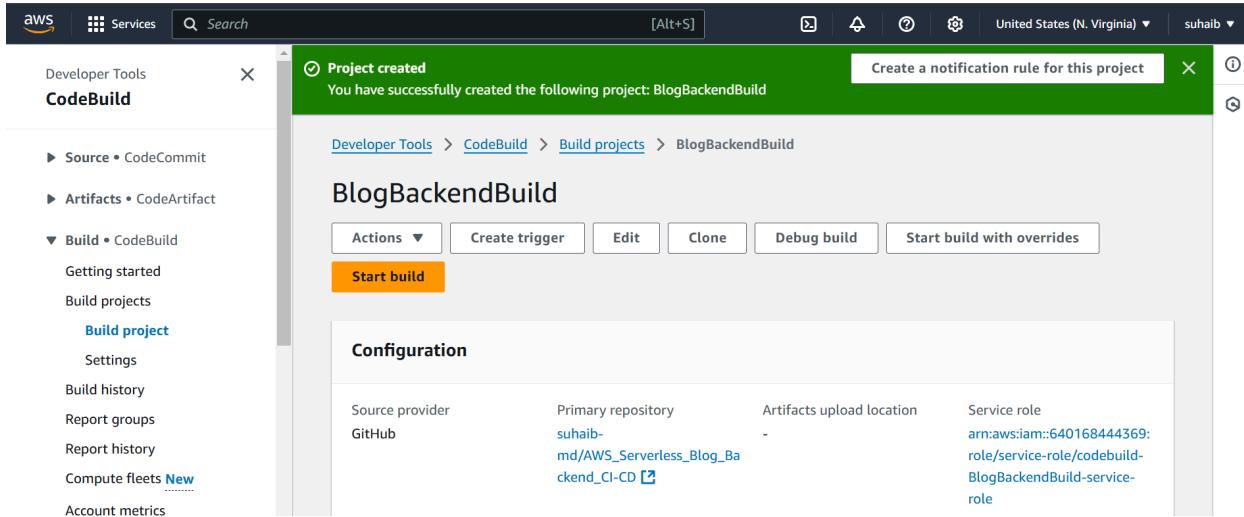
```
post_build:  
  commands:  
    - ls -lah $CODEBUILD_SRC_DIR # List files to confirm deploy.zip exists  
    - test -f $CODEBUILD_SRC_DIR/deploy.zip && echo "deploy.zip found" ||  
echo "deploy.zip MISSING"  
    - aws lambda update-function-code --function-name CreatePost --zip-file  
fileb://$CODEBUILD_SRC_DIR/deploy.zip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



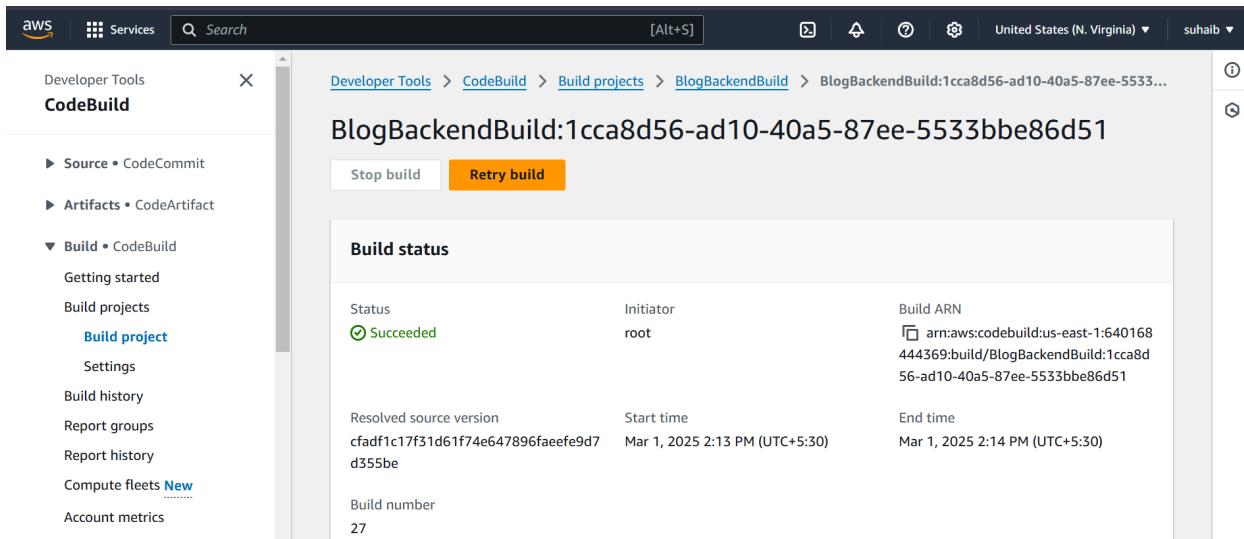
- **Artifacts:**

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





- Click Start Build:



ISSUES:

- Permission for updating lambda functions

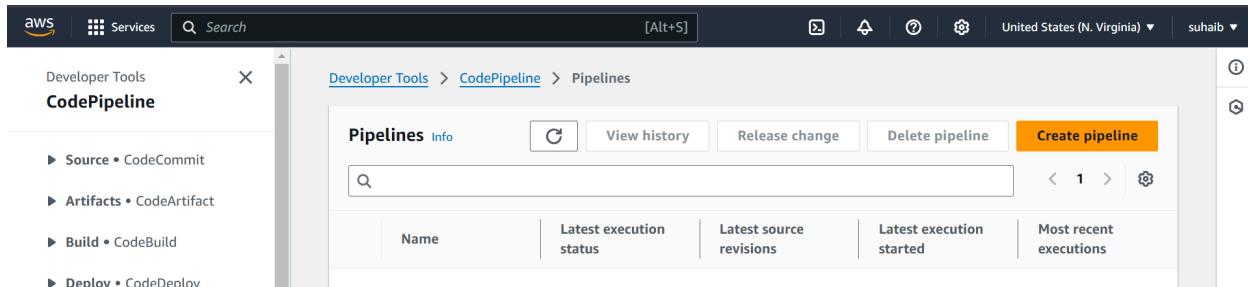
6. Create the CodePipeline

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**

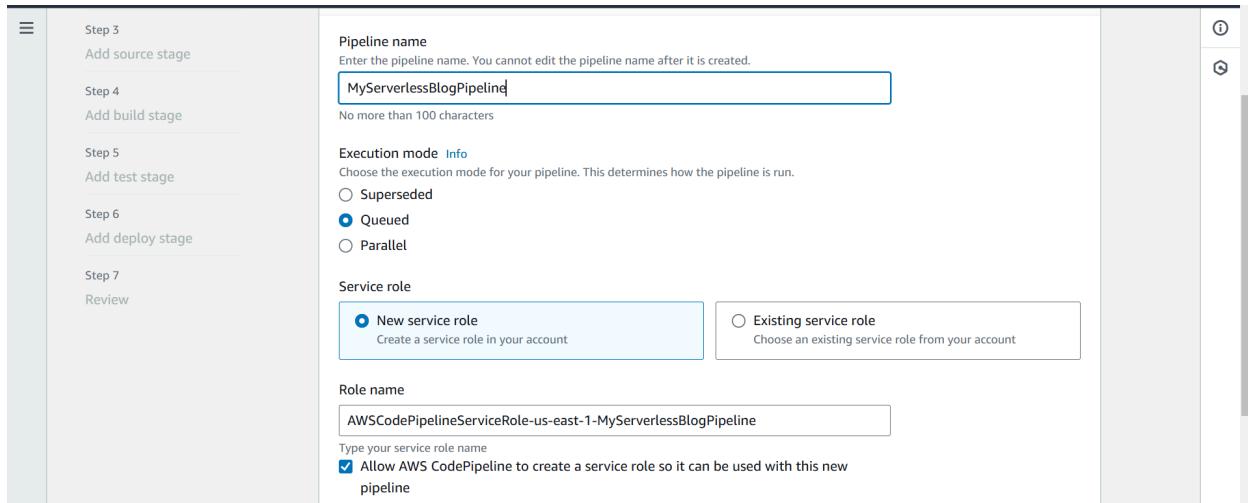
1. Sign in to the **AWS Management Console**.
2. Navigate to **AWS CodePipeline**.
3. Click **Create pipeline**.



The screenshot shows the AWS Management Console with the 'CodePipeline' service selected under 'Developer Tools'. The main pane displays a table of existing pipelines, with a prominent orange 'Create pipeline' button at the top right. The left sidebar lists various AWS services: Source (CodeCommit), Artifacts (CodeArtifact), Build (CodeBuild), and Deploy (CodeDeploy).

◆ **Step 2: Configure Pipeline Settings**

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



This screenshot shows the 'Create Pipeline' wizard, Step 2: Set Pipeline Details. On the left, a sidebar lists steps: Step 3 (Add source stage), Step 4 (Add build stage), Step 5 (Add test stage), Step 6 (Add deploy stage), and Step 7 (Review). The main panel contains fields for 'Pipeline name' (MyServerlessBlogPipeline), 'Execution mode' (set to 'Queued'), and 'Service role' (selected 'New service role' with role name 'AWSCodePipelineServiceRole-us-east-1-MyServerlessBlogPipeline'). A checkbox 'Allow AWS CodePipeline to create a service role so it can be used with this new pipeline' is checked.

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

Choose pipeline settings

Step 3
Add source stage

Step 4
Add build stage

Step 5
Add test stage

Step 6
Add deploy stage

Step 7
Review

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (via GitHub App)

Connection
Choose an existing connection that you have already configured, or create a new one and then return to this task.

arn:aws:codeconnections:us-east-1:640168444369:connection/ca90... X or Connect to GitHub

Repository name
Choose a repository in your GitHub account.

suaib-md/Serverless_Blog_Platform_with_Content_Moderation_AWS X

You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.

Default branch
Default branch will be used only when pipeline execution starts from a different source or manually started.

main X

◆ 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**

Choose pipeline settings

Step 3
Add source stage

Step 4
Add build stage

Step 5
Add test stage

Step 6
Add deploy stage

Step 7
Review

Build - optional

Build provider
Choose the tool you want to use to run build commands and specify artifacts for your build action.

Commands Other build providers

AWS CodeBuild

Project name
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.

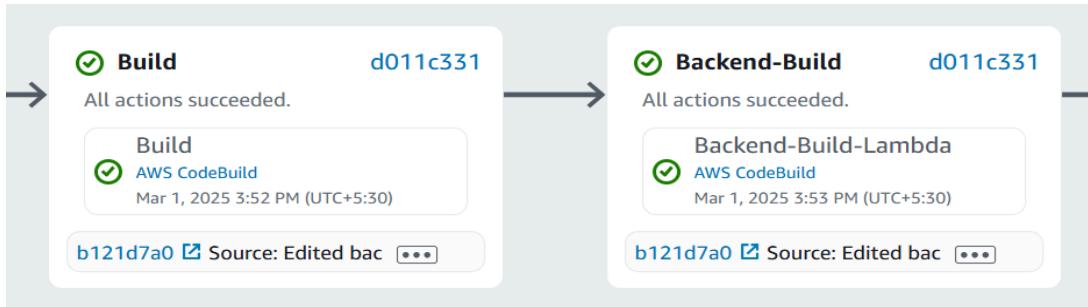
BlogFrontendBuild X or Create project ↗

Environment variables - optional
Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. Learn more ↗

➤ 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**

The screenshot shows the configuration of a Lambda action. The 'Action name' is 'Backend-Deploy-2'. The 'Action provider' is 'AWS Lambda'. The 'Region' is 'United States (N. Virginia)'. Under 'Input artifacts', there is a selected item 'BackendArtifact' with a note 'Defined by: Backend-Build-Lambda'. In the 'Function name' section, the value 'GetUploadUrl' is entered. A note below says '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-3
No more than 100 characters

Action provider
▶ Sourc... AWS Lambda ▾

Region
▶ Build... United States (N. Virginia) ▾

Input artifacts
Choose an input artifact for this action. [Learn more](#)

▶ Pipe... BackendArtifact

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.
Q. 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
▶ Sourc... AWS Lambda ▾

Region
▶ Build... United States (N. Virginia) ▾

Input artifacts
Choose an input artifact for this action. [Learn more](#)

▶ Pipe... BackendArtifact

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.
Q. 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](#)

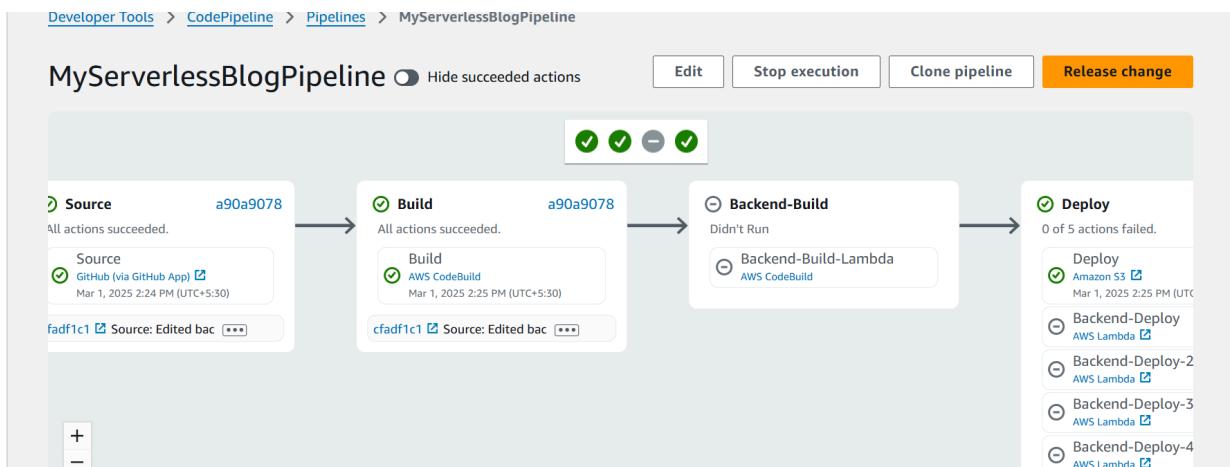
Deploy Amazon S3	Backend-Deploy AWS Lambda	Backend-Deploy-2 AWS Lambda
Backend-Deploy-3 AWS Lambda	Backend-Deploy-4 AWS Lambda	

Automated stage configuration: Enable automatic rollback on stage failure

◆ Step 6: Review & Create Pipeline

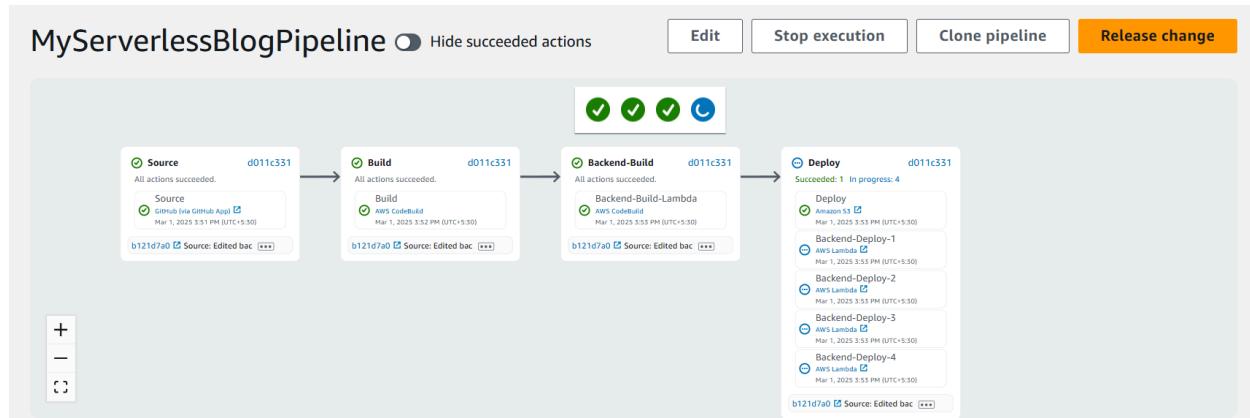
1. Review all stages
2. Click **Create pipeline**

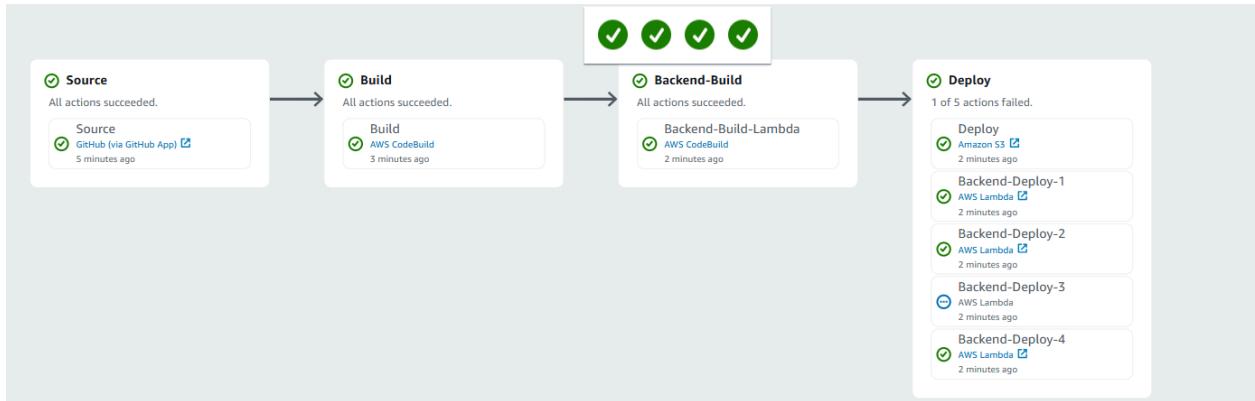
Your pipeline will automatically start running!



◆ 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/ge
t-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">
            <img alt="Choose an Image" /> 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;
```