

cprime

Serverless Blog Platform with Content
Moderation

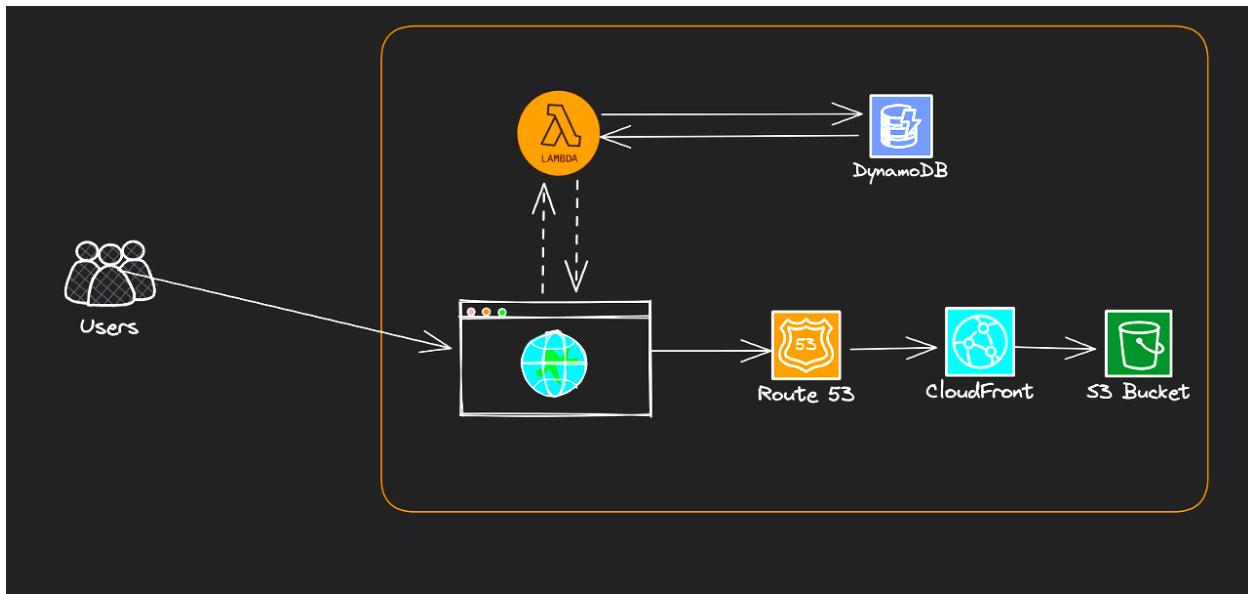


NAME : T MANOJ

EMPID : LYAKE2KHS

I. Overview

The Serverless Blog Application is a fully functional web application that demonstrates the capabilities of AWS serverless technologies. It includes features like user registration, login, blog creation, image upload, and moderation. The application is designed to be scalable, secure, and cost-effective, making it suitable for a wide range of use cases. The project is divided into multiple phases, each focusing on a specific set of functionalities and AWS services.



II. Introduction

The Serverless Blog Application is a modern, scalable, and cost-effective web application built using AWS serverless technologies. It allows users to register, log in, create blog posts, upload images, and view posts. The application leverages AWS services like Lambda, API Gateway, DynamoDB, S3, SNS, SQS, Rekognition, Route53, CloudFront, and Certificate Manager to provide a seamless and secure user experience. The project is designed to demonstrate the power of serverless architecture for building scalable and maintainable web applications.

III. Prerequisites

Technical Prerequisites

- AWS Account:** An active AWS account with necessary permissions to create and manage resources.
- AWS CLI:** Installed and configured with appropriate credentials.

3. **Python:** Installed for Lambda function development.
4. **GitHub Account:** For version control and CI/CD pipeline integration.
5. **Domain Name:** A registered domain name (e.g., manojconnects.space) for hosting the application.

Knowledge Prerequisites

1. Basic understanding of **AWS services** (Lambda, API Gateway, DynamoDB, S3, etc.).
2. Familiarity with **Python** for Lambda functions.
3. Knowledge of **HTML, CSS, and JavaScript** for frontend development.
4. Understanding of **REST APIs** and **serverless architecture**.

IV. Technologies & AWS Services Utilized

1. AWS Lambda

- **Description:** AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources. It allows you to build applications without provisioning or managing servers.
- **Role in the Project:**
 - Handles backend logic for user registration, login, blog creation, and image moderation.
 - Executes Python code to interact with DynamoDB, S3, SNS, SQS, and Rekognition.
 - Scales automatically based on the number of requests.

2. Amazon API Gateway

- **Description:** Amazon API Gateway is a fully managed service that makes it easy to create, publish, maintain, monitor, and secure APIs at any scale.
- **Role in the Project:**
 - Acts as the front door for the application, routing requests from the frontend to the appropriate Lambda functions.
 - Provides RESTful endpoints for user registration (/INSERTLOGIN), login (/GETDETAILS), and blog operations (/uploadBlog, /getBlogs, /getMyPosts).
 - Enables CORS (Cross-Origin Resource Sharing) for secure communication between the frontend and backend.

3. Amazon DynamoDB

- **Description:** Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.
- **Role in the Project:**
 - Stores user data (e.g., userId, email, password, userSnsTopicArn) in the LOGIN table.
 - Stores blog data (e.g., blogId, userId, title, content, imageUrl) in the BLOGS table.
 - Uses Global Secondary Indexes (GSIs) for efficient querying (e.g., filtering blogs by userId).

4. Amazon S3 (Simple Storage Service)

- **Description:** Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance.
- **Role in the Project:**
 - Stores static website files (HTML, CSS, JavaScript) for the frontend.
 - Stores blog images uploaded by users in the blogdata-12 bucket.
 - Enables static website hosting for the application.

5. Amazon SNS (Simple Notification Service)

- **Description:** Amazon SNS is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication.
- **Role in the Project:**
 - Sends email notifications to users after successful registration.
 - Sends notifications to the admin when a new user registers.
 - Creates a dedicated SNS topic for each user to send personalized notifications.

6. Amazon SQS (Simple Queue Service)

- **Description:** Amazon SQS is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications.
- **Role in the Project:**
 - Queues user registration details for admin processing.
 - Ensures reliable message delivery between the INSERTLOGIN Lambda function and admin systems.

7. Amazon Rekognition

- **Description:** Amazon Rekognition is a deep learning-based image and video analysis service that can identify objects, people, text, scenes, and activities.
- **Role in the Project:**
 - Scans images uploaded by users for inappropriate content.
 - Triggers notifications and deletions if inappropriate content is detected.
 - Stores moderation labels in the MODERATION DynamoDB table.

8. Amazon Route 53

- **Description:** Amazon Route 53 is a scalable and highly available Domain Name System (DNS) web service.
- **Role in the Project:**
 - Manages domain routing for the application (e.g., manojconnects.space).
 - Routes traffic to the CloudFront distribution for the application.

9. Amazon CloudFront

- **Description:** Amazon CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally.
- **Role in the Project:**
 - Delivers static website files (HTML, CSS, JavaScript) and blog images with low latency.
 - Provides HTTPS support using SSL/TLS certificates from AWS Certificate Manager.

10. AWS Certificate Manager

- **Description:** AWS Certificate Manager is a service that lets you easily provision, manage, and deploy Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates.
- **Role in the Project:**
 - Provides SSL/TLS certificates for secure communication between users and the application.
 - Enables HTTPS for the custom domain (manojconnects.space).

11. AWS CodePipeline

- **Description:** AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates.
- **Role in the Project:**
 - Automates the build, test, and deployment process for the application.
 - Integrates with GitHub and AWS CodeBuild for seamless CI/CD.

12. AWS CodeBuild

- **Description:** AWS CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages.
- **Role in the Project:**
 - Builds and packages the Lambda function code.
 - Runs tests to ensure code quality before deployment.

13. Amazon VPC (Virtual Private Cloud)

- **Description:** Amazon VPC lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network.
- **Role in the Project:**
 - Secures backend resources (e.g., Lambda functions, DynamoDB) in a private network.
 - Provides network-level security and isolation.

14. AWS IAM (Identity and Access Management)

- **Description:** AWS IAM is a web service that helps you securely control access to AWS resources.
- **Role in the Project:**
 - Manages permissions for Lambda functions to access DynamoDB, S3, SNS, SQS, and Rekognition.
 - Ensures least privilege access for enhanced security.

15. AWS CloudFormation

- **Description:** AWS CloudFormation is a service that helps you model and set up AWS resources so you can spend less time managing resources and more time focusing on your applications.
- **Role in the Project:**
 - Automates the creation and management of all AWS resources (e.g., Lambda, DynamoDB, S3, API Gateway).
 - Provides a reusable template for deploying the application in different environments.

16. AWS CloudWatch

- **Description:** Amazon CloudWatch is a monitoring and observability service built for DevOps engineers, developers, and IT managers.
- **Role in the Project:**
 - Monitors Lambda function performance and logs.
 - Tracks API Gateway metrics (e.g., request count, latency).

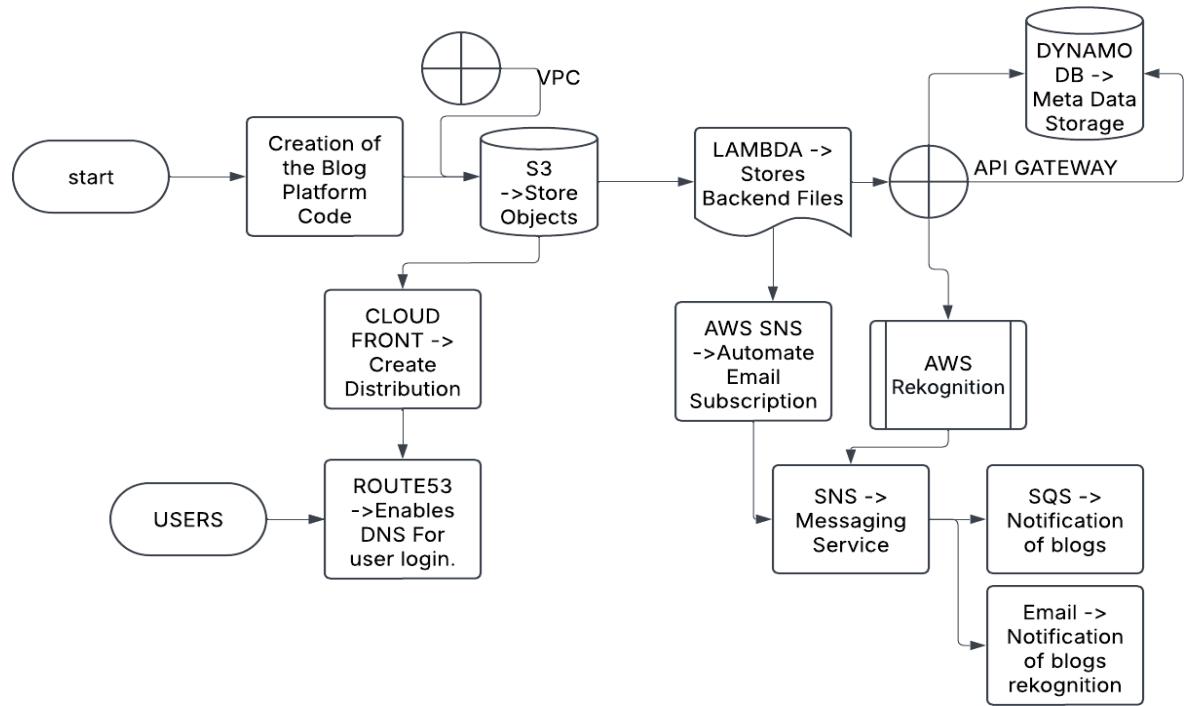
17. Git

- **Description:** Git is a distributed version control system used to track changes in source code and facilitate collaboration among developers.
- **Role in the Project:**
 - Tracks changes in the project's source code (e.g., HTML, CSS, JavaScript, and Python files).
 - Enables seamless integration with CI/CD pipelines for automated deployment.

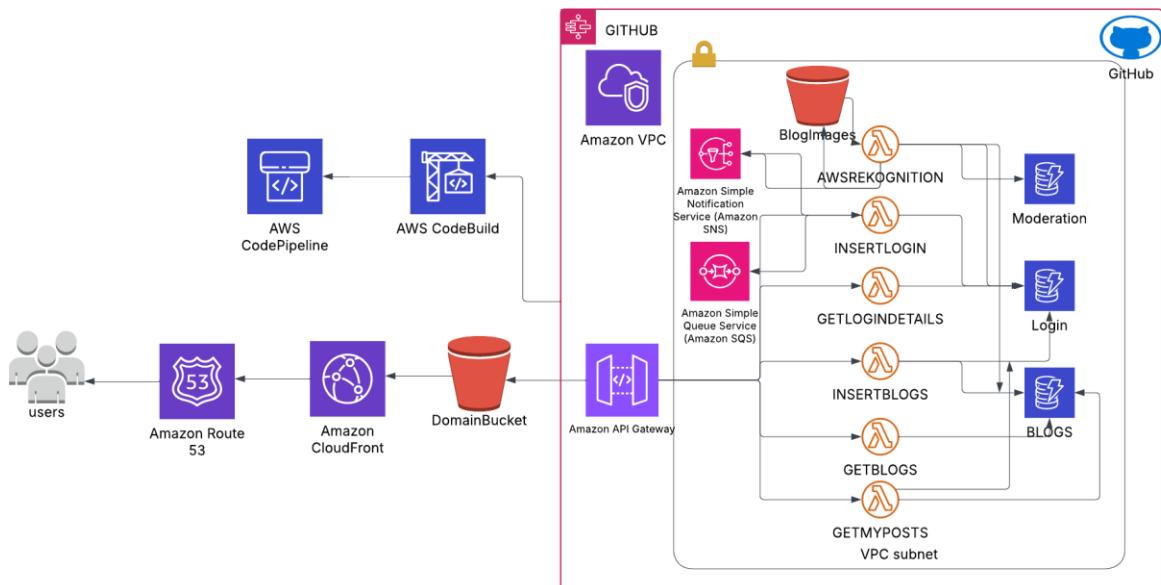
Summary of Services

<i>Service</i>	<i>Role in the Project</i>
AWS Lambda	Executes backend logic for user registration, login, blog creation, and image moderation.
API Gateway	Provides RESTful endpoints for the frontend to interact with backend services.
DynamoDB	Stores user and blog data in a scalable NoSQL database.
S3	Stores static website files and blog images.
SNS	Sends email notifications to users and admins.
SQS	Queues user registration details for admin processing.
Rekognition	Scans uploaded images for inappropriate content.
Route 53	Manages domain routing for the application.
CloudFront	Delivers static content with low latency and high performance.
Certificate Manager	Provides SSL/TLS certificates for secure HTTPS communication.
CodePipeline	Automates the CI/CD pipeline for the application.
CodeBuild	Builds and packages Lambda function code.
VPC	Secures backend resources in a private network.
IAM	Manages permissions for AWS resources.
CloudFormation	Automates resource creation and management.
CloudWatch	Monitors application performance and logs.

V. Proposed Workflow :



VI. SYSTEM ARCHITECTURE :

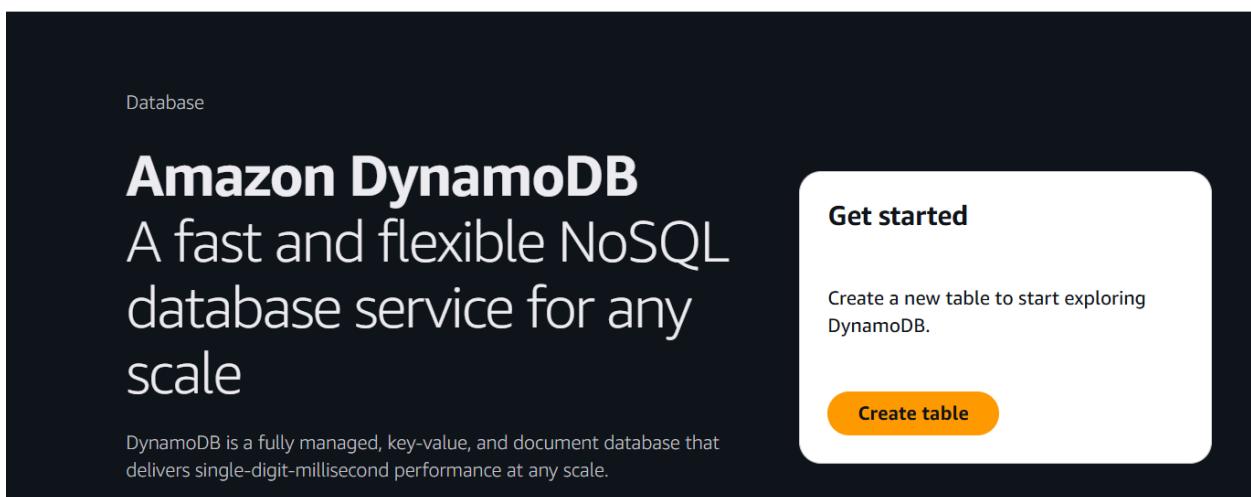


VII. Development Phases & Workflow

Phase 1: Login and Signup Page with Admin SNS, Admin SQS, User SNS, and DynamoDB

Step 1: Create DynamoDB Table

1. **Table Name:** LOGIN
2. **Partition Key:** userId (String)
3. **Attributes:**
 - a. userId (Primary Key)
 - b. email (String)
 - c. password (String)
 - d. userSnsTopicArn (String)
4. **Settings:**
 - a. Enable encryption at rest using AWS KMS.
 - b. Set read/write capacity units (or enable auto-scaling).
5. **Create Table:**
 - a. Use AWS Management Console or AWS CLI (aws dynamodb create-table) or cloud formation by mentioning in the template.



General information

Partition key userId (String)	Sort key -	Capacity mode On-demand	Table status Active
Alarms No active alarms	Point-in-time recovery (PITR) Info Off	Resource-based policy Info Not active	

Items returned (2)

	userId (String)	email	password	snsTopicArn
<input type="checkbox"/>				
<input type="checkbox"/>				

Step 2: Create Lambda Function INSERTLOGIN

- Function Logic:**
 - Extract userId, email, and password from the API Gateway request body.
 - Validate input data (e.g., check for empty fields, valid email format).
 - Generate a unique userSnsTopicArn for the user.
 - Store userId, email, password, and userSnsTopicArn in the LOGIN DynamoDB table.
 - Publish user registration details to **Admin SNS Topic**.
 - Send a message to **Admin SQS Queue** with user details.
 - Create a new **User SNS Topic** for the registered user.
 - Subscribe the user's email to the SNS topic for notifications.
 - Return a success response with userId and userSnsTopicArn.

Lambda > Functions > INSERTLOGIN

INSERTLOGIN

Function overview Info

Diagram Template

INSERTLOGIN

Layers (0)

API Gateway (2)

+ Add destination

+ Add trigger

Description

Last modified 5 hours ago

Function ARN arn:aws:lambda:us-east-1:491085415620:function:INSERTLOGIN

Function URL Info

Lambda > Functions > INSERTLOGIN

Code source Info

EXPLORER

INSERTLOGIN

lambda_function.py

DEPLOY

Deploy (Ctrl+Shift+F)

Test (Ctrl+Shift+I)

TEST EVENTS [SELECTED: PYTHONCODE]

Upload a file from Amazon S3

When you upload a new .zip file package, it overwrites the existing code.

Amazon S3 link URL

Paste an S3 link URL to your function code.zip.

s3://www.manojconnects.space/lambda/INSERTLOGIN

Enable encryption with an AWS KMS customer managed key

By default, Lambda encrypts the .zip file archive using an AWS owned key.

Cancel Save

CloudShell Feedback

aws Search [Alt+S] United States (N. Virginia) T MANOJ

Amazon SQS > Queues > BLOG

BLOG

Edit Delete Purge Send and receive messages Start DLQ redrive

Details Info

Name	Type	ARN
BLOG	Standard	arn:aws:sqs:us-east-1:491085415620:BLOG
Encryption	URL	Dead-letter queue
Disabled	https://sqs.us-east-1.amazonaws.com/491085415620/BLOG	-

More

Topics (3)		
Edit Delete Publish message Create topic		
	Name	Type
<input checked="" type="radio"/>	ADMIN-BLOG	Standard

2. IAM Role:

- a. Attach the following policies:
 - i. AmazonDynamoDBFullAccess
 - ii. AmazonSNSFullAccess
 - iii. AmazonSQSFullAccess
 - iv. AWSLambdaBasicExecutionRole

LambdaBlogAppRole Info	
Allows Lambda functions to call AWS services on your behalf.	
Summary	Edit
Creation date February 27, 2025, 10:26 (UTC+05:30)	ARN  arn:aws:iam::491085415620:role/LambdaBlogAppRole
Last activity  5 hours ago	Maximum session duration 1 hour

3. Environment Variables:

- a. ADMIN_SNS_TOPIC_ARN:

`arn:aws:sqs:us-east-1:491085415620:BLOG.`

- b. ADMIN_SQS_QUEUE_URL:

`https://sns.us-east-1.amazonaws.com/491085415620/BLOG.`

4. Testing:

- a. Create a test event with sample data:

<code>"userId":</code>	<code>"manoj"</code>
<code>"email":</code>	<code>"dreamers2k22@gmail.com"</code>
<code>"password":</code>	<code>"securepassword"</code>

}

- b. Verify the function logs, DynamoDB entries, SNS notifications, and SQS messages.

The screenshot shows the AWS Lambda function execution results. The 'OUTPUT' tab is selected, displaying the response object and function logs.

```
Response:
{
  "statusCode": 201,
  "headers": {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "POST, OPTIONS",
    "Access-Control-Allow-Headers": "Content-Type, Authorization"
  },
  "body": "{\"success\": true, \"message\": \"User registered successfully!\", \"userId\": \"manoj23\"}"
}

Function Logs:
START RequestId: 1714f4db-6a9b-4729-976c-f3df10391d28 Version: $LATEST
END RequestId: 1714f4db-6a9b-4729-976c-f3df10391d28
REPORT RequestId: 1714f4db-6a9b-4729-976c-f3df10391d28 Duration: 1936.02 ms Billed Duration: 1937
ms Memory Size: 128 MB Max Memory Used: 82 MB Init Duration: 507.96 ms
```

The screenshot shows the AWS SNS topic configuration for 'user-manoj-topic'. The 'Details' section displays the topic's name, ARN, and type.

Name	Display name
user-manoj-topic	-

ARN	Topic owner
arn:aws:sns:us-east-1:491085415620:user-manoj-topic	491085415620

Type
Standard

Buttons: Edit, Delete, Publish message

The screenshot shows the details of a received message. The message ID is 818a956e-2366-4413-bf9a-3fba530f6ee9.

Message: 818a956e-2366-4413-bf9a-3fba530f6ee9

Body: {"userId": "manoj", "email": "dreamers2k22@gmail.com", "message": "User registered successfully!"}

Attributes: (empty)

Details: (empty)

Buttons: Done



AWS Notification - Subscription Confirmation Inbox x

AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾

15:48 (1 minute ago)

2 of 638 < >

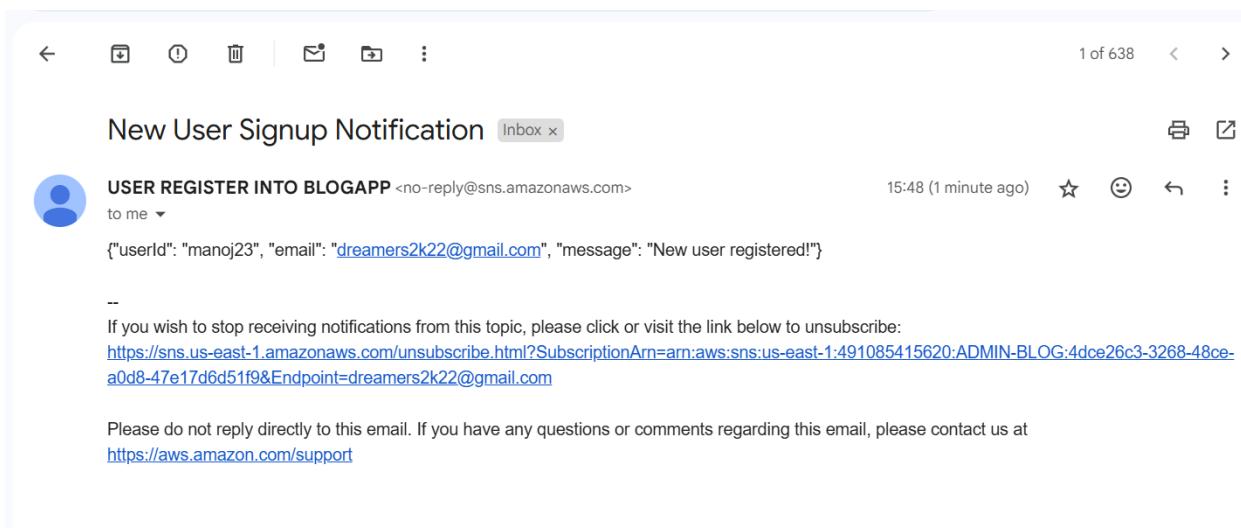
You have chosen to subscribe to the topic:

arn:aws:sns:us-east-1:491085415620:user-manoj23-topics

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

Confirm subscription

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)



Step 3: Create API Gateway LOGINAPI

1. API Name: LOGINAPI

APIs (1/1)						 Delete	Create API				
<input type="text"/> Find APIs						< 1 >					
Name	▲	Description	▼	ID	▼	Protocol	▼	API endpoint type	▼	Created	▼
 LOGINAPI		x0t22jrakh		REST		Edge-optimized		2025-02-27			

2. **Resource:** /INSERTLOGIN

The screenshot shows the 'Resource details' page for the '/INSERTDETAILS' endpoint. On the left, a tree view lists methods: GET, OPTIONS, POST, and two custom methods: /getBlogs and /GETLOGIN. The right panel displays the resource path (/INSERTDETAILS), a resource ID (7o4sfy), and three buttons: Delete, Update documentation, and Enable CORS. Below these are sections for 'Methods (2)' and 'Method type' filtering.

Method	Type	Integration type	Authorization	API key
OPTIONS	Mock	None	Not required	
POST	Lambda	None	Not required	

3. Integration:

- Connect to the INSERTLOGIN Lambda function.
- Enable CORS with headers: Content-Type, Authorization, OPTIONS.

4. Methods:

- POST: For user registration.
- OPTIONS: For CORS preflight requests.

The screenshot shows the 'Enable CORS' configuration page. It includes sections for 'Gateway responses' (checkboxes for Default 4XX and Default 5XX) and 'Access-Control-Allow-Methods' (checkboxes for OPTIONS and POST). A note at the top states: 'To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.'

5. Deploy API:

- Create a deployment stage (e.g., prod).
- Note the API endpoint URL.

Deploy API

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

⚠️ When you deploy an API to an existing stage, you immediately overwrite the current stage configuration with a new active deployment.

Stage

Prod

Deployment description

DEPLOYING NEW ENVIRONMENT

[Cancel](#)

[Deploy](#)

Step 4: Create Lambda Function GETLOGIN

1. Function Logic:

- Extract userId and password from the API request.
- Query the LOGIN DynamoDB table using userId.
- Compare the provided password with the stored password.
- Return a success response if credentials match, else return an error.

GETLOGIN-role-76nfb8m [Info](#)

[Delete](#)

Summary

[Edit](#)

Creation date

February 27, 2025, 20:14 (UTC+05:30)

ARN

[arn:aws:iam::491085415620:role/service-role/GETLOGIN-role-76nfb8m](#)

Last activity

-

Maximum session duration

1 hour

The screenshot shows the AWS Lambda Functions interface. At the top, there's a search bar and navigation links for Lambda > Functions > GETDETAILS. Below the header, the function name 'GETDETAILS' is displayed. On the left, a 'Function overview' section includes a 'Diagram' tab (selected) and a 'Template' tab. The diagram shows a Lambda icon labeled 'GETDETAILS' with 'Layers (0)' underneath. To its right is an 'API Gateway' box containing '(6)' triggers. Buttons for '+ Add destination' and '+ Add trigger' are visible. On the right side, there are sections for 'Description' (empty), 'Last modified' (5 hours ago), 'Function ARN' (arn:aws:lambda:us-east-1:491085415620:function:GETDETAILS), and 'Function URL' (Info). Buttons for 'Throttle', 'Copy ARN', and 'Actions' are at the top right.

The screenshot shows the AWS Lambda code editor. On the left, the 'Code source' tab is selected, showing an 'EXPLORER' view with a file named 'lambda_function.py'. In the center, a modal dialog titled 'Upload a file from Amazon S3' is open. It contains a note: 'When you upload a new .zip file package, it overwrites the existing code.' Below this is an 'Amazon S3 link URL' field containing 's3://www.manojconnects.space/lambda/GETDETAILS.zip'. There's also a checkbox for 'Enable encryption with an AWS KMS customer managed key'. At the bottom of the dialog are 'Cancel' and 'Save' buttons. The background shows a preview of the Python code in 'TEST EVENTS' and a status bar with copyright information.

2. IAM Role:

- Attach the following policies:
 - AmazonDynamoDBReadOnlyAccess
 - AWSLambdaBasicExecutionRole

GETLOGIN-role-76nfb8m [Info](#)

Summary	
Creation date February 27, 2025, 20:14 (UTC+05:30)	ARN arn:aws:iam::491085415620:role/service-role/GETLOGIN-role-76nfb8m
Last activity -	Maximum session duration 1 hour

3. Testing:

- Create a test event with sample data:

```

        "userId": "omnamoshivaya",
        "password": "securepassword"
    }
    
```
- Verify the function logs and response.

Lambda > Functions > GETDETAILS

Code source [Info](#)

Upload from [...](#)

EXPLORER

DEPLOY

Deploy (Ctrl+Shift+U) Test (Ctrl+Shift+I)

TEST EVENTS [SELECTED: PYTHONCODE]

- + Create new test event
- Private saved events

lambda_function.py

```

def lambda_handler(event, context):
    """
    Lambda function that processes a POST request to validate user login.
    """
    # Extract user credentials from the request body
    user_id = event['body']['userId']
    password = event['body']['password']

    # Verify user credentials (simulated)
    if user_id == "omnamoshivaya" and password == "securepassword":
        return {
            "statusCode": 200,
            "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
            "body": "Login successful"
        }
    else:
        return {
            "statusCode": 401,
            "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
            "body": "Invalid credentials"
        }
    
```

Function Logs:

```

START RequestId: 638880ef-6b2f-49fe-a48b-e575087094a8 Version: $LATEST
Received Event: {"body": {"userId": "omnamoshivaya", "password": "omnamoshivaya"}}
omnamoshivaya omnamoshivaya
{'ResponseMetadata': {'RequestId': 'PBQ2EL3TRSDQCDFNUN64TUIIMNVV4KQNS05AEMVJF66Q9ASUAAJG', 'HTTPStatusCode': 200, 'HTTPHeaders': {'server': 'Server', 'date': 'Sun, 02 Mar 2025 10:27:49 GMT', 'content-type': 'application/x-amz-json-1.0', 'content-length': '2', 'connection': 'keep-alive', 'x-amzn-requestid': 'PBQ2EL3TRSDQCDFNUN64TUIIMNVV4KQNS05AEMVJF66Q9ASUAAJG', 'x-amz-crc32': '2745614147'}, 'RetryAttempts': 0}} www
DynamoDB Response: {'ResponseMetadata': {'RequestId': 'PBQ2EL3TRSDQCDFNUN64TUIIMNVV4KQNS05AEMVJF66Q9ASUAAJG', 'HTTPStatusCode': 200, 'HTTPHeaders': {'server': 'Server', 'date': 'Sun, 02 Mar 2025 10:27:49 GMT', 'content-type': 'application/x-amz-json-1.0', 'content-length': '2', 'connection': 'keep-alive', 'x-amzn-requestid': 'PBQ2EL3TRSDQCDFNUN64TUIIMNVV4KQNS05AEMVJF66Q9ASUAAJG', 'x-amz-crc32': '2745614147'}}}
    
```

Step 5: Add GETLOGIN Method to API Gateway

- Resource:** /GETDETAILS
- Integration:**
 - Connect to the GETDETAILS Lambda function.
 - Enable CORS with headers: Content-Type, Authorization, OPTIONS.
- Methods:**
 - POST: For login validation.
 - OPTIONS: For CORS preflight requests.
- Deploy API:**
 - Update the deployment stage.

Resource details

[Delete](#)[Update documentation](#)[Enable CORS](#)**Path**

/GETLOGIN

Resource ID

gep09b

Methods (3)

[Delete](#)[Create method](#)

	Method type ▲	Integration type ▼	Authorization ▼	API key ▼
<input type="radio"/>	GET	Lambda	None	Not required
<input type="radio"/>	OPTIONS	Mock	None	Not required
<input type="radio"/>	POST	Lambda	None	Not required

[Edit](#) [View history](#)**CORS settings** Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.

Gateway responses

API Gateway will configure CORS for the selected gateway responses.

- Default 4XX
- Default 5XX

Access-Control-Allow-Methods

- GET
- OPTIONS
- POST

Access-Control-Allow-Headers

API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

Deploy API

X

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

⚠️ When you deploy an API to an existing stage, you immediately overwrite the current stage configuration with a new active deployment.

Stage

Prod



Deployment description

DEPLOYING NEW ENVIRONMENT

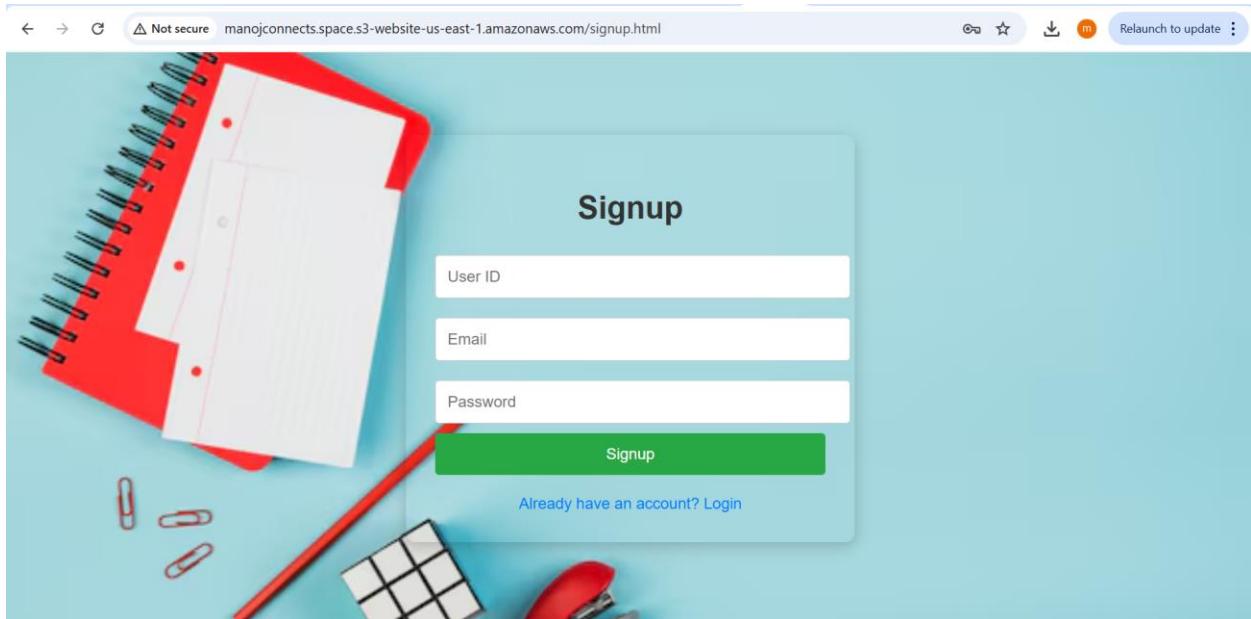
Cancel

Deploy

Step 6: Create Frontend (HTML, CSS, JavaScript)

1. Signup Page:

- a. Form fields: userId, email, password.
- b. On submit, call /INSERTLOGIN API using fetch or axios.
- c. Display success/error messages based on the API response.



www.manojconnects.space.s3-website-us-east-1.amazonaws.com says

Signup successful!

Console Sources Network Performance > Default levels ▾ No Issues | 1 hidden

OK

Group similar messages in console

Show CORS errors in console

Raw Response: `signup.html:161`
Response {type: 'cors', url: 'https://x0t22jrakh.execute-api.us-east-1.amazonaws.com/Prod/INSERTDETAILS', redirected: false, status: 200, ok: true, ...}

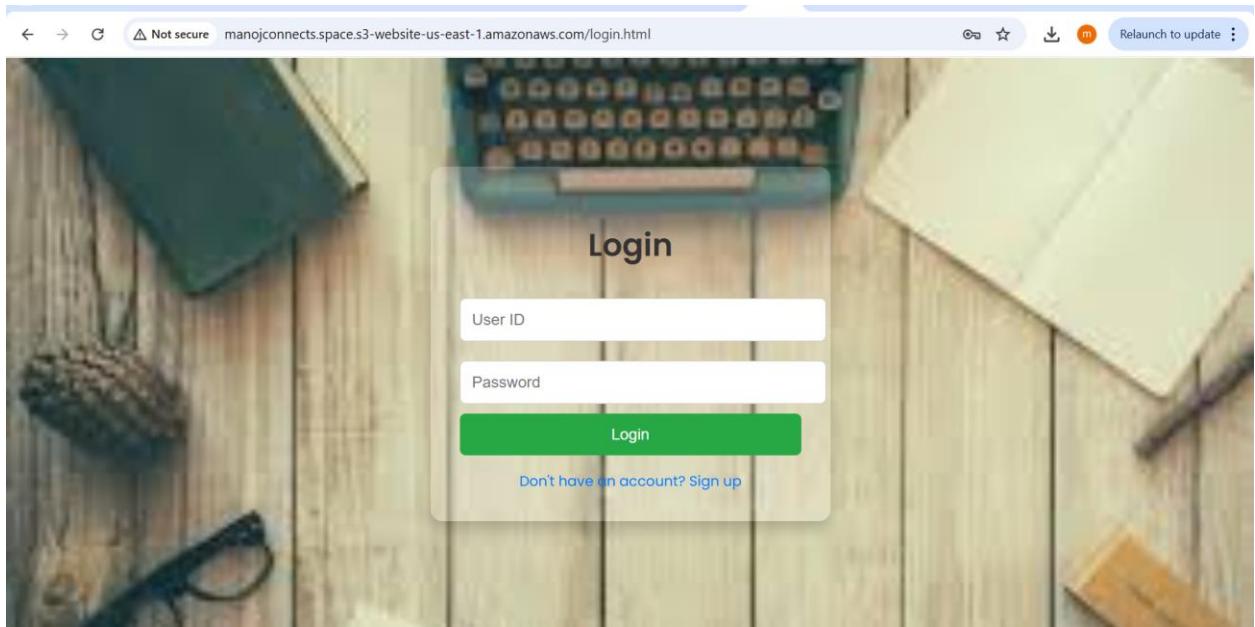
Parsed Response: `signup.html:164`
► {statusCode: 201, headers: {}, body: {"success": true, "message": "User registered successfully!", "userId": "manojcprime"}}

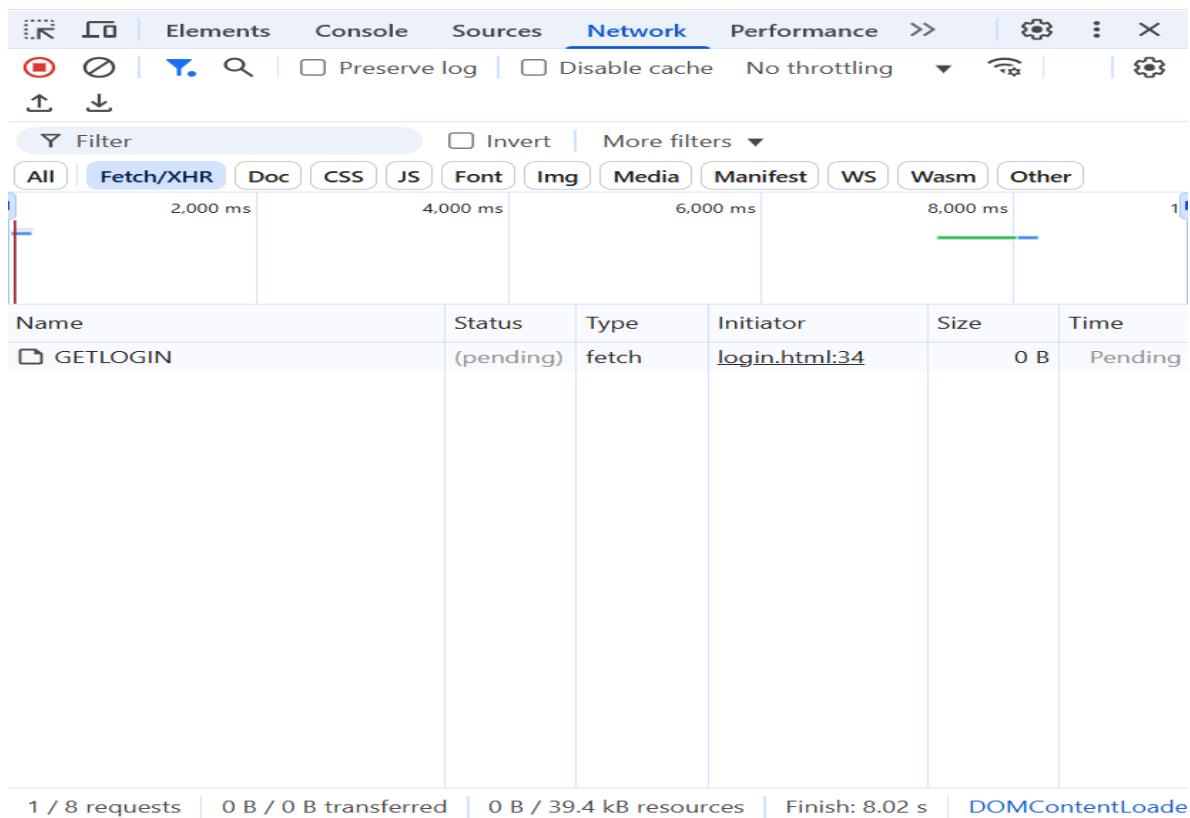
Final Parsed Data: `signup.html:174`
► {success: true, message: 'User registered successfully!', userId: 'manojcprime'}

Fetch finished loading: POST "`https://x0t22jrakh.execute-api.us-east-1.amazonaws.com/Prod/INSERTDETAILS`". `signup.html:155`

2. Login Page:

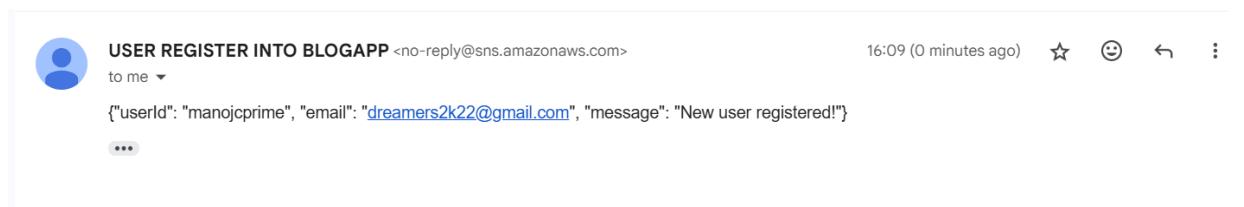
- Form fields: userId, password.
- On submit, call /GETDETAILS API.
- Redirect to the blog home page on success.
- Display an error message for invalid credentials.





3. Notifications:

- a. Ensure the user receives an email via SNS after signup.
- b. Ensure the admin receives user details via SNS and SQS.





AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾

16:09 (0 minutes ago) ⭐ 😊 ← ⋮

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:491085415620:user-manojcprime-topic

...

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

Phase 2: Blog Home Page, Blog Post Page, and Database

Step 1: Create S3 Bucket and DynamoDB Table

1. S3 Bucket:

- a. Name: blogdata-12
- b. Enable versioning and encryption.
- c. Set bucket policy to allow public read access for images.

blogdata-12 Info

Objects | Metadata | **Properties** | Permissions | Metrics | Management | Access Points

Bucket overview

AWS Region
US East (N. Virginia) us-east-1

Amazon Resource Name (ARN)
 arn:aws:s3:::blogdata-12

Creation date
February 28, 2025, 01:12:52 (UTC+05:30)

Bucket Versioning

[Edit](#)

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#) 

Bucket Versioning
Enabled

Multi-factor authentication (MFA) delete

Bucket Policy :

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "rekognition.amazonaws.com"  
      },  
      "Action": "s3:GetObject",  
      "Resource": "arn:aws:s3:::blogdata-12/blogs/*"  
    },  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::491085415620:role/LambdaBlogAppRole"  
      },  
      "Action": "s3:GetObject",  
      "Resource": "arn:aws:s3:::blogdata-12/blogs/*"  
    },  
    {  
      "Effect": "Allow",  
      "Principal": "*","
```

```

        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::blogdata-12/blogs/*"
    }
]
}

```

2. **DynamoDB Table:**

- a. Name: BLOGS
- b. Partition Key: blogId (String)
- c. GSI: userId (for filtering user-specific posts)
- d. Attributes:
 - i. blogId (Primary Key)
 - ii. userId (GSI Key)
 - iii. title (String)
 - iv. content (String)
 - v. imageUrl (String)

The screenshot shows the AWS DynamoDB console interface for the 'BLOGS' table. At the top, there are navigation icons (star, refresh, actions), a 'Explore table items' button, and tabs for Overview, Indexes, Monitor, Global tables, Backups, Exports and streams, and Per. The 'Overview' tab is selected.

A callout box highlights a notification about Point-in-time recovery (PITR):

- Protect your DynamoDB table from accidental writes and deletes**
- When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply. [Learn more](#)

Below the notification, the 'General information' section displays the following details:

Partition key BlogId (String)	Sort key -	Capacity mode On-demand	Table status Active
Alarms No active alarms	Point-in-time recovery (PITR) Info Off	Resource-based policy Info Not active	

At the bottom of the page, there are links for [© 2025, Amazon Web Services, Inc. or its affiliates.](#), [Privacy](#), [Terms](#), and [Cookie preferences](#).

BLOGS

Overview **Indexes** Monitor Global tables Backups Exports and streams Permissions

Global secondary indexes (1)

Name	Status	Partition key	Sort key	Read capacity	Write capacity	Projection type
userId-index	Active	userId (String)	-	On-demand	On-demand	All

Items returned (2)

BlogId	content	imageUrl	title	userId

Step 2: Create Lambda Functions

1. uploadBlog:

- Input: userId, blogId, imageUrl, content, title.
- Save data in DynamoDB and image in S3.
- Return success/error response.

Lambda > Functions > uploadBlog

Function overview

Description

Last modified
6 hours ago

Function ARN
arn:aws:lambda:us-east-1:491085415620:function:uploadBlog

Function URL

Upload a file from Amazon S3

X

i When you upload a new .zip file package, it overwrites the existing code.

Amazon S3 link URL

Paste an S3 link URL to your function code .zip.

s3://www.manojconnects.space/lambda/uploadBlog.zip

Enable encryption with an AWS KMS customer managed key

By default, Lambda encrypts the .zip file archive using an AWS owned key.

Cancel

Save

```
PROBLEMS    OUTPUT    CODE REFERENCE LOG    TERMINAL    Execution Results    ▾    ⌂    ⌄    ⌅    ⌆    ⌇    ⌈    ⌉
```

Status: Succeeded
Test Event Name: Pythoncode

Response:

```
{
  "statusCode": 200,
  "body": "{\"success\": true, \"blogId\": \"f05425fe-2631-44f1-9099-309743c31161\"}",
  "headers": {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*"
  }
}
```

Function Logs:

```
START RequestId: e0364c77-7270-47e5-a60b-90f48532cab1 Version: $LATEST
END RequestId: e0364c77-7270-47e5-a60b-90f48532cab1
REPORT RequestId: e0364c77-7270-47e5-a60b-90f48532cab1 Duration: 1318.87 ms      Billed Duration: 1319
ms      Memory Size: 128 MB Max Memory Used: 87 MB

Request ID: e0364c77-7270-47e5-a60b-90f48532cab1
```

2. getBlogs:

- a. Scan DynamoDB to retrieve all blogs.
- b. Return blog data.

The screenshot shows the AWS Lambda Function Overview page for a function named 'getBlogs'. The top navigation bar includes the AWS logo, a search bar, and account information for 'United States (N. Virginia)'. Below the navigation, the function name 'getBlogs' is displayed, along with 'Throttle', 'Copy ARN', and 'Actions' buttons.

The main area is divided into several sections:

- Function overview**: Includes tabs for 'Diagram' (selected) and 'Template'. The diagram shows a single step labeled 'getBlogs' with a 'Layers' section below it. An 'API Gateway' section indicates 2 triggers, with buttons for '+ Add destination' and '+ Add trigger'.
- Description**: A text input field.
- Last modified**: Shows '6 hours ago'.
- Function ARN**: Displays the ARN: arn:aws:lambda:us-east-1:491085415620:function:getBlogs.
- Function URL**: A text input field.

Upload a file from Amazon S3



i When you upload a new .zip file package, it overwrites the existing code.

Amazon S3 link URL

Paste an S3 link URL to your function code .zip.

s3://www.manojconnects.space/lambda/getBlogs

Enable encryption with an AWS KMS customer managed key

By default, Lambda encrypts the .zip file archive using an AWS owned key.

[Cancel](#)

[Save](#)

The screenshot shows the AWS Lambda Test Event interface. The event name is "getBlogs". The response body is a JSON object representing a successful HTTP response with status code 200, headers including Content-Type: application/json and Access-Control-Allow-Origin: *, and a body containing two blog posts. The posts have titles "hello", user IDs "manoj", and URLs like "https://blogdata-12.s3.amazonaws.com/blogs/ef07c004-aabd-45e2-9071-f4699e032b0e.jpg". The response is displayed in a code editor-like interface with syntax highlighting for JSON.

```
Response:
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*"
  },
  "body": "[{"success": true, "blogs": [{"content": "hello", "imageUrl": "https://blogdata-12.s3.amazonaws.com/blogs/ef07c004-aabd-45e2-9071-f4699e032b0e.jpg", "BlogId": "ef07c004-aabd-45e2-9071-f4699e032b0e", "userId": "manoj", "title": "hello"}, {"content": "This is my blog post content.", "imageUrl": "https://blogdata-12.s3.amazonaws.com/blogs/f05425fe-2631-44f1-9099-309743c31161.jpg", "BlogId": "f05425fe-2631-44f1-9099-309743c31161", "userId": "user678", "title": "My First Blog"}]}"
}
```

3. getMyPosts:

- a. Input: userId (from query string).
- b. Query DynamoDB using GSI to filter user-specific posts.
- c. Return filtered data.

aws | Search [Alt+S] | United States (N. Virginia) | T MANOJ

Lambda > Functions > getMyPosts

getMyPosts

Throttle Copy ARN Actions ▾

▼ Function overview [Info](#)

[Diagram](#) [Template](#)

 **getMyPosts**
Layers (0)

 API Gateway (2)
+ Add destination
+ Add trigger

Export to Infrastructure Composer Download ▾

Description -

Last modified 6 hours ago

Function ARN [arn:aws:lambda:us-east-1:491085415620:function:getMyPosts](#)

Function URL [Info](#)

Upload a file from Amazon S3

X

ⓘ When you upload a new .zip file package, it overwrites the existing code.

Amazon S3 link URL

Paste an S3 link URL to your function code .zip.

s3://www.manojconnects.space/lambda/getMyPosts.zip

Enable encryption with an AWS KMS customer managed key

By default, Lambda encrypts the .zip file archive using an AWS owned key.

Cancel

Save

Upload a file from Amazon S3

X

ⓘ When you upload a new .zip file package, it overwrites the existing code.

Amazon S3 link URL

Paste an S3 link URL to your function code .zip.

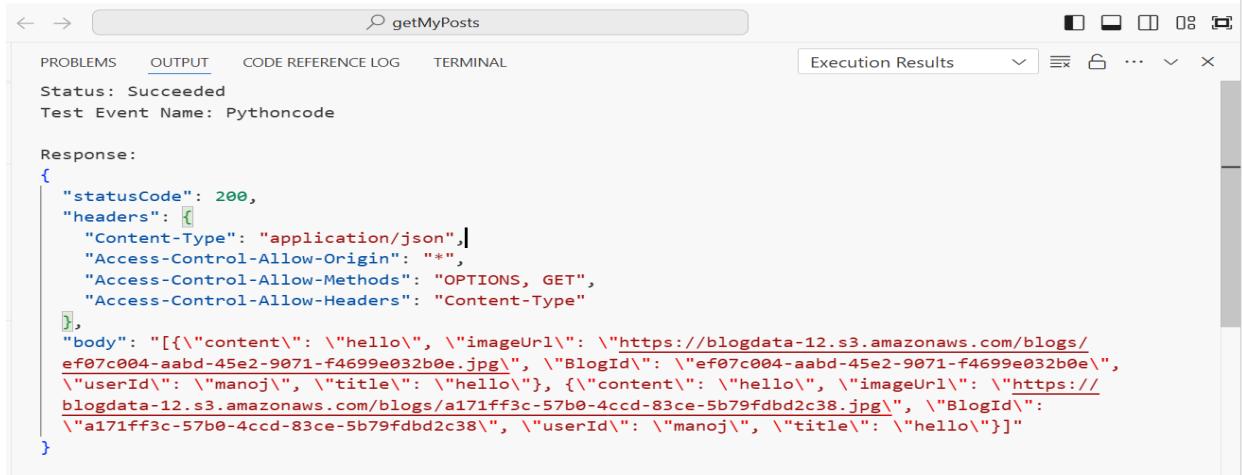
s3://www.manojconnects.space/lambda/getMyPosts.zip

Enable encryption with an AWS KMS customer managed key

By default, Lambda encrypts the .zip file archive using an AWS owned key.

Cancel

Save



The screenshot shows the AWS Lambda console interface. At the top, there's a navigation bar with back, forward, and search buttons, followed by the function name "getMyPosts". Below the navigation is a toolbar with icons for PROBLEMS, OUTPUT, CODE REFERENCE LOG, TERMINAL, and EXECUTION RESULTS. The EXECUTION RESULTS tab is selected. The status is shown as "Succeeded". The test event name is "Pythoncode". The response body is displayed as follows:

```
Response:
{
  "statusCode": 200,
  "headers": [
    {
      "Content-Type": "application/json",
      "Access-Control-Allow-Origin": "*",
      "Access-Control-Allow-Methods": "OPTIONS, GET",
      "Access-Control-Allow-Headers": "Content-Type"
    }
  ],
  "body": "[{"content": "hello", "imageUrl": "https://blogdata-12.s3.amazonaws.com/blogs/ef07c004-aabd-45e2-9071-f4699e032b0e.jpg", "BlogId": "ef07c004-aabd-45e2-9071-f4699e032b0e", "userId": "manoj", "title": "hello"}, {"content": "hello", "imageUrl": "https://blogdata-12.s3.amazonaws.com/blogs/a171ff3c-57b0-4cccd-83ce-5b79fdbd2c38.jpg", "BlogId": "a171ff3c-57b0-4cccd-83ce-5b79fdbd2c38", "userId": "manoj", "title": "hello"}]"
}
```

Step 3: Create API Gateway Methods

1. **getMyPosts:**

- a. Methods: POST, OPTIONS.

- b. Enable CORS.
- c. Add Method Responses and Integration Response.

Resource details

[Delete](#) [Update documentation](#) [Enable CORS](#)

Path	/getMyPosts	Resource ID	ru337o
-------------	-------------	--------------------	--------

Methods (3)

[Delete](#) [Create method](#)

Method type	Integration type	Authorization	API key
GET	Lambda	None	Not required
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required

Enable CORS

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.

Gateway responses

API Gateway will configure CORS for the selected gateway responses.

- Default 4XX
- Default 5XX

Access-Control-Allow-Methods

- GET
- OPTIONS
- POST

Access-Control-Allow-Headers

API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

Method responses Info

[Create response](#)

Response 200

[Edit](#)[Delete](#)

Response headers (1)

Name

Access-Control-Allow-Origin

< 1 >

▼

Response body (1)

Content type

▼ | Model

application/json

Empty

< 1 >

▼

Integration responses

[Create response](#)

Default - Response

[Edit](#)[Delete](#)

Lambda error regex Info

-

Content handling Learn more ↗

Passthrough

Method response status code

200

Default mapping

True

Header mappings (1)

< 1 >

Name

▲ | Mapping value ▼

method.response.header.Access-Control-Allow-Origin

**

Mapping templates (1)

▶ application/json

© 2025, Amazon Web Services, Inc. or its affiliates.

[Privacy](#)[Terms](#)[Cookie preferences](#)

2. getBlogs:

- a. Methods: GET, OPTIONS.
- b. Enable CORS.

Resource details

[Delete](#)[Update documentation](#)[Enable CORS](#)

Path
/getBlogs

Resource ID
ybzl8

Methods (3)

[Delete](#)[Create method](#)

Method type	Integration type	Authorization	API key
<input type="radio"/> GET	Lambda	None	Not required
<input type="radio"/> OPTIONS	Mock	None	Not required
<input type="radio"/> POST	Lambda	None	Not required

Enable CORS

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.

Gateway responses

API Gateway will configure CORS for the selected gateway responses.

- Default 4XX
- Default 5XX

Access-Control-Allow-Methods

- GET
- OPTIONS
- POST

Access-Control-Allow-Headers

API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

3. uploadBlog:

- a. Methods: GET, OPTIONS.
- b. Add query string parameter: userId.
- c. Enable CORS.

API actions ▾ Deploy API

Resource details

Path /uploadBlog Resource ID bc3j9t

Delete Update documentation Enable CORS

Methods (3)

Method type	Integration type	Authorization	API key
GET	Lambda	None	Not required
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required

Delete Create method

Enable CORS

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.

Gateway responses
API Gateway will configure CORS for the selected gateway responses.

Default 4XX
 Default 5XX

Access-Control-Allow-Methods

GET
 OPTIONS
 POST

Access-Control-Allow-Headers
API Gateway will configure CORS for the selected gateway responses.

Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token

4. Deploy API

Deploy API

X

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

⚠️ When you deploy an API to an existing stage, you immediately overwrite the current stage configuration with a new active deployment.

Stage

Prod



Deployment description

DEPLOYING NEW ENVIRONMENT

[Cancel](#)

[Deploy](#)

Step 4: Create Frontend Pages

1. Home Page (home.html):

- a. Form for blog creation: title, content, image.
- b. Buttons: "Submit Post", "My Posts", "All Posts".
- c. On submit, call uploadBlog API.

Create a New Blog.

Logout

This is a testing upload blog

 download.jpg

Upload Post

Show Posts

My Posts

Not secure manojconnects.space.s3-website-us-east-1.amazonaws.com/home.html

Create a New

Testing Blog

This is a testing upload blog

Choose file download.jpg

Upload Post

Show Posts

My Posts

www.manojconnects.space.s3-website-us-east-1.amazonaws.com says

Blog posted successfully!

OK

Console Sources Network Performance > Relaunch to update

Name Status Type Initiator Size Time

uploadBlog	200	fetch	home.js:35	466 B	3.41 s
------------	-----	-------	------------	-------	--------

2.000 ms 3.000 ms 4.000 ms 5.00

1 / 2 requests | 466 B / 466 B transferred | 67 B / 67 B resources

2. My Posts Page (posts.html):

- a. Fetch user-specific posts using getMyPosts API.
- b. Display posts with images, titles, and content.

My Blog Posts

Back to Home

Your Blogs

Testing Blog

This is a testing upload blog



Elements Console Sources Network Performance > Relaunch to update

Name Status Type Initiator Size Time

getMyPosts?userId=manoj	200	fetch	myposts.js:18	1.3 kB	1.66 s
-------------------------	-----	-------	---------------	--------	--------

500 ms 1.000 ms 1.500 ms 2.000 ms 2.500 ms 3.000 ms 3.500 ms 4.00

3. All Posts Page (blogs.html):

- a. Fetch all posts using getBlogs API.
- b. Display all posts.

The screenshot shows a web browser window with the URL manojconnects.space.s3-website-us-east-1.amazonaws.com/blogs.html. The page title is "All Blog Posts". Below it is a button "Back to Home". The main content area is titled "Testing Blog" and includes the text "By: manoj" and "This is a testing upload blog". A sunflower image is displayed. On the right side of the browser window, the developer tools Network tab is open, showing a single entry for the request "getBlogs". The entry details are: Status: 200, Type: fetch, Initiator: blogs.js:7, Size: 1.5 kB, and Time: 1.83 s.

Phase 3: Amazon Rekognition for Image Moderation

Step 1: Create DynamoDB Table

1. **Table Name:** MODERATION
2. **Partition Key:** imageld (String)
3. **Attributes:**
 - a. imageld (Primary Key)
 - b. labels (String)
 - c. Title(String)
 - d. Content (String)
 - e. UserId (String)
 - f. timestamp (String)

Moderation

Overview Indexes Monitor Global tables Backups Exports and streams Peri >

Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply. [Learn more](#)

General information [Info](#)

Partition key BlogId (String)	Sort key -	Capacity mode On-demand	Table status Active
Alarms No active alarms	Point-in-time recovery (PITR) Info Off	Resource-based policy Info Not active	

Items returned (17)

Actions Create item

BlogId (String) Content ImageUrl ModerationLabels Timestamp Title

Step 2: Create Lambda Function for Image Moderation

aws | [Search](#) [Alt+S] | [United States \(N. Virginia\)](#) | T MANOJ

Lambda > Functions > AWSRECOGNITION

AWSRECOGNITION

Function overview [Info](#) Throttle Copy ARN Actions

Diagram **Template** Export to Infrastructure Composer Download

AWSRECOGNITION Layers (0)

S3 + Add destination + Add trigger

Description
-

Last modified
6 hours ago

Function ARN
arn:aws:lambda:us-east-1:491085415620:function:AWSRECOGNITION

Function URL [Info](#)

1. Trigger:

- Set up S3 event trigger for blogdata-12 bucket.

Triggers (1) [Info](#)

[Fix errors](#) [Edit](#) [Delete](#) [Add trigger](#)

Find triggers < 1 >

Trigger

 S3: [blogdata-12](#)
arn:aws:s3:::blogdata-12

▼ Details

Bucket arn: arn:aws:s3:::blogdata-12
Event types: s3:ObjectCreated:*, s3:ObjectRemoved:*, s3:ObjectRestore:*, s3:ObjectTagging:*, s3:ObjectAcl:Put
isComplexStatement: No
Notification name: Rekognition
Prefix: None
Service principal: s3.amazonaws.com
Source account: 491085415620
Statement ID: 491085415620_event_permissions_from_blogdata-12_for_AWSRECOGNITION

2. Function Logic:

- a. Use Amazon Rekognition to scan uploaded images.
- b. If inappropriate content is detected:
 - i. Send email notifications to admin and user.
 - ii. Delete image from S3 and DynamoDB.
- c. Save moderation labels in MODERATION table.

<    |   : 1 of 640 < >

Blog Post Removed - Policy Violation Inbox x



AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾

17:22 (0 minutes ago)    

Dear manojcprime,

Your blog post 'fh' was removed due to inappropriate content: ['Weapons', 'Violence'].

--
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-1:491085415620:user-manojcprime-topic:b0f43d6c-7ec8-42ba-9758-82929513f5ef&Endpoint=dreamers2k22@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>

A screenshot of a database table titled "Items returned (23)". The table has columns: BlogId (String), Content, ImageUrl, ModerationLabels, Timestamp, and Title. One row is visible, showing values: 7238de6a-3e25-471a..., TEST BLOG ..., https://blo..., [{"S": "Weapons"}], 1740915198, TESTING. The table includes standard navigation buttons like back, forward, and search.

	BlogId (String)	Content	ImageUrl	ModerationLabels	Timestamp	Title
	7238de6a-3e25-471a...	TEST BLOG ...	https://blo...	[{"S": "Weapons"}]	1740915198	TESTING

Phase 4: Uploading Files, Route53, CloudFront, Certificate Manager

Step 1: Upload Code to S3

1. Buckets:

- a. www.manojconnects.space (public, static hosting enabled).
- b. manojconnects.space (private, redirects to www.manojconnects.space).

www.manojconnects.space [Info](#)

Objects | Metadata | **Properties** | Permissions | Metrics | Management | Access Points

Bucket overview

AWS Region
US East (N. Virginia) us-east-1

Amazon Resource Name (ARN)
 arn:aws:s3:::www.manojconnects.space

Creation date
February 28, 2025, 12:05:03 (UTC+05:30)

Bucket Versioning

[Edit](#)

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#) 

Bucket Versioning
Enabled

Multi-factor authentication (MFA) delete

manojconnects.space [Info](#)

Objects | Metadata | **Properties** | Permissions | Metrics | Management | Access Points

Bucket overview

AWS Region
US East (N. Virginia) us-east-1

Amazon Resource Name (ARN)
 arn:aws:s3:::manojconnects.space

Creation date
March 1, 2025, 14:46:28 (UTC+05:30)

Bucket Versioning

[Edit](#)

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#) 

Bucket Versioning
Disabled

Multi-factor authentication (MFA) delete

2. Bucket Policies:

- Allow public read access for [www.manojconnects.space](#).

{

"Version": "2012-10-17",

"Statement": [

```
{
    "Sid": "PublicReadGetObject",
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::www.manojconnects.space/*"
}
]
}
```

Step 2: Configure Route53

1. Hosted Zone:

- Create a hosted zone for manojoconnects.space.

The screenshot shows the 'Create hosted zone' configuration page in the AWS Route 53 console. The domain name is set to 'manojoconnects.space'. The description 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 at the top and bottom.

2. Records:

- Create A records pointing to CloudFront distributions.

Records (2) DNSSEC signing Hosted zone tags (0)

Records (2) [Info](#) [Delete record](#) [Import zone file](#) [Create record](#)

Automatic mode is the current search behavior optimized for best filter results. [To change modes go to settings.](#)

Filter records by property or value [Type](#) [Routing p...](#) [Alias](#) [Value/Route traffic to](#) [TTL \(s...\)](#)

	Record ...	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...)
<input type="checkbox"/>	manojcon...	NS	Simple	-	No	ns-76.awsdns-09.com. ns-670.awsdns-19.net. ns-1073.awsdns-06.org. ns-1772.awsdns-29.co.uk.	172800
<input type="checkbox"/>	manojcon...	SOA	Simple	-	No	ns-76.awsdns-09.com. awsdn...	900

Nameservers changed!

Your nameservers has been changed to:

ns-1073.awsdns-06.org

ns-1772.awsdns-29.co.uk

ns-670.awsdns-19.net

ns-76.awsdns-09.com

 It might take up to 24 hours for the domain to propagate to the new nameservers.

[Close](#)

Step 3: Create SSL Certificates

1. Certificate Manager:

- Request certificates for manojconnects.space and www.manojconnects.space.

The screenshot shows the AWS Certificate Manager interface. At the top, there's a large identifier: **58f67379-27c0-4f5f-bf58-4d30c33e71cb**. To the right is a blue **Delete** button. Below this, under **Certificate status**, there are details: Identifier (**58f67379-27c0-4f5f-bf58-4d30c33e71cb**), Status (**Issued** with a green checkmark), ARN (**arn:aws:acm:us-east-1:491085415620:certificate/58f67379-27c0-4f5f-bf58-4d30c33e71cb**), and Type (**Amazon Issued**). In the middle section, titled **Domains (2)**, there are two entries: **manojconnects.space** and **www.manojconnects.space**. To the right of the domains are buttons for **Create records in Route 53** and **Export to CSV**. Below the domains, there's a table header with columns: Domain, Status, Renewal status, Type, and CNAME. The table has one row corresponding to each domain.

Step 4: Configure CloudFront

1. Distributions:

- Create two distributions (one for each bucket).
- Set alternate domain names and SSL certificates.
- Set origin to the S3 static hosting URL.

The screenshot shows the AWS CloudFront **Distributions** page. The distribution ID is **E3IEXP8NXTDX4S**. The **General** tab is selected. In the **Details** section, the distribution domain name is **db4fz6oavxoja.cloudfront.net**, ARN is **arn:aws:cloudfront::491085415620:distribution/E3IEXP8NXTDX4S**, and last modified status is **Deploying**. In the **Settings** section, the description is empty, price class is **Use all edge locations (best performance)**, alternate domain names include **manojconnects.space** (with a green checkmark), custom SSL certificate is set to **www.manojconnects.space** (with a green checkmark), standard logging is off, and cookie logging is off. At the bottom, there are links for **CloudShell** and **Feedback**, and a footer with copyright information: **© 2025, Amazon Web Services, Inc. or its affiliates.** and links for **Privacy**, **Terms**, and **Cookie preferences**.

Screenshot of the AWS CloudFront Distribution Details page for distribution E17MGWVGDHZW4G.

General Tab:

- Distribution domain name:** d2pyvme90h42uo.cloudfront.net
- ARN:** arn:aws:cloudfront:491085415620:distribution/E17MGWVGDHZW4G
- Last modified:** Deploying

Settings Tab:

- Description:** -
- Alternate domain names:** www.manojconnects.space
- Custom SSL certificate:** www.manojconnects.space
- Standard logging:** Off
- Cookie logging:** Off

Record name: subdomain manojconnects.space

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.

Record type: 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: Alias to CloudFront distribution

An alias to a CloudFront distribution and another record in the same hosted zone are global and available only in US East (N. Virginia).

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

Buttons: Cancel, Define simple record

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	.manoconnects.space
-----	---------------------

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 CloudFront distribution	▼
US East (N. Virginia)	▼

An alias to a CloudFront distribution and another record in the same hosted zone are global and available only in US East (N. Virginia).

<input type="text"/> d2pyvme90h42uo.cloudfront.net	<input type="button" value="X"/>
--	----------------------------------

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

[Cancel](#) [Define simple record](#)

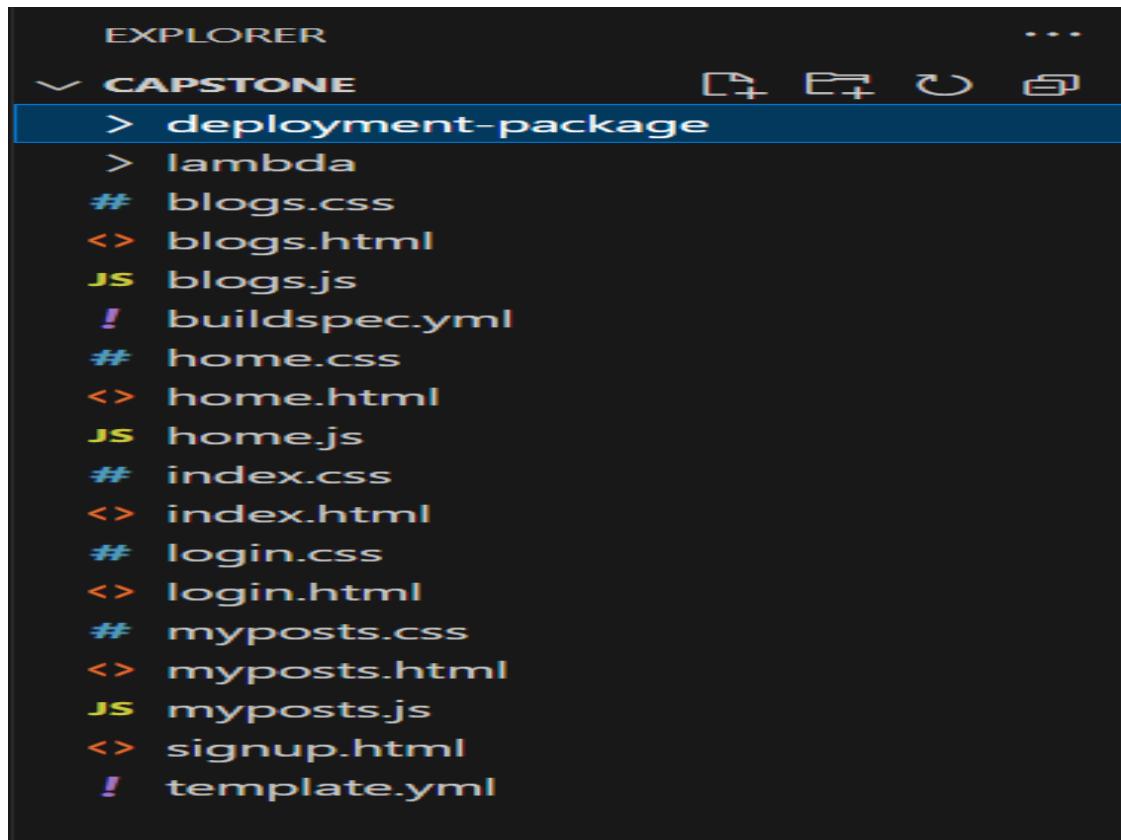
	Record ...	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...)
<input type="checkbox"/>	manocon...	A	Simple	-	Yes	db4fz6oavxoja.cloudfront.net.	-
<input type="checkbox"/>	manocon...	NS	Simple	-	No	ns-76.awsdns-09.com. ns-670.awsdns-19.net. ns-1073.awsdns-06.org. ns-1772.awsdns-29.co.uk.	172800
<input type="checkbox"/>	manocon...	SOA	Simple	-	No	ns-76.awsdns-09.com. awsdn...	900
<input type="checkbox"/>	_883d34b...	CNAME	Simple	-	No	_185578de8e03c4ec86feeca...	300
<input type="checkbox"/>	www.man...	A	Simple	-	Yes	d2pyvme90h42uo.cloudfron...	-
<input type="checkbox"/>	_714a3db...	CNAME	Simple	-	No	_bb0d692a7063d3d4126f3d...	300

Phase 5: CodeDeploy and CodePipeline

Step 1: Create GitHub Repository

1. Push Code:

- a. Upload all project files (HTML, CSS, JS, Lambda code) to GitHub.



```
● PS C:\Users\T Manoj\Downloads\CAPSTONE> git add .
② PS C:\Users\T Manoj\Downloads\CAPSTONE> git commit -m "commits"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
● PS C:\Users\T Manoj\Downloads\CAPSTONE> git push -u origin main
branch 'main' set up to track 'origin/main'.
Everything up-to-date
○ PS C:\Users\T Manoj\Downloads\CAPSTONE> █
```

Step 2: Create Buildspec File

1. **File Name:** buildspec.yml
2. **Content:**

a. Define build commands for Lambda deployment.

YML :

```
version: 0.2
```

```
phases:
```

```
install:
  runtime-versions:

    python: 3.11
    commands:
      - echo "Checking AWS CLI installation..."
      - if ! command -v aws &> /dev/null; then pip install awscli; else echo "AWS CLI already installed"; fi
```

```
pre_build:
  commands:
    - echo "Ensuring lambda folder exists..."
    - mkdir -p lambda # Ensure the lambda folder exists
```

```
build:
```

```
  commands:
    - echo "Zipping Lambda function folders..."
    - cd lambda
    - for dir in *; do
        if [ -d "$dir" ]; then
            zip -r "$dir.zip" "$dir";
        fi
```

```
done
```

```
  - cd ..
```

```
post_build:
  commands:
    - echo "Deploying website to S3..."
    - aws s3 sync . s3://www.manojconnects.space --exclude "lambda/*" --exclude ".git/*" --exclude "node_modules/*" --delete
```

```

- echo "Deploying Lambda function ZIPs to S3..."
- for zipfile in lambda/*.zip; do
    aws s3 cp "$zipfile" "s3://www.manojconnects.space/$zipfile";
done

- echo "Updating Lambda functions..."
- for function in AWSREKOGNITION uploadBlog getMyPosts getBlogs INSERTLOGIN
GETDETAILS; do
    aws lambda update-function-code --function-name "$function" --s3-bucket
www.manojconnects.space --s3-key "lambda/$function.zip" || echo "Failed to update $function";
done

- echo "Deployment completed successfully!"
```

```

artifacts:
  files:
    - '**/*'
  base-directory: .
```

Step 3: Configure CodeBuild

1. **Project:**
 - a. Connect to GitHub repository.
 - b. Select Lambda runtime.
 - c. Auth the Github Credentials and select the repo.
 - d. Mention the buildspec.yml code
 - e. Create the build

Build projects [Info](#)

[Actions ▾](#) [Create trigger](#) [View details](#) [Start build ▾](#) [Create project](#)

[Your projects ▾](#) ◀ 1 ▶

Name	Source provider	Repository	Latest build status	Description	Last Modified
BLOG	GitHub	manojmanu94 41/Serverless- Blog- Platform-with- Content- Moderation	Succeeded	-	1 day ago

[Developer Tools](#) > [CodeBuild](#) > [Build projects](#) > [BLOG](#)

BLOG

[Actions ▾](#) [Create trigger](#) [Edit](#) [Clone](#) [Debug build](#) [Start build with overrides](#) [Start build](#)

Configuration

Source provider	Primary repository	Artifacts upload location	Service role
GitHub	manojmanu9441/Serverless- Blog-Platform-with-Content- Moderation	-	arn:aws:iam::491085415620: role/service-role/AWS-BLOG- CICD
Public builds	Disabled		

Build history Batch history Project details Build triggers Metrics

Build history G Stop build View artifacts View logs Delete builds Retry build

< 1 2 3 4 5 ... > ⚙️

<input type="checkbox"/>	Build run	Status	Build number	Source version	Submitter	Duration	Comp
<input type="checkbox"/>	BLOG:895f7 f60-913c- 44a7-a462- c342b5fb3 17	✔️ Succeeded	107	arn:aws:s3::: codepipeline e-us-east-1- 754620458 484/BLOGCI CD/BuildArt if/hWLyk12	codepipeline e/BLOGCIC D	35 seconds	6 hours

⌚ **Build started**
You have successfully started the following build: BLOG:c5979346-9440-442b-83fa-1e49787fb947

Developer Tools > CodeBuild > Build projects > BLOG > BLOG:c5979346-9440-442b-83fa-1e49787fb947

BLOG:c5979346-9440-442b-83fa-1e49787fb947

Stop build Retry build

Build status

Status	Initiator	Build ARN
✔️ Succeeded	root	🔗 arn:aws:codebuild:us-east-1:491085 415620:build/BLOG:c5979346-9440-44 2b-83fa-1e49787fb947
Resolved source version	Start time	End time
8d322d5988f72edf8096b8baf4c6af0ed e2fb139	Mar 2, 2025 5:36 PM (UTC+5:30)	Mar 2, 2025 5:36 PM (UTC+5:30)

Showing the last 354 lines of the build log. [View entire log](#)

[Tail logs](#)

No previous logs

```

1 [Container] 2025/03/02 12:06:18.543657 Running on CodeBuild On-demand
2 [Container] 2025/03/02 12:06:18.543666 Waiting for agent ping
3 [Container] 2025/03/02 12:06:19.748307 Waiting for DOWNLOAD_SOURCE
4 [Container] 2025/03/02 12:06:22.777110 Phase is DOWNLOAD_SOURCE
5 [Container] 2025/03/02 12:06:23.136107
CODEBUILD_SRC_DIR=/codebuild/output/src4260246881/src/github.com/manojmanu9441/Serverless-Blog-Platform-with-
Content-Moderation
6 [Container] 2025/03/02 12:06:23.138132 YAML location is
/codebuild/output/src4260246881/src/github.com/manojmanu9441/Serverless-Blog-Platform-with-Content-
Moderation/buildspec.yml
7 [Container] 2025/03/02 12:06:23.140387 Setting HTTP client timeout to higher timeout for Github and GitHub
Enterprise sources
8 [Container] 2025/03/02 12:06:23.140514 Processing environment variables
9 [Container] 2025/03/02 12:06:23.947617 Selecting 'python' runtime version '3.11' based on manual selections...
10 [Container] 2025/03/02 12:06:23.948336 Running command echo "Installing Python version 3.11 ..."
11 Installing Python version 3.11 ...
12
13 [Container] 2025/03/02 12:06:23.955921 Running command pyenv global $PYTHON_311_VERSION
14
15 [Container] 2025/03/02 12:06:24.715062 Moving to directory
/codebuild/output/src4260246881/src/github.com/manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation
16 [Container] 2025/03/02 12:06:24.715086 Cache is not defined in the buildspec

```

Step 4: Create CodePipeline

1. Pipeline:

- Add source (GitHub) and select the github repo .
- Add the Build (CodeBuild), and deploy (Lambda) stages.
- Create the service role with necessary permission[lambda,code build,code deploy]
- Add the Build stage and test stage .
- Create the code Pipeline.

The screenshot shows the 'Choose creation option' step of the AWS CodePipeline 'Create new pipeline' wizard. The page title is 'Choose creation option' with an 'Info' link. It indicates 'Step 1 of 7'. On the left, there's a sidebar with steps: Step 1 (Choose creation option), Step 2 (Choose pipeline settings), 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 content area has a 'Category' section with three options: 'Deployment' (radio button), 'Continuous Integration' (radio button), and 'Build custom pipeline' (radio button, which is selected and highlighted with a blue border). At the bottom right are 'Cancel' and 'Next' buttons.

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-west-2:491085415620:connection/7cd8 or Connect to GitHub

Repository name

Choose a repository in your GitHub account.

 manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation

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

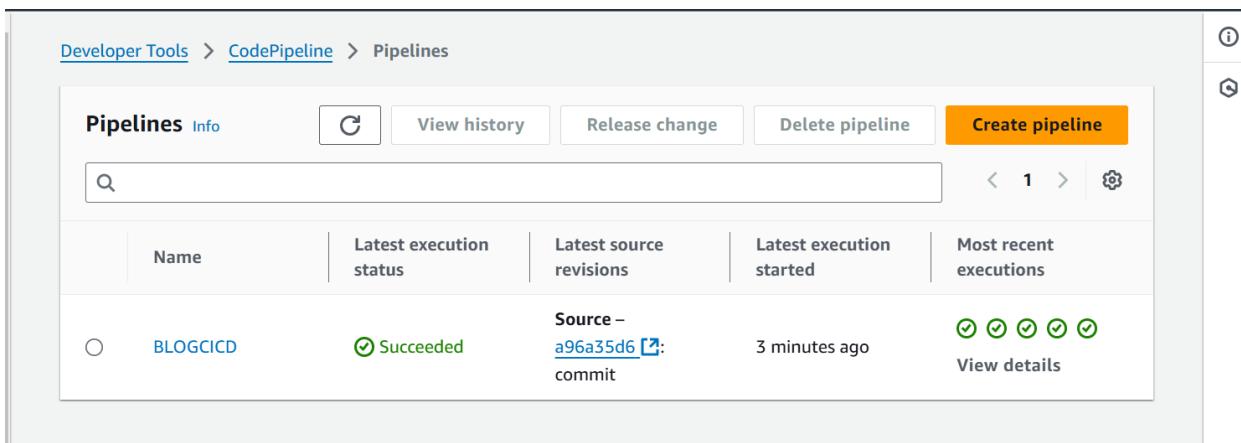
Default branch will be used only when pipeline execution starts from a different source or manually started.

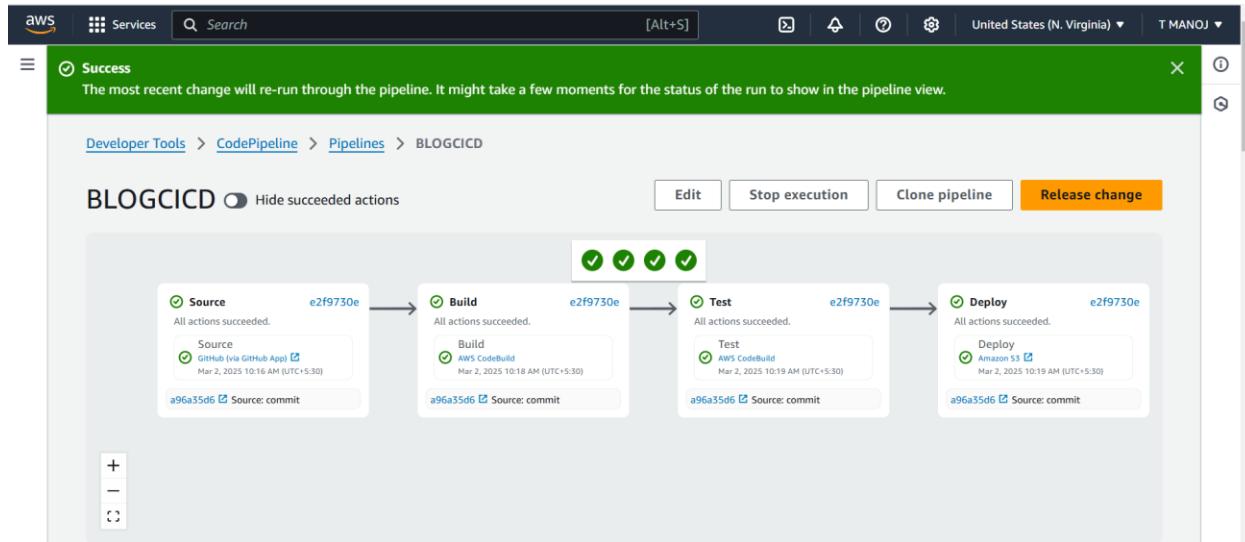
 main

Output artifact format

Choose the output artifact format.

CodePipeline default





Execution ID | Status | Source revisions | Trigger | Started | Duration | Completed

e2f9730e	Success	Source - a96a35d6 commit	StartPipelineExecution - root	Mar 2, 2025 10:16 AM (UTC+5:30)	2 minutes 13 seconds	Mar 2, 2025 10:19 AM (UTC+5:30)
96a15215	Success	Source - a96a35d6 commit	Commit a96a35d6 pushed in manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation/main commit	Mar 1, 2025 11:33 PM (UTC+5:30)	2 minutes 12 seconds	Mar 1, 2025 11:35 PM (UTC+5:30)
6cb43e0f	Success	Source - 9cf48616 commits	Commit 9cf48616 pushed in manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation/main commit	Mar 1, 2025 11:27 PM (UTC+5:30)	2 minutes 12 seconds	Mar 1, 2025 11:29 PM (UTC+5:30)

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

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.

BLOG 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 environment variable

Build type

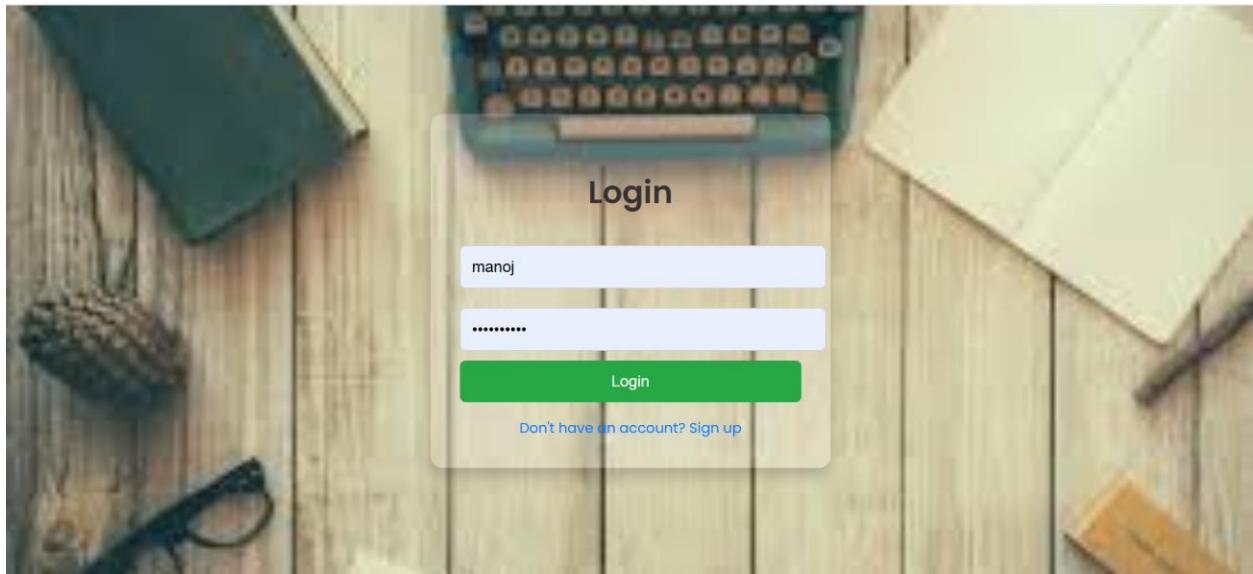
Single build
Triggers a single build. Batch build
Triggers multiple builds as a single execution.

Phase 6: Final Testing

1. **Test All Features:**
 - a. Signup, login, blog creation, image moderation, notifications.
2. **Check CloudFront and Route53:**
 - a. Ensure the application is accessible via the custom domain.

The screenshot shows the homepage of a blog platform. At the top, there's a header bar with browser controls and a URL bar showing "manojconnects.space". The main content area has a background image of a spiral notebook and a cup of coffee. A large blue title "Welcome to the Serverless AWS Blog Website" is centered. Below it is a section titled "About the Blog" with a paragraph of text: "This is a serverless blog platform built on AWS. It allows users to create, share, and explore blog posts seamlessly. Whether you're a writer or a reader, this platform is designed to provide a smooth and engaging experience." There's also a section titled "Content Moderation" with a similar paragraph: "We ensure a safe and respectful environment for all users. All blog posts are moderated to prevent spam, hate speech, and inappropriate content. Please adhere to our community guidelines when posting." A green button labeled "Create a Blog" is visible. The overall theme is professional and modern.

The screenshot shows a "Signup" form overlaid on a background image of office supplies like a red spiral notebook, a pen, and paperclips. The form has three input fields: a username field containing "manoj", an email field containing "dreamers2k22@gmail.com", and a password field with masked input. A green "Signup" button is at the bottom. Below the button, a link says "Already have an account? Login". The form is clean and user-friendly, set against a light blue background.



Create a New Blog.

Logout

OUTPUT TESTING BLOG

THIS IS BLOG POSTED FOR TESTING

Choose file download.png

Upload Post

Show Posts

My Posts

All Blog Posts

[Back to Home](#)

OUTPUT TESTING BLOG

By: manoj

THIS IS BLOG POSTED FOR TESTING



My Blog Posts

[Back to Home](#)

Your Blogs

OUTPUT TESTING BLOG

THIS IS BLOG POSTED FOR TESTING



IMAGE MODERATION : BLOG POSTED WITH THIS IMAGE



 AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾

18:03 (0 minutes ago)    

Dear manoj,

Your blog post 'TESTING FOR IMAGE MODERATION' was removed due to inappropriate content: ['Weapons', 'Violence'].

...

Items returned (23)   

< 1 >  

<input type="checkbox"/>	BlogId (String)	Content	ImageUrl	ModerationLabels	Timestamp	Title
<input type="checkbox"/>	7238de6a-3e25-471a...	TEST BLOG ...	https://blo...	[{ "S": "Weapons" }, ...	1740915198	TEST

Phase 7: CloudFormation Template

1. **Template:**
 - a. Define all resources (DynamoDB, Lambda, S3, API Gateway, etc.) in a CloudFormation template.

The template will include:

2. **DynamoDB Tables:** LOGIN and BLOGS.
3. **S3 Bucket:** blogdata-12.
4. **Lambda Functions:** INSERTLOGIN, GETDETAILS, uploadBlog, getBlogs, getMyPosts.
5. **API Gateway:** REST API with resources and methods.

6. IAM Roles: For Lambda execution permissions.

The screenshot shows the AWS IAM 'Create access key' page. At the top, there's a green success message: 'Access key created' with the 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 this, a vertical navigation bar on the left lists 'Step 2 - optional' (Set description tag), 'Step 3' (Retrieve access keys), and 'Step 2 optional' (Set description tag again). The main area contains an 'Access key' section with fields for 'Access key' (AKIAJAXEVXYYTCDQFRLFWI) and 'Secret access key' (***** Show). Below this is an 'Access key best practices' section with a bulleted list: 'Never store your access key in plain text, in a code repository, or in code.', 'Disable or delete access key when no longer needed.', 'Enable least-privilege permissions.', and 'Rotate access keys regularly.' A link to 'best practices for managing AWS access keys' is provided at the bottom of this section. The bottom of the page includes standard AWS footer links: CloudShell, Feedback, © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

The screenshot shows a Windows command prompt window titled 'C:\Windows\System32\cmd.e'. The output of the 'aws configure' command is displayed:

```
Microsoft Windows [Version 10.0.22631.4890]
(c) Microsoft Corporation. All rights reserved.

C:\Users\T Manoj\Downloads>cd blog-app
C:\Users\T Manoj\Downloads\blog-app>cd ../
C:\Users\T Manoj\Downloads>cd CAPSTONE
C:\Users\T Manoj\Downloads\CAPSTONE>code .

C:\Users\T Manoj\Downloads\CAPSTONE>aws configure
AWS Access Key ID [*****]: AKIAJAXEVXYYTCDQFRLFWI
AWS Secret Access Key [*****]: n+6i3Zy6ei336qUkhw1dgJ/HgNAuqKb9jqxZSlFQ
Default region name [us-east-1]: us-east-1
Default output format [None]:
```

7. Outputs: API Gateway endpoint URLs, S3 bucket name, etc.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: CloudFormation template for CodeDeploy using existing S3 bucket
```

```

Parameters:
  ExistingS3Bucket:
    Type: String
    Default: "www.manojconnects.space"
    Description: "Existing S3 bucket to store deployment artifacts"

Resources:
  # IAM Role for CodeDeploy
  CodeDeployServiceRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: CodeDeployServiceRole
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - codedeploy.amazonaws.com
            Action:
              - sts:AssumeRole
      Policies:
        - PolicyName: CodeDeployPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - s3:GetObject
                  - s3>ListBucket
                Resource:
                  - !Sub "arn:aws:s3:::${ExistingS3Bucket}"
                  - !Sub "arn:aws:s3:::${ExistingS3Bucket}/*"
              - Effect: Allow
                Action:
                  - ec2:Describe*
                  - autoscaling:*
                Resource: "*"

  # Create a CodeDeploy Application
  CodeDeployApplication:
    Type: AWS::CodeDeploy::Application
    Properties:
      ApplicationName: MyCodeDeployApp

```

```

ComputePlatform: Server

# Create a CodeDeploy Deployment Group
CodeDeployDeploymentGroup:
  Type: AWS::CodeDeploy::DeploymentGroup
  Properties:
    ApplicationName: !Ref CodeDeployApplication
    DeploymentGroupName: MyDeploymentGroup
    ServiceRoleArn: !GetAtt CodeDeployServiceRole.Arn
    DeploymentConfigName: CodeDeployDefault.OneAtATime
    AutoRollbackConfiguration:
      Enabled: true
      Events:
        - DEPLOYMENT_FAILURE
    DeploymentStyle:
      DeploymentOption: WITH_TRAFFIC_CONTROL
      DeploymentType: IN_PLACE
    Ec2TagFilters:
      - Key: Name
        Value: CodeDeployInstance
        Type: KEY_AND_VALUE

Outputs:
  CodeDeployAppName:
    Description: "Name of the CodeDeploy application"
    Value: !Ref CodeDeployApplication
  DeploymentGroupName:
    Description: "Name of the CodeDeploy deployment group"
    Value: !Ref CodeDeployDeploymentGroup
  ExistingBucket:
    Description: "Using existing S3 bucket for deployment"
    Value: !Ref ExistingS3Bucket

```

8. Deploy:

- Use the template to automate resource creation.

```
C:\Users\T Manoj\Downloads\CAPSTONE>aws cloudformation create-stack --stack-name S3CodeDeployStack --template-body file:///C:/Users/T Manoj/Downloads/CAPSTONE/template.yml --capabilities CAPABILITY_NAMED_IAM
{
  "StackId": "arn:aws:cloudformation:us-east-1:491085415620:stack/S3CodeDeployStack/3f7b57f0-f5f7-11ef-bd67-12cafb6e945b"
}

C:\Users\T Manoj\Downloads\CAPSTONE>
```

Phase 8: VPC Configuration

1. Create VPC:

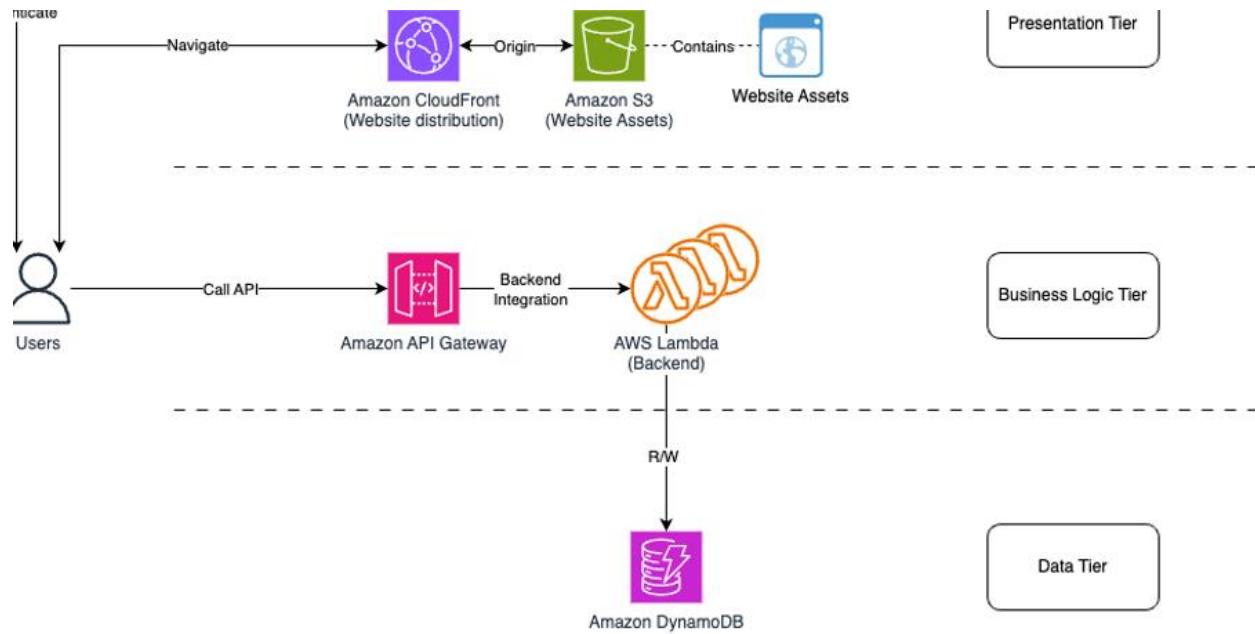
- Define subnets, route tables, and security groups.

Details	
VPC ID	vpc-054ade8bb3705318f
DNS resolution	Enabled
Main network ACL	acl-03a11d059e8aa6b48
IPv6 CIDR (Network border group)	-
State	Available
Tenancy	default
Default VPC	No
Network Address Usage metrics	Disabled
Block Public Access	<input checked="" type="checkbox"/> Off
DHCP option set	dopt-0887c831259938960
IPv4 CIDR	10.0.0.0/16
Route 53 Resolver DNS Firewall rule groups	-
DNS hostnames	Enabled
Main route table	rtb-0feb0f2b6f67cb054
IPv6 pool	-
Owner ID	491085415620

Place Resources:

- Move Lambda functions, DynamoDB, and other resources into the private subnet.

Final Infrastructure Design



VIII. Advantages

The Serverless Blog Application offers several advantages:

1. Scalability

- Automatically scales with the number of users and requests without manual intervention.

2. Cost-Effectiveness

- Pay only for the resources you use, with no upfront costs or idle resource charges.

3. High Availability

- Built on AWS's global infrastructure, ensuring high availability and fault tolerance.

4. Security

- Uses AWS IAM roles, encryption, and Amazon Rekognition for secure data handling and image moderation.

5. Ease of Maintenance

- Serverless architecture eliminates the need for server management, reducing operational overhead.

6. Rapid Development

- Leverages AWS managed services, enabling faster development and deployment.

IX. Use Cases

The Serverless Blog Application can be used in various scenarios:

Personal Blogging

- Individuals can use the application to create and manage their personal blogs.

Corporate Blogs

- Companies can use the application to share updates, news, and insights with employees or customers.

Educational Platforms

- Educators and students can use the application to share knowledge and collaborate on projects.

Community Forums

- Communities can use the application to create discussion forums and share content.

Image Moderation

- Platforms requiring image uploads can use the application's moderation feature to ensure content compliance.

X. Conclusion

The Serverless Blog Application is a powerful example of how AWS serverless technologies can be used to build modern, scalable, and cost-effective web applications. By leveraging services like Lambda, API Gateway, DynamoDB, S3, and Rekognition, the application provides a seamless user experience while minimizing operational overhead. The project also demonstrates best practices for security, scalability, and maintainability, making it an excellent reference for developers and organizations looking to adopt serverless architecture.

XI. GITHUB LINK

> <https://github.com/manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation.git>

XII. References

- <https://docs.aws.amazon.com/rekognition/latest/dg/moderation.html>
- <https://aws.amazon.com/blogs/compute/building-scalable-serverless-applications-with-amazon-s3-and-aws-lambda/>
- <https://numericaideas.com/blog/aws-serverless-web-application/>