cprime

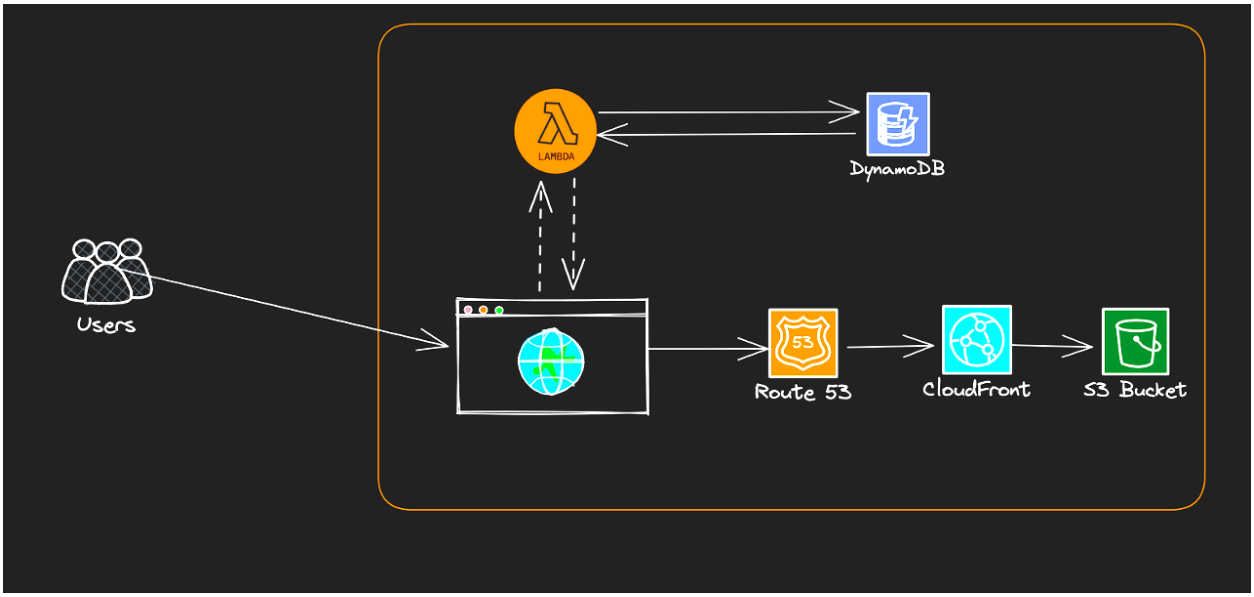
Serverless Blog Platform with Content Moderation

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

1. **AWS Account**: An active AWS account with necessary permissions to create and manage resources.
2. **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).

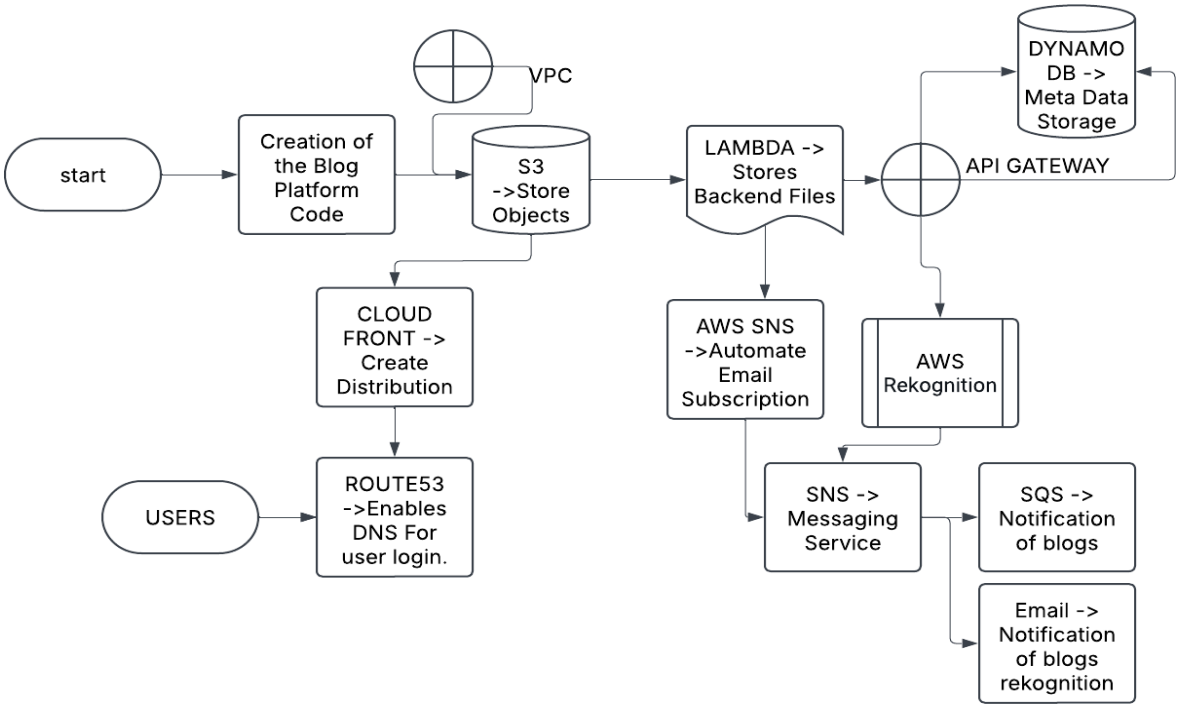
17.Git

- **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.

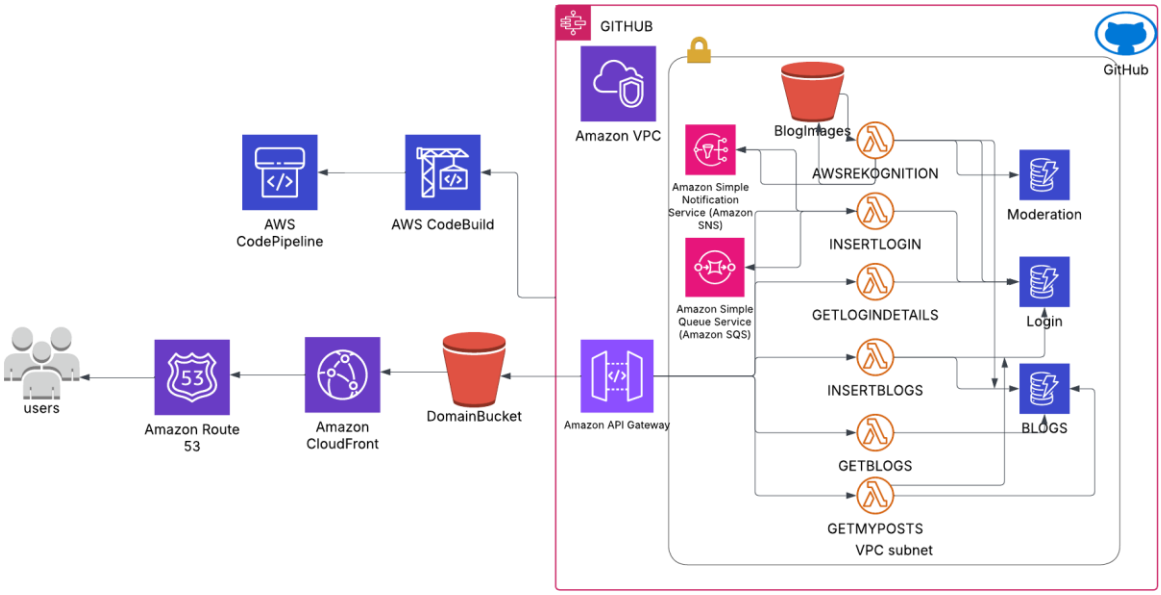
Summary of Services

Service	***Role in the Project***
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 :

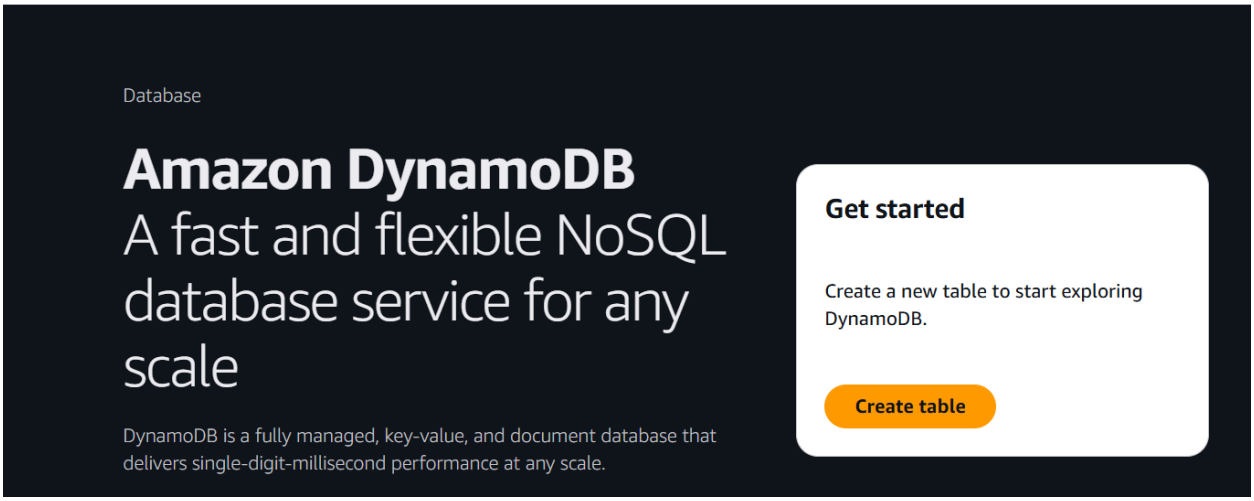


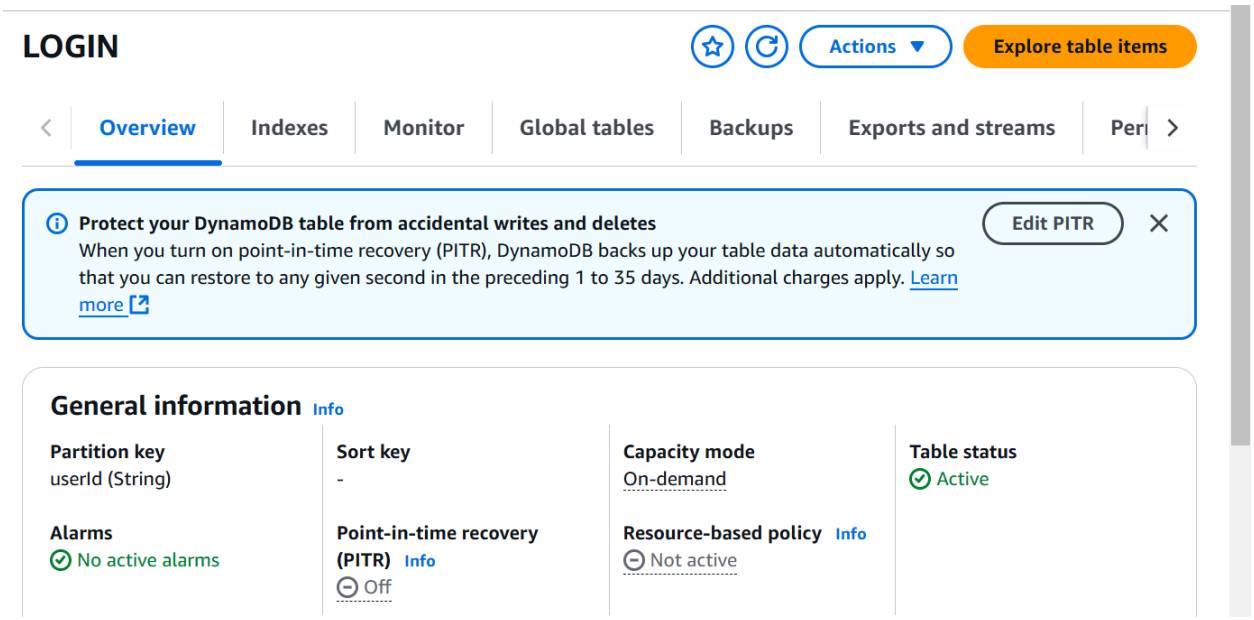
VII. Development Phases & Workflow

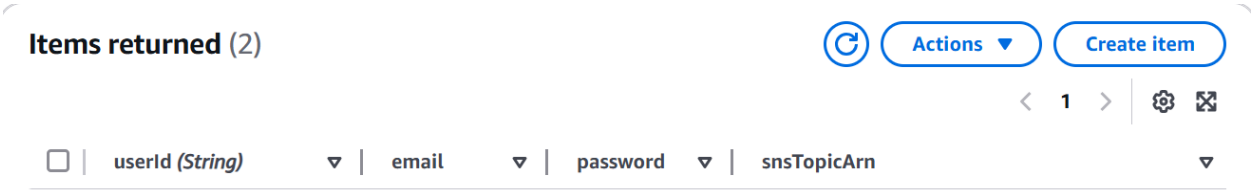
Phase 1: Login and Signup Page with Admin SNS, Admin SQS, User SNS, and DynamoDB

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.

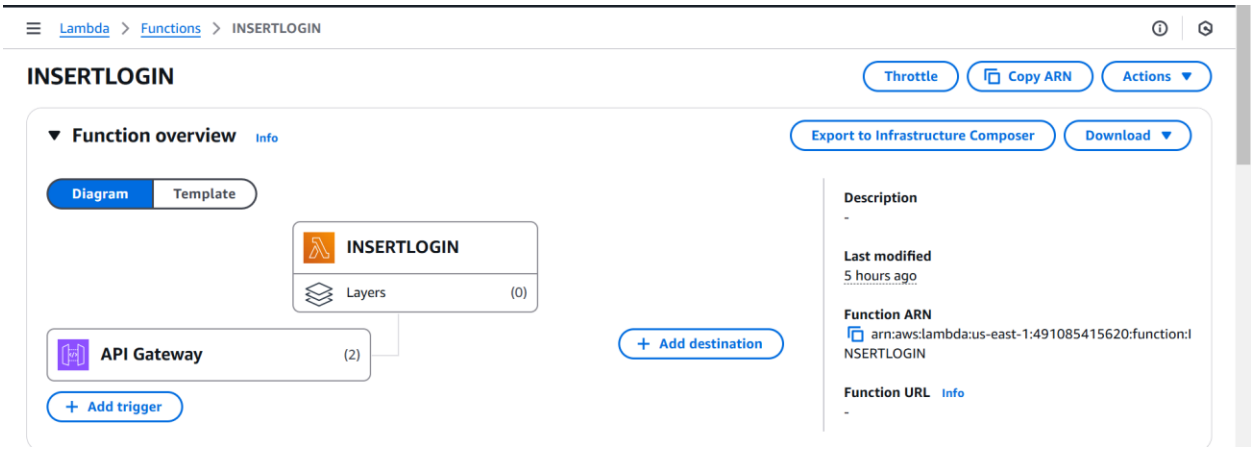


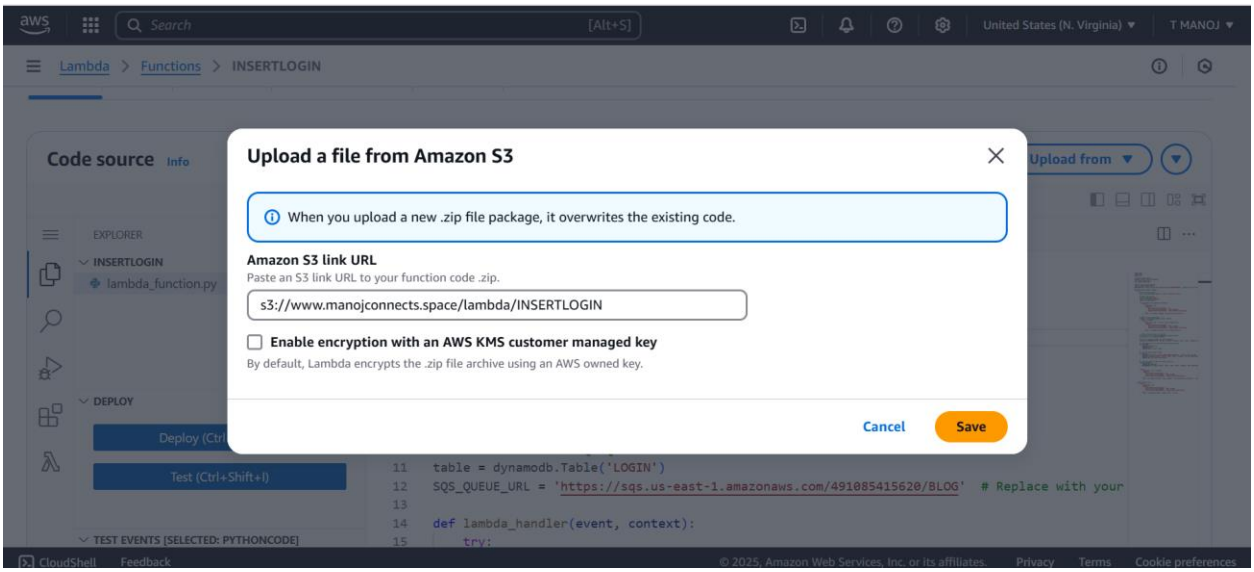


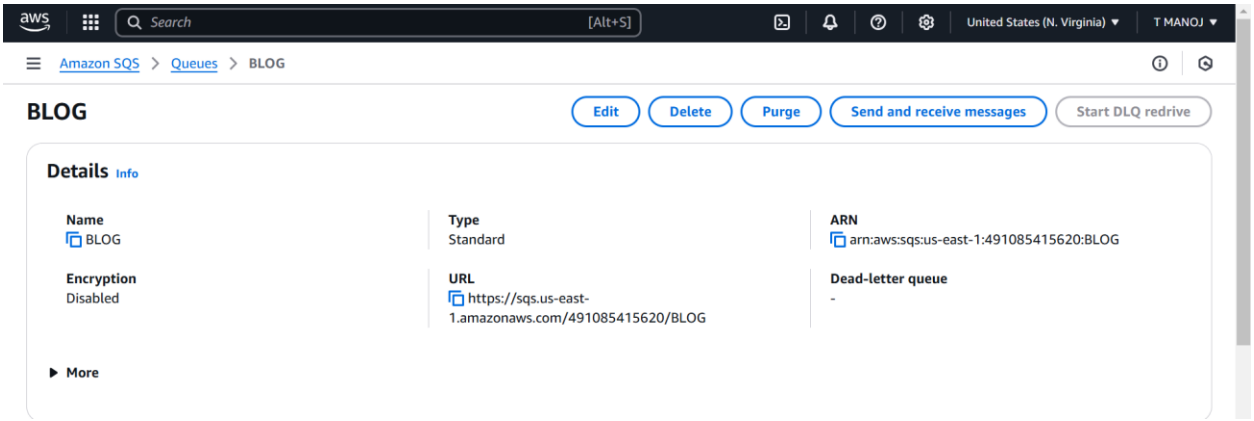


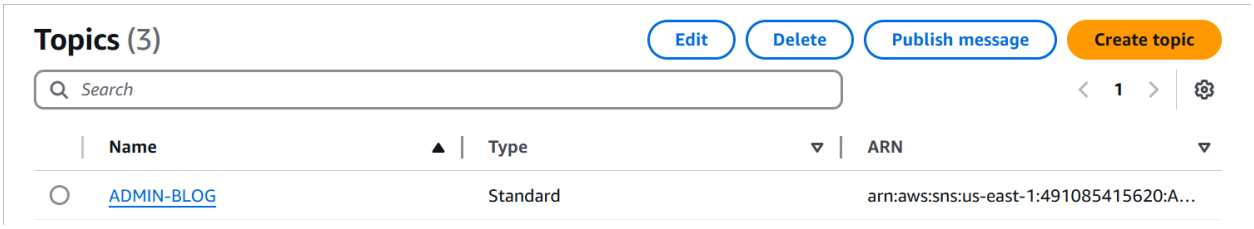
Step 2: Create Lambda Function INSERTLOGIN

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

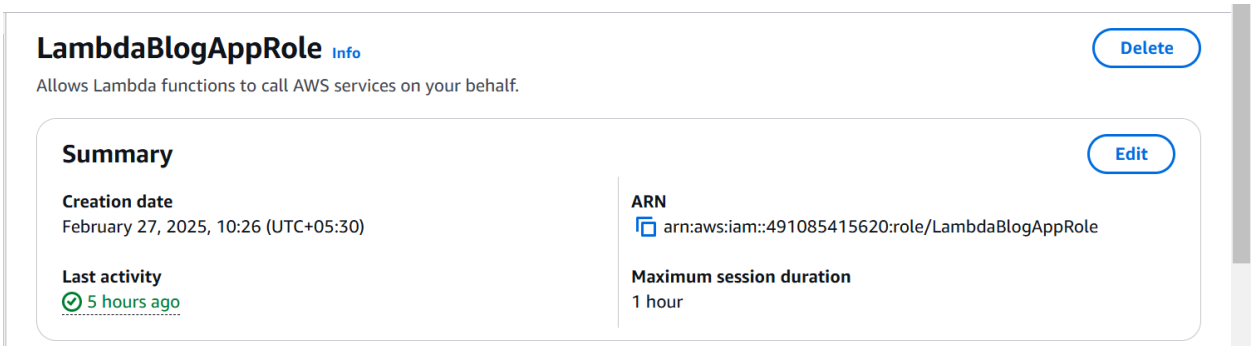

Lambda > Functions > INSERTLOGIN
INSERTLOGIN
Throttle
Copy ARN
Actions
Function overview Info
Export to Infrastructure Composer
Download
Diagram
Template
INSERTLOGIN
Layers (0)
API Gateway (2)
Add destination
Add trigger
Description
-
Last modified
5 hours ago
Function ARN
arn:aws:lambda:us-east-1:491085415620:function:INSERTLOGIN
Function URL Info
-


Lambda > Functions > INSERTLOGIN
Code source Info
Upload from
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
EXPLORER
INSERTLOGIN
lambda_function.py
DEPLOY
Deploy (Ctrl
Test (Ctrl+Shift+I)
TEST EVENTS [SELECTED: PYTHONCODE]
table = dynamodb.Table('LOGIN')
SQS_QUEUE_URL = 'https://sqs.us-east-1.amazonaws.com/491085415620/BLOG' # Replace with your
def lambda_handler(event, context):
try:
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Amazon SQS > Queues > BLOG
BLOG
Edit
Delete
Purge
Send and receive messages
Start DLQ redrive
Details Info
Name
BLOG
Type
Standard
ARN
arn:aws:sqs:us-east-1:491085415620:BLOG
Encryption
Disabled
URL
https://sqs.us-east-1.amazonaws.com/491085415620/BLOG
Dead-letter queue
-
More



2. **IAM Role**:
    a. Attach the following policies:
        i. AmazonDynamoDBFullAccess
        ii. AmazonSNSFullAccess
        iii. AmazonSQSFullAccess
        iv. AWSLambdaBasicExecutionRole



3. **Environment Variables**:
    a. ADMIN_SNS_TOPIC_ARN:

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

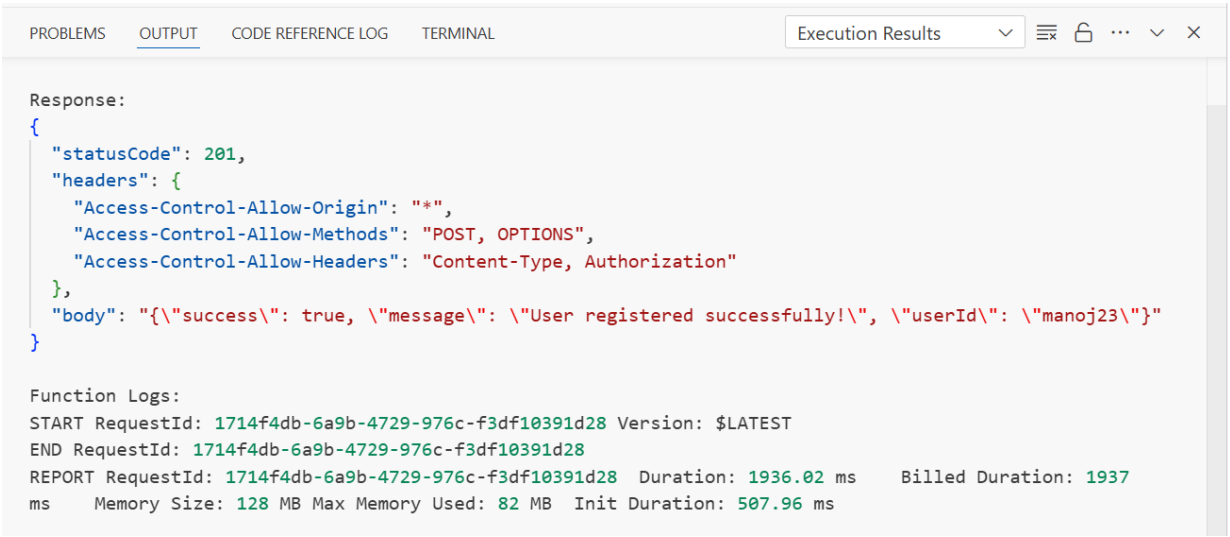
    b. ADMIN_SQS_QUEUE_URL:

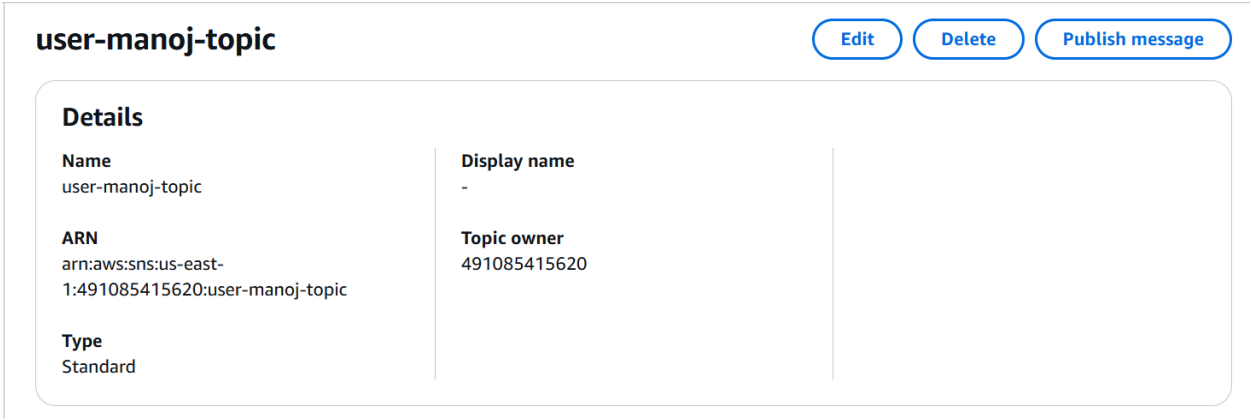
https://sqs.us-east-1.amazonaws.com/491085415620/BLOG.

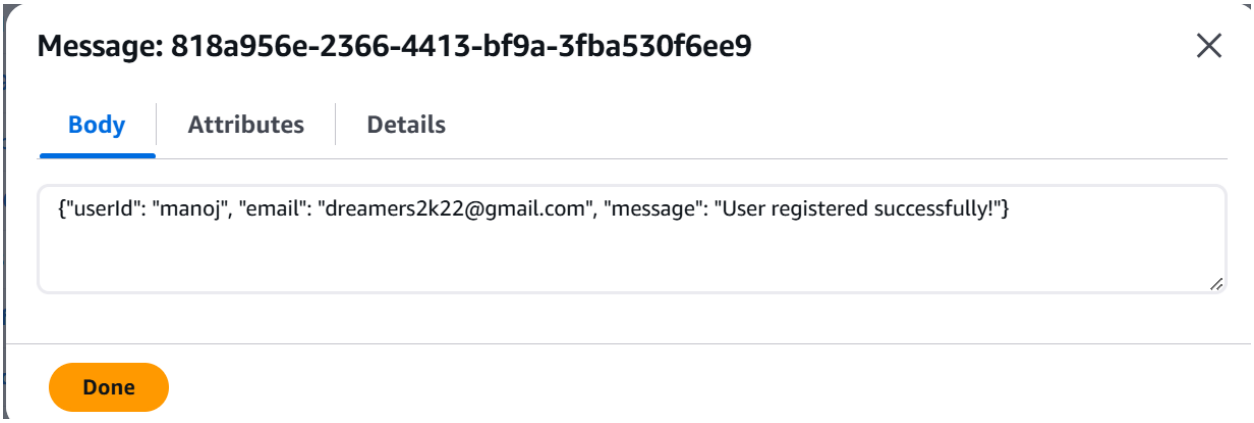
4. **Testing**:
    a. Create a test event with sample data:{
    "userId": "manoj",
    "email": "dreamers2k22@gmail.com",
    "password": "securepassword"

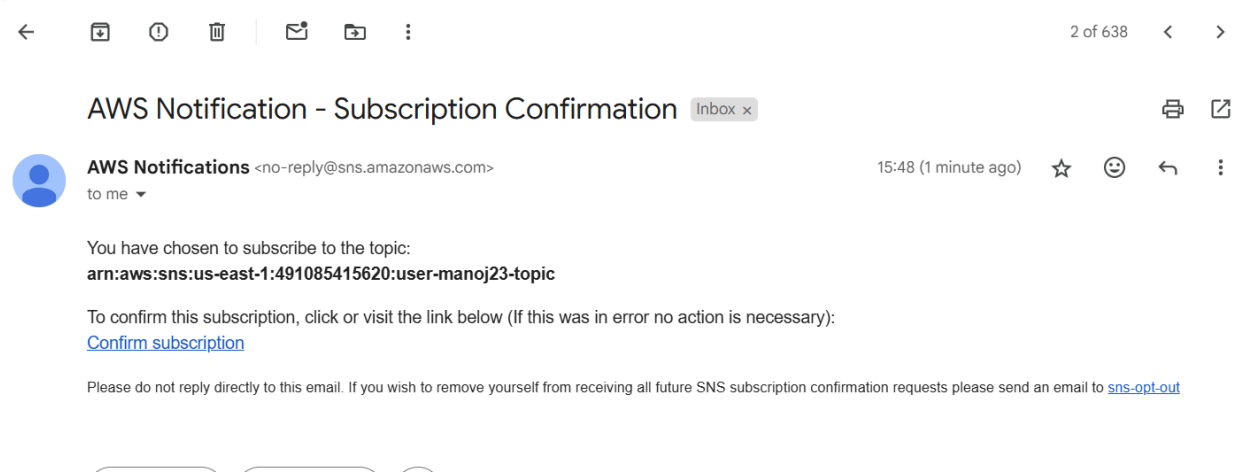
}

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









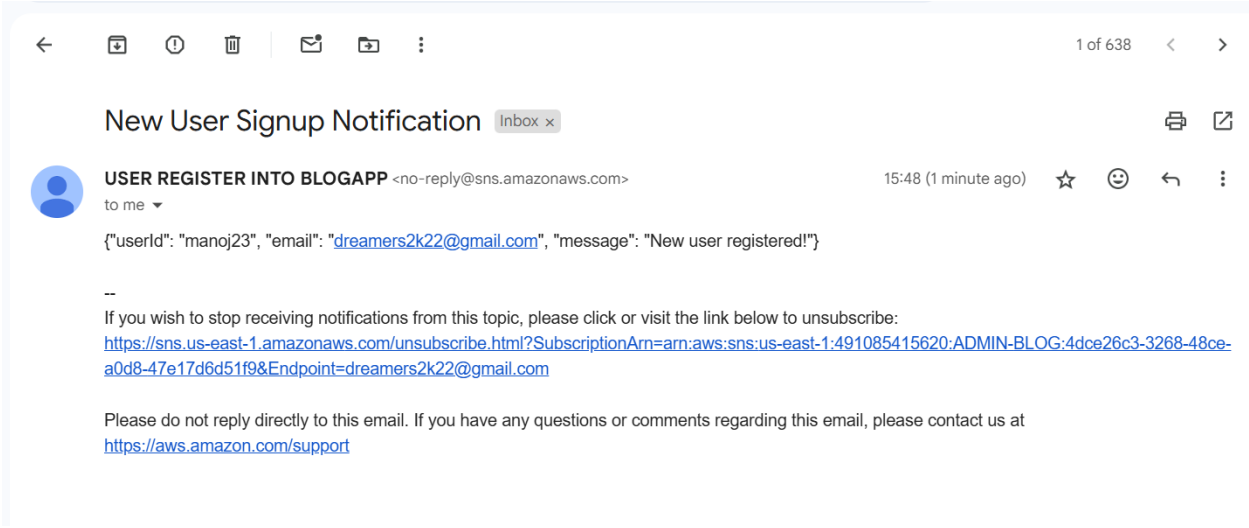
AWS Notification - Subscription Confirmation Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com> 15:48 (1 minute ago)
to me

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:491085415620:user-manoj23-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



New User Signup Notification Inbox ×

USER REGISTER INTO BLOGAPP <no-reply@sns.amazonaws.com> 15:48 (1 minute ago)
to me

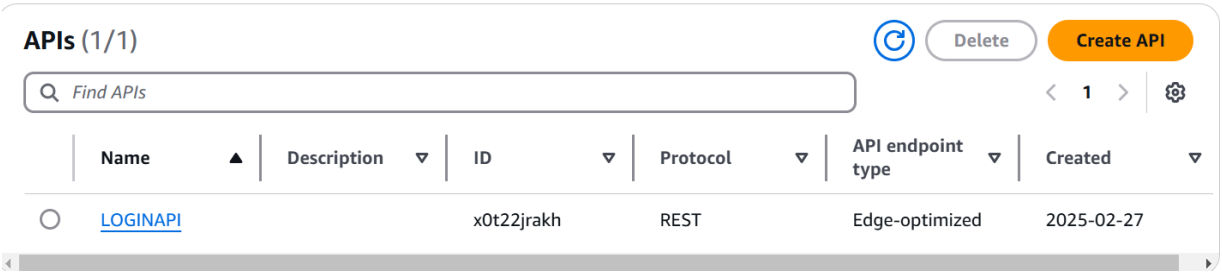
{"userId": "manoj23", "email": "dreamers2k22@gmail.com", "message": "New user registered!"}

--
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:ADMIN-BLOG:4dce26c3-3268-48ce-a0d8-47e17d6d51f9&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

Step 3: Create API Gateway LOGINAPI

1. **API Name**: LOGINAPI

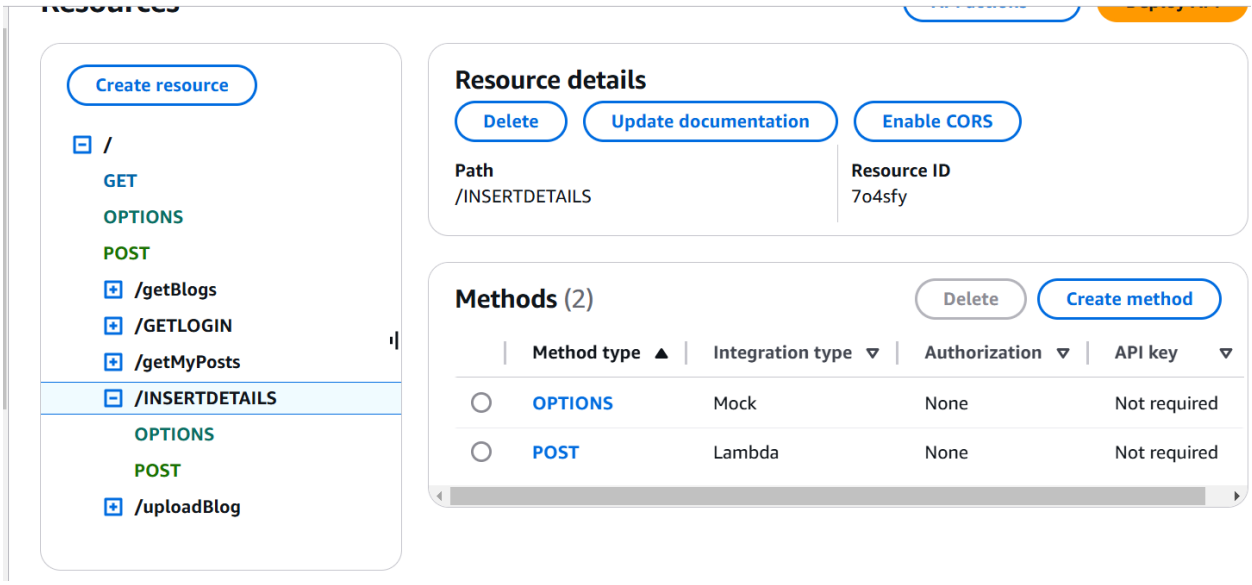


APIs (1/1) Delete Create API

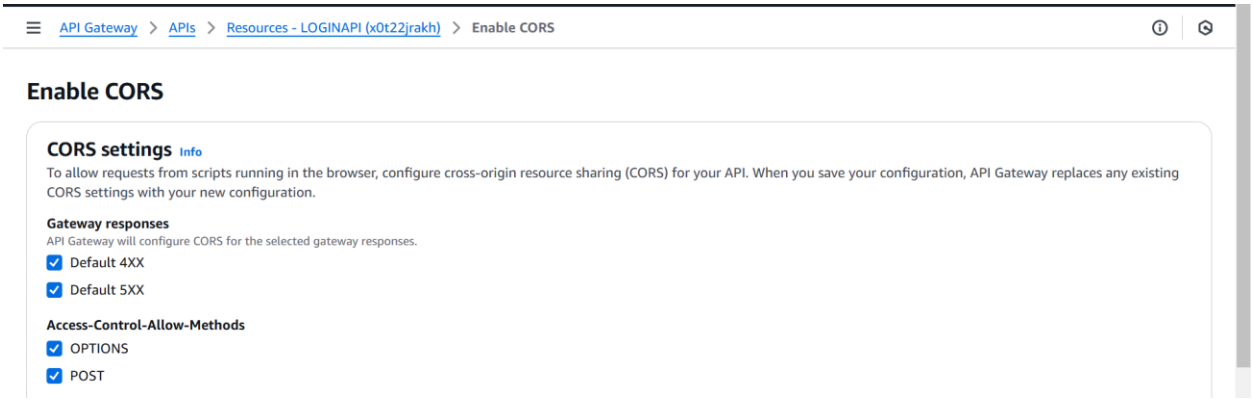
Find APIs 1

Name	Description	ID	Protocol	API endpoint type	Created
LOGINAPI		x0t22jrakh	REST	Edge-optimized	2025-02-27

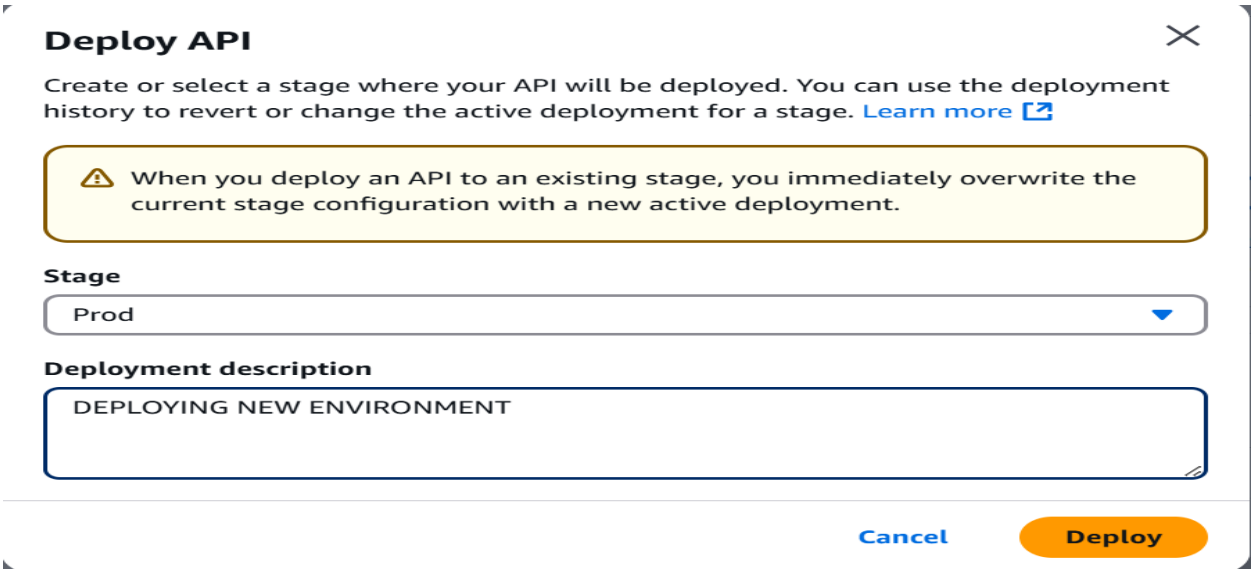
2. **Resource**: /INSERTLOGIN



3. **Integration**:
    a. Connect to the INSERTLOGIN Lambda function.
    b. Enable CORS with headers: Content-Type, Authorization, OPTIONS.
4. **Methods**:
    a. POST: For user registration.
    b. OPTIONS: For CORS preflight requests.

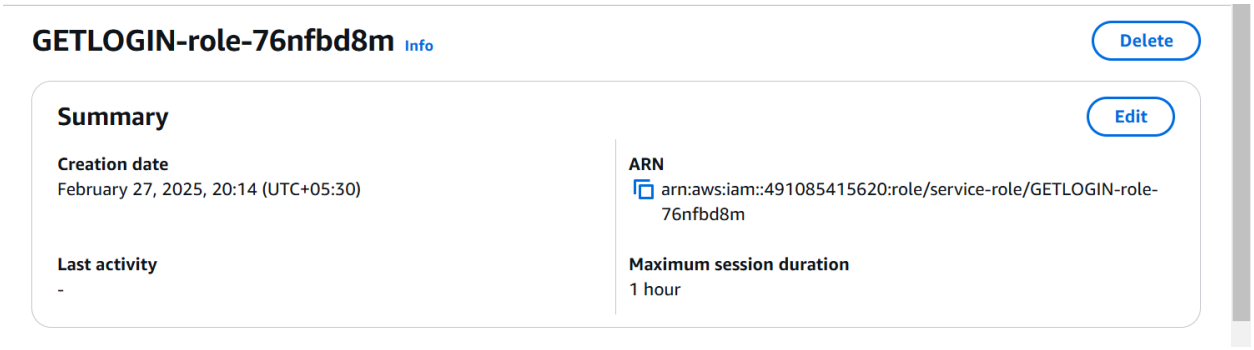


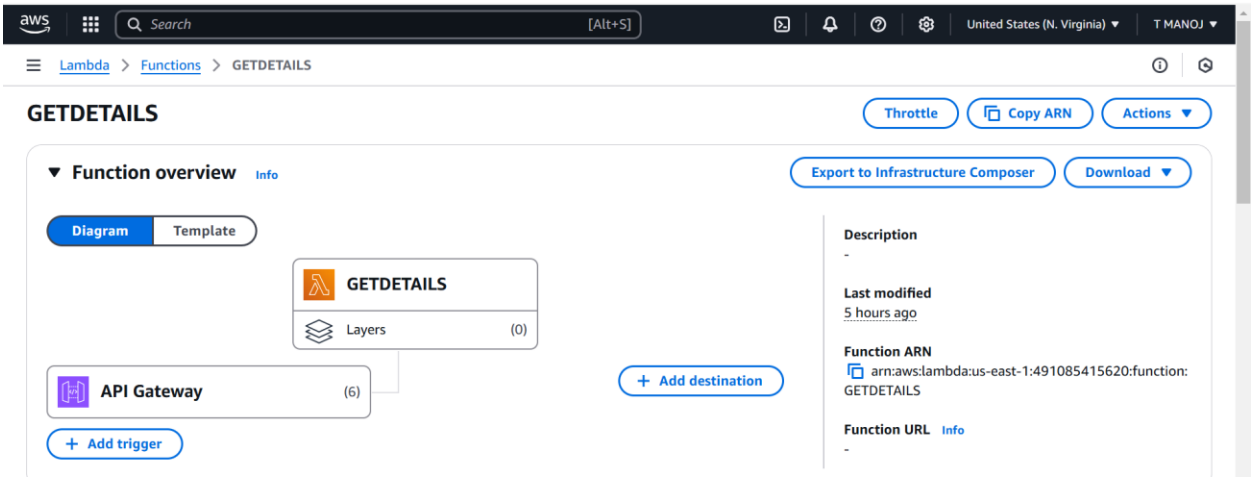
5. **Deploy API**:
    a. Create a deployment stage (e.g., prod).
    b. Note the API endpoint URL.

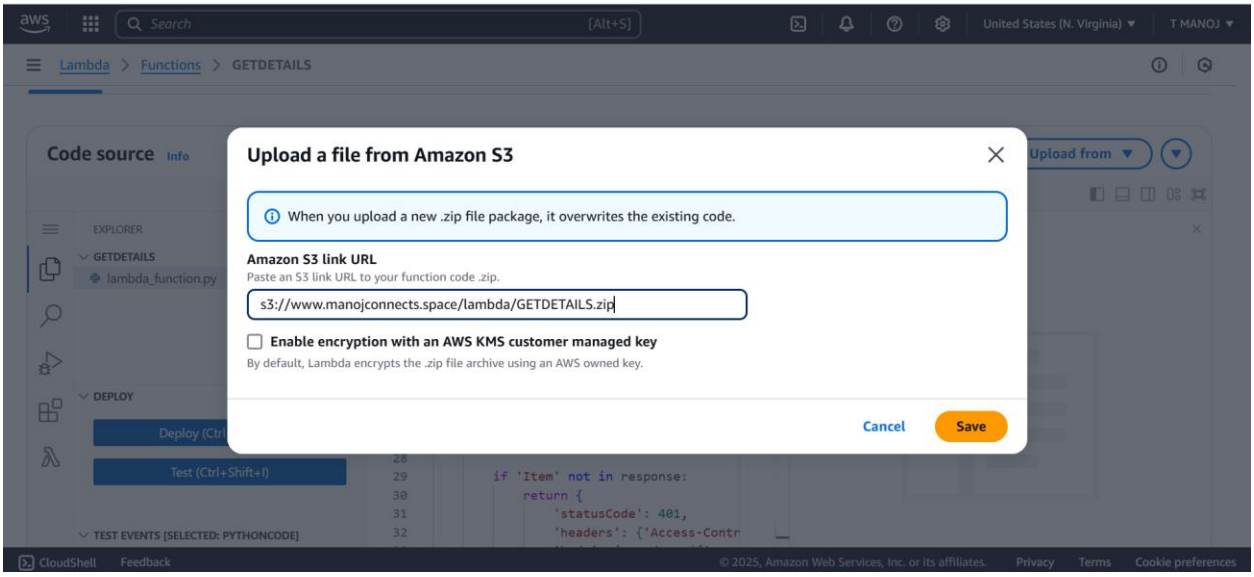


Step 4: Create Lambda Function GETLOGIN

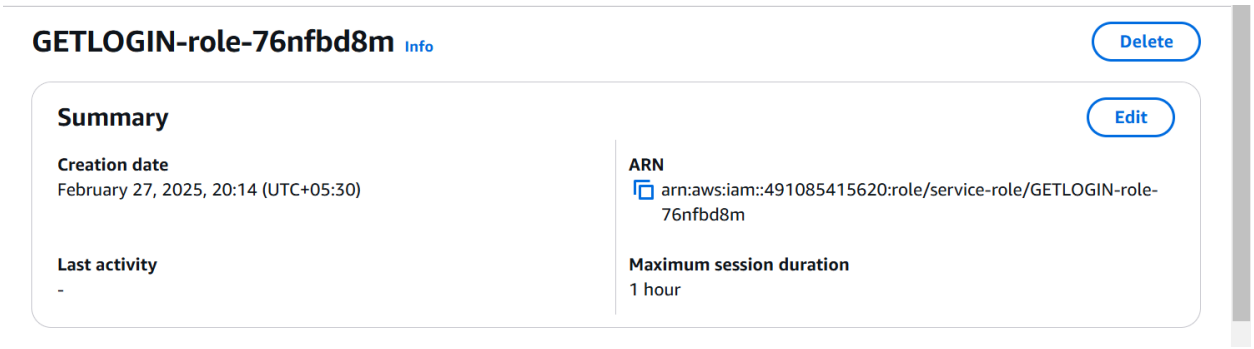
1. **Function Logic**:
    a. Extract userId and password from the API request.
    b. Query the LOGIN DynamoDB table using userId.
    c. Compare the provided password with the stored password.
    d. Return a success response if credentials match, else return an error.





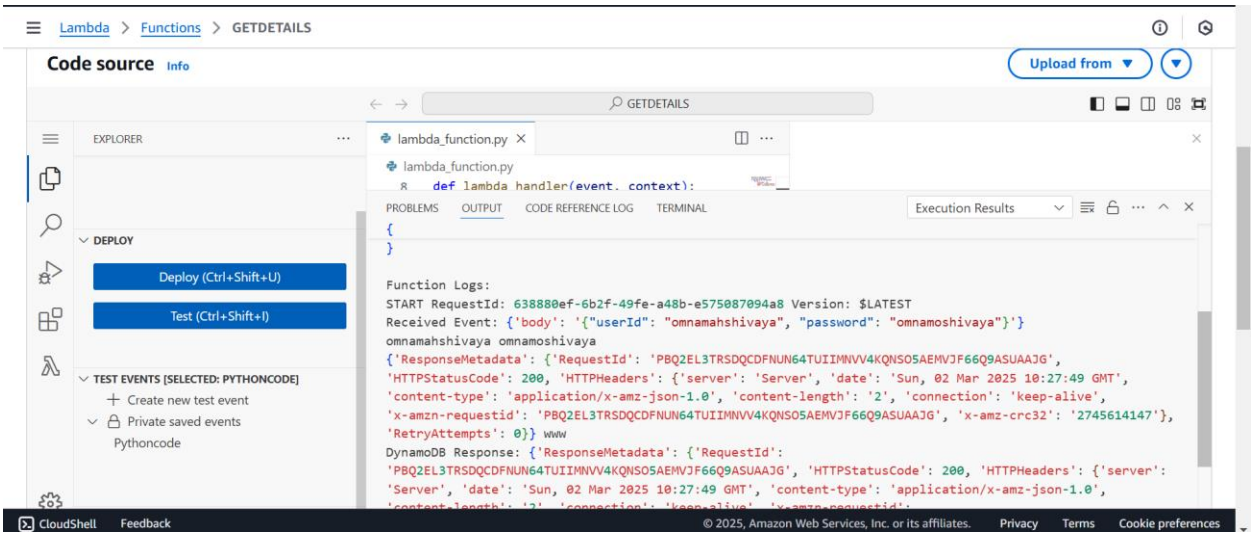


2. **IAM Role**:
    a. Attach the following policies:
        i. AmazonDynamoDBReadOnlyAccess
        ii. AWSLambdaBasicExecutionRole



3. **Testing**:
    a. Create a test event with sample data:{
    "userId": "omnamoshiavya",
    "password": "securepassword"
    }

    b. Verify the function logs and response.



Step 5: Add GETLOGIN Method to API Gateway

1. **Resource**: /GETDETAILS
2. **Integration**:
    a. Connect to the GETDETAILS Lambda function.
    b. Enable CORS with headers: Content-Type, Authorization, OPTIONS.
3. **Methods**:
    a. POST: For login validation.
    b. OPTIONS: For CORS preflight requests.
4. **Deploy API**:
    a. 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
GET	Lambda	None	Not required
OPTIONS	Mock	None	Not required
POST	Lambda	None	Not required

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.

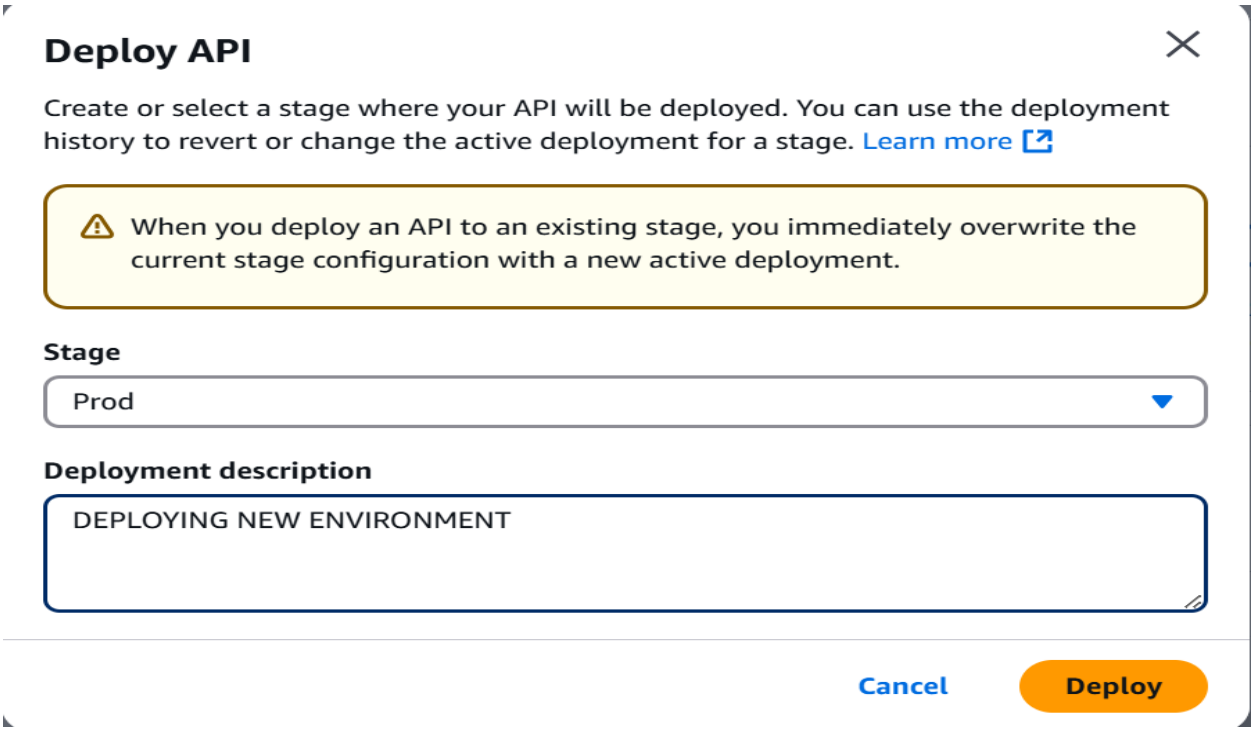
- [x] Default 4XX
- [x] Default 5XX

Access-Control-Allow-Methods

- [x] GET
- [x] OPTIONS
- [ ] POST

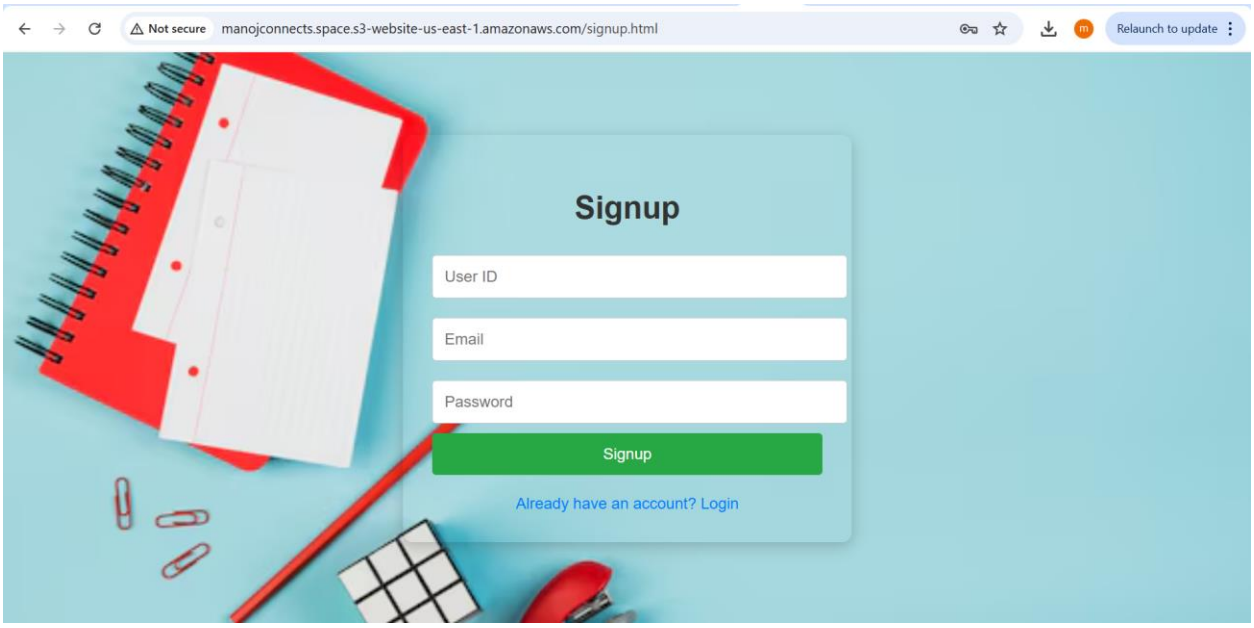
Access-Control-Allow-Headers
API Gateway will configure CORS for the selected gateway responses.

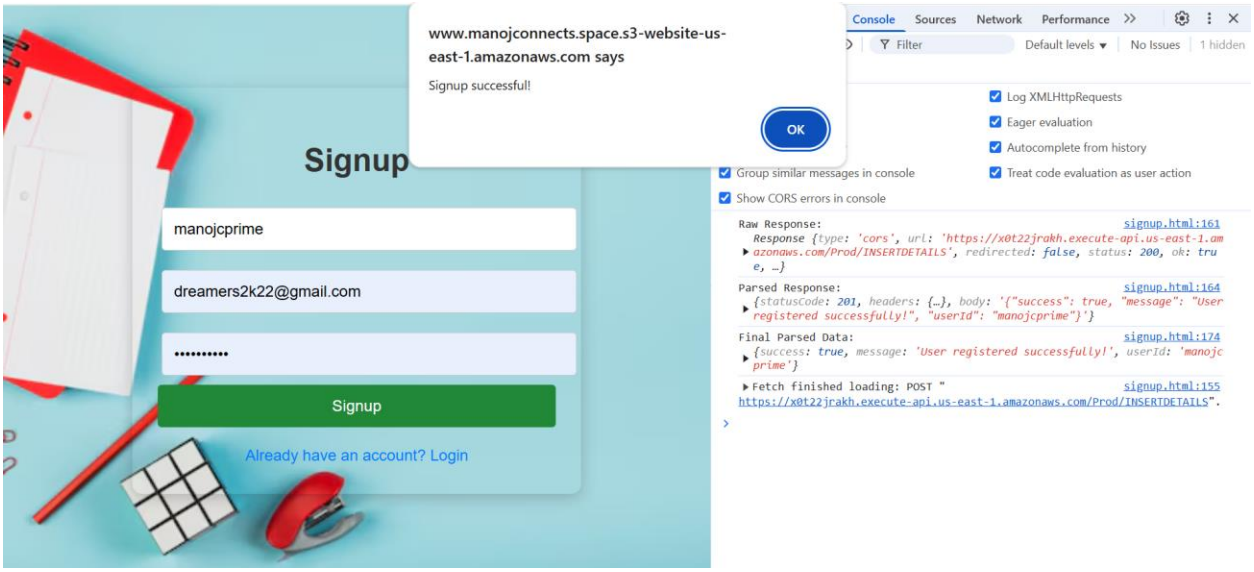
Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token



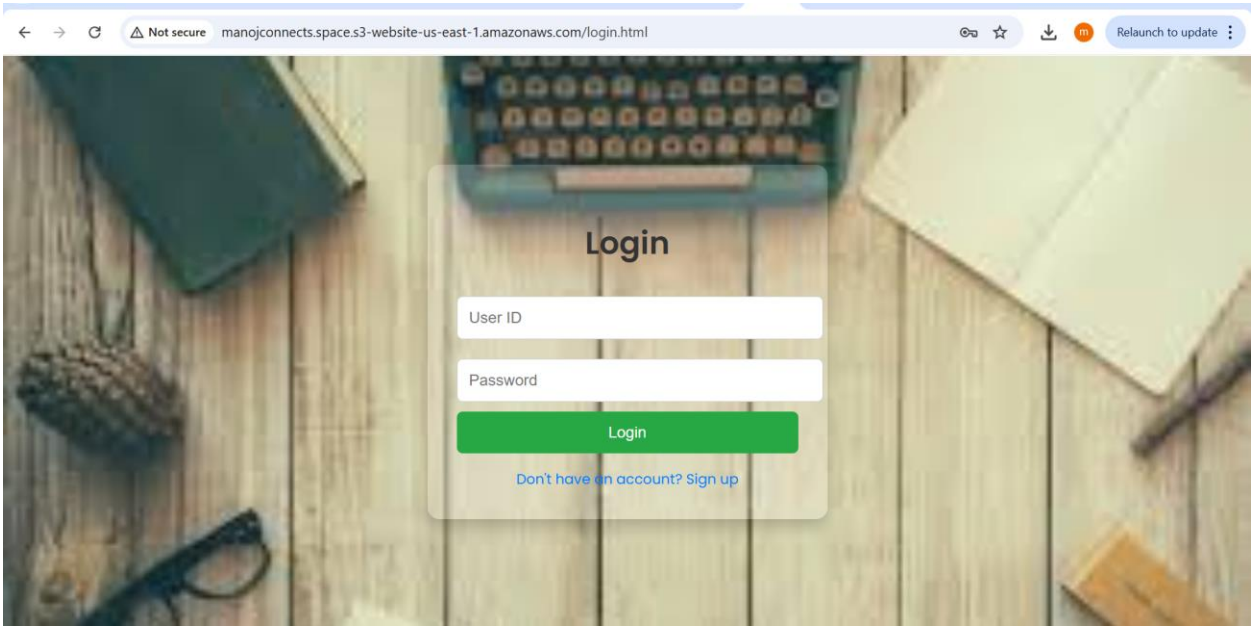
Step 6: Create Frontend (HTML, CSS, JavaScript)

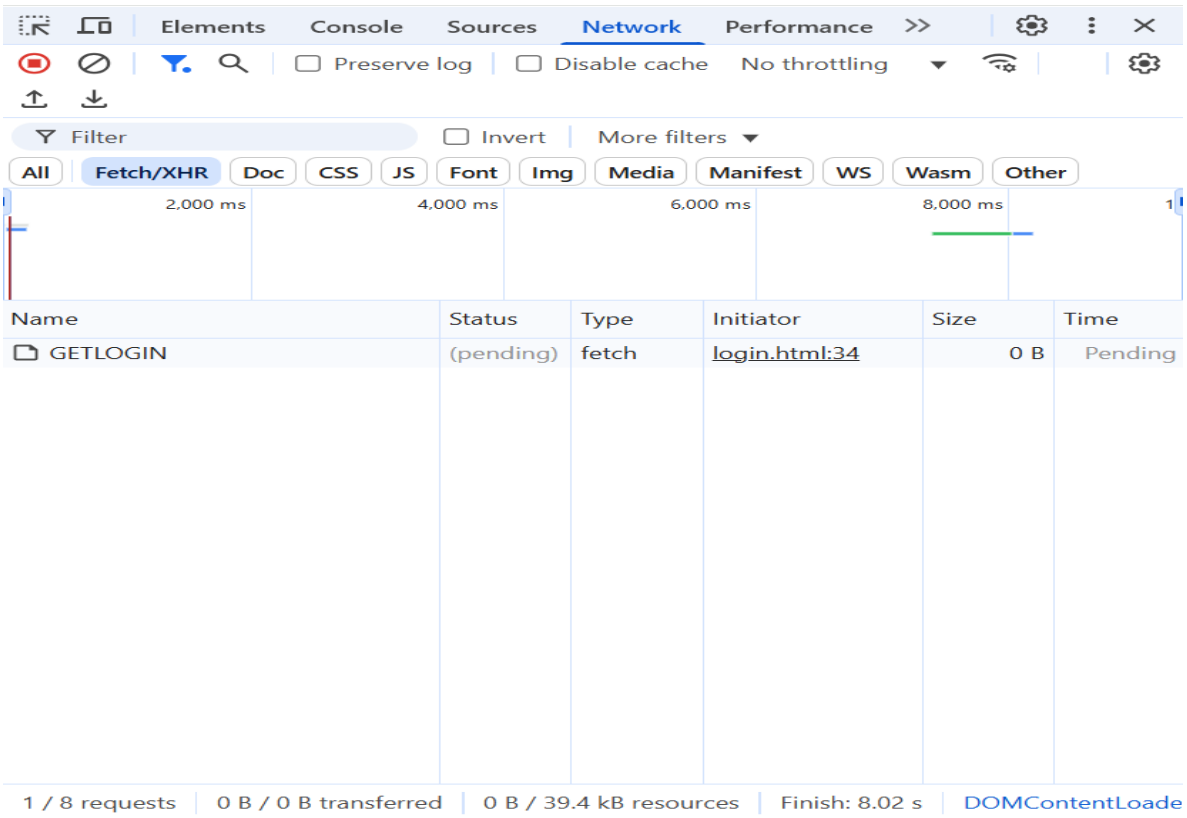
1. **Signup Page**:
    a. Form fields: userId, email, password.
    b. On submit, call /INSERTLOGIN API using fetch or axios.
    c. Display success/error messages based on the API response.



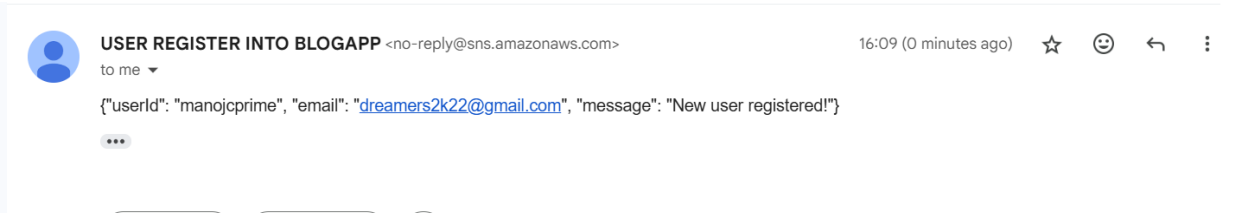


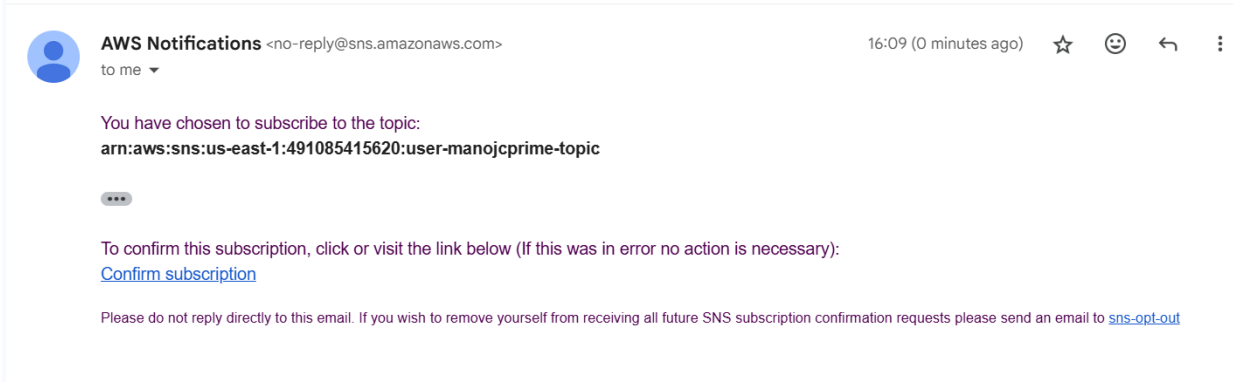
2. **Login Page**:
    a. Form fields: userId, password.
    b. On submit, call /GETDETAILS API.
    c. Redirect to the blog home page on success.
    d. Display an error message for invalid credentials.





3. **Notifications**:
    a. Ensure the user receives an email via SNS after signup.
    b. Ensure the admin receives user details via SNS and SQS.

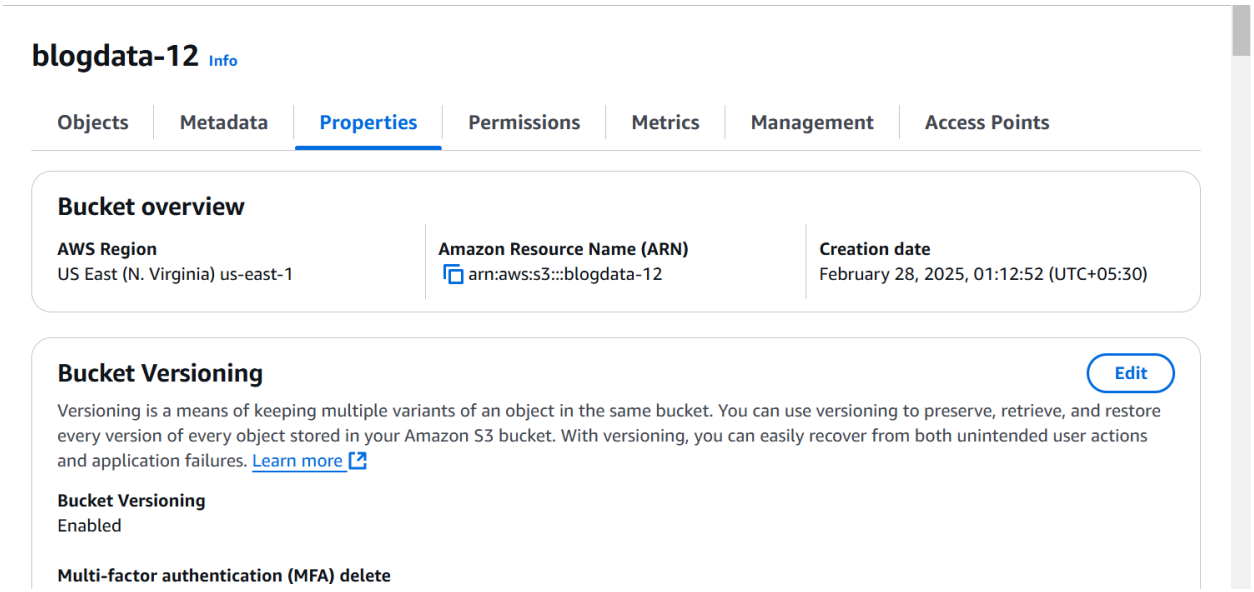




Phase 2: Blog Home Page, Blog Post Page, and Database

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.

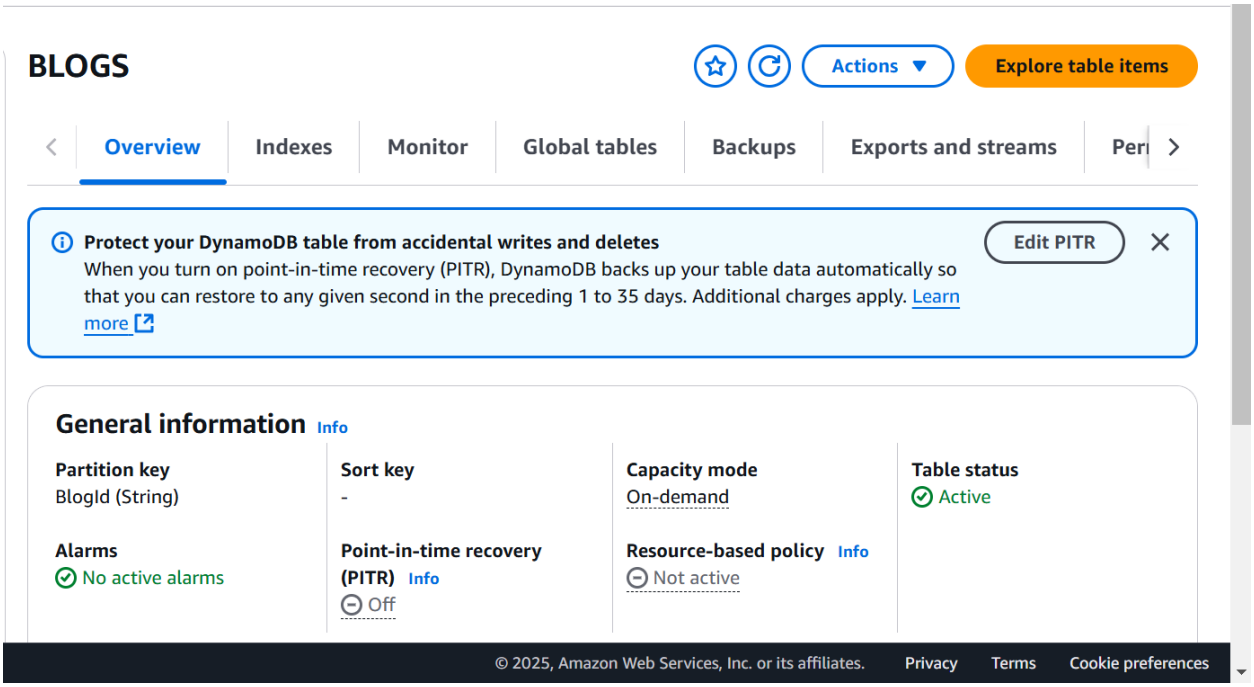


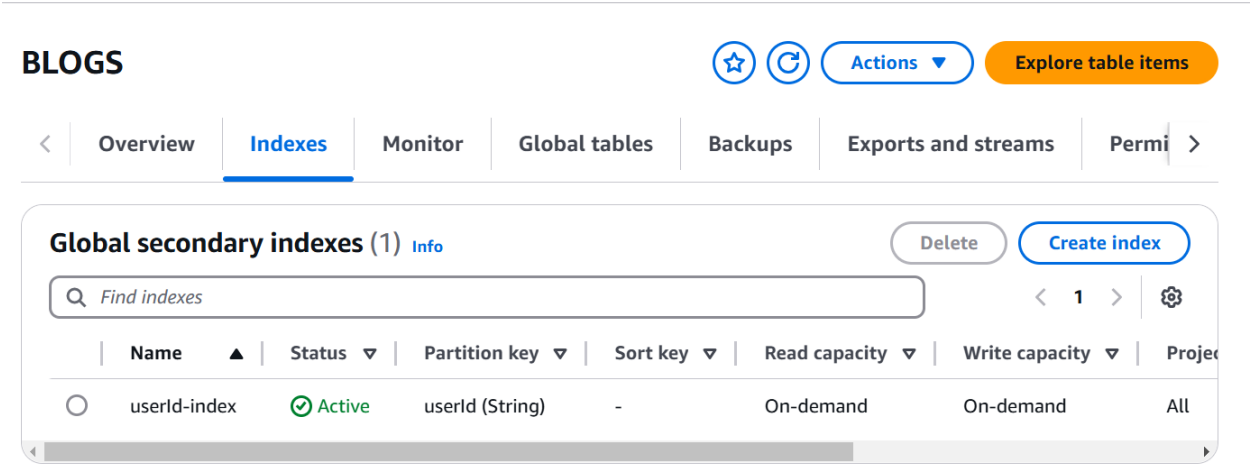
Bucket Policy :

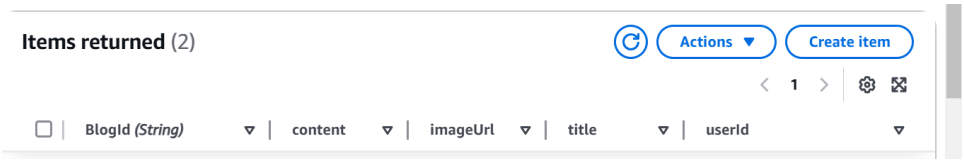
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rekognition.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::blogdata-12/blogs/*"
        },
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

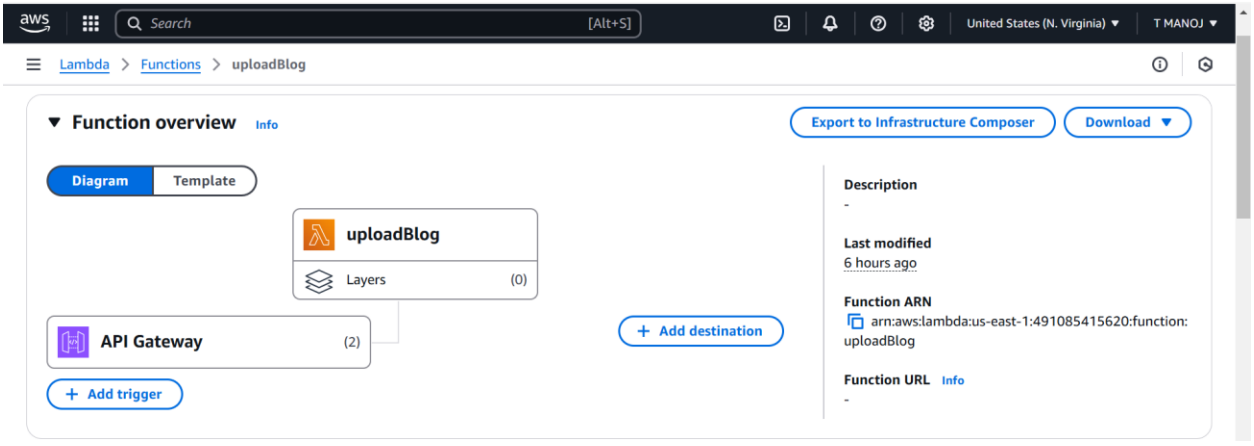


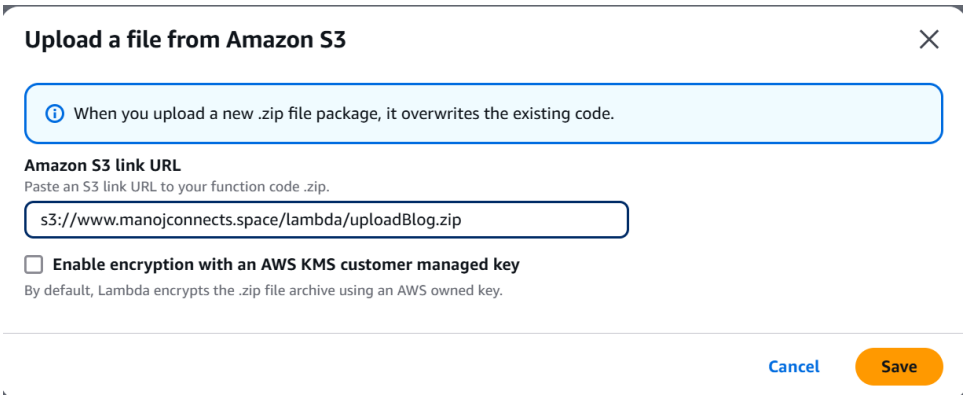




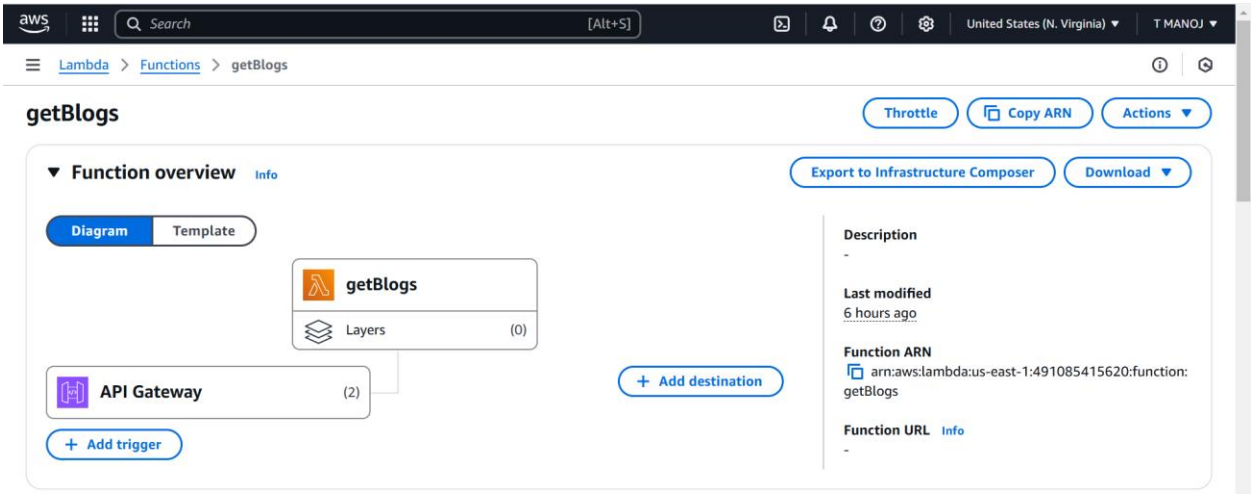
Step 2: Create Lambda Functions

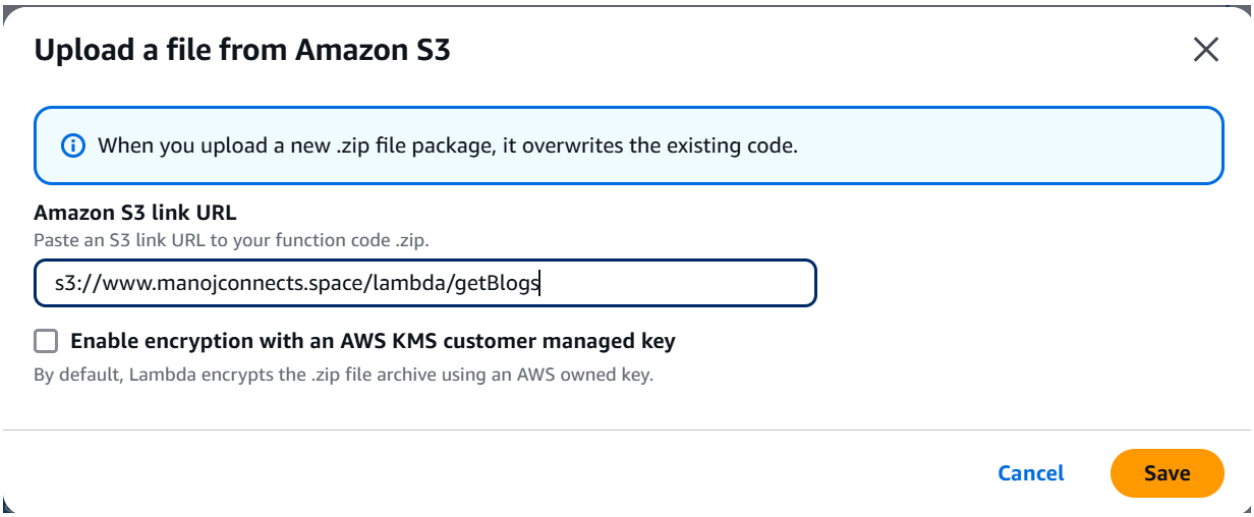
1. **uploadBlog**:
    a. Input: userId, blogId, imageUrl, content, title.
    b. Save data in DynamoDB and image in S3.
    c. Return success/error response.

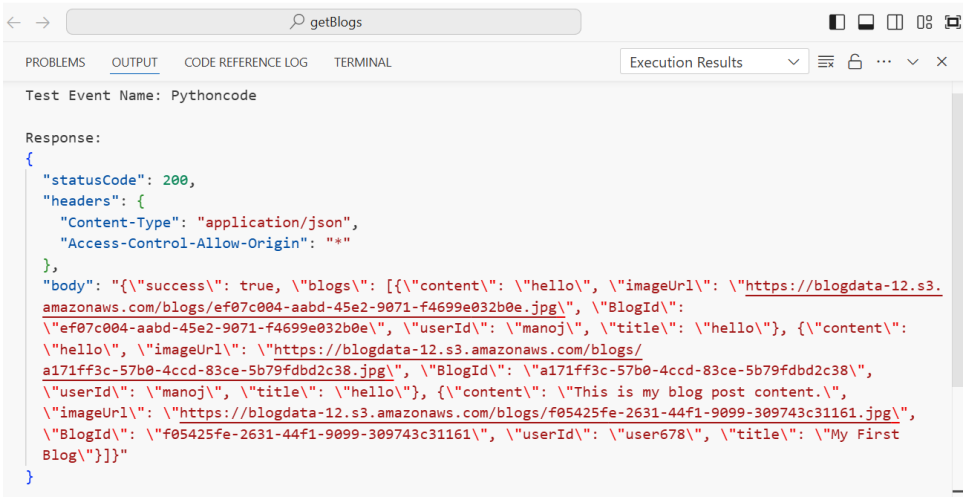




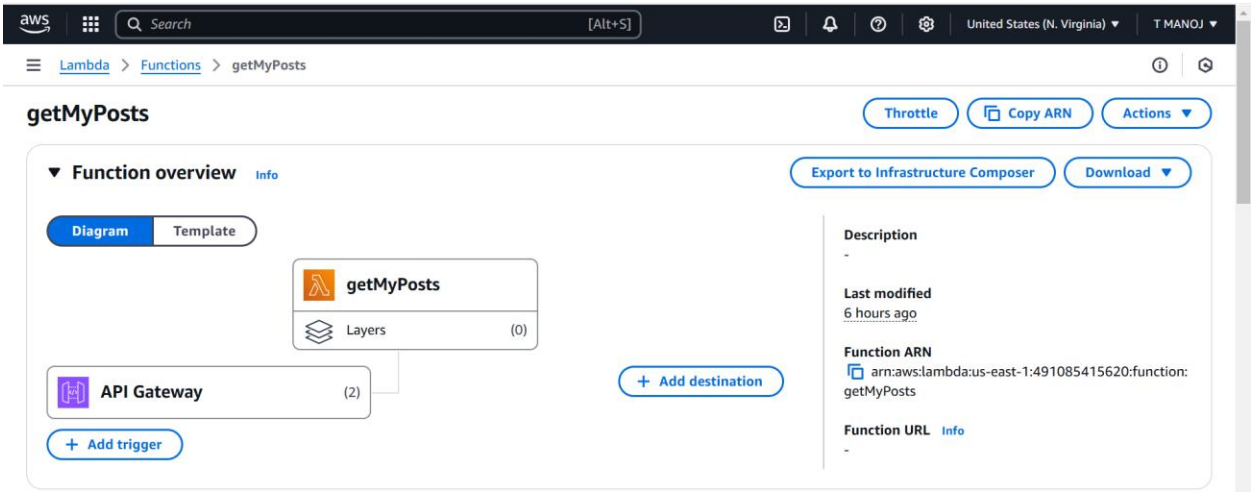
2. **getBlogs**:
    a. Scan DynamoDB to retrieve all blogs.
    b. Return blog data.

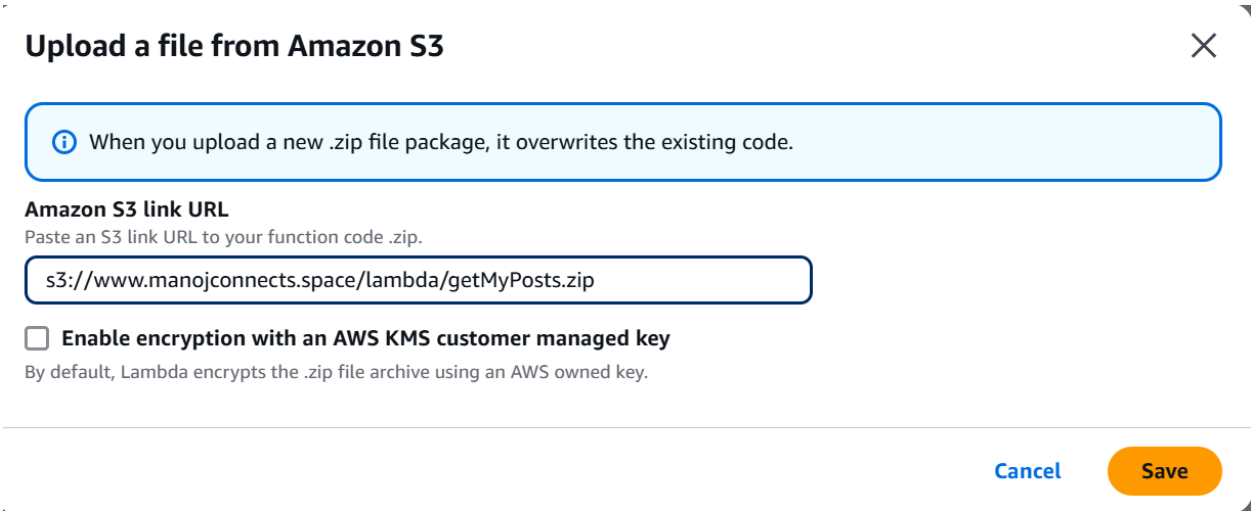


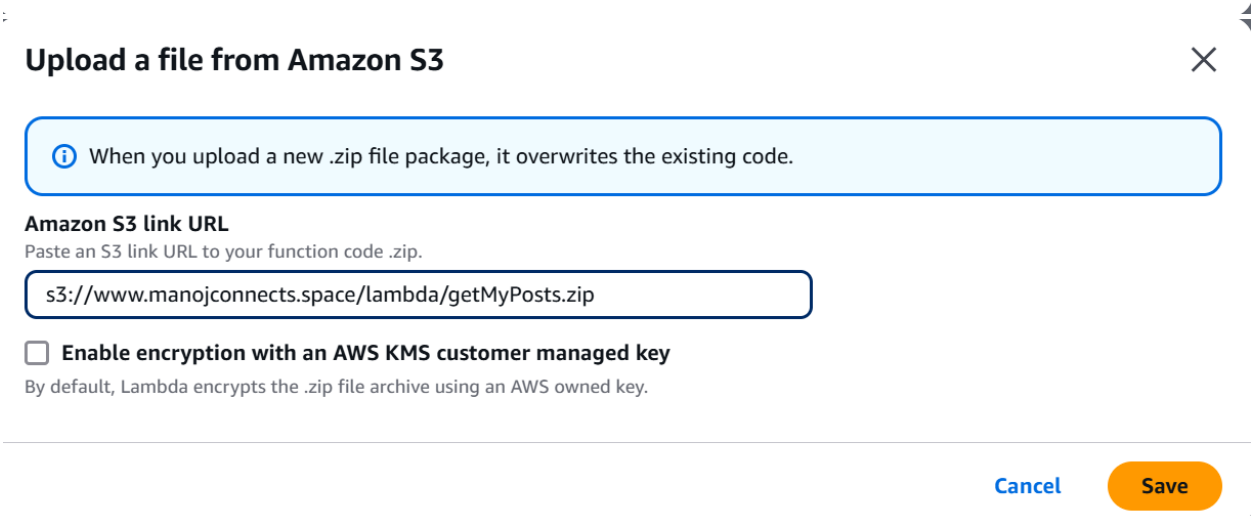


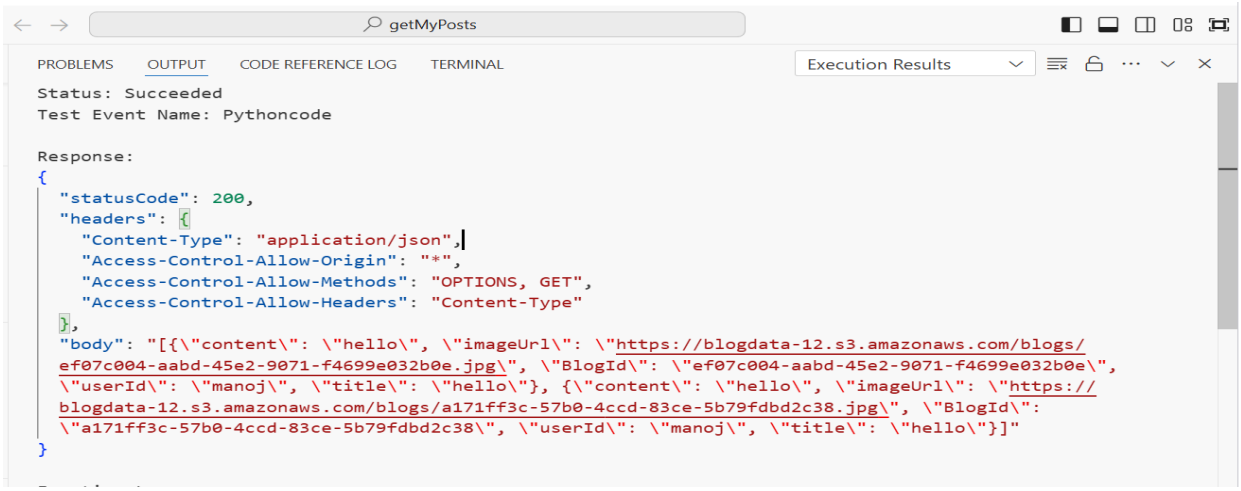


3. **getMyPosts**:
    a. Input: userId (from query string).
    b. Query DynamoDB using GSI to filter user-specific posts.
    c. Return filtered data.


aws
Search
[Alt+S]
United States (N. Virginia)
T MANOJ
Lambda
Functions
getMyPosts
getMyPosts
Throttle
Copy ARN
Actions
Function overview
Info
Export to Infrastructure Composer
Download
Diagram
Template
getMyPosts
Layers
(0)
API Gateway
(2)
Add destination
Add trigger
Description
-
Last modified
6 hours ago
Function ARN
arn:aws:lambda:us-east-1:491085415620:function:
getMyPosts
Function URL
Info
-



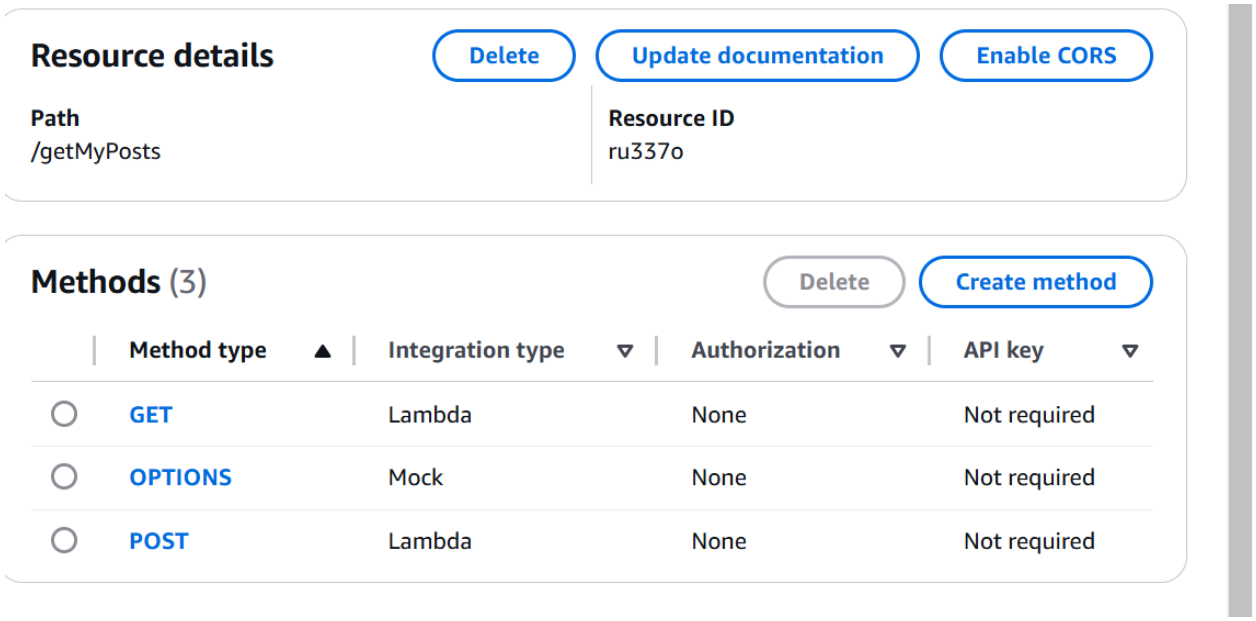


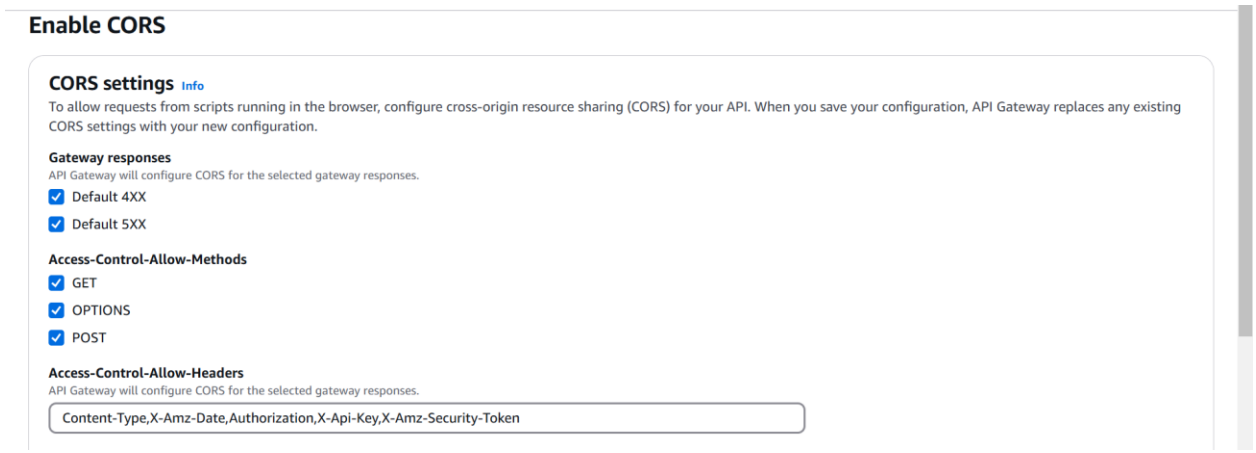


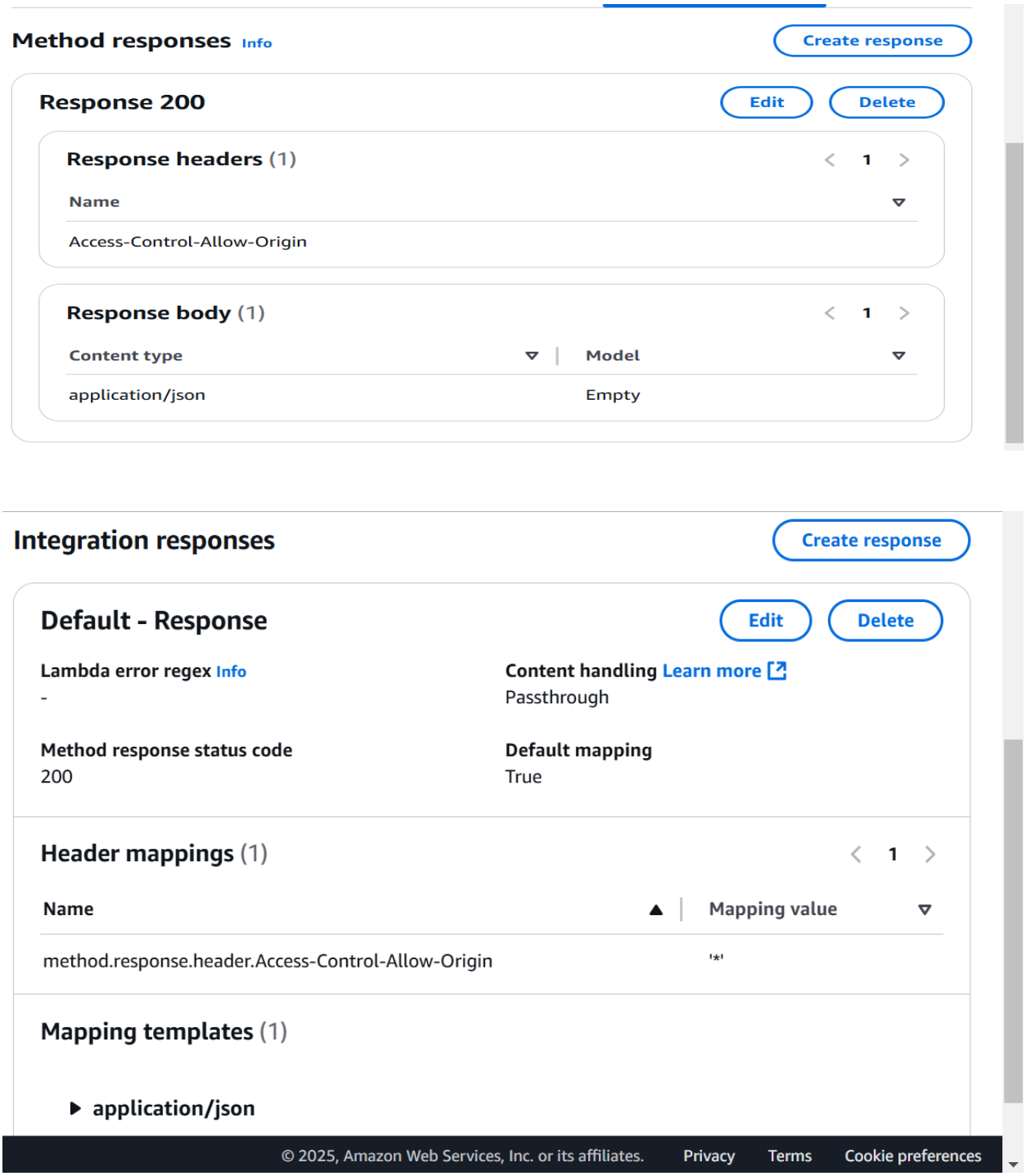
Step 3: Create API Gateway Methods

1. **getMyPosts:**
    a. Methods: POST, OPTIONS.

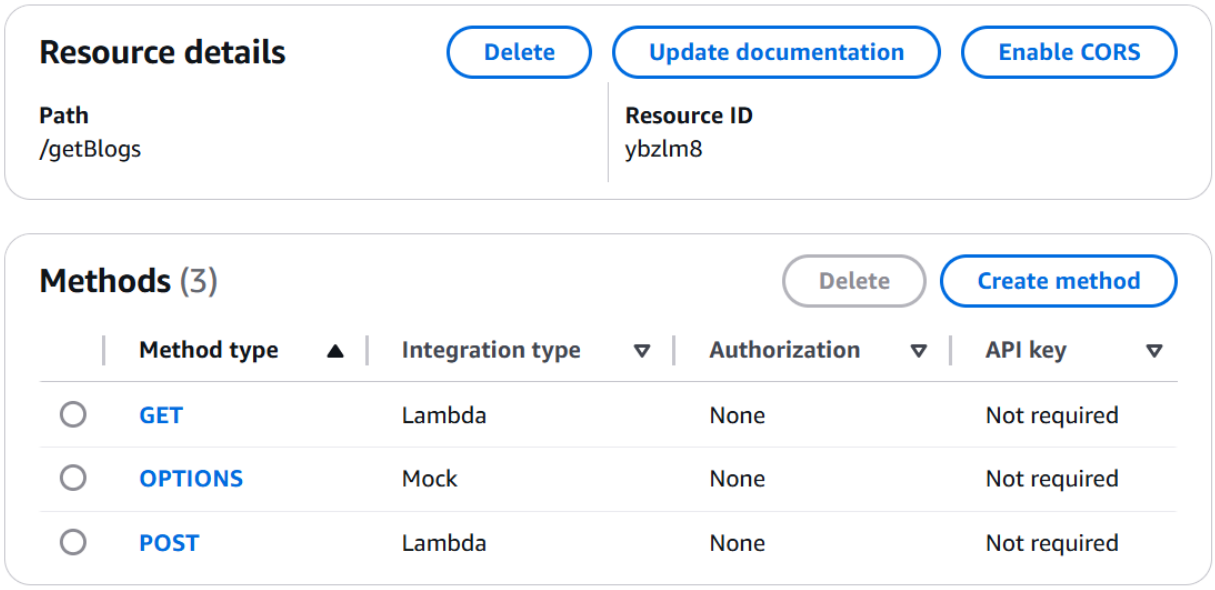
b. Enable CORS.
c. Add Method Responses and Integration Response.

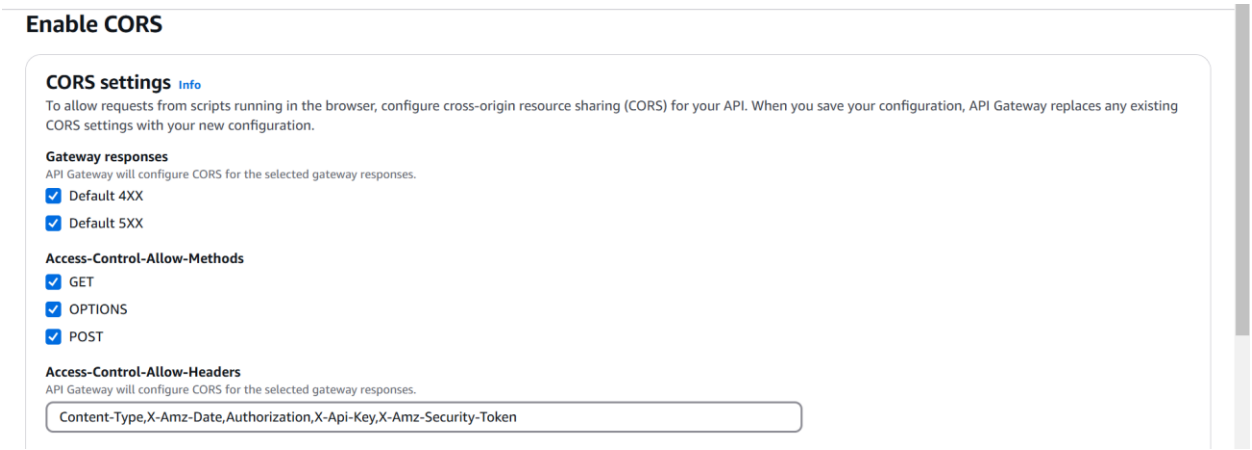




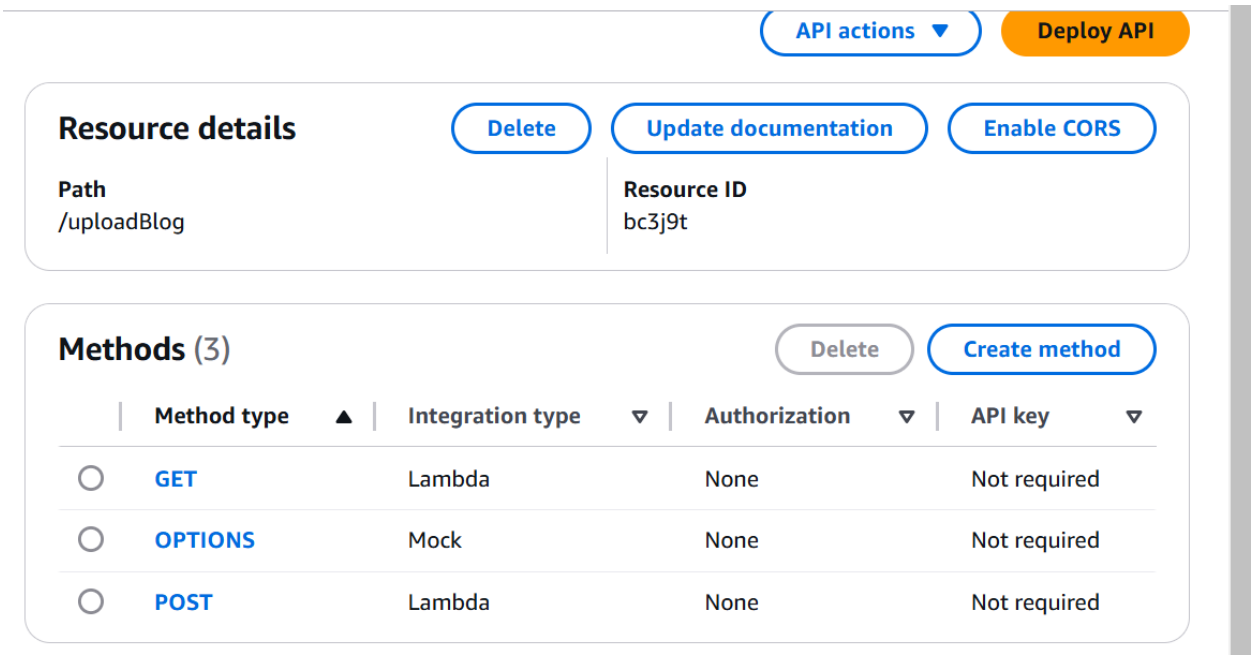


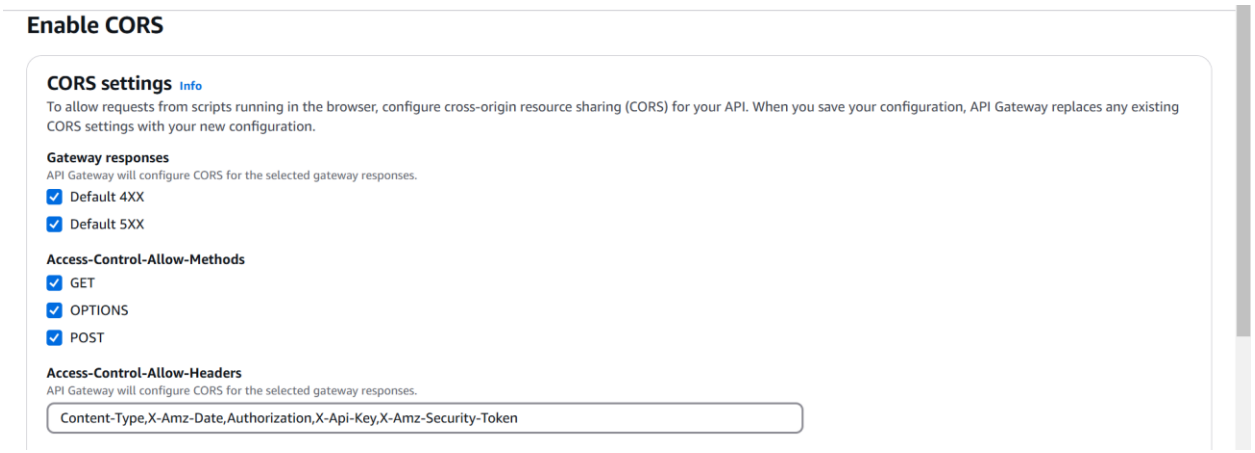
2. **getBlogs**:
    a. Methods: GET, OPTIONS.
    b. Enable CORS.



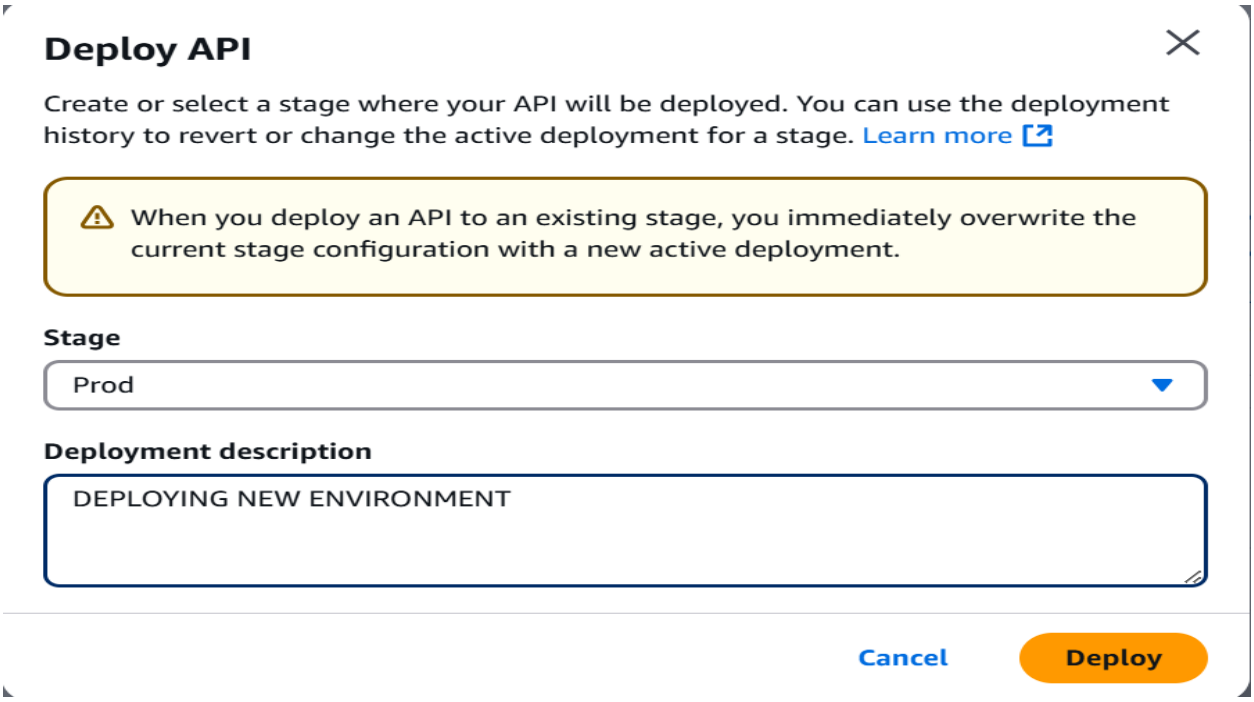


3. **uploadBlog**:
    a. Methods: GET, OPTIONS.
    b. Add query string parameter: userId.
    c. Enable CORS.


API actions
Deploy API
Resource details
Delete
Update documentation
Enable CORS
Path
/uploadBlog
Resource ID
bc3j9t
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

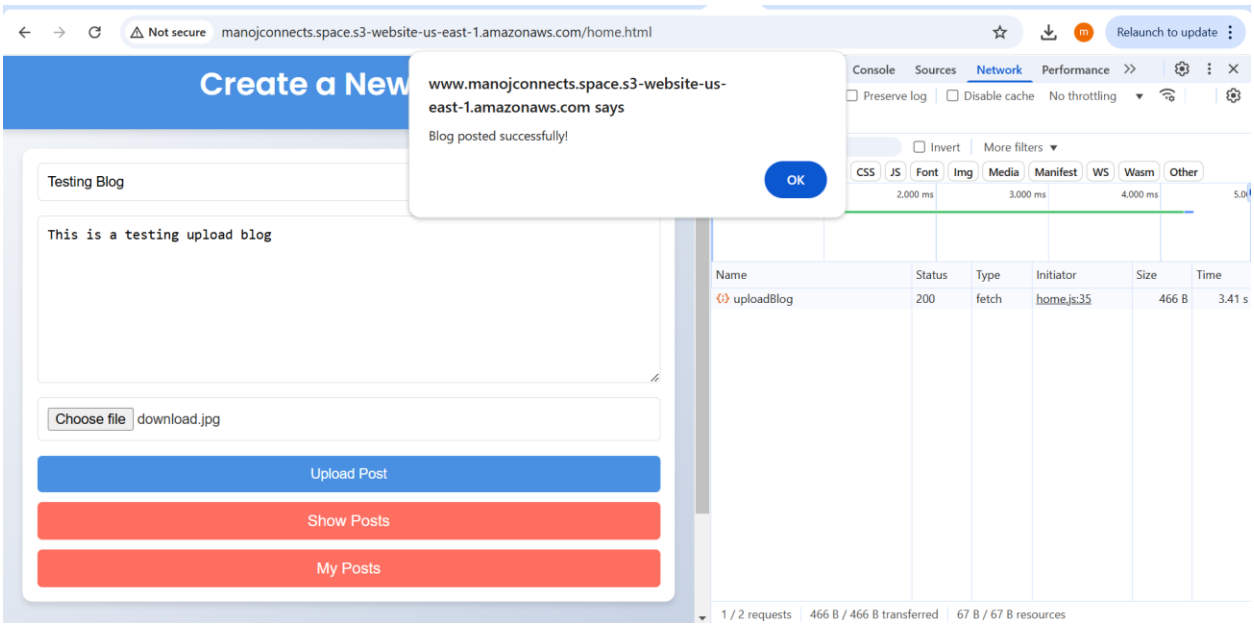

Enable CORS
CORS settings Info
To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API. When you save your configuration, API Gateway replaces any existing CORS settings with your new configuration.
Gateway responses
API Gateway will configure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

4. **Deploy API**

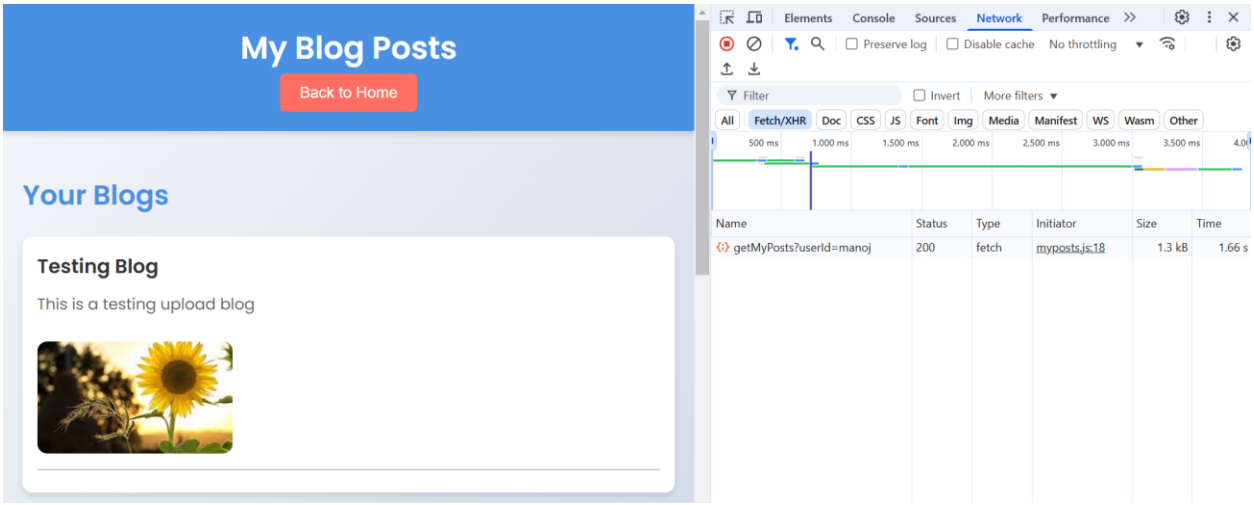


Step 4: Create Frontend Pages

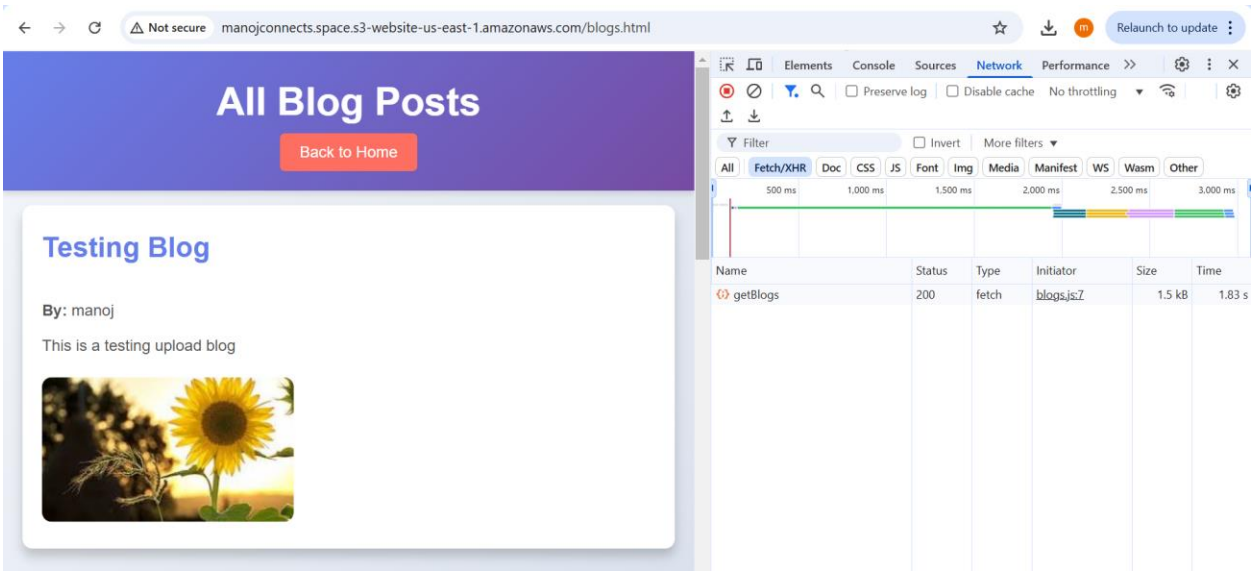
1. **Home Page (home.html)**:
    a. Form for blog creation: title, content, image.
    b. Buttons: "Submit Post", "My Posts", "All Posts".
    c. On submit, call uploadBlog API.



2. **My Posts Page (posts.html)**:
    a. Fetch user-specific posts using getMyPosts API.
    b. Display posts with images, titles, and content.



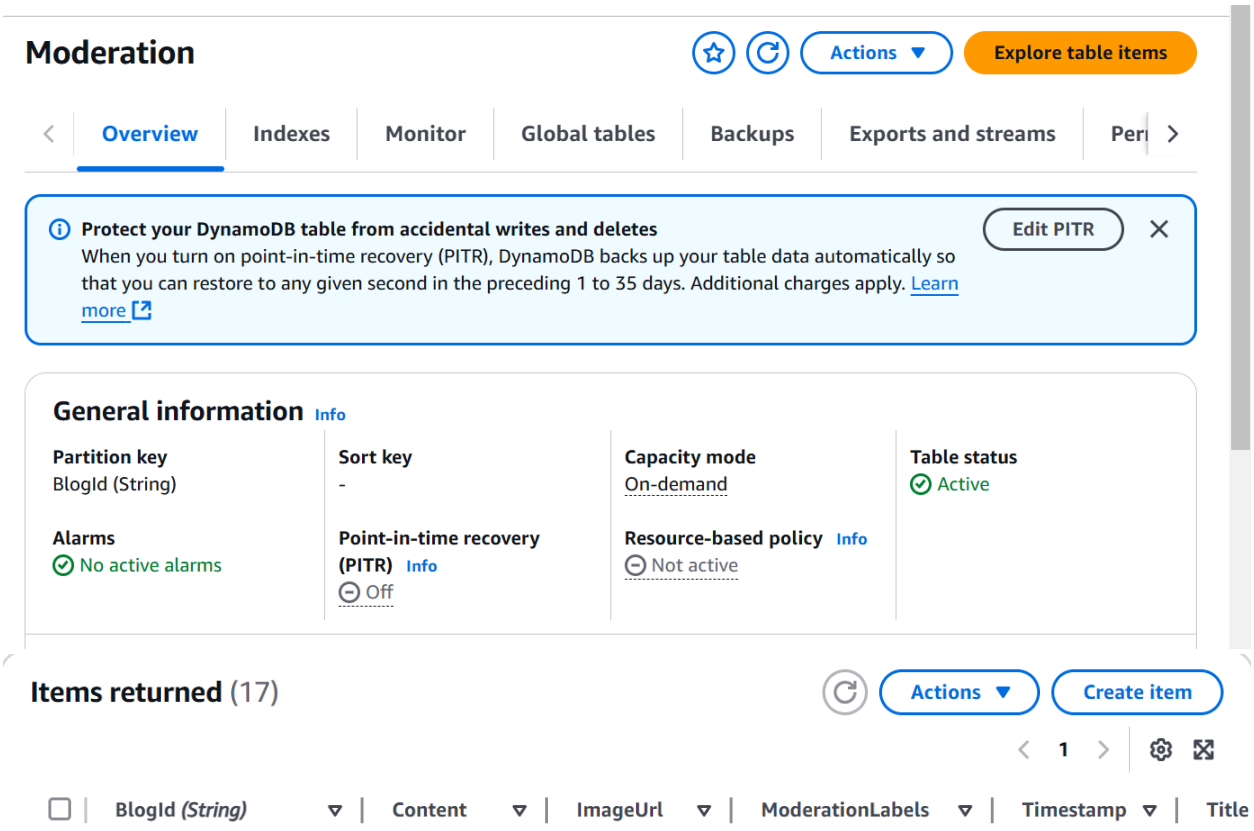
3. **All Posts Page (blogs.html)**:
    a. Fetch all posts using getBlogs API.
    b. Display all posts.



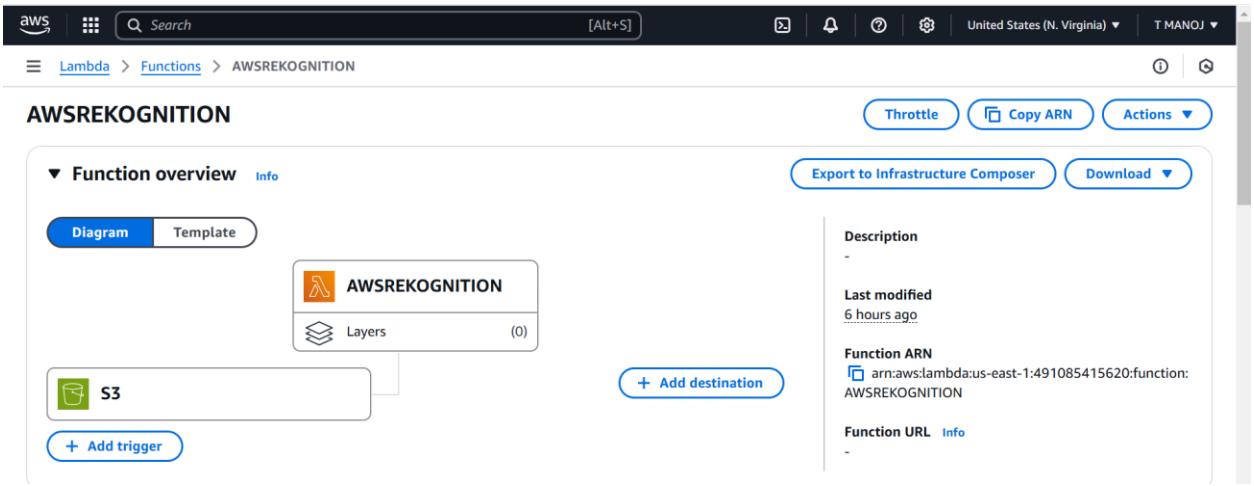
Phase 3: Amazon Rekognition for Image Moderation

Step 1: Create DynamoDB Table

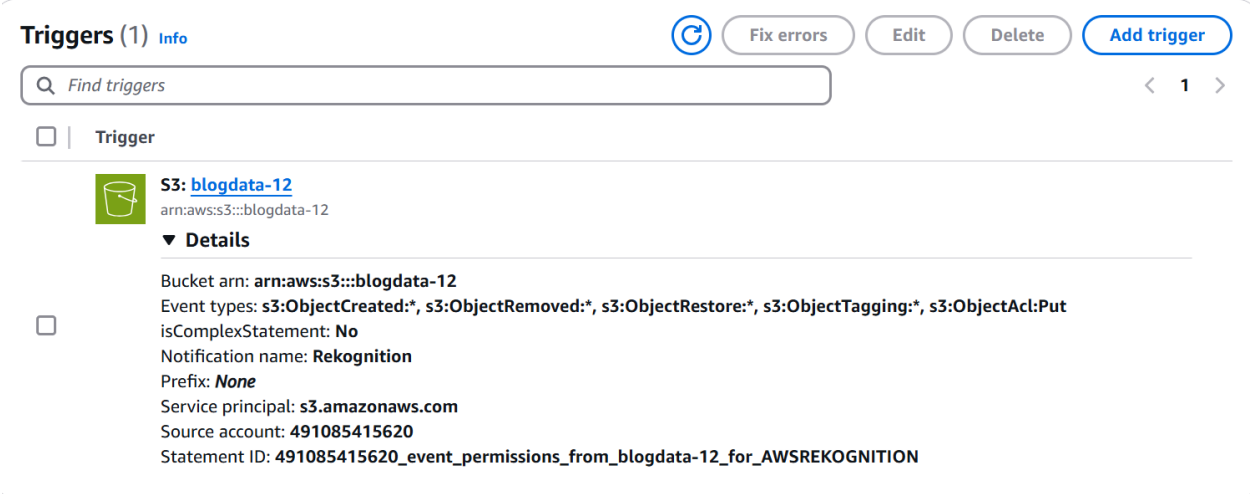
1. **Table Name**: MODERATION
2. **Partition Key**: imageId (String)
3. **Attributes**:
    a. imageId (Primary Key)
    b. labels (String)
    c. Title(String)
    d. Content (String)
    e. UserId (String)
    f. timestamp (String)



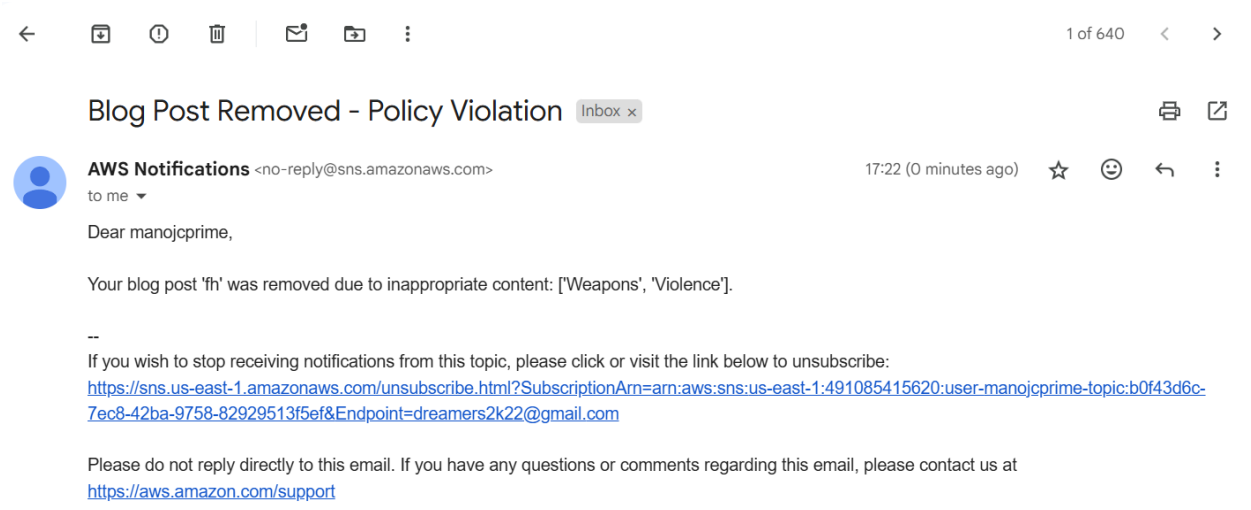
Step 2: Create Lambda Function for Image Moderation

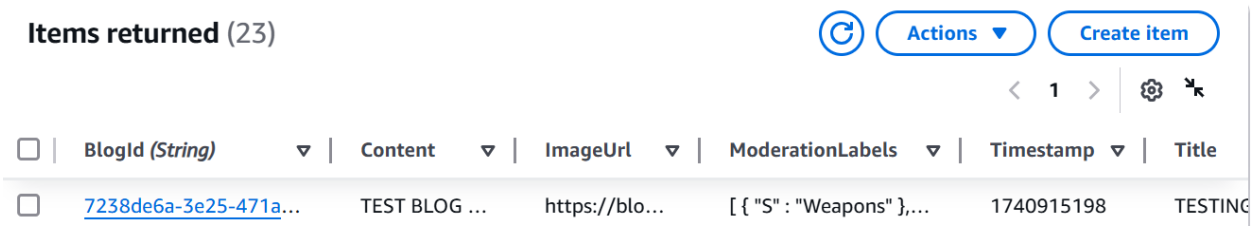


1. **Trigger**:
    a. Set up S3 event trigger for blogdata-12 bucket.



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.

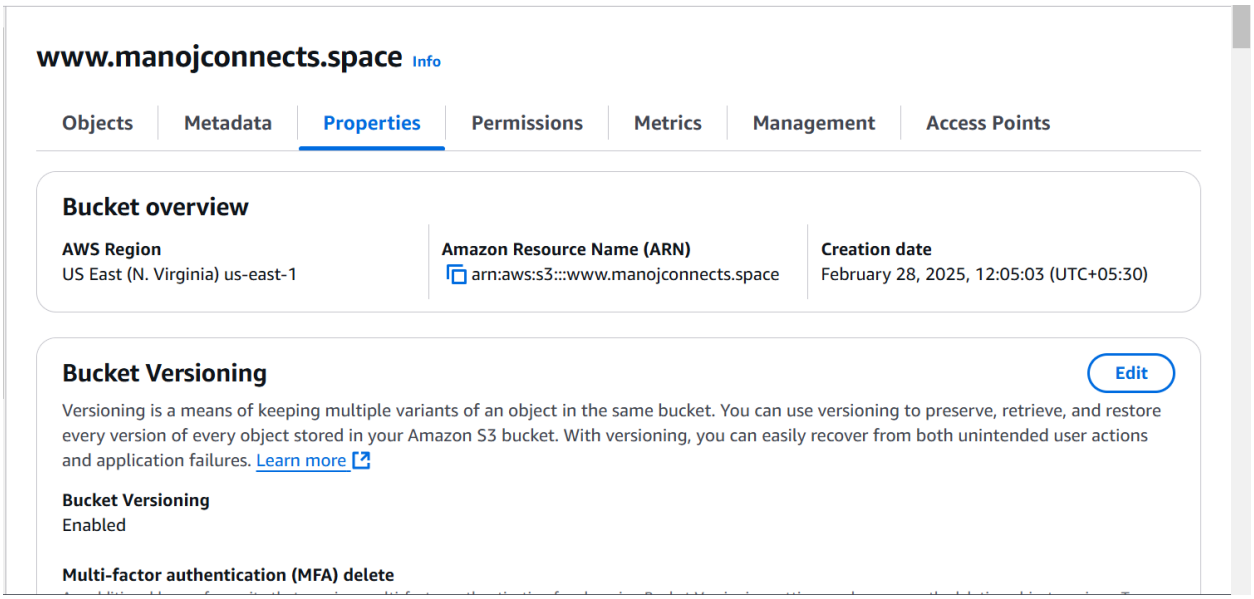


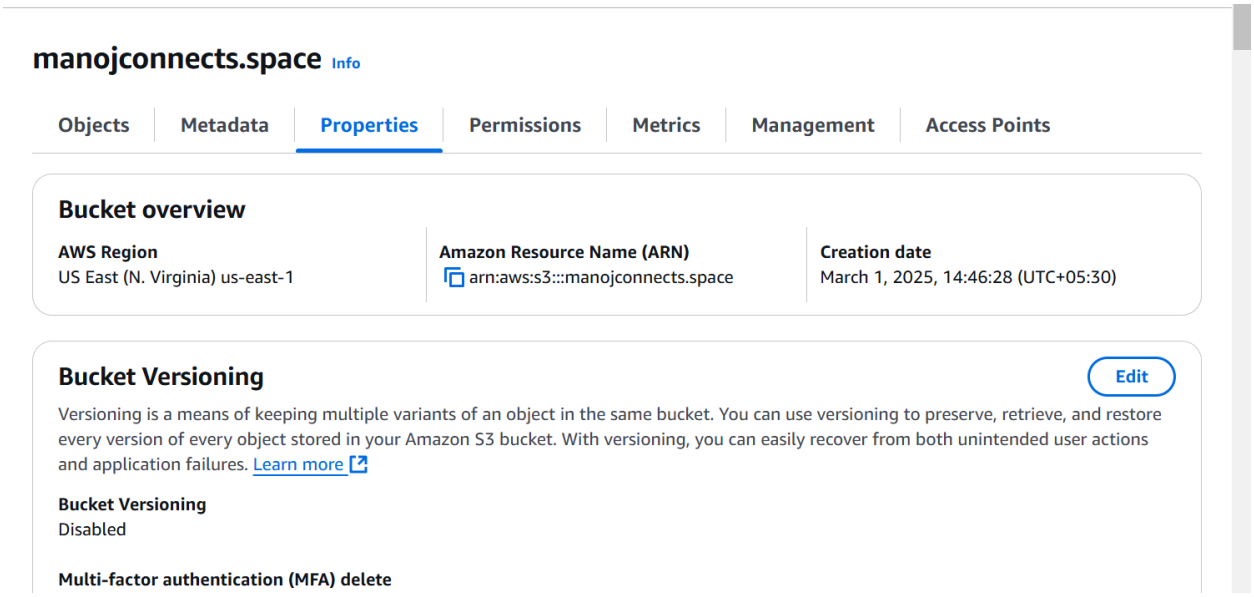


Phase 4: Uploading Files, Route53, CloudFront, Certificate Manager

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).





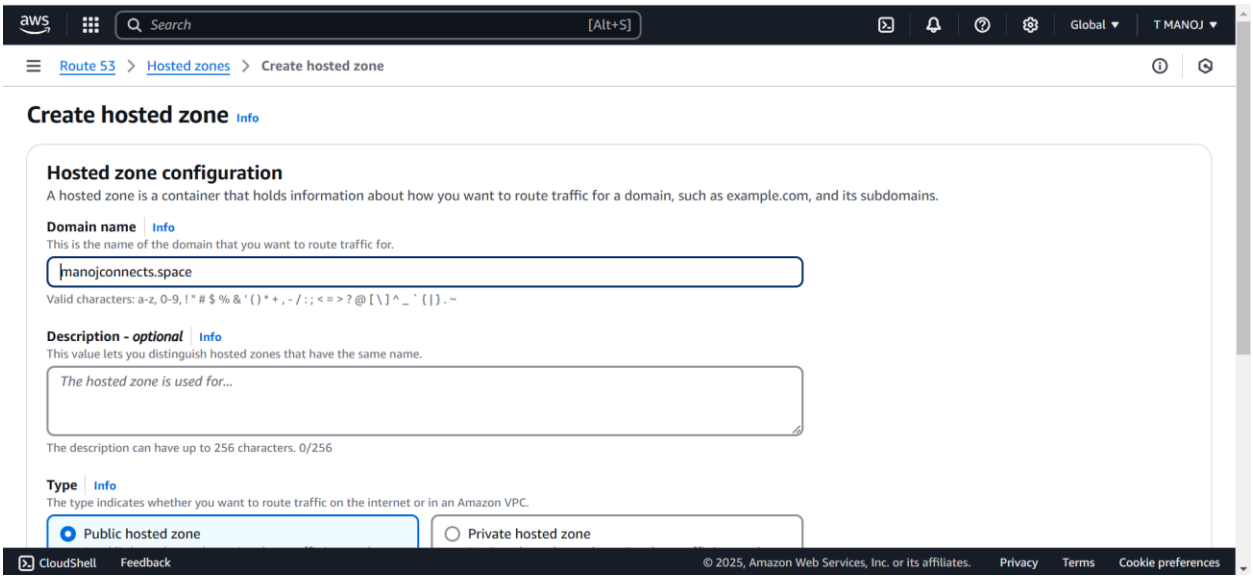
2. **Bucket Policies**:
    a. Allow public read access for www.manojconnects.space.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

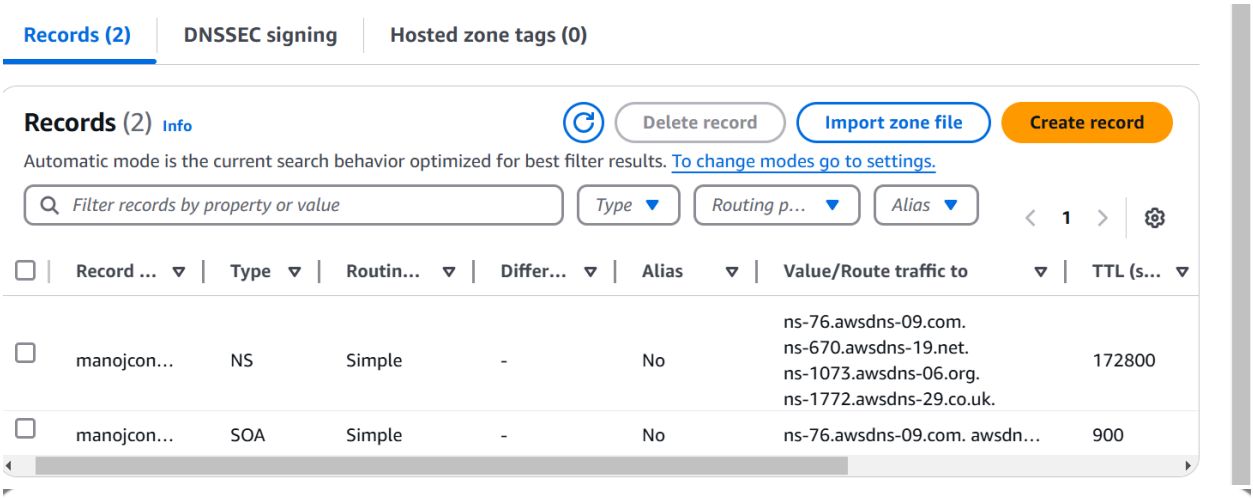
```
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::www.manojconnects.space/*"
    }
  ]
}
```

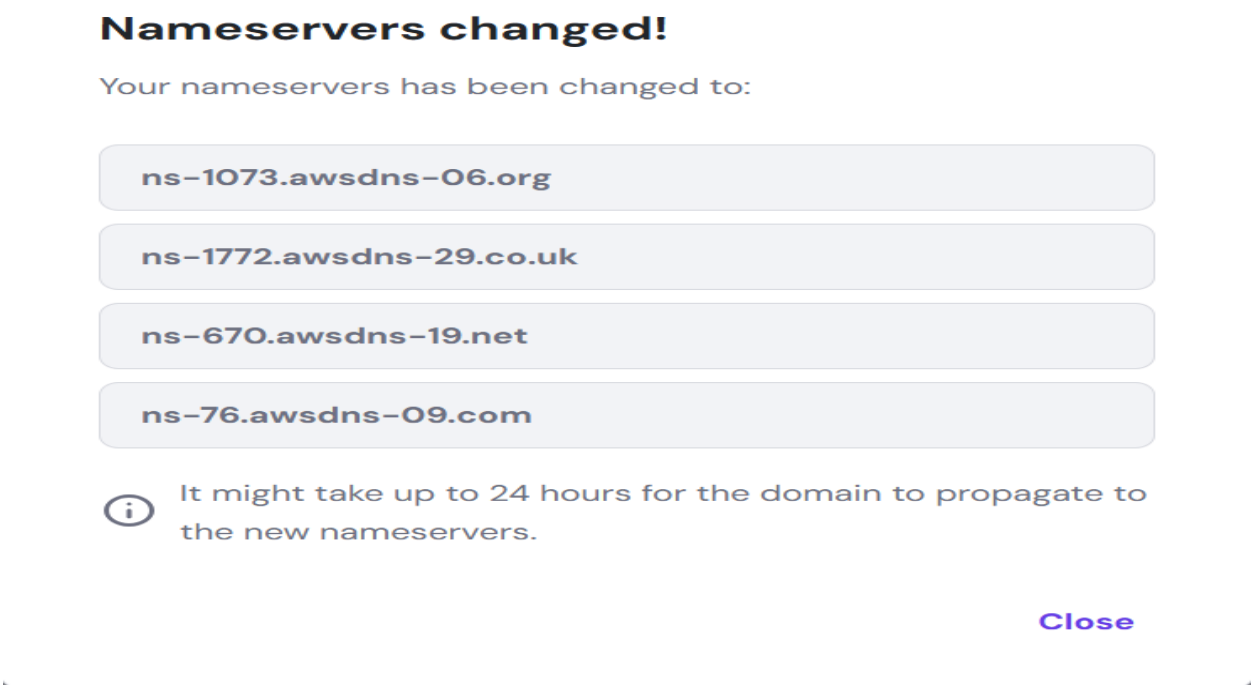
Step 2: Configure Route53

1. **Hosted Zone**:
    a. Create a hosted zone for manojconnects.space.



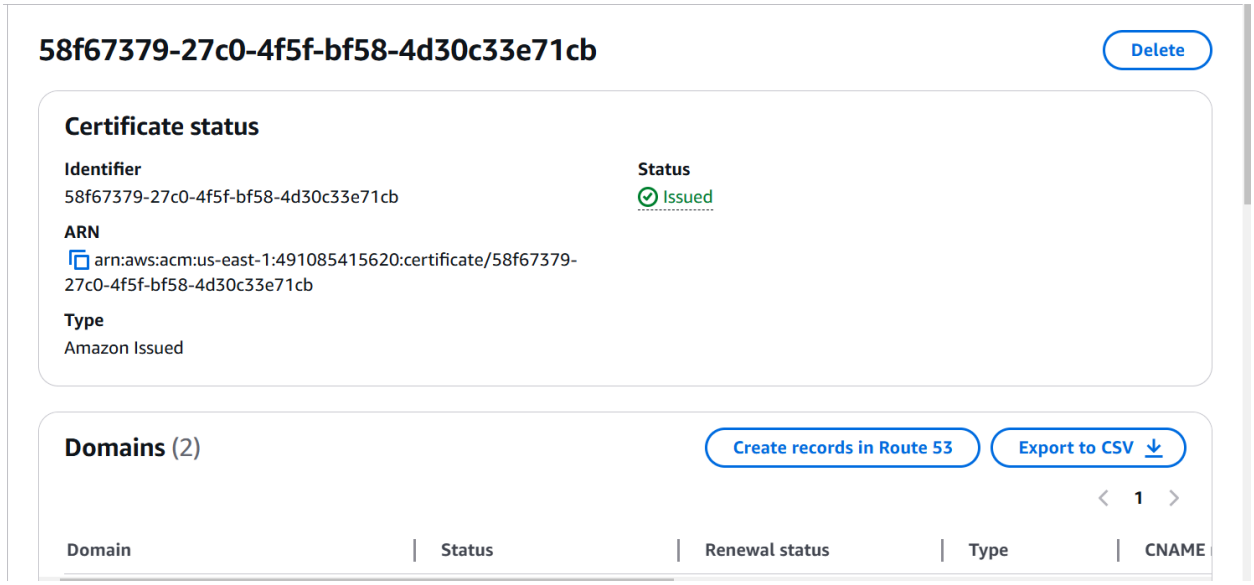
2. **Records**:
    a. Create A records pointing to CloudFront distributions.





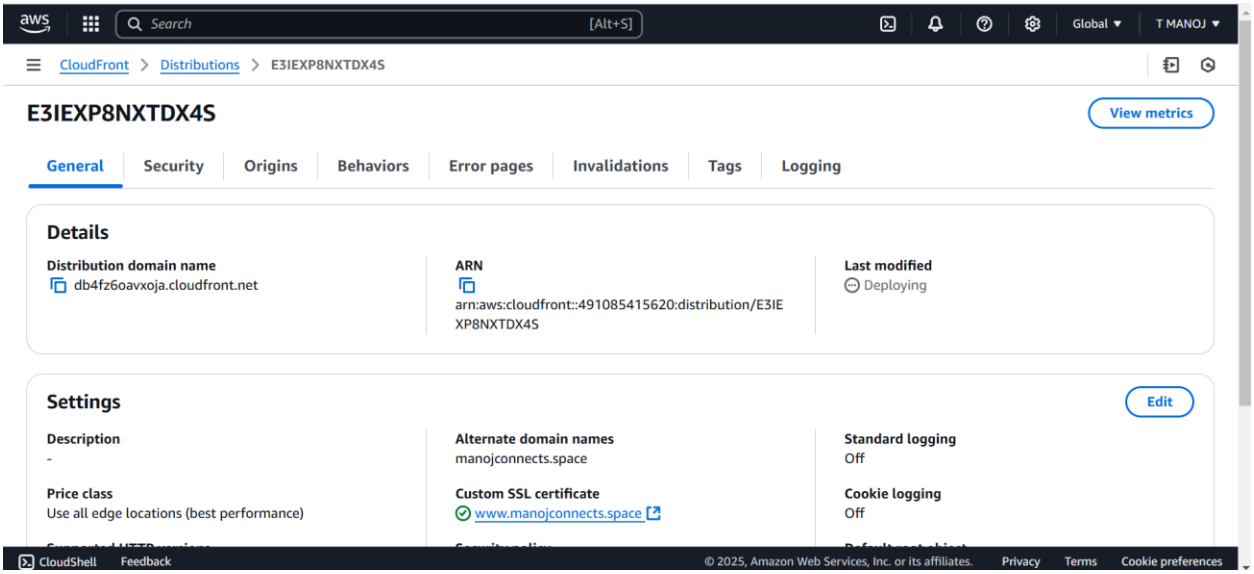
Step 3: Create SSL Certificates

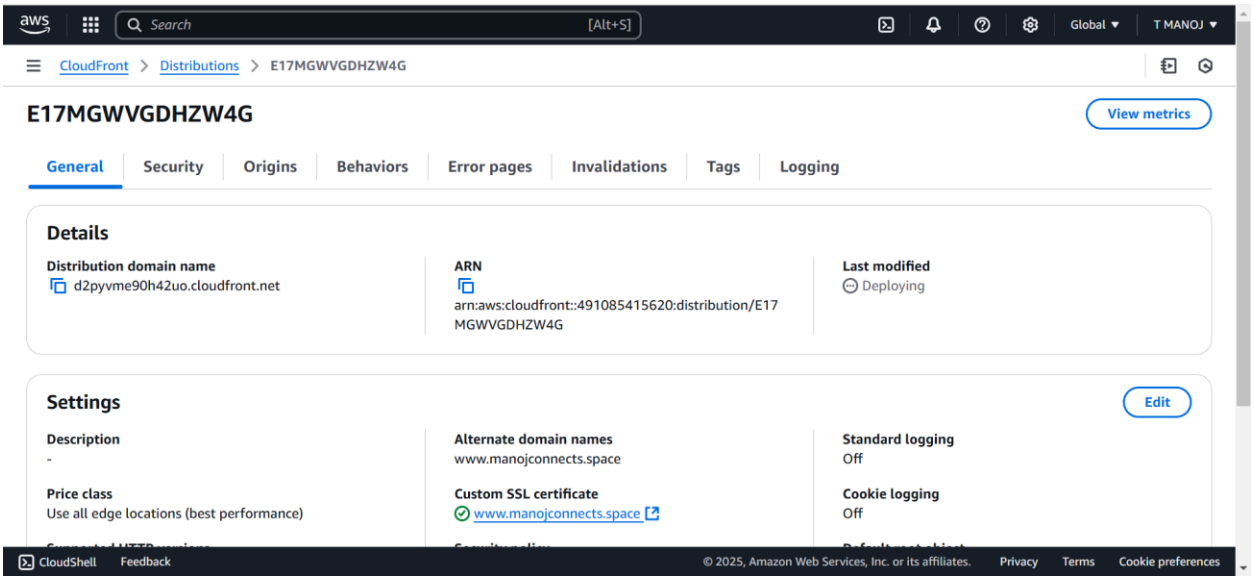
1. **Certificate Manager**:
    a. Request certificates for manojconnects.space and www.manojconnects.space.



Step 4: Configure CloudFront

1. **Distributions**:
    a. Create two distributions (one for each bucket).
    b. Set alternate domain names and SSL certificates.
    c. Set origin to the S3 static hosting URL.





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.

subdomain manojconnects.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)

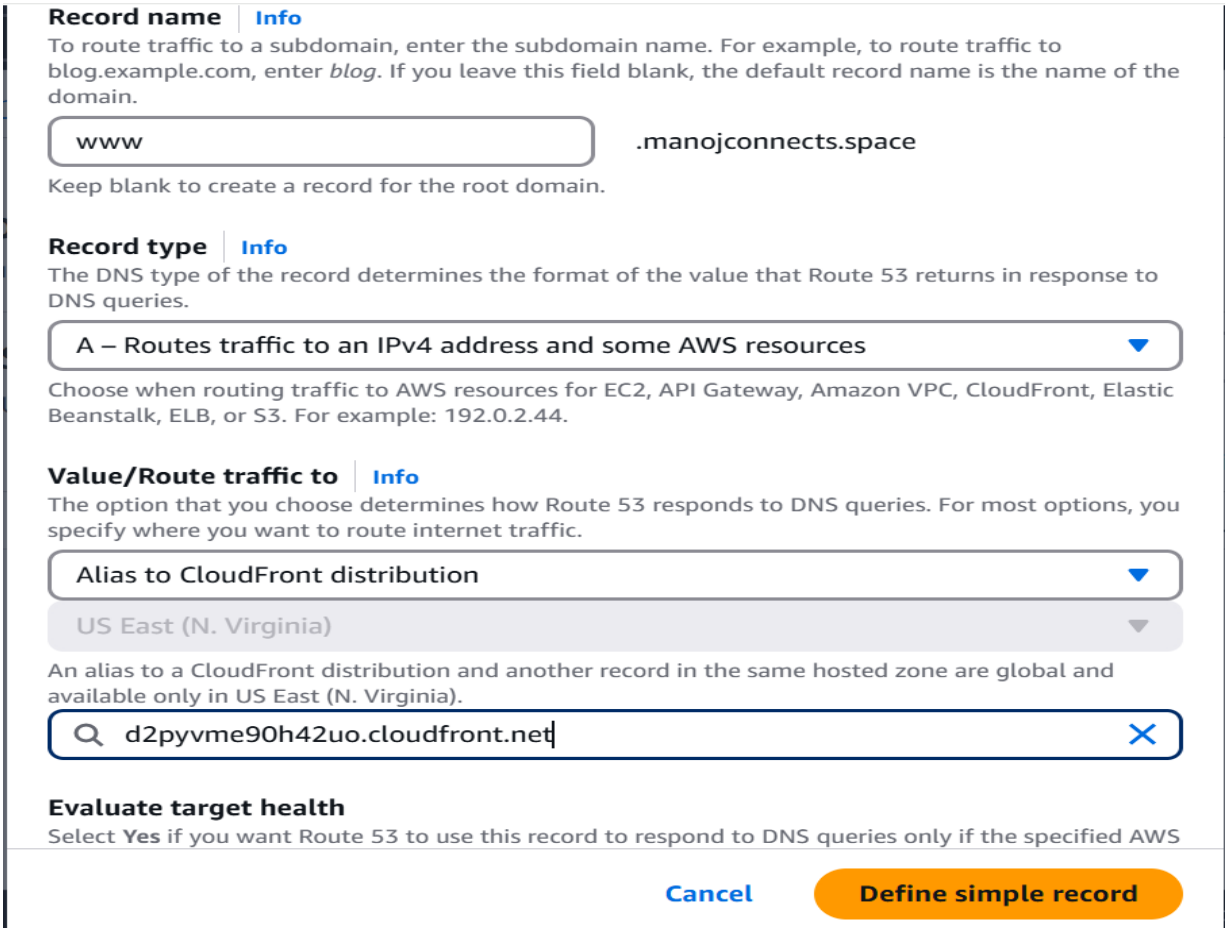
An alias to a CloudFront distribution and another record in the same hosted zone are global and available only in US East (N. Virginia).

db4fz6oavxoja.cloudfront.net

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



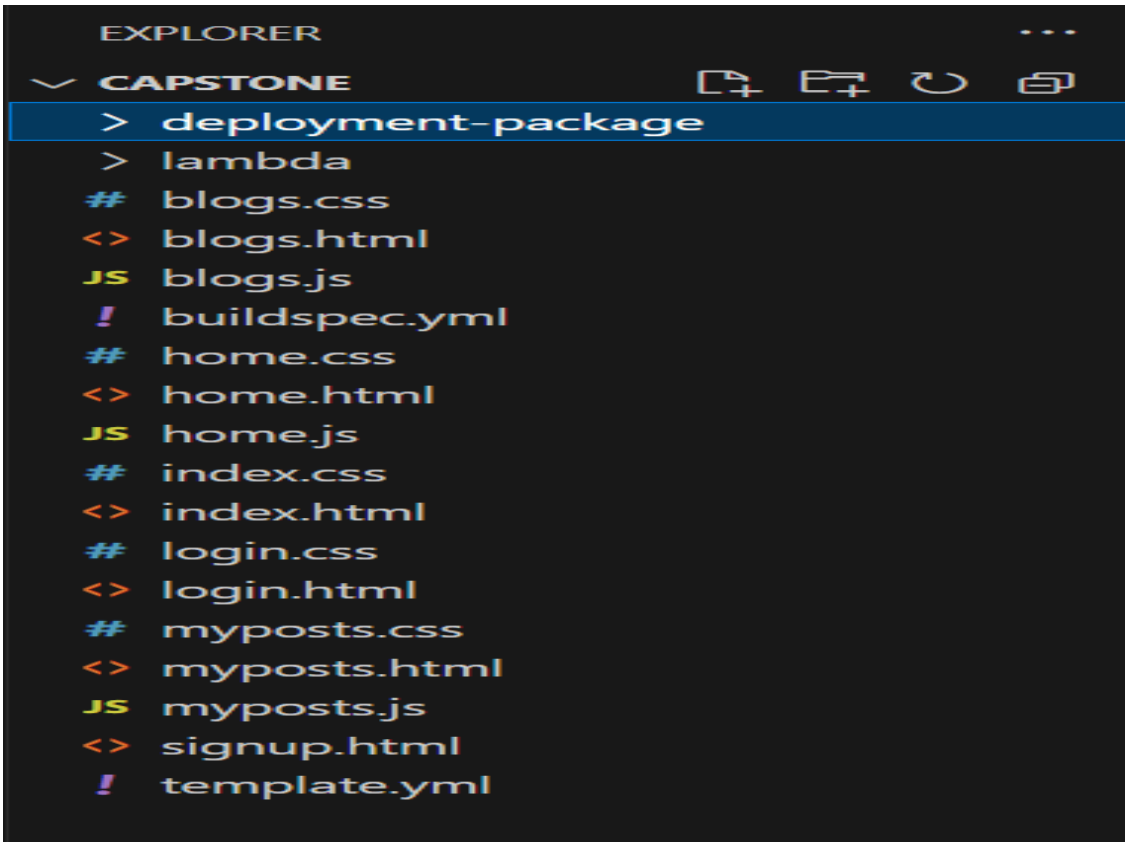
Record ...	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...
manojcon...	A	Simple	-	Yes	db4fz6oavxoja.cloudfront.net.	-
manojcon...	NS	Simple	-	No	ns-76.awsdns-09.com. ns-670.awsdns-19.net. ns-1073.awsdns-06.org. ns-1772.awsdns-29.co.uk.	172800
manojcon...	SOA	Simple	-	No	ns-76.awsdns-09.com.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

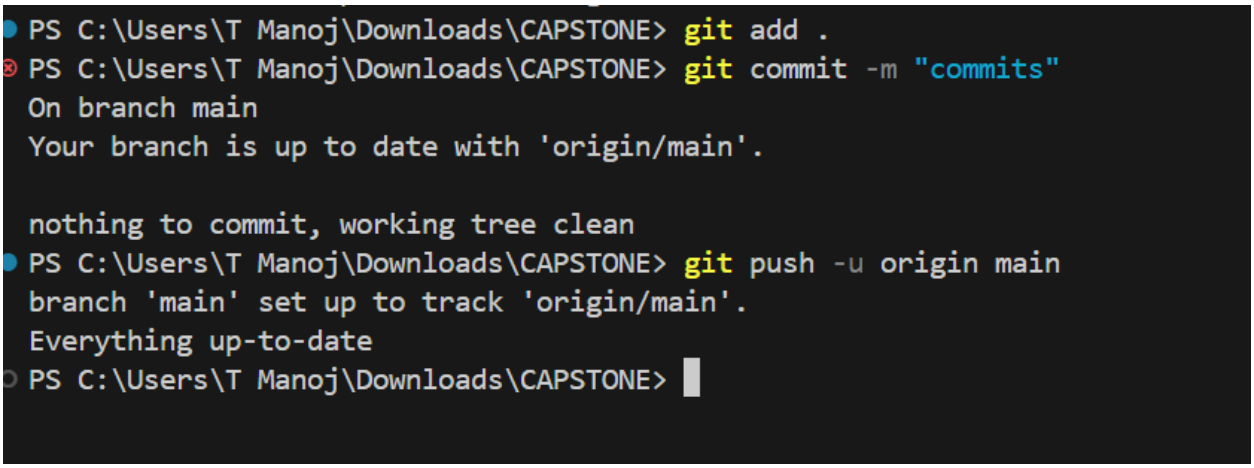
Phase 5: CodeDeploy and CodePipeline

Step 1: Create GitHub Repository

1. **Push Code**:

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





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

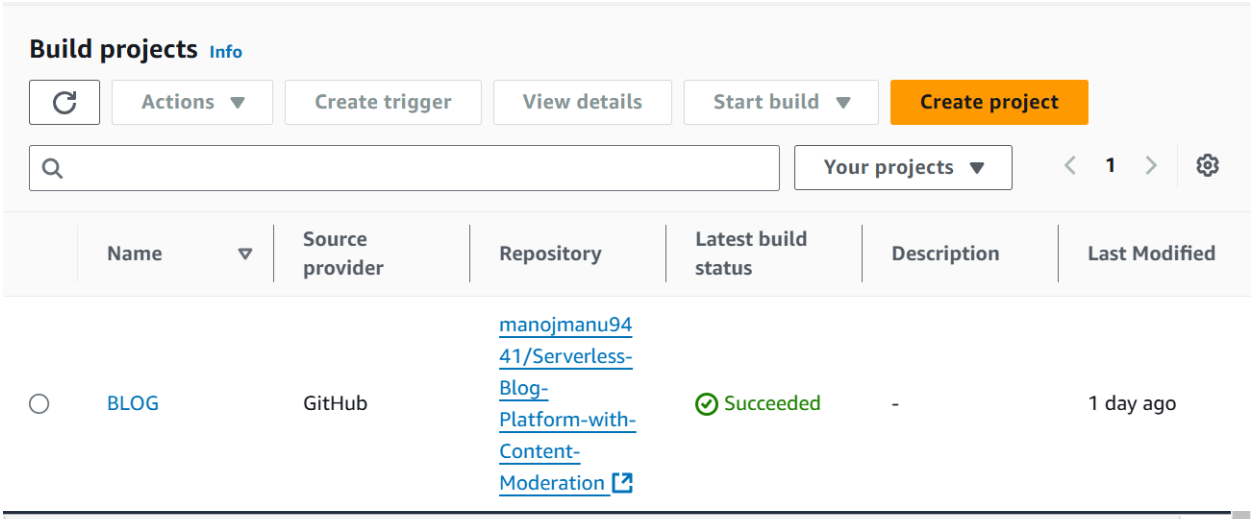
```
      - cd ..
```

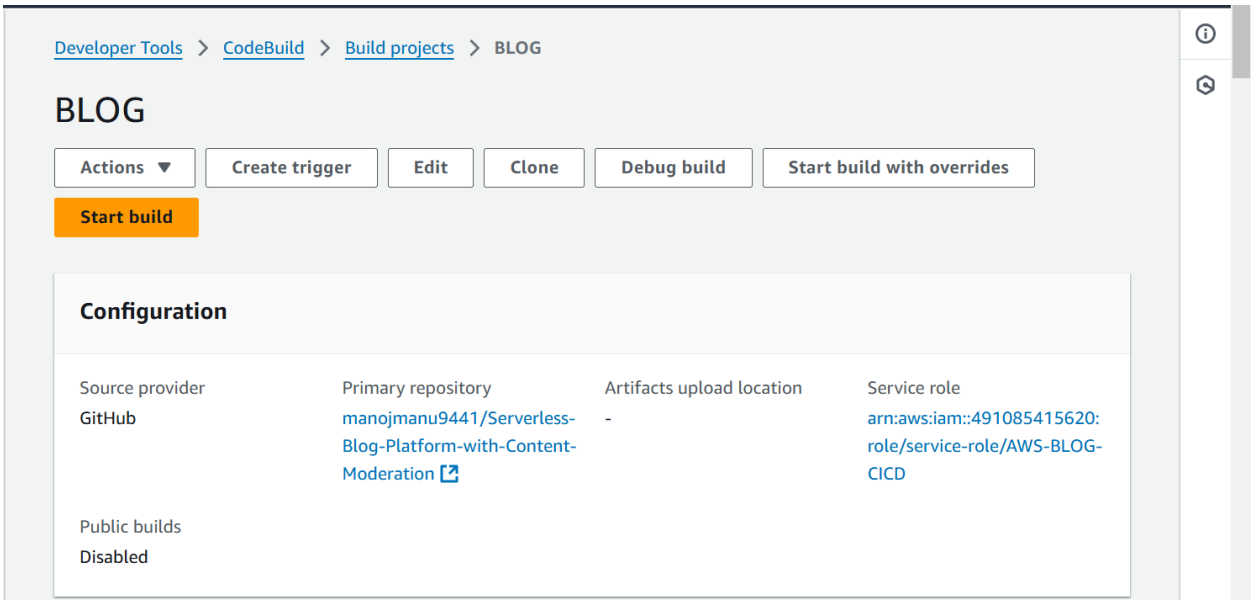
```
  post_build:
    commands:
      - echo "Deploying website to S3..."
      -  aws  s3  sync  .  s3://www.manojconnects.space  --exclude  "lambda/*"  --exclude
".git/*" --exclude "node_modules/*" --delete
```

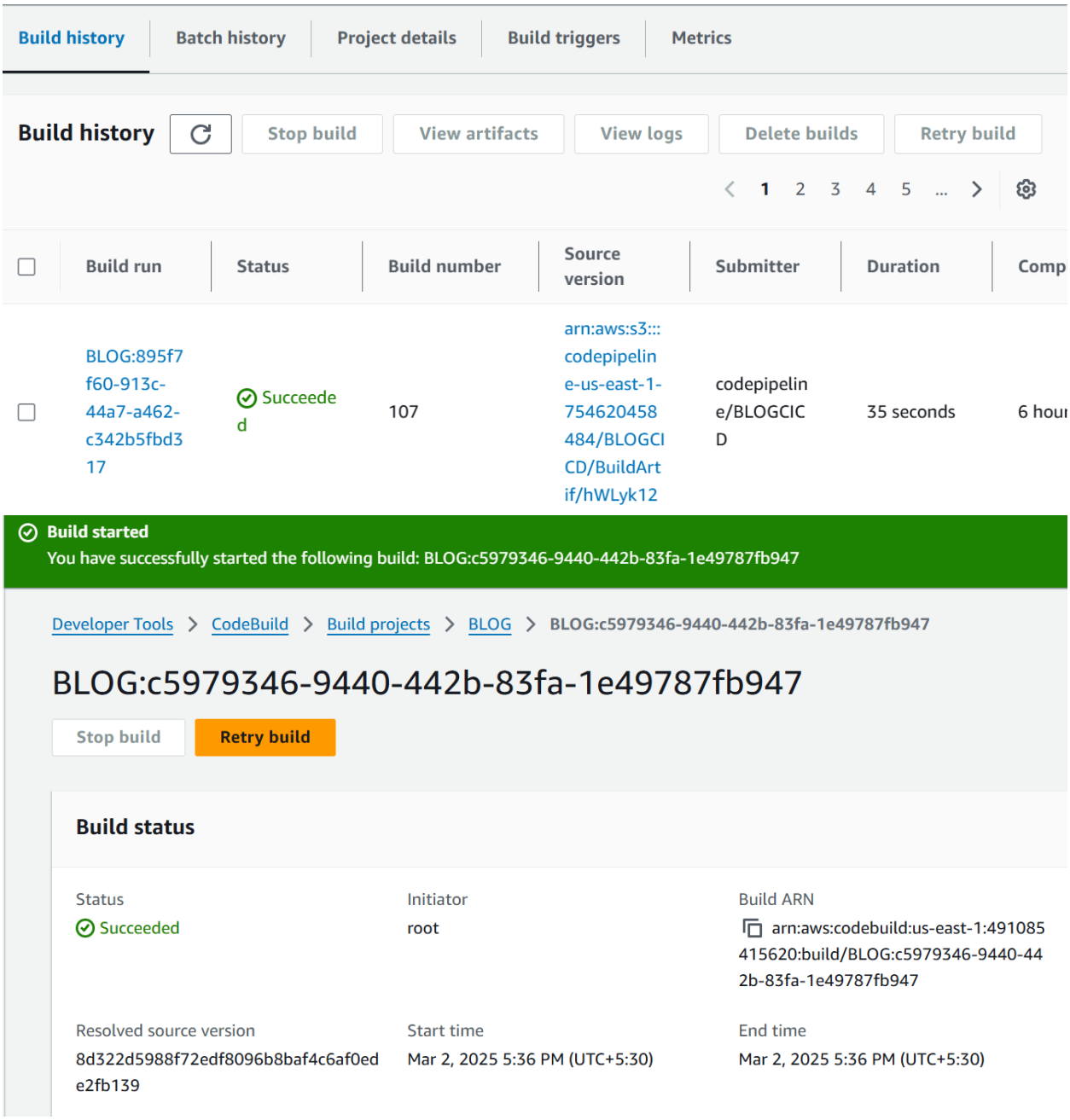
```
      - echo "Deploying Lambda function ZIPs to S3..."
      - for zipfile in lambda/*.zip; do
          aws s3 cp "$zipfile" "s3://www.manojconnects.space/$zipfile";
      done

      - 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: .
```

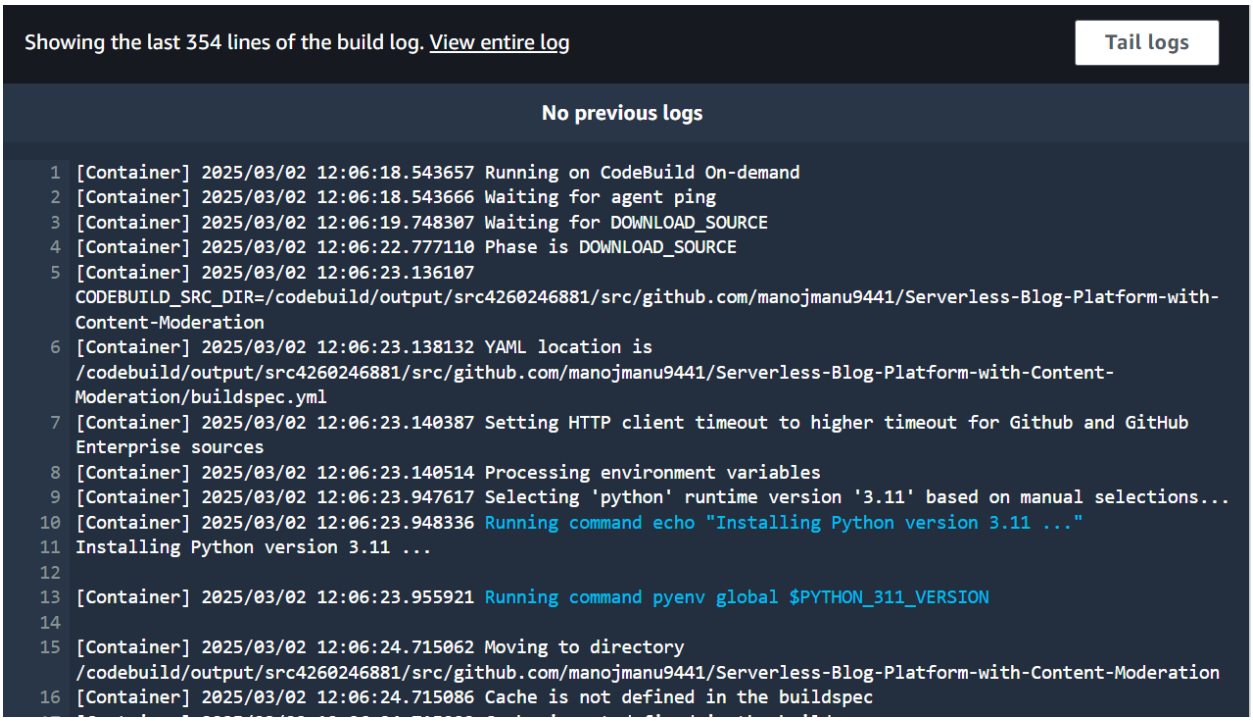
Step 3: Configure CodeBuild

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
manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation
Succeeded
-
1 day ago

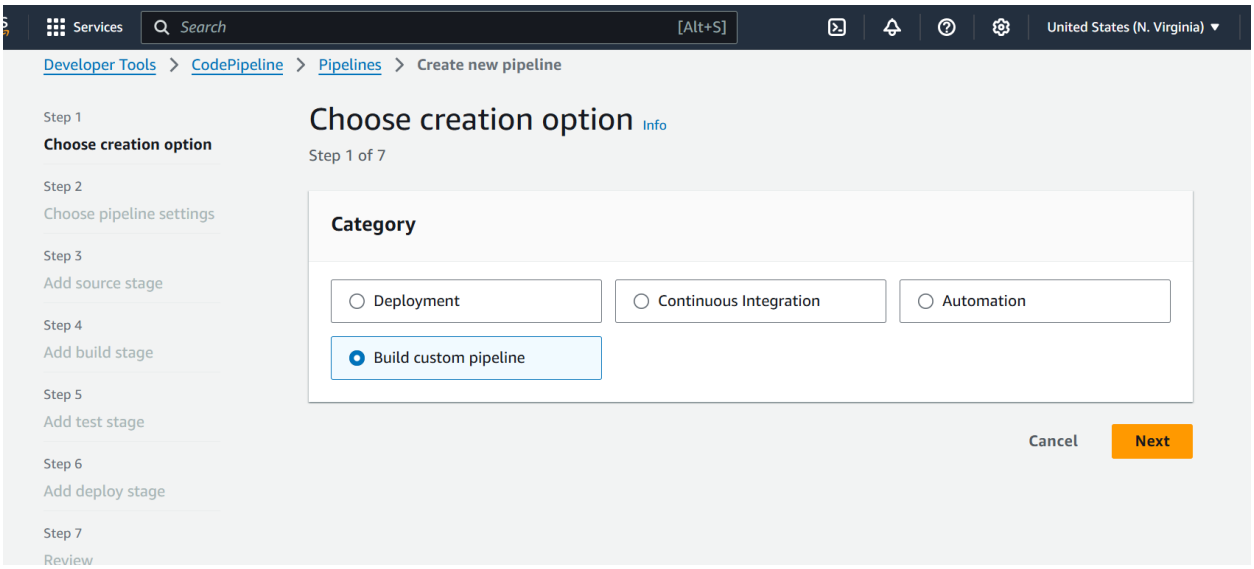

Developer Tools
CodeBuild
Build projects
BLOG
BLOG
Actions
Create trigger
Edit
Clone
Debug build
Start build with overrides
Start build
Configuration
Source provider
GitHub
Primary repository
manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation
Artifacts upload location
-
Service role
arn:aws:iam::491085415620:role/service-role/AWS-BLOG-CICD
Public builds
Disabled


Build history
Batch history
Project details
Build triggers
Metrics
Build history
Stop build
View artifacts
View logs
Delete builds
Retry build
1 2 3 4 5 ...
Build run
Status
Build number
Source version
Submitter
Duration
Comp
BLOG:895f7f60-913c-44a7-a462-c342b5fbd317
Succeeded
107
arn:aws:s3:::codepipeline-us-east-1-754620458484/BLOGCICD/BuildArtif/hWLyk12
codepipeline/BLOGCICD
35 seconds
6 hour
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
Succeeded
Initiator
root
Build ARN
arn:aws:codebuild:us-east-1:491085415620:build/BLOG:c5979346-9440-442b-83fa-1e49787fb947
Resolved source version
8d322d5988f72edf8096b8baf4c6af0ede2fb139
Start time
Mar 2, 2025 5:36 PM (UTC+5:30)
End time
Mar 2, 2025 5:36 PM (UTC+5:30)



Step 4: Create CodePipeline

1. **Pipeline**:
    a. Add source (GitHub) and select the github repo .
    b. Add the Build (CodeBuild), and deploy (Lambda) stages.
    c. Create the service role with necessary permission[lambda,code build,code deploy]
    d. Add the Build stage and test stage .
    e. Create the code Pipeline.



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

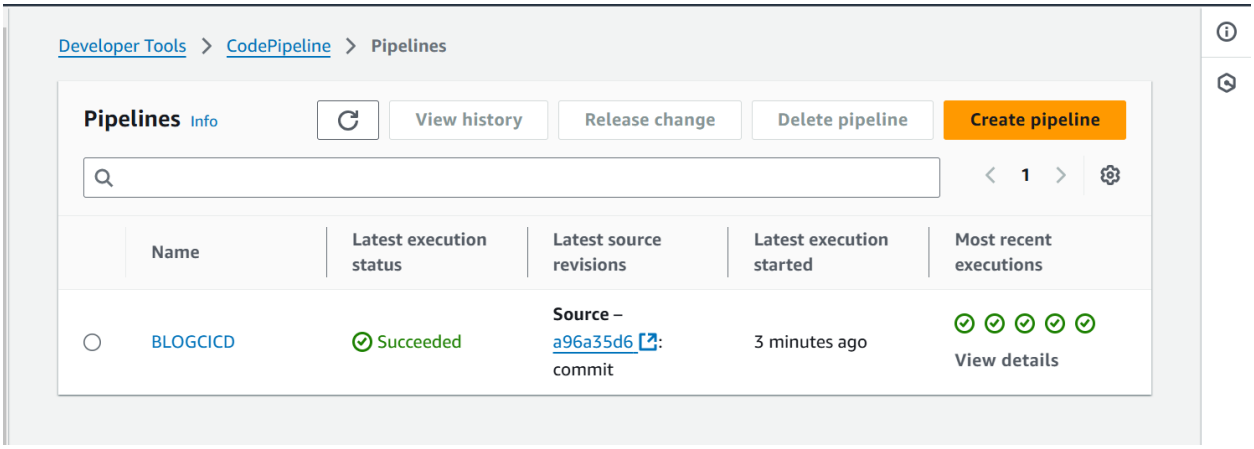
Default branch will be used only when pipeline execution starts from a different source or manually started.

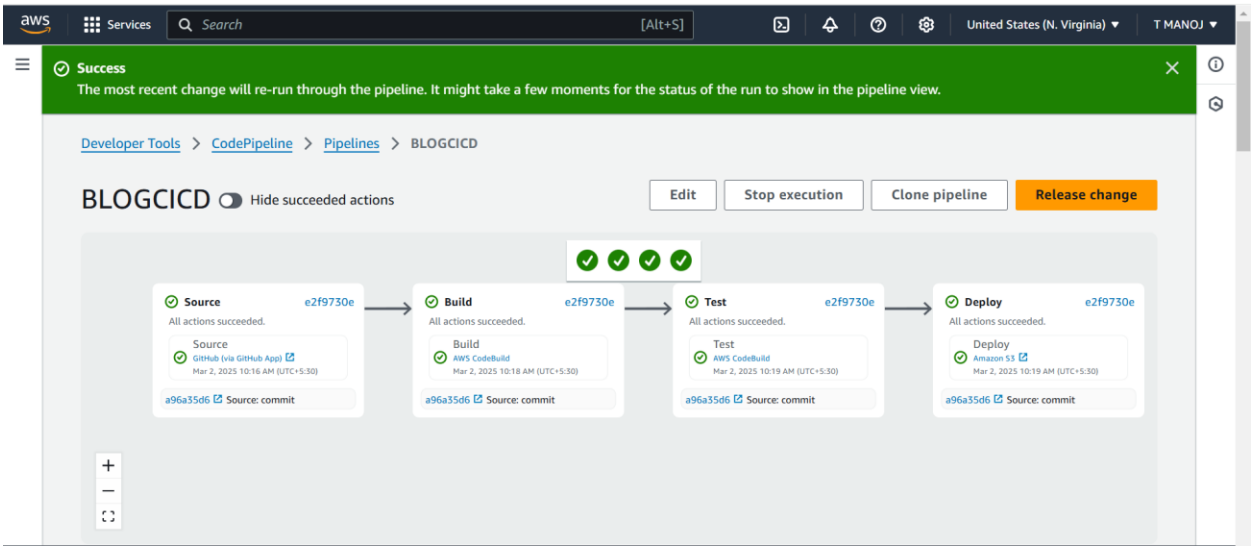
main

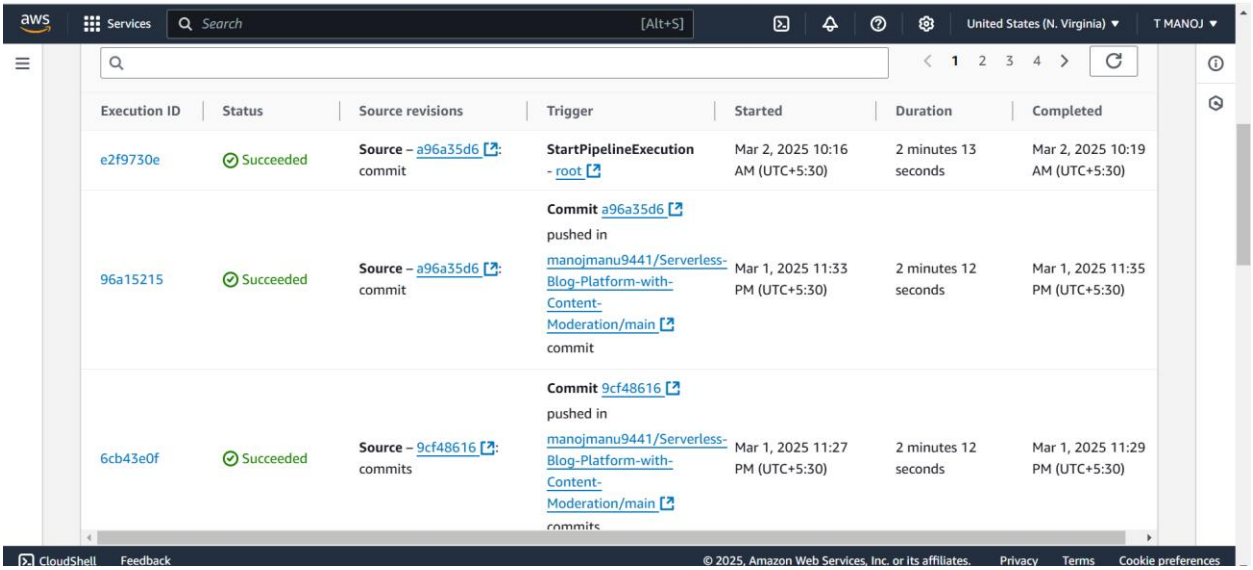
Output artifact format

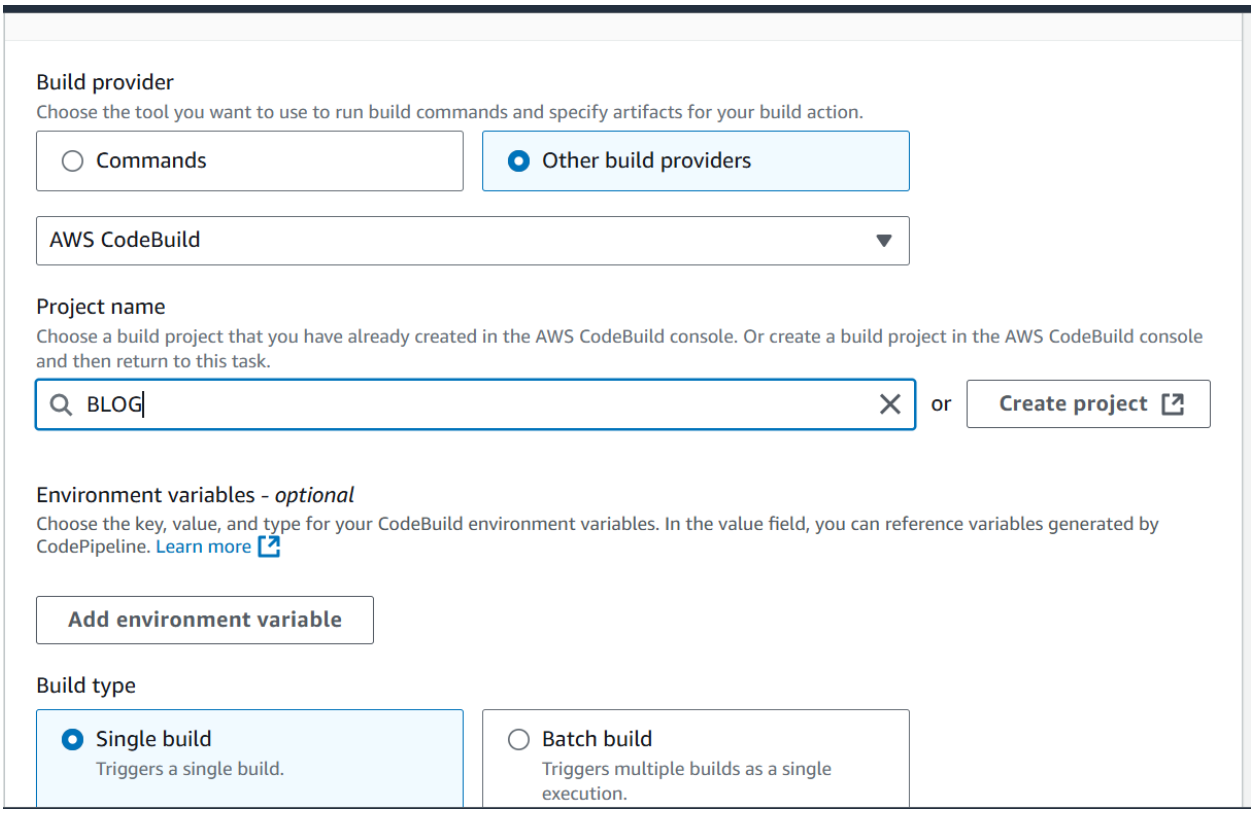
Choose the output artifact format.

CodePipeline default | Full clone



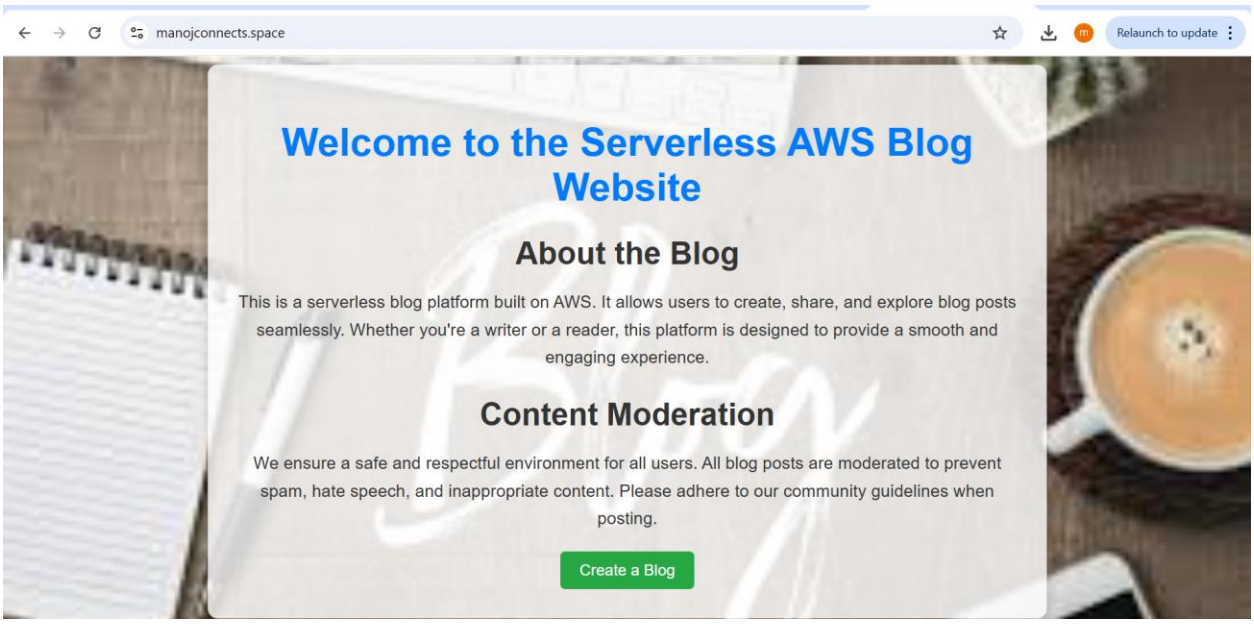

aws
Services
Search
[Alt+S]
United States (N. Virginia)
T MANOJ
Success
The most recent change will re-run through the pipeline. It might take a few moments for the status of the run to show in the pipeline view.
Developer Tools > CodePipeline > Pipelines > BLOGCICD
BLOGCICD
Hide succeeded actions
Edit
Stop execution
Clone pipeline
Release change
Source e2f9730e
All actions succeeded.
Source
GitHub (via GitHub App)
Mar 2, 2025 10:16 AM (UTC+5:30)
a96a35d6 Source: commit
Build e2f9730e
All actions succeeded.
Build
AWS CodeBuild
Mar 2, 2025 10:18 AM (UTC+5:30)
a96a35d6 Source: commit
Test e2f9730e
All actions succeeded.
Test
AWS CodeBuild
Mar 2, 2025 10:19 AM (UTC+5:30)
a96a35d6 Source: commit
Deploy e2f9730e
All actions succeeded.
Deploy
Amazon S3
Mar 2, 2025 10:19 AM (UTC+5:30)
a96a35d6 Source: commit

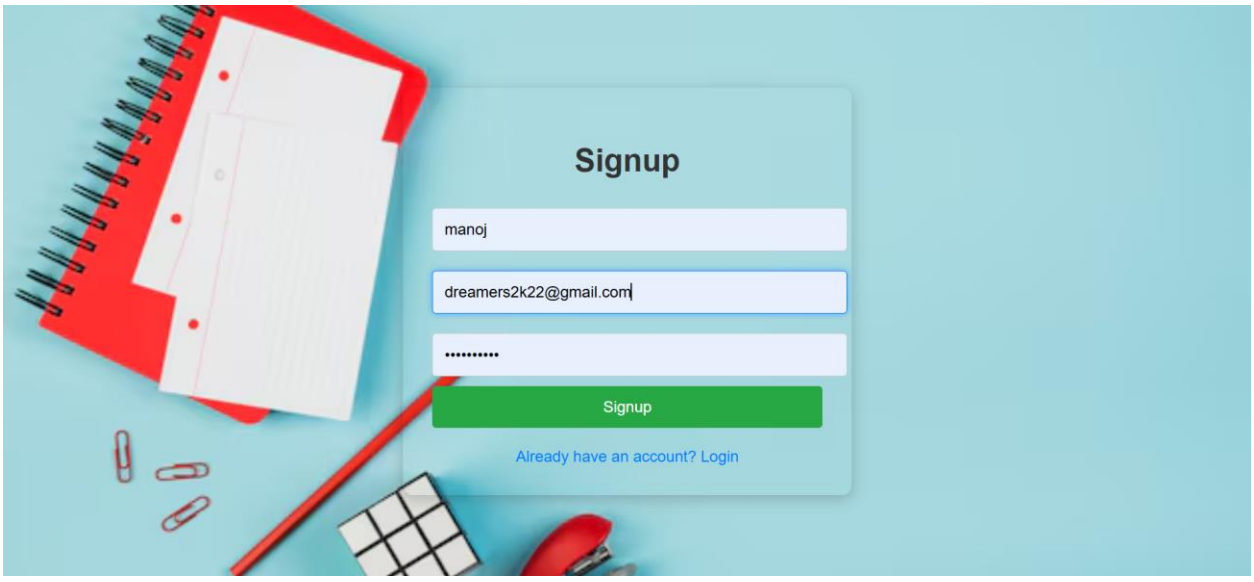

aws
Services
Search
[Alt+S]
United States (N. Virginia)
T MANOJ
1 2 3 4
Execution ID
Status
Source revisions
Trigger
Started
Duration
Completed
e2f9730e
Succeeded
Source – a96a35d6: commit
StartPipelineExecution - root
Mar 2, 2025 10:16 AM (UTC+5:30)
2 minutes 13 seconds
Mar 2, 2025 10:19 AM (UTC+5:30)
96a15215
Succeeded
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
Succeeded
Source – 9cf48616: commits
Commit 9cf48616 pushed in manojmanu9441/Serverless-Blog-Platform-with-Content-Moderation/main commits
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

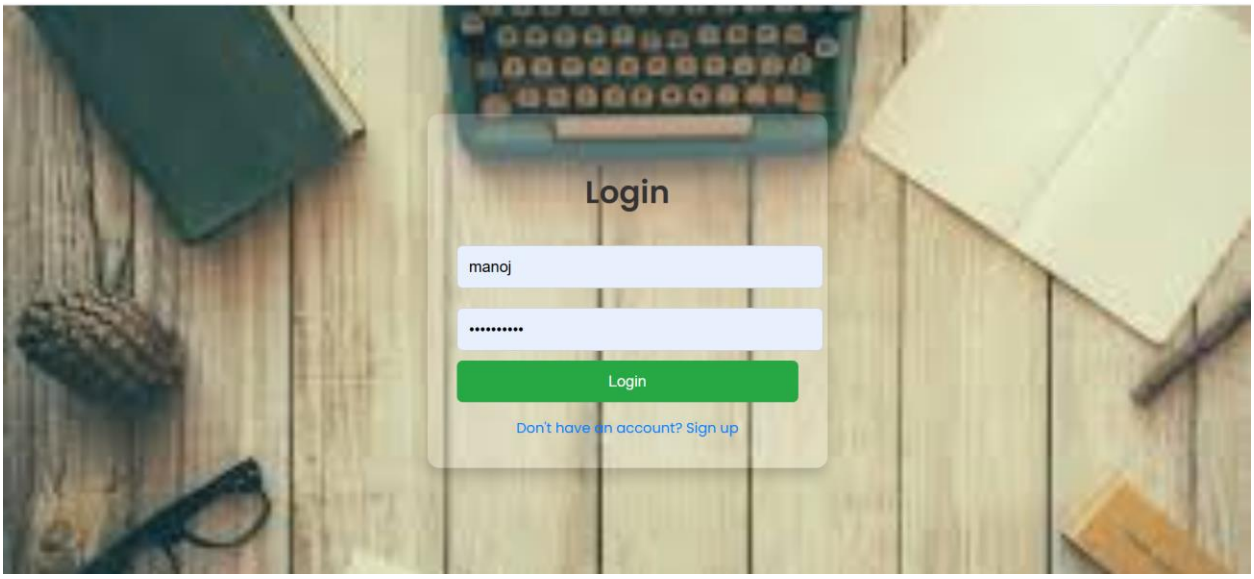


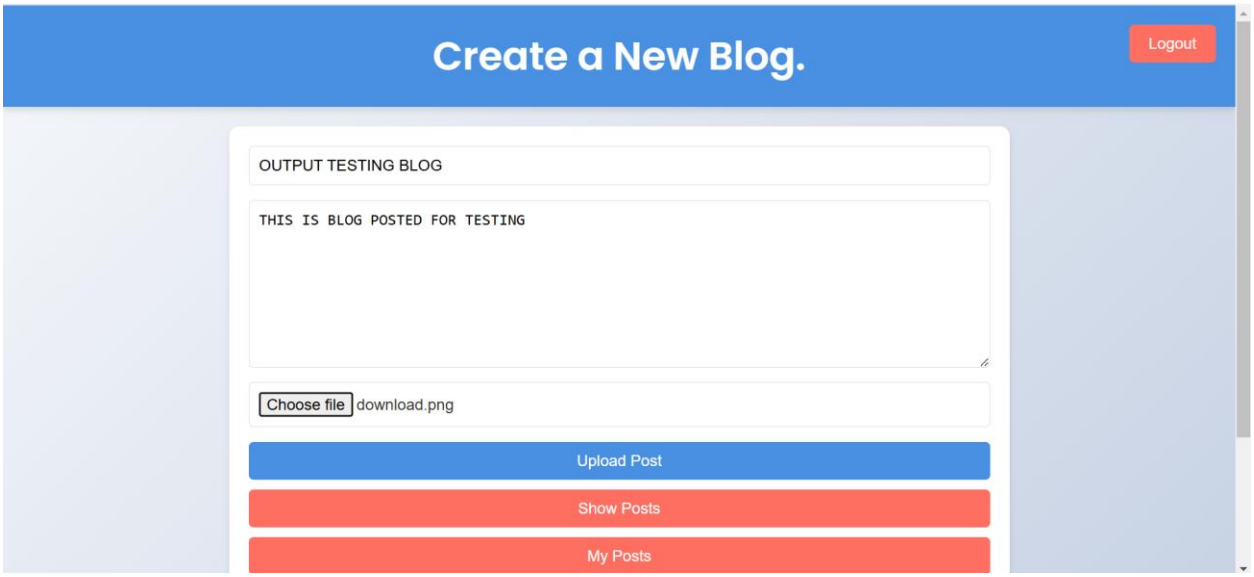
Phase 6: Final Testing

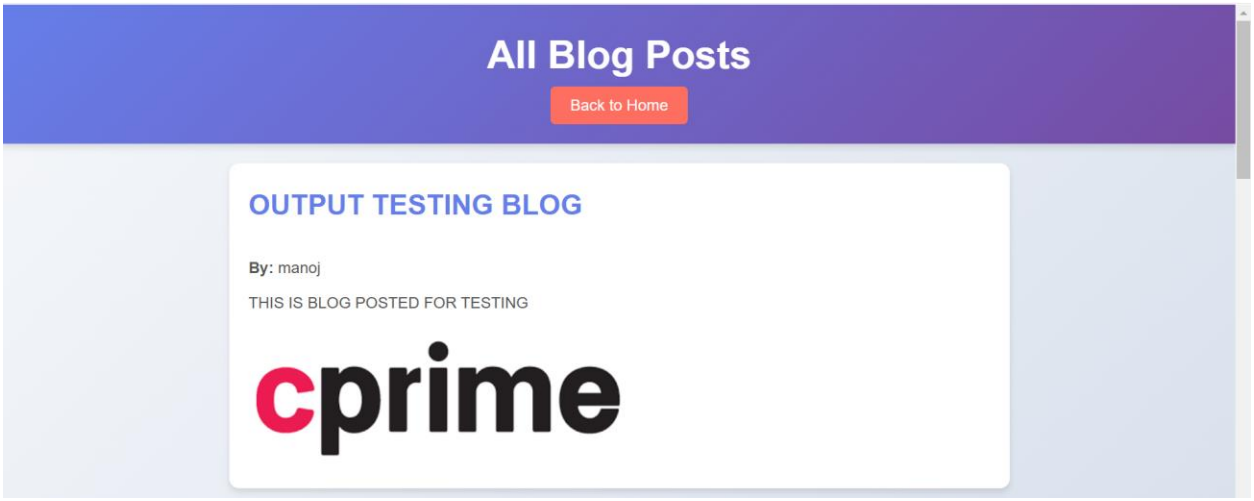
1. **Test All Features**:
    a. Signup, login, blog creation, image moderation, notifications.
2. **Check CloudFront and Route53**:
    a. Ensure the application is accessible via the custom domain.


manojconnects.space
Relaunch to update
Welcome to the Serverless AWS Blog Website
About the Blog
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.
Content Moderation
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.
Create a Blog


Signup
manoj
dreamers2k22@gmail.com
Signup
Already have an account? Login


Login
manoj
Login
Don't have an account? Sign up


Create a New Blog.
Logout
OUTPUT TESTING BLOG
THIS IS BLOG POSTED FOR TESTING
Choose file
download.png
Upload Post
Show Posts
My Posts



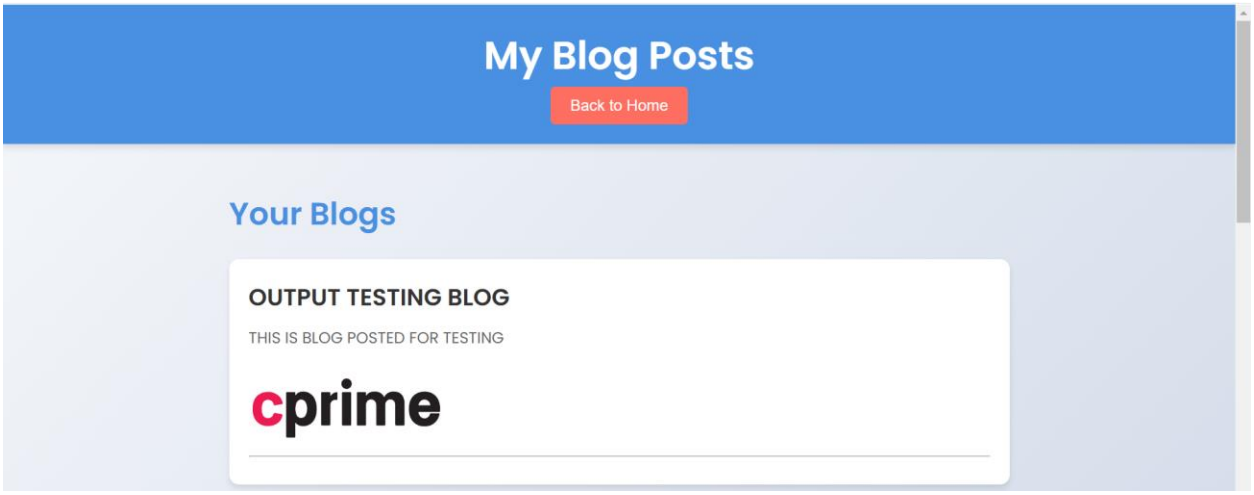
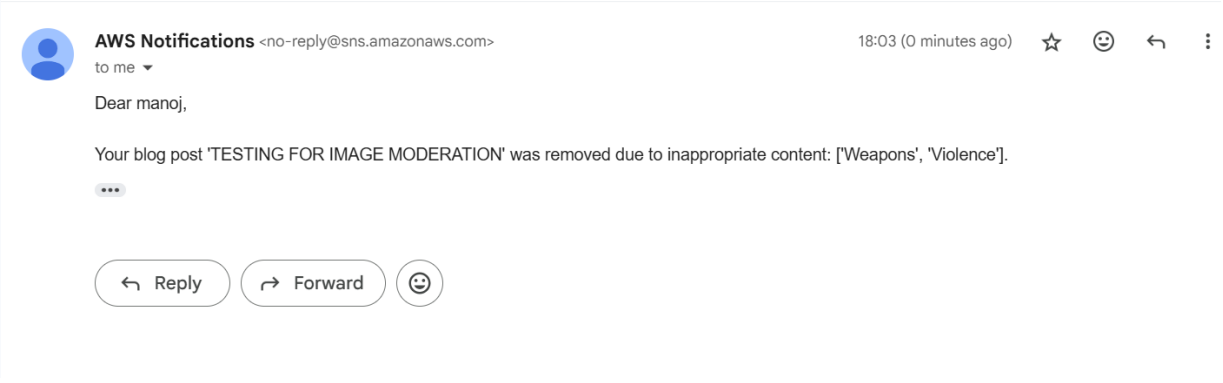
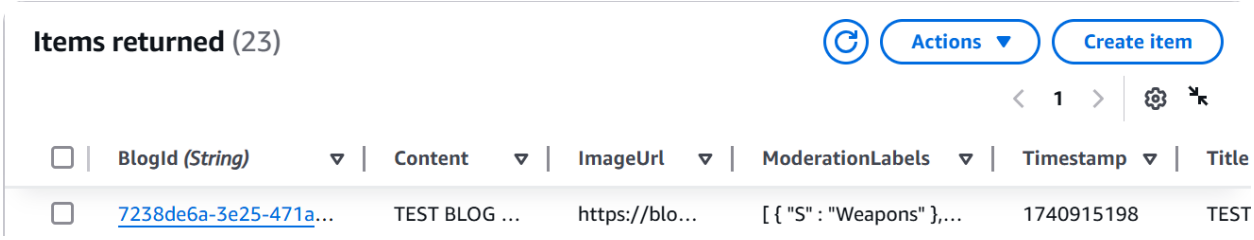


IMAGE MODERATION : BLOG POSTED WITH THIS IMAGE





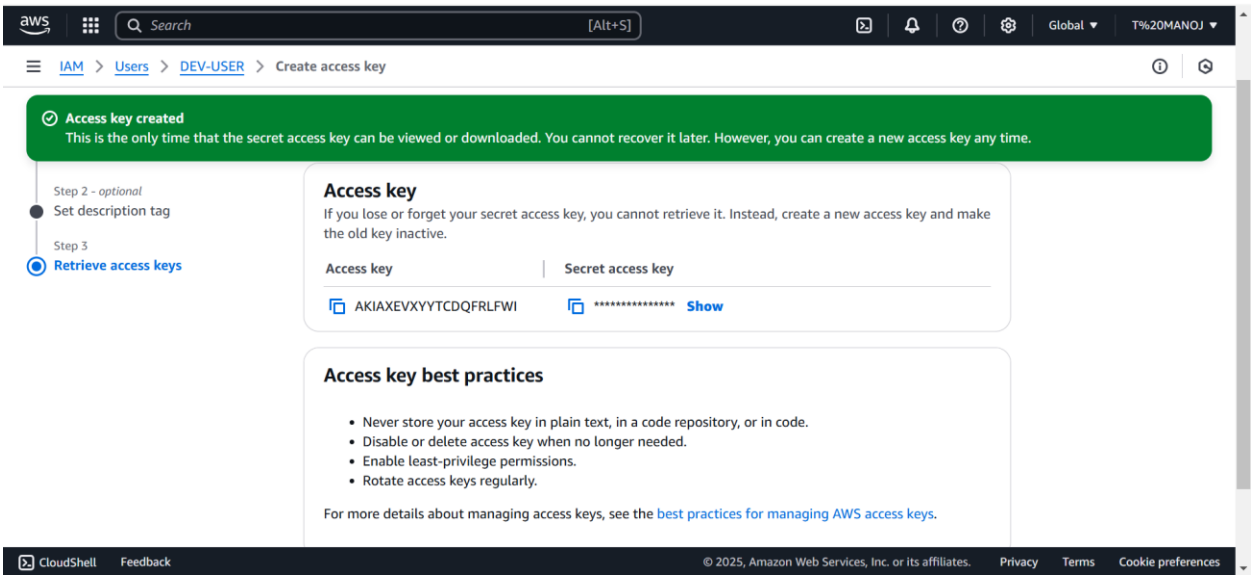
Phase 7: CloudFormation Template

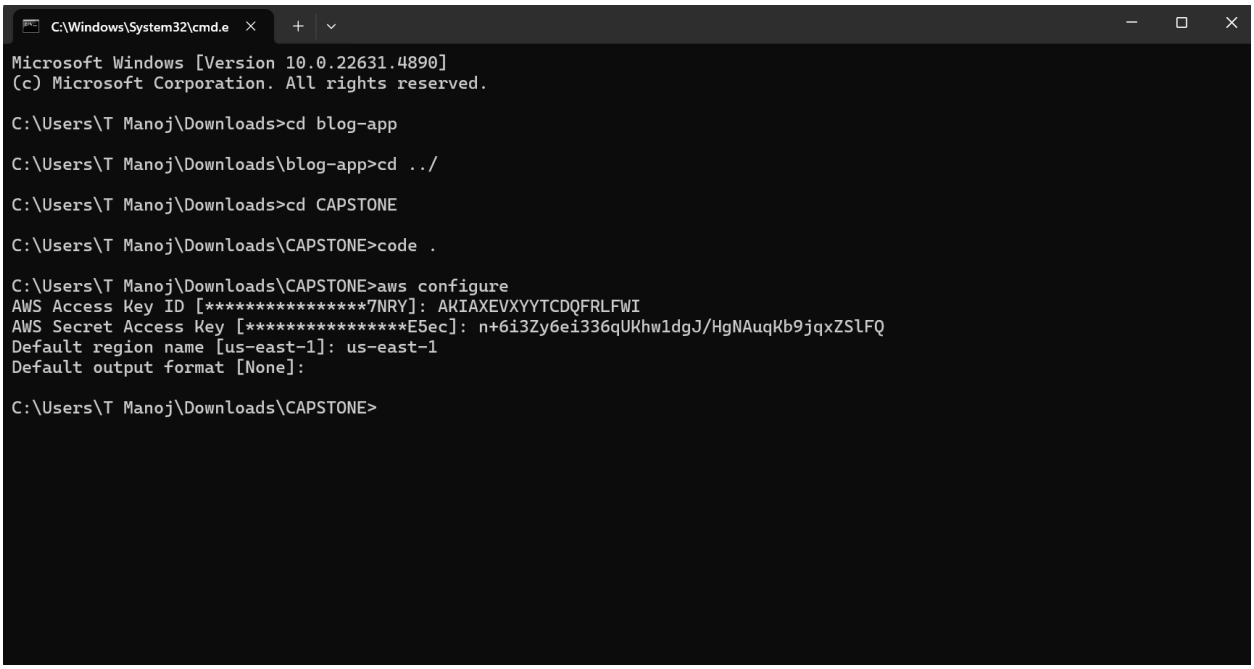
1. **Template**:
    a. Define all resources (DynamoDB, Lambda, S3, API Gateway, etc.) in a CloudFormation template.

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.





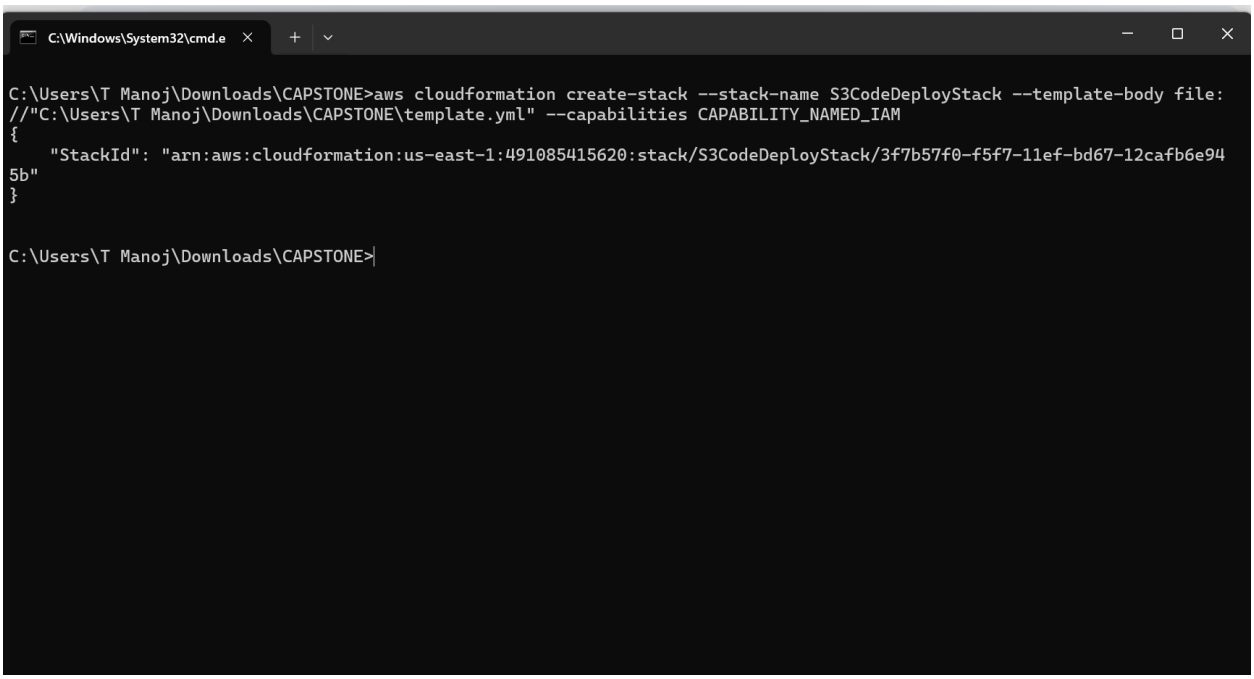
7. **Outputs**: API Gateway endpoint URLs, S3 bucket name, etc.

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

```yaml
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
```

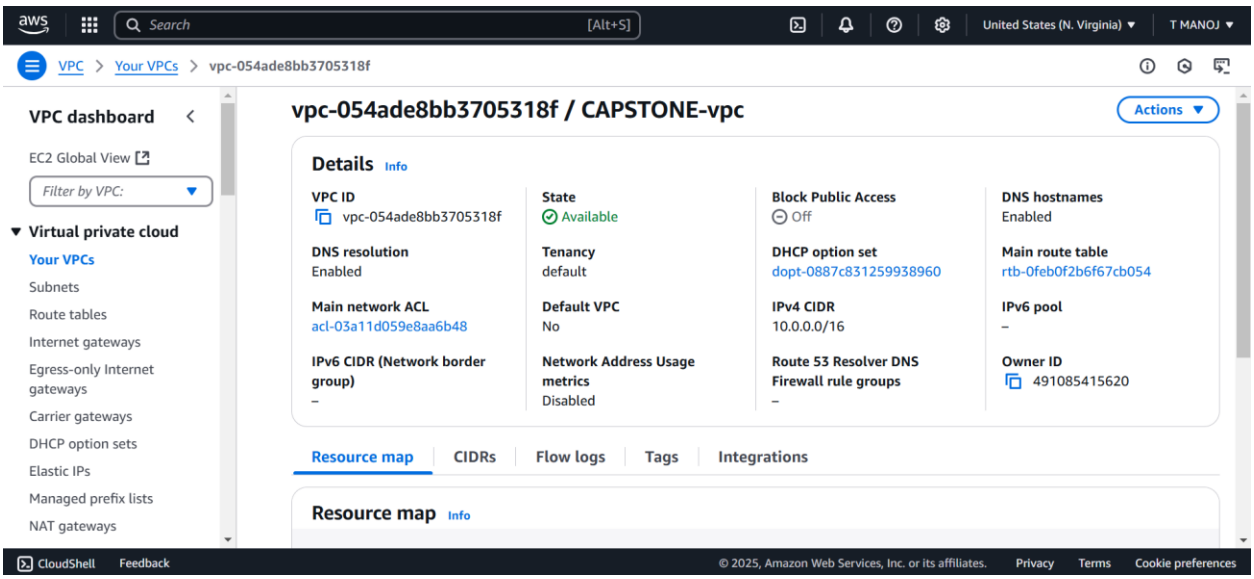
```
      ComputePlatform: Server

  # 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**:
    a. Use the template to automate resource creation.



Phase 8: VPC Configuration

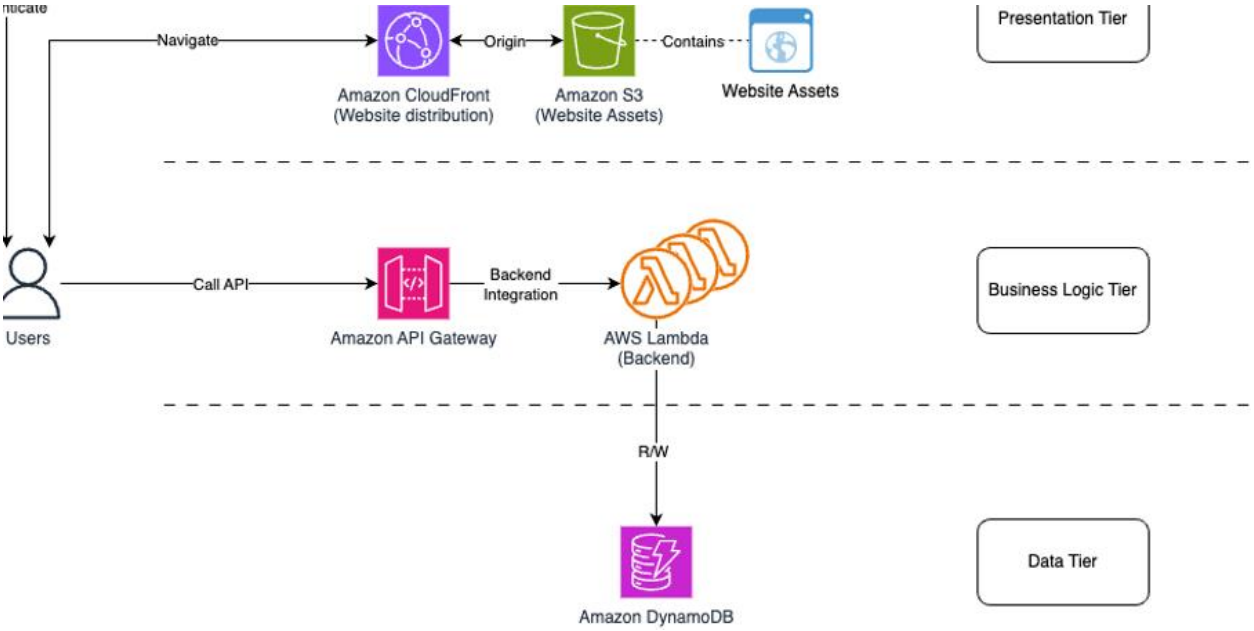
1. **Create VPC**:
    a. Define subnets, route tables, and security groups.



Place Resources:

b. 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/