AWS Cloud-Based Discussion Forum

by

Project Team: **Cloud Crafters**

Raahul Krishna	- 821934171
Komal Kiri	- 825935760
Sumit Singh	- 884416991
Vidhi Sharma	- 820240299
Prem Brahmbhatt	- 826308843

Discussion Forum Title: TopicTribe

A project report submitted for the fulfilment of

CPSC 465: Modern Software Deployment and Operations

Department of Computer Science
California State University, Fullerton

Submitted on 2nd December 2024

Contents

Table Of Figures

Abstract

This project implements a cloud-based discussion forum making use of the different resources provided by Amazon Web Services (AWS). The said platform allows for an interaction between users in which the users can talk to each other, exchange ideas and form communities. The front end of the forum has been built using Next.js, Amazon Cognito as a means of authenticated user interaction securely, and AWS Lambda incorporated through API Gateway as the backend for the system, thus providing a robust and scalable architecture for the forum. Its broad capabilities manage to include, but are not limited to, real-time post updates, email updates, and elastic scalability dealing with a constant flow of traffic or even moderate flow of it. The platform provides a dependable and consistent user experience by maximizing the available capabilities of the AWS elements while guaranteeing protection of the sensitive matters and making the system user friendly. The system is autoscaling in nature meaning it grows seamlessly to meet demand without performance or security risk. This project highlights the advantages of modern web development techniques and integrating them with cloud-based architecture to improve the building of highly functional online community platforms.

Keywords

AWS, Discussion Forum, Cognito, AWS Lambda, Next.js, DynamoDB, Scalability, Serverless

1. Introduction

1.1 Background

- **Growing Need for Online Communities**: As more individuals and organizations seek digital spaces for collaboration, learning, and engagement, cloud-based discussion forums are essential for fostering inclusive, scalable, and real-time online communities.

- **Balancing Scalability and Cost:** Ensuring the platform can scale dynamically to handle traffic spikes while maintaining cost efficiency is a significant challenge when building a cloud-based discussion forum.

1.2 Objective

The objective of this project is to create a secure, scalable, and feature-rich cloud-based discussion forum that provides a seamless and engaging user experience. The key goals are:

1. **Ensure Secure User Authentication**: Provide a secure and reliable authentication process, protecting user data and interactions from unauthorized access.

2. **Deliver a Seamless User Experience**: Create a fast, responsive, and intuitive interface that facilitates smooth user interactions, fostering engagement and community-building.

3. **Enable Scalability**: Design the platform to handle growing user traffic and data efficiently, ensuring the system can scale dynamically without compromising performance.

4. **Support Real-Time Interaction**: Implement features that enable users to participate in real-time discussions, receive instant updates, and stay engaged through timely notifications.

5. **Ensure Modular and Maintainable Architecture**: Adopt a flexible architecture that allows for independent management of different forum components (such as forums, posts, and votes), making future updates and enhancements easier.

6. **Ensure Reliable Messaging**: Guarantee the reliable delivery of important messages, such as notifications to users via e-mail, ensuring consistent communication within the platform.

1.3 Scope

This project focuses on developing a secure, scalable cloud-based discussion forum that allows users to create and participate in discussions, vote on posts, and receive real-time notifications. Key features include secure user authentication, real-time updates, email notifications, and a serverless architecture for easy scalability. The platform will support a growing user base, providing a seamless user experience.

However, the project will not include advanced content moderation tools, third-party integrations, or mobile-specific features in its initial version. The scope is also limited to English-language support, with future phases planned for expanding functionalities.

2. Related Work

- Existing cloud-based discussion forums, such as those built on Firebase or Azure, can face challenges scaling efficiently during periods of high traffic. AWS offers a more robust solution with services like AWS Lambda for serverless computing and Amazon DynamoDB for scalable data storage, ensuring the platform can automatically adjust resources to meet growing demand without compromising performance.

- While platforms like Firebase or local servers require manual configuration for security and real-time features, AWS provides built-in solutions like Amazon Cognito for secure user authentication and API Gateway for routing requests. This enables seamless, secure interactions and real-time updates, enhancing user engagement while maintaining a reliable and secure platform.

3. System Design and Architecture

3.1 System Overview

The AWS Cloud-Based Discussion Forum is a secure, scalable platform built to facilitate real-time user interactions and community engagement. On the front end, the platform uses Next.js to provide a responsive and dynamic user interface for creating forums, posting content, and interacting with discussions. Amazon Cognito handles secure user authentication, ensuring only registered users can access certain features. On the backend, AWS Lambda processes API requests routed through API Gateway, while Amazon DynamoDB stores user data, posts, and votes. Real-time interactions and notifications are managed through Amazon SES, keeping users informed about new posts or replies. The serverless architecture ensures the platform can scale seamlessly, providing high availability and reliability as user traffic grows.

3.2 Architecture

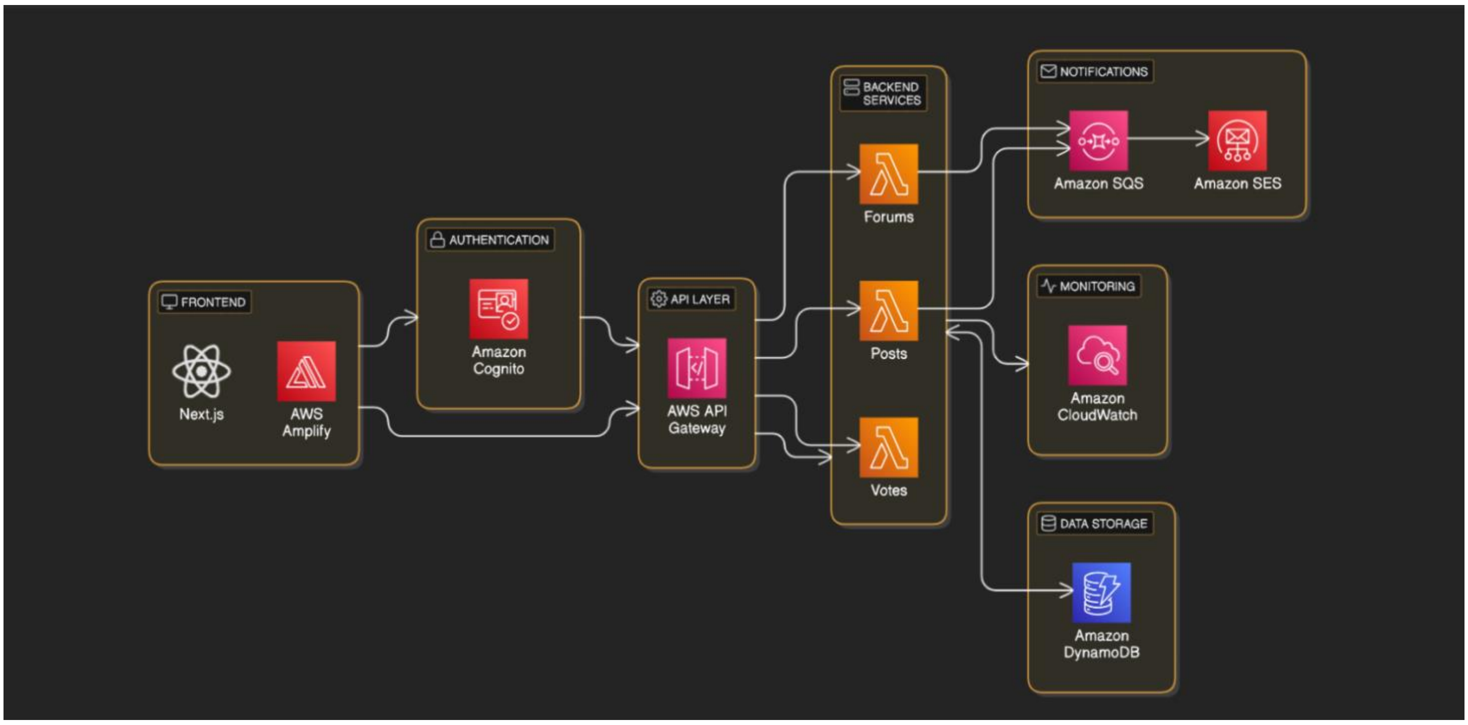


Figure 1: Process Flow Diagram

3.3 Workflow

The architecture is built on AWS cloud services, using a microservices approach:

1. Front-end: Next.js for frontend framework. AWS Amplify is used to host the front-end.

2. User Authentication: Amazon Cognito will manage user sign-in and sign-up processes.

3. API Layer: AWS API Gateway routes incoming requests to respective microservices.

4. Backend Services: AWS Lambda for serverless backend logic (Python-based) for handling forums, posts, and votes.

5. Data Storage: Amazon DynamoDB as the NoSQL database to store forum-related data.

6. Monitoring and Logging: Amazon CloudWatch for monitoring application health and storing logs.

7. Notifications: Amazon SES sends email notifications to users when a forum or post is created. Amazon SQS stores email requests, ensuring reliable message delivery.

This architecture ensures scalability, high availability, and modularity, allowing for future enhancements and optimizations.

4. Implementation

4.1 Technologies Used

- Front-End: Next.js
- Front-End Deployment: AWS Amplify
- Authentication: Amazon Cognito
- Back-End Services: AWS Lambda (Python)
- Database: Amazon DynamoDB (NoSQL)
- API Layer: AWS API Gateway
- Notifications: Amazon SES (Email), Amazon SQS (Queuing)
- Logging: AWS CloudWatch

4.2 Key Features

- **Secure User Authentication and Authorization**: Users can securely sign up, log in, and manage their accounts with Amazon Cognito, ensuring protected access to forum features.

- **Real-Time Post and Interaction**: Users can create and interact with posts and comments in real-time, with changes instantly reflected on the platform.

- **Scalable, Serverless Backend**: The platform uses AWS Lambda for serverless processing and Amazon DynamoDB for scalable, high-performance data storage, ensuring smooth operation even with increasing traffic.

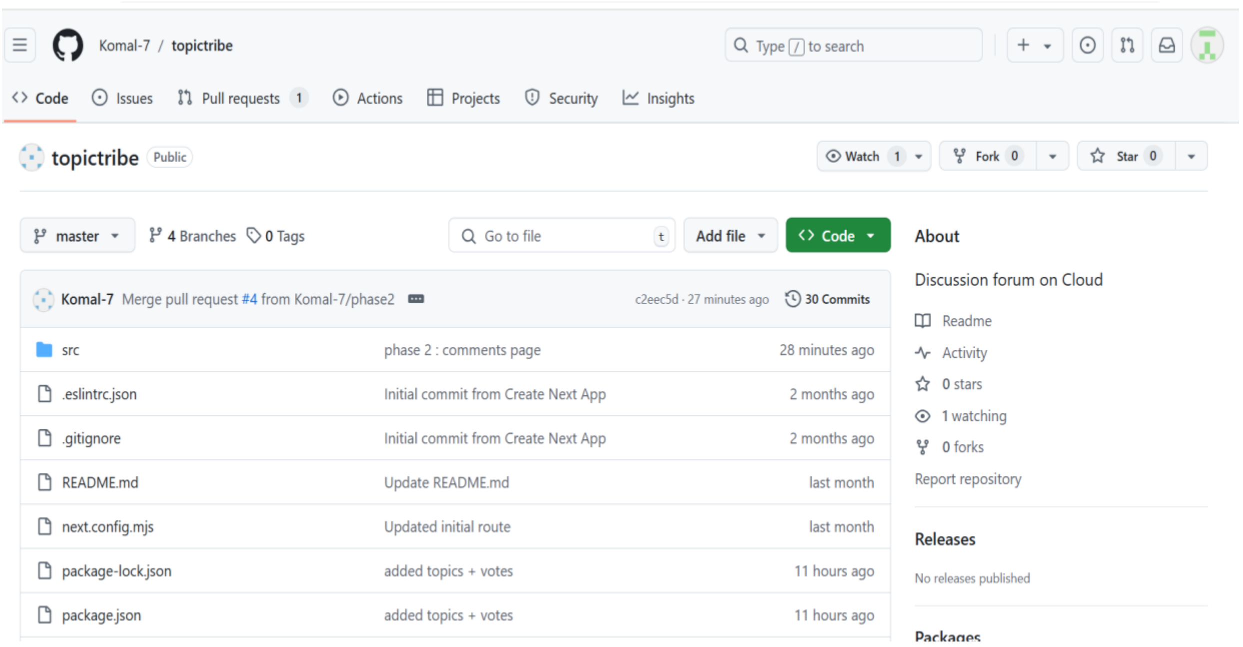
- **Real-Time Notifications**: Users receive timely email notifications through Amazon SES when new content is posted or when their posts receive responses.

- **Modular Microservices Architecture**: The platform follows a microservices-based architecture, allowing independent management of forums, posts, votes, and user data for easier maintenance and future scalability.

4.3 Code Snippets and Configurations

4.3.1 AWS Configurations

1. **Frontend Deployment using AWS Amplify-** The frontend code has been created in Next.js framework and has been deployed on AWS Amplify. The frontend code resides in a GitHub repository, from where Amplify fetches it for deployment.



Komal-7 Merge pull request #4 from Komal-7/phase2 c2eec5d · 27 minutes ago 30 Commits

src	phase 2 : comments page	28 minutes ago
.eslintrc.json	Initial commit from Create Next App	2 months ago
.gitignore	Initial commit from Create Next App	2 months ago
README.md	Update README.md	last month
next.config.mjs	Updated initial route	last month
package-lock.json	added topics + votes	11 hours ago
package.json	added topics + votes	11 hours ago

About

Discussion forum on Cloud

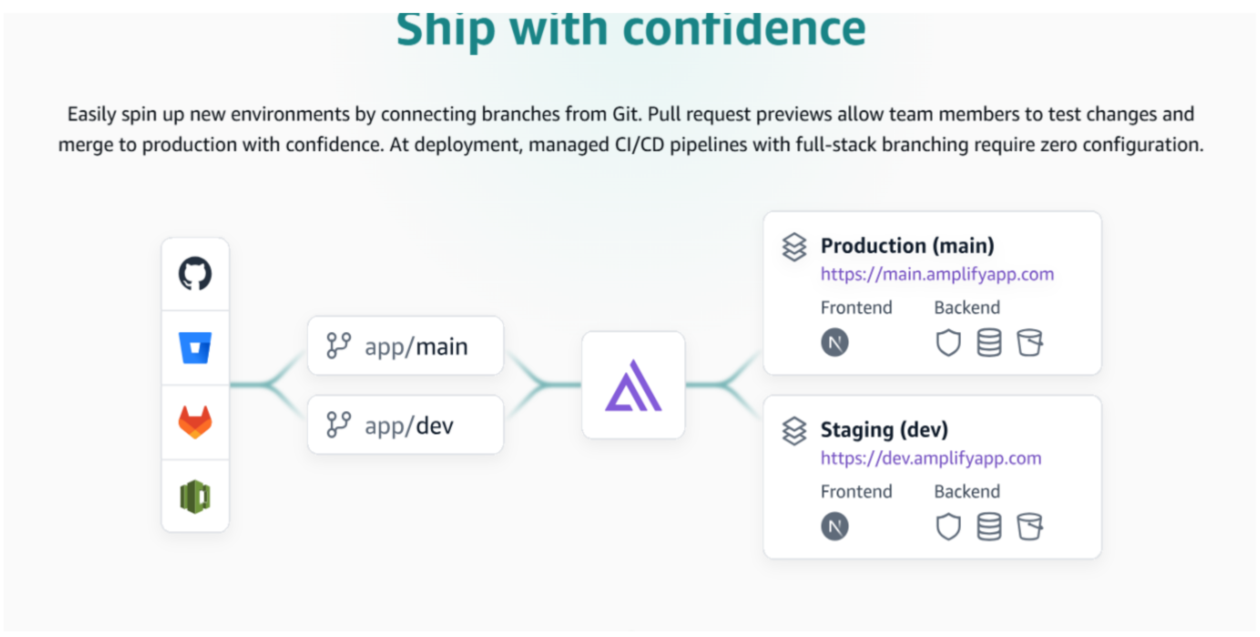
Readme
Activity
0 stars
1 watching
0 forks
Report repository

Releases

No releases published

Packages

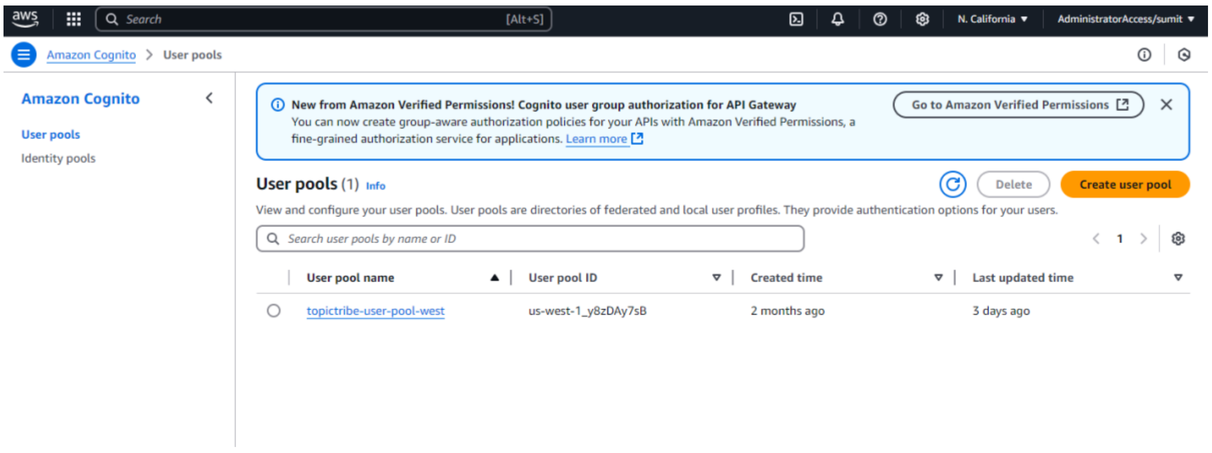
Figure 2:GitHub Repository


Ship with confidence

Easily spin up new environments by connecting branches from Git. Pull request previews allow team members to test changes and merge to production with confidence. At deployment, managed CI/CD pipelines with full-stack branching require zero configuration.

Figure 3: AWS Amplify

2. **User Authentication Via Amazon Cognito -** A user pool in Amazon Cognito has been created which will be storing the user information such as name, email and account passwords.



New from Amazon Verified Permissions! Cognito user group authorization for API Gateway
You can now create group-aware authorization policies for your APIs with Amazon Verified Permissions, a fine-grained authorization service for applications. Learn more

User pools (1) Info

View and configure your user pools. User pools are directories of federated and local user profiles. They provide authentication options for your users.

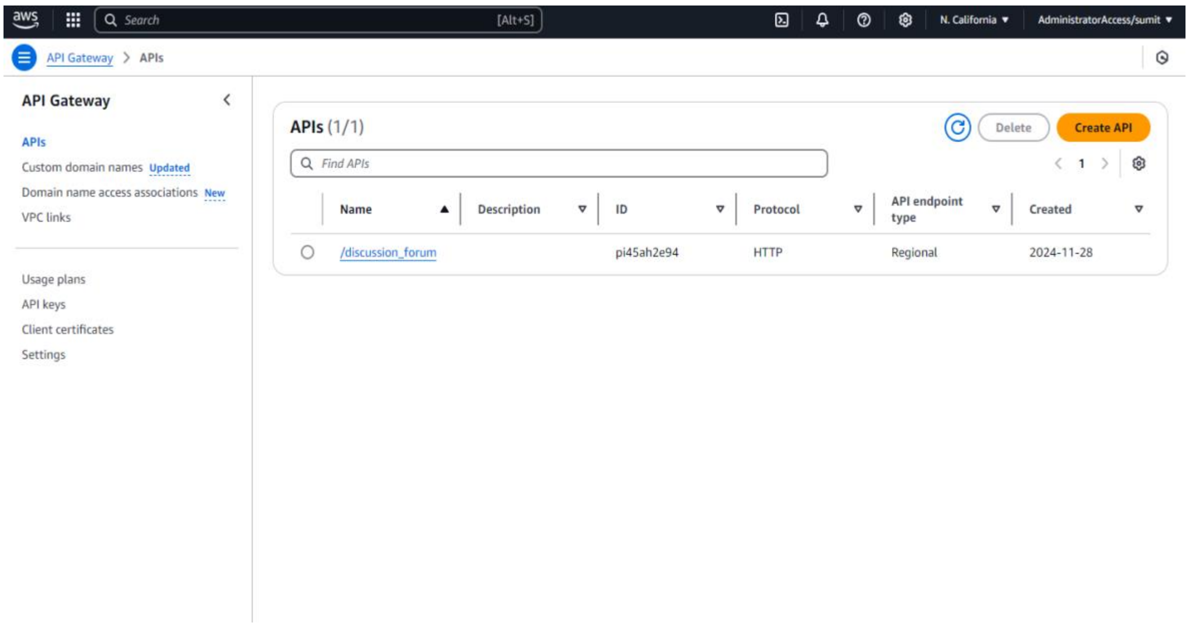
Search user pools by name or ID

User pool name	User pool ID	Created time	Last updated time
topictribe-user-pool-west	us-west-1_y8zDAy7sB	2 months ago	3 days ago

Figure 4: User Authentication Via Amazon Cognito

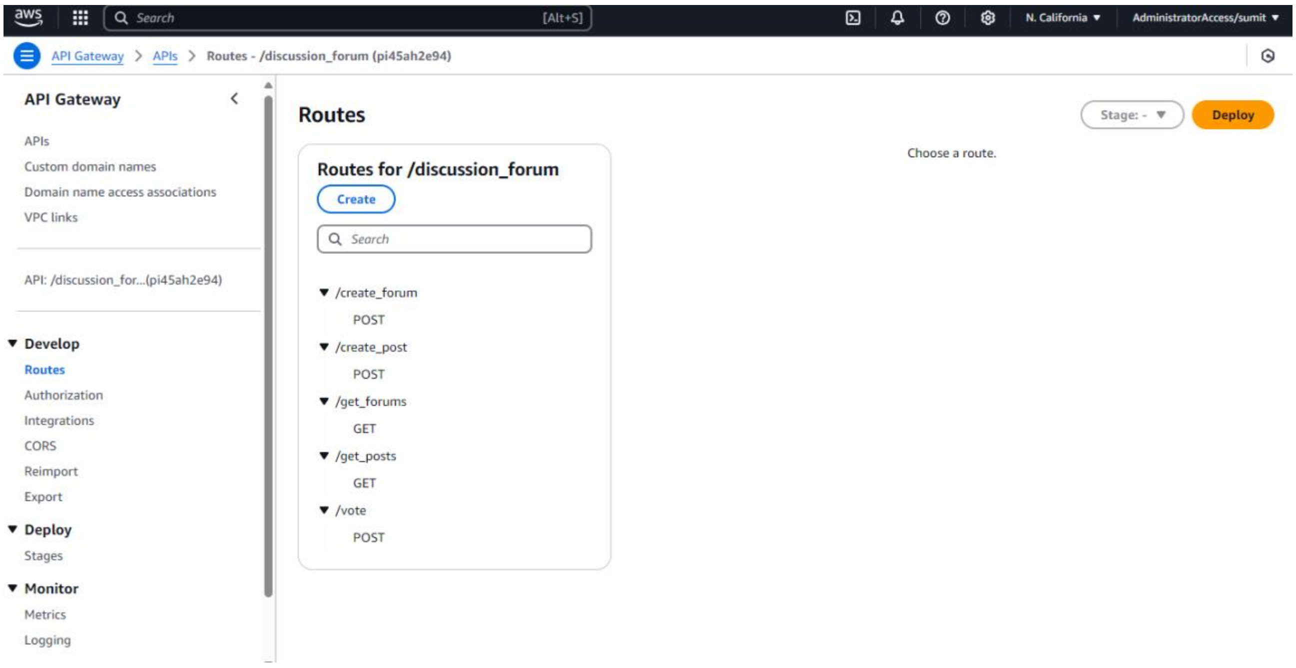
3. **API Gateway for Redirecting User Requests to Lambda Functions:** After the user has been authenticated, Amazon API Gateway will be directing these user

requests to the appropriate Lambda functions, where the backend logic for various operations in the discussion forum is defined.



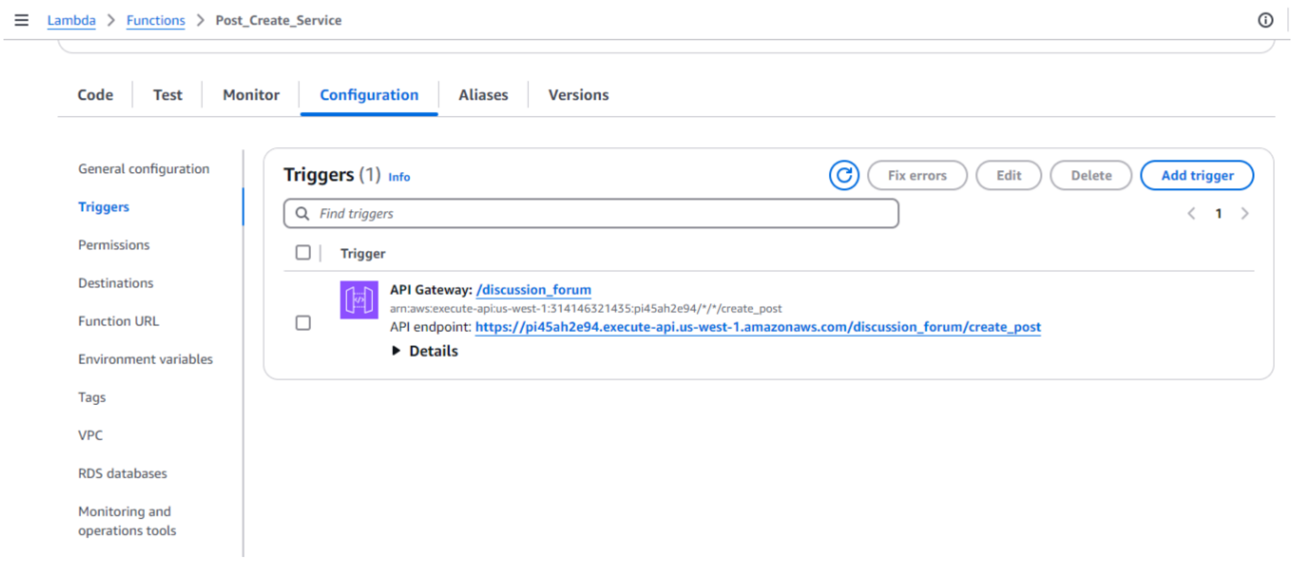
Name	Description	ID	Protocol	API endpoint type	Created
/discussion_forum		pi45ah2e94	HTTP	Regional	2024-11-28

Figure 5: API Gateway for Redirecting User Requests to Lambda Functions



- /create_forum
  - POST
- /create_post
  - POST
- /get_forums
  - GET
- /get_posts
  - GET
- /vote
  - POST

Figure 6: API Routes



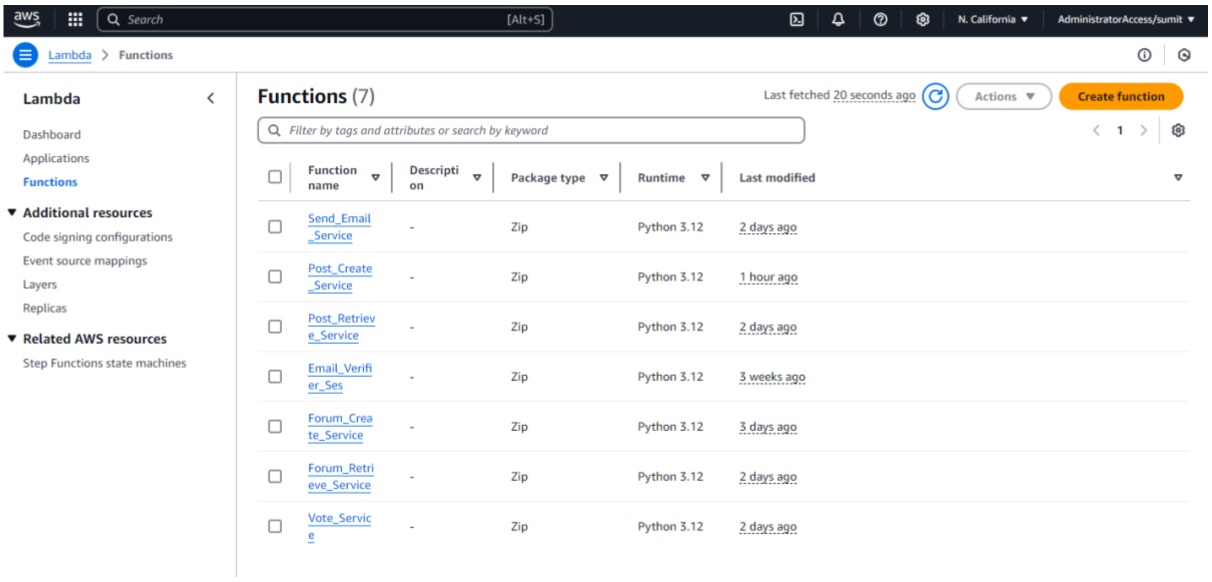
Triggers (1) Info

Trigger

API Gateway: /discussion_forum
arn:aws:execute-api:us-west-1:314146321435:pi45ah2e94/*/*/create_post
API endpoint: https://pi45ah2e94.execute-api.us-west-1.amazonaws.com/discussion_forum/create_post
▶ Details

Figure 7: Triggers

4. **Backend Logic Implemented Via AWS Lambda:** The backend logic for various operations like post, vote and comment in the discussion forum have been stored in AWS Lambda functions written Python.



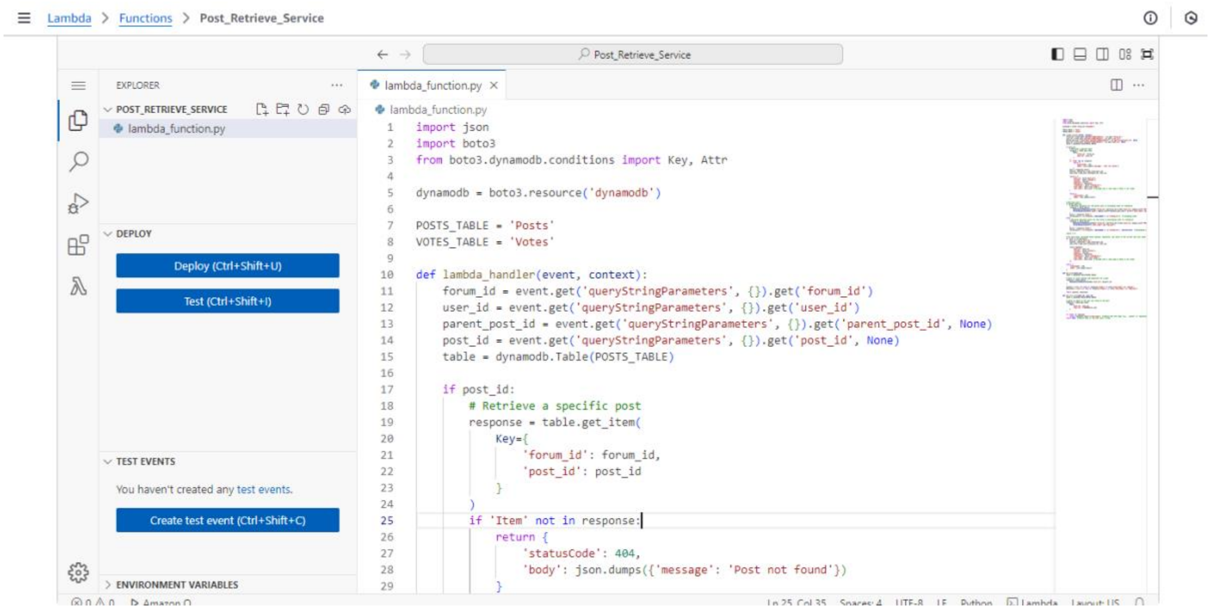
Lambda > Functions

Functions (7)

Last fetched 20 seconds ago

Function name	Description	Package type	Runtime	Last modified
Send_Email_Service	-	Zip	Python 3.12	2 days ago
Post_Create_Service	-	Zip	Python 3.12	1 hour ago
Post_Retrieve_Service	-	Zip	Python 3.12	2 days ago
Email_Verifier_Ses	-	Zip	Python 3.12	3 weeks ago
Forum_Create_Service	-	Zip	Python 3.12	3 days ago
Forum_Retrieve_Service	-	Zip	Python 3.12	2 days ago
Vote_Service	-	Zip	Python 3.12	2 days ago

Figure 8: AWS Lambda Functions



Lambda > Functions > Post_Retrieve_Service

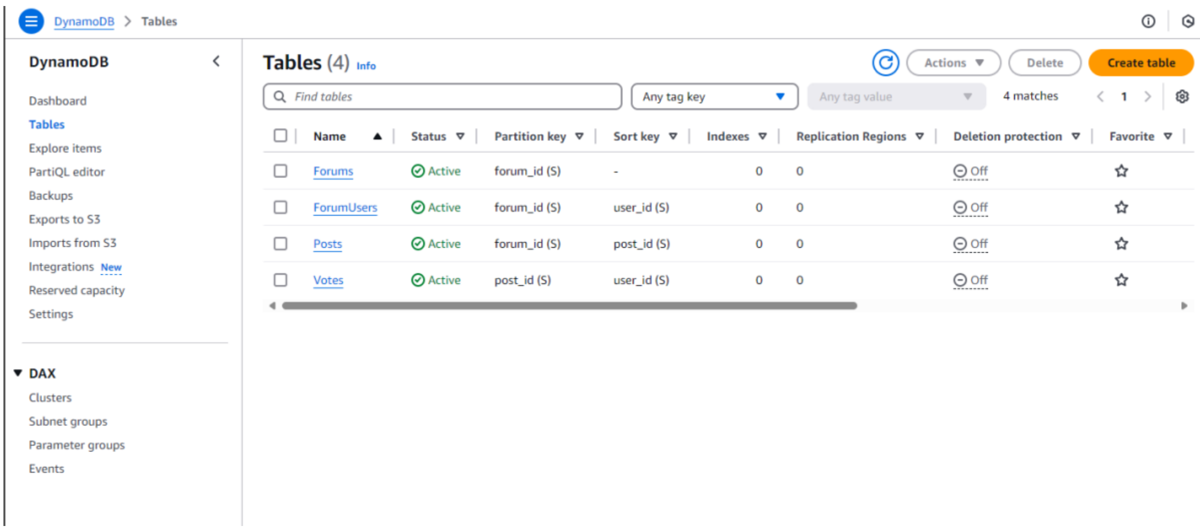
lambda_function.py

```python
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource('dynamodb')

POSTS_TABLE = 'Posts'
VOTES_TABLE = 'Votes'

def lambda_handler(event, context):
    forum_id = event.get('queryStringParameters', {}).get('forum_id')
    user_id = event.get('queryStringParameters', {}).get('user_id')
    parent_post_id = event.get('queryStringParameters', {}).get('parent_post_id', None)
    post_id = event.get('queryStringParameters', {}).get('post_id', None)
    table = dynamodb.Table(POSTS_TABLE)

    if post_id:
        # Retrieve a specific post
        response = table.get_item(
            Key={
                'forum_id': forum_id,
                'post_id': post_id
            }
        )
        if 'Item' not in response:
            return {
                'statusCode': 404,
                'body': json.dumps({'message': 'Post not found'})
            }
```

EXPLORER
POST_RETRIEVE_SERVICE
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS
You haven't created any test events.
Create test event (Ctrl+Shift+C)
ENVIRONMENT VARIABLES
Ln 25, Col 35 Spaces: 4 UTF-8 LF Python Lambda Layout: US

Figure 9: Lambda Functions Code

5. **Forum Data Stored in DynamoDB:** Forum data generated by various user operations such as new posts, new comments, vote count, etc. are stored in DynamoDB.

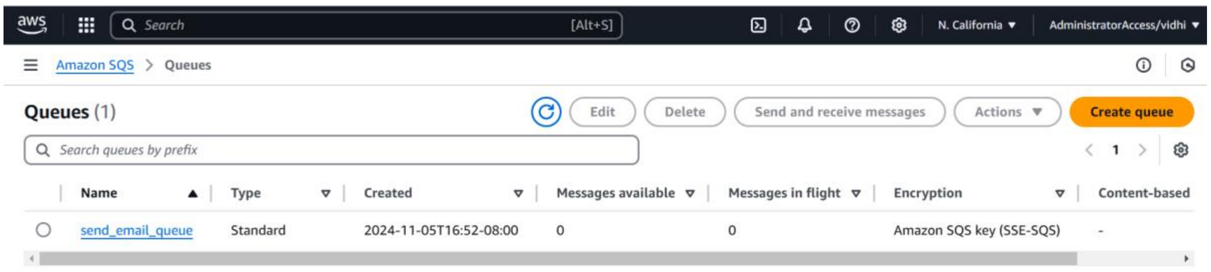


Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite
Forums	Active	forum_id (S)	-	0	0	Off	☆
ForumUsers	Active	forum_id (S)	user_id (S)	0	0	Off	☆
Posts	Active	forum_id (S)	post_id (S)	0	0	Off	☆
Votes	Active	post_id (S)	user_id (S)	0	0	Off	☆

Figure 10: DynamoDB Database

6. **Amazon SQS and SES for Managing Notifications:** The new notifications generated by the user operations in the forum will be conveyed to the users using amazon SQS and SES. Since the architecture is serverless and all operations are carried out in the form of lambda function calls, with each new change the user

needs to be notified about, a request will be coming in amazon SQS, which will then be forwarded to the user using SES.



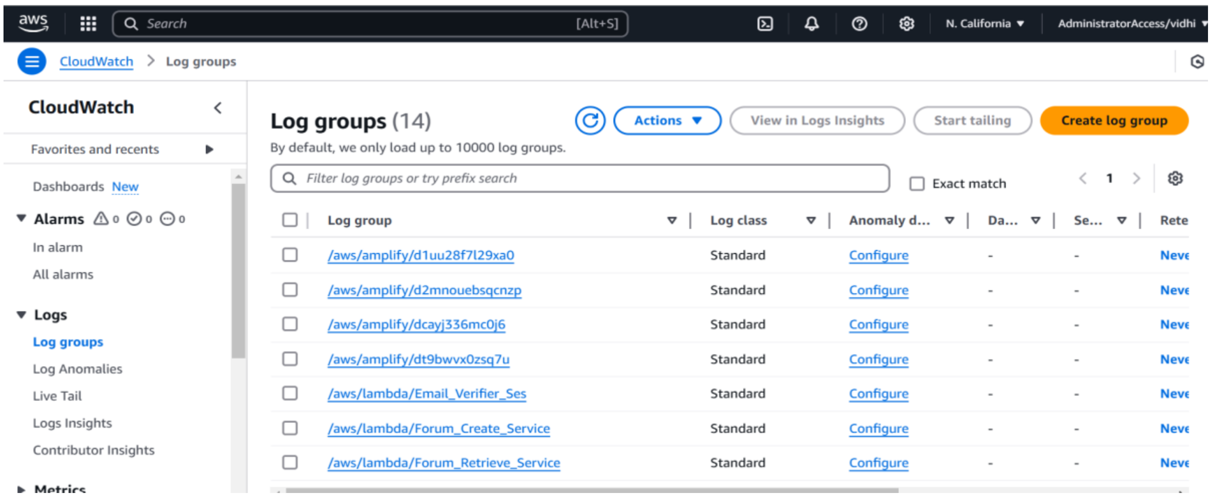
Amazon SQS > Queues

Queues (1)

	Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based
○	send_email_queue	Standard	2024-11-05T16:52-08:00	0	0	Amazon SQS key (SSE-SQS)	-

Figure 11: Amazon SQS

7. **AWS CloudWatch for Logging:** The logs generated by various user operations in the discussion forum will be logged and recorded by AWS CloudWatch.



CloudWatch > Log groups

Log groups (14)

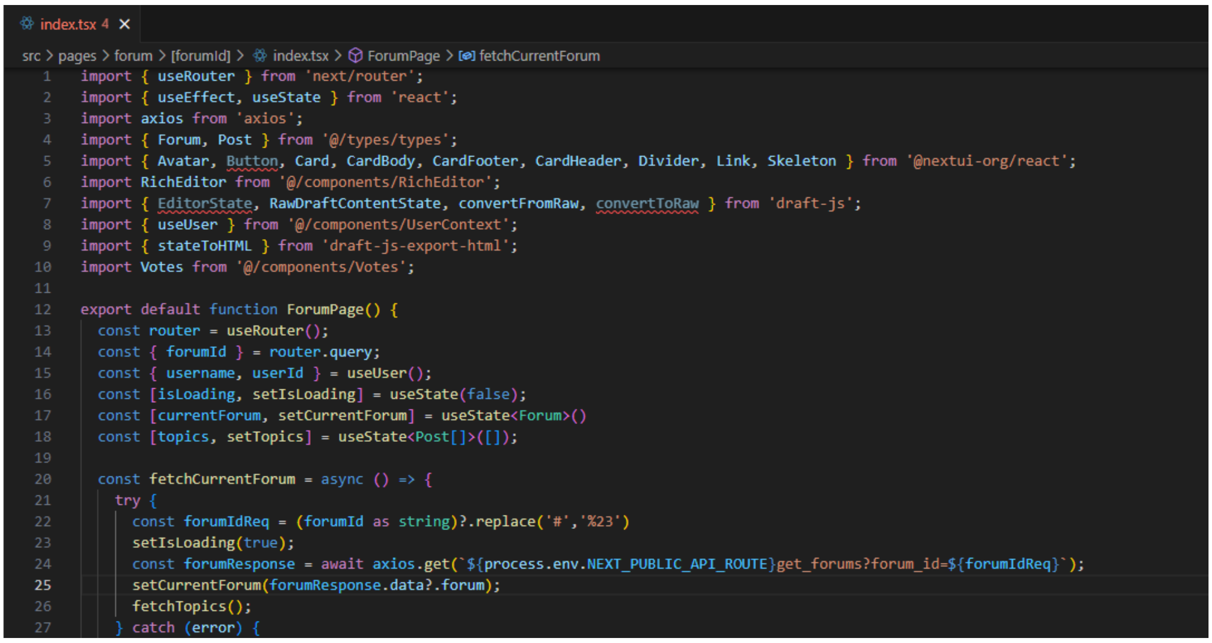
By default, we only load up to 10000 log groups.

Log group	Log class	Anomaly d...	Da...	Se...	Rete
/aws/amplify/d1uu28f7l29xa0	Standard	Configure	-	-	Neve
/aws/amplify/d2mnouebsqcnzp	Standard	Configure	-	-	Neve
/aws/amplify/dcayj336mc0j6	Standard	Configure	-	-	Neve
/aws/amplify/dt9bwvx0zsq7u	Standard	Configure	-	-	Neve
/aws/lambda/Email_Verifier_Ses	Standard	Configure	-	-	Neve
/aws/lambda/Forum_Create_Service	Standard	Configure	-	-	Neve
/aws/lambda/Forum_Retrieve_Service	Standard	Configure	-	-	Neve

Figure 12: Amazon CloudWatch for logs

4.3.2 Code Snippets

1. Front End Code (Next.js)

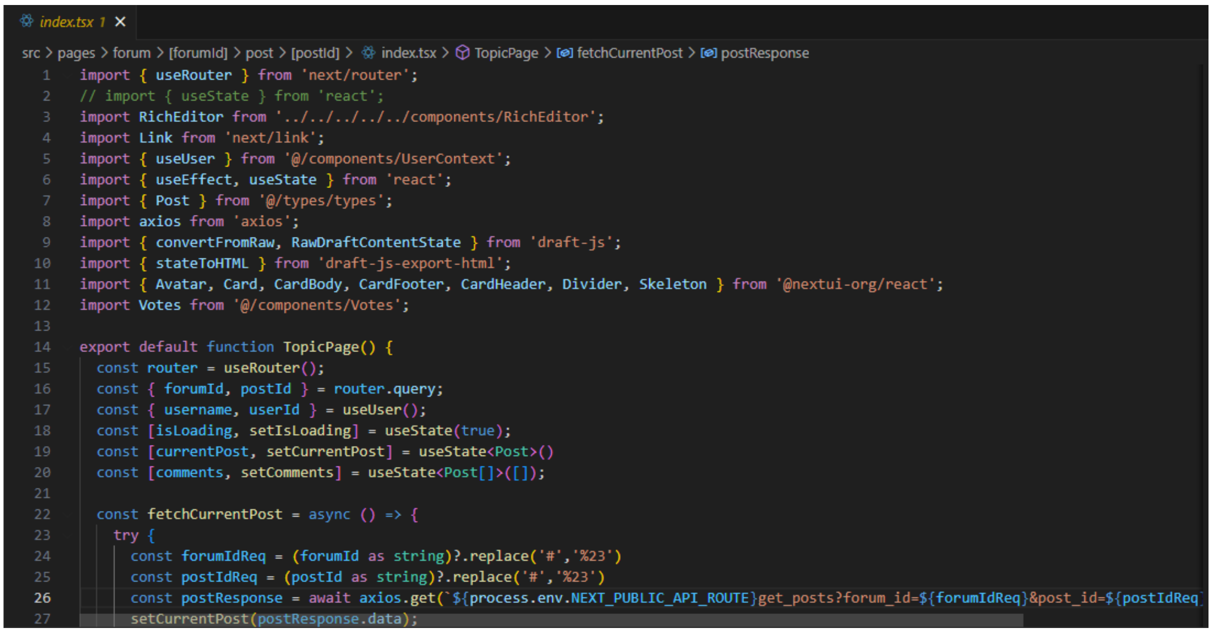


index.tsx 4

src > pages > forum > [forumId] > index.tsx > ForumPage > fetchCurrentForum

```tsx
import { useRouter } from 'next/router';
import { useEffect, useState } from 'react';
import axios from 'axios';
import { Forum, Post } from '@/types/types';
import { Avatar, Button, Card, CardBody, CardFooter, CardHeader, Divider, Link, Skeleton } from '@nextui-org/react';
import RichEditor from '@/components/RichEditor';
import { EditorState, RawDraftContentState, convertFromRaw, convertToRaw } from 'draft-js';
import { useUser } from '@/components/UserContext';
import { stateToHTML } from 'draft-js-export-html';
import Votes from '@/components/Votes';

export default function ForumPage() {
  const router = useRouter();
  const { forumId } = router.query;
  const { username, userId } = useUser();
  const [isLoading, setIsLoading] = useState(false);
  const [currentForum, setCurrentForum] = useState<Forum>()
  const [topics, setTopics] = useState<Post[]>([]);

  const fetchCurrentForum = async () => {
    try {
      const forumIdReq = (forumId as string)?.replace('#','%23')
      setIsLoading(true);
      const forumResponse = await axios.get(`${process.env.NEXT_PUBLIC_API_ROUTE}get_forums?forum_id=${forumIdReq}`);
      setCurrentForum(forumResponse.data?.forum);
      fetchTopics();
    } catch (error) {
```

Figure 13: Front End Code (Forum Page)

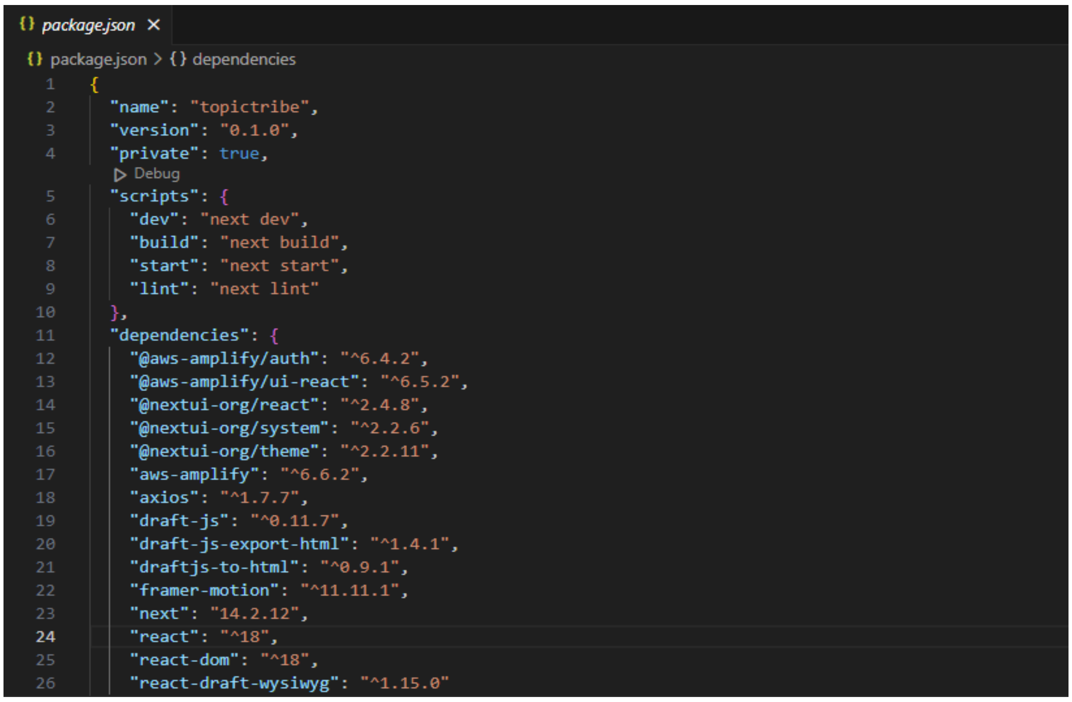


index.tsx 1

src > pages > forum > [forumId] > post > [postId] > index.tsx > TopicPage > fetchCurrentPost > postResponse

```tsx
import { useRouter } from 'next/router';
// import { useState } from 'react';
import RichEditor from '../../../../../components/RichEditor';
import Link from 'next/link';
import { useUser } from '@/components/UserContext';
import { useEffect, useState } from 'react';
import { Post } from '@/types/types';
import axios from 'axios';
import { convertFromRaw, RawDraftContentState } from 'draft-js';
import { stateToHTML } from 'draft-js-export-html';
import { Avatar, Card, CardBody, CardFooter, CardHeader, Divider, Skeleton } from '@nextui-org/react';
import Votes from '@/components/Votes';

export default function TopicPage() {
  const router = useRouter();
  const { forumId, postId } = router.query;
  const { username, userId } = useUser();
  const [isLoading, setIsLoading] = useState(true);
  const [currentPost, setCurrentPost] = useState<Post>()
  const [comments, setComments] = useState<Post[]>([]);

  const fetchCurrentPost = async () => {
    try {
      const forumIdReq = (forumId as string)?.replace('#','%23')
      const postIdReq = (postId as string)?.replace('#','%23')
      const postResponse = await axios.get(`${process.env.NEXT_PUBLIC_API_ROUTE}get_posts?forum_id=${forumIdReq}&post_id=${postIdReq
      setCurrentPost(postResponse.data);
```

Figure 14: Front End Code (Forum Post Page)

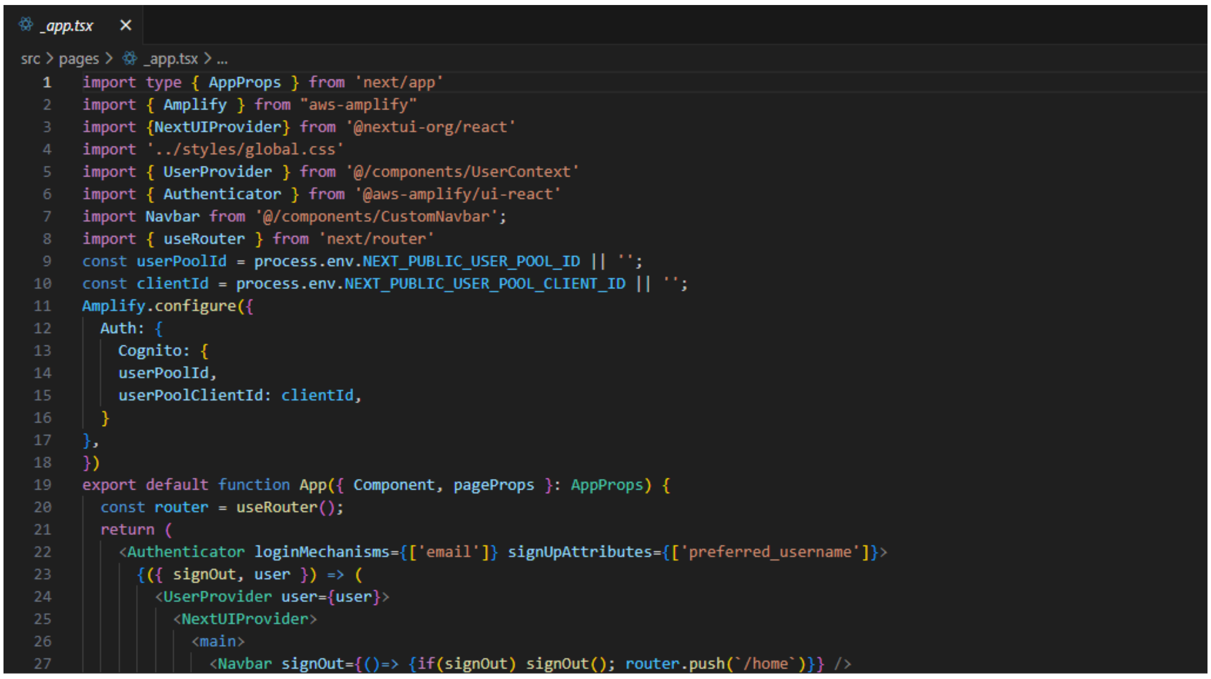


{} package.json ×

{} package.json > {} dependencies

```json
{
  "name": "topictribe",
  "version": "0.1.0",
  "private": true,
  ▷ Debug
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "@aws-amplify/auth": "^6.4.2",
    "@aws-amplify/ui-react": "^6.5.2",
    "@nextui-org/react": "^2.4.8",
    "@nextui-org/system": "^2.2.6",
    "@nextui-org/theme": "^2.2.11",
    "aws-amplify": "^6.6.2",
    "axios": "^1.7.7",
    "draft-js": "^0.11.7",
    "draft-js-export-html": "^1.4.1",
    "draftjs-to-html": "^0.9.1",
    "framer-motion": "^11.11.1",
    "next": "14.2.12",
    "react": "^18",
    "react-dom": "^18",
    "react-draft-wysiwyg": "^1.15.0"
```

Figure 15: Front End Code 3 (Package.json file)



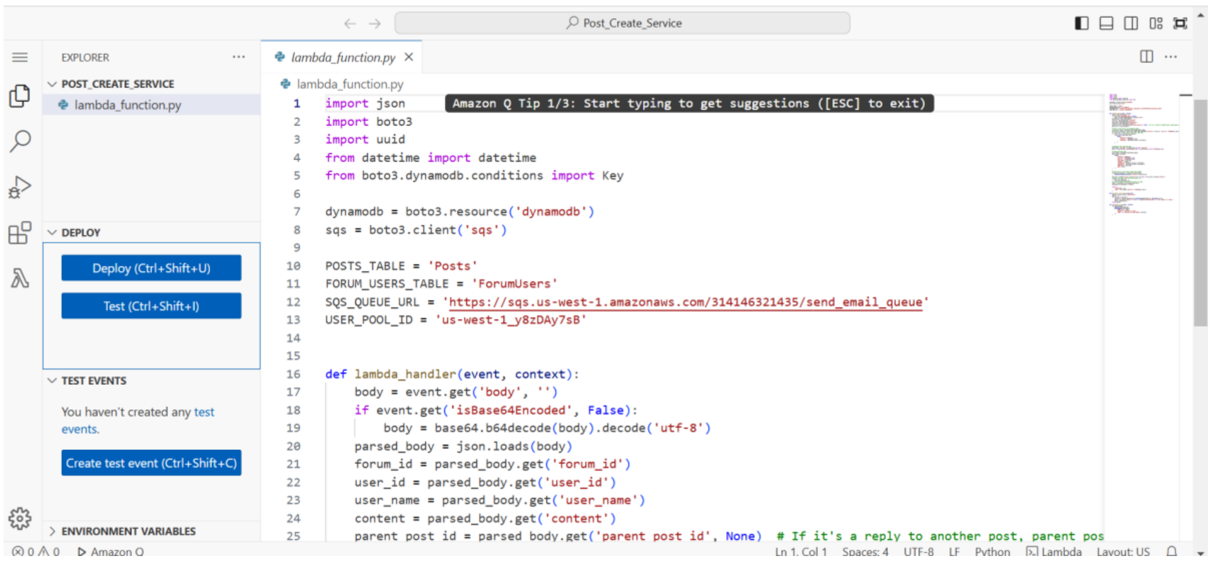
_app.tsx ×

src > pages > _app.tsx > ...

```tsx
import type { AppProps } from 'next/app'
import { Amplify } from "aws-amplify"
import {NextUIProvider} from '@nextui-org/react'
import '../styles/global.css'
import { UserProvider } from '@/components/UserContext'
import { Authenticator } from '@aws-amplify/ui-react'
import Navbar from '@/components/CustomNavbar';
import { useRouter } from 'next/router'
const userPoolId = process.env.NEXT_PUBLIC_USER_POOL_ID || '';
const clientId = process.env.NEXT_PUBLIC_USER_POOL_CLIENT_ID || '';
Amplify.configure({
  Auth: {
    Cognito: {
      userPoolId,
      userPoolClientId: clientId,
    }
  },
})
export default function App({ Component, pageProps }: AppProps) {
  const router = useRouter();
  return (
    <Authenticator loginMechanisms={['email']} signUpAttributes={['preferred_username']}>
      {({ signOut, user }) => (
        <UserProvider user={user}>
          <NextUIProvider>
            <main>
              <Navbar signOut={()=> {if(signOut) signOut(); router.push(`/home`)}} />
```

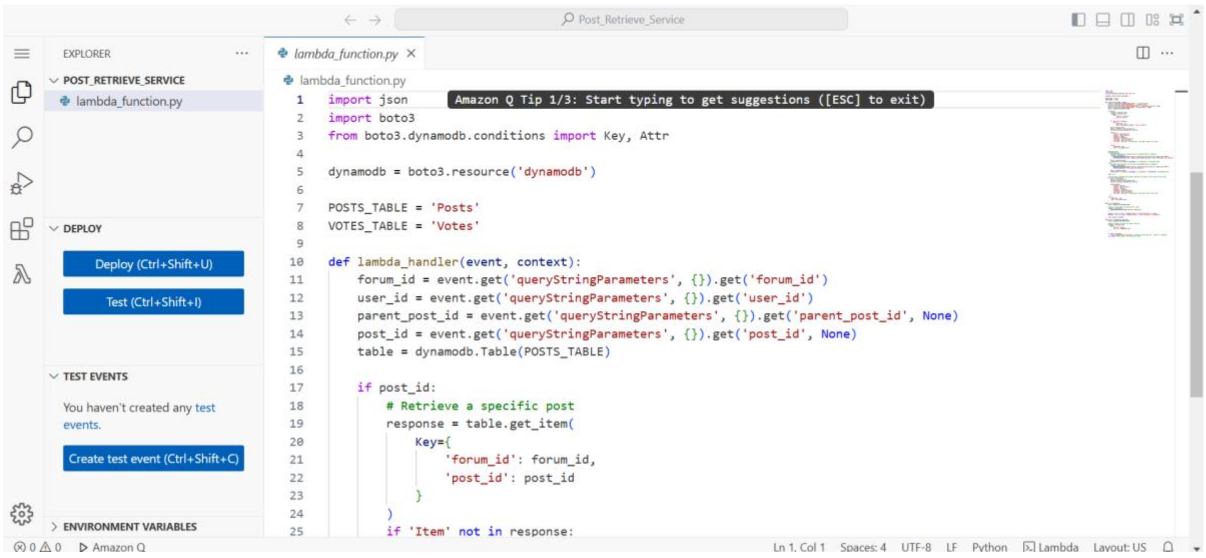
Figure 16: Front Code (_app.tsx)

2. Backend Code - AWS Lambda Functions



```python
import json
import boto3
import uuid
from datetime import datetime
from boto3.dynamodb.conditions import Key

dynamodb = boto3.resource('dynamodb')
sqs = boto3.client('sqs')

POSTS_TABLE = 'Posts'
FORUM_USERS_TABLE = 'ForumUsers'
SQS_QUEUE_URL = 'https://sqs.us-west-1.amazonaws.com/314146321435/send_email_queue'
USER_POOL_ID = 'us-west-1_y8zDAy7sB'


def lambda_handler(event, context):
    body = event.get('body', '')
    if event.get('isBase64Encoded', False):
        body = base64.b64decode(body).decode('utf-8')
    parsed_body = json.loads(body)
    forum_id = parsed_body.get('forum_id')
    user_id = parsed_body.get('user_id')
    user_name = parsed_body.get('user_name')
    content = parsed_body.get('content')
    parent_post_id = parsed_body.get('parent_post_id', None)  # If it's a reply to another post, parent pos
```

Figure 17: Backend Code (Post Create Service)



```python
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource('dynamodb')

POSTS_TABLE = 'Posts'
VOTES_TABLE = 'Votes'

def lambda_handler(event, context):
    forum_id = event.get('queryStringParameters', {}).get('forum_id')
    user_id = event.get('queryStringParameters', {}).get('user_id')
    parent_post_id = event.get('queryStringParameters', {}).get('parent_post_id', None)
    post_id = event.get('queryStringParameters', {}).get('post_id', None)
    table = dynamodb.Table(POSTS_TABLE)

    if post_id:
        # Retrieve a specific post
        response = table.get_item(
            Key={
                'forum_id': forum_id,
                'post_id': post_id
            }
        )
        if 'Item' not in response:
```

Figure 18: Backend Code (Post Retrieve Service)

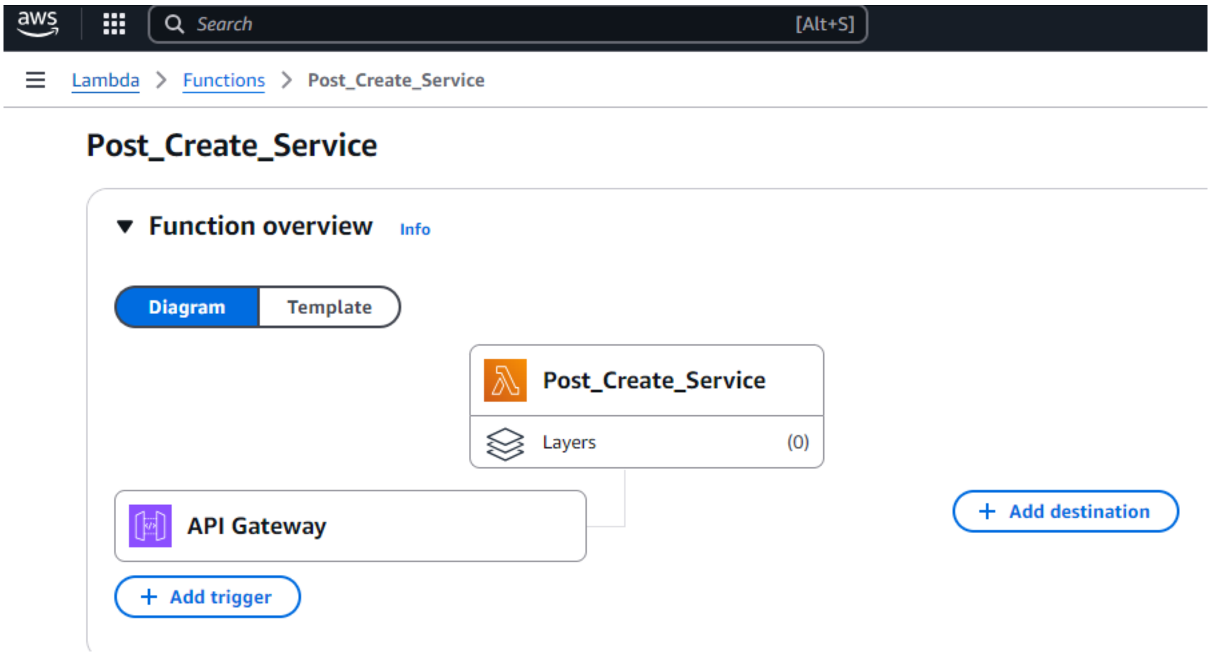


Figure 19: Post Create Service function overview

5. Frontend Design

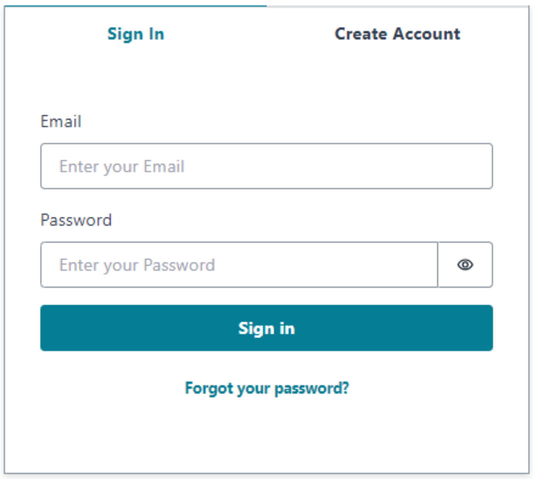
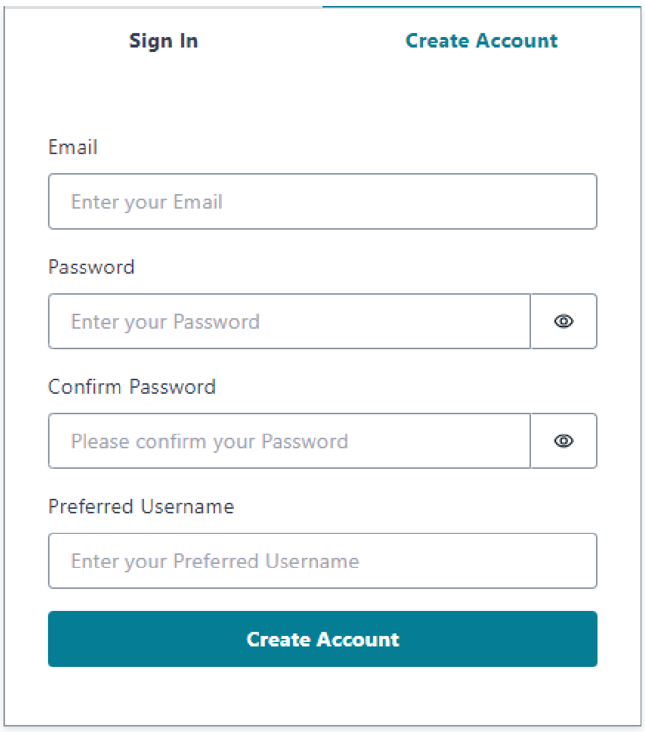


Figure 20: Login Page



Email

Enter your Email

Password

Enter your Password

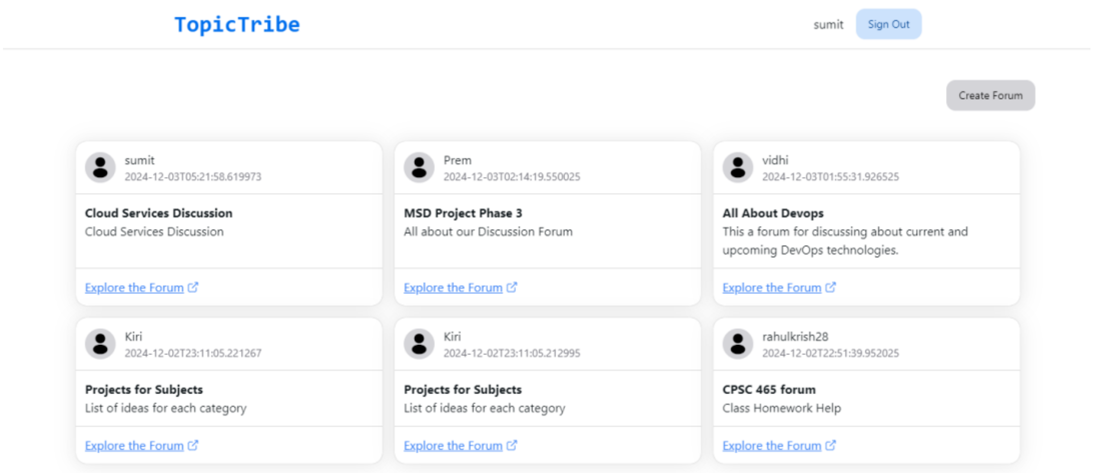
Confirm Password

Please confirm your Password

Preferred Username

Enter your Preferred Username

Figure 21: Sign Up Page



sumit
2024-12-03T05:21:58.619973
Cloud Services Discussion
Cloud Services Discussion
Explore the Forum

Prem
2024-12-03T02:14:19.550025
MSD Project Phase 3
All about our Discussion Forum
Explore the Forum

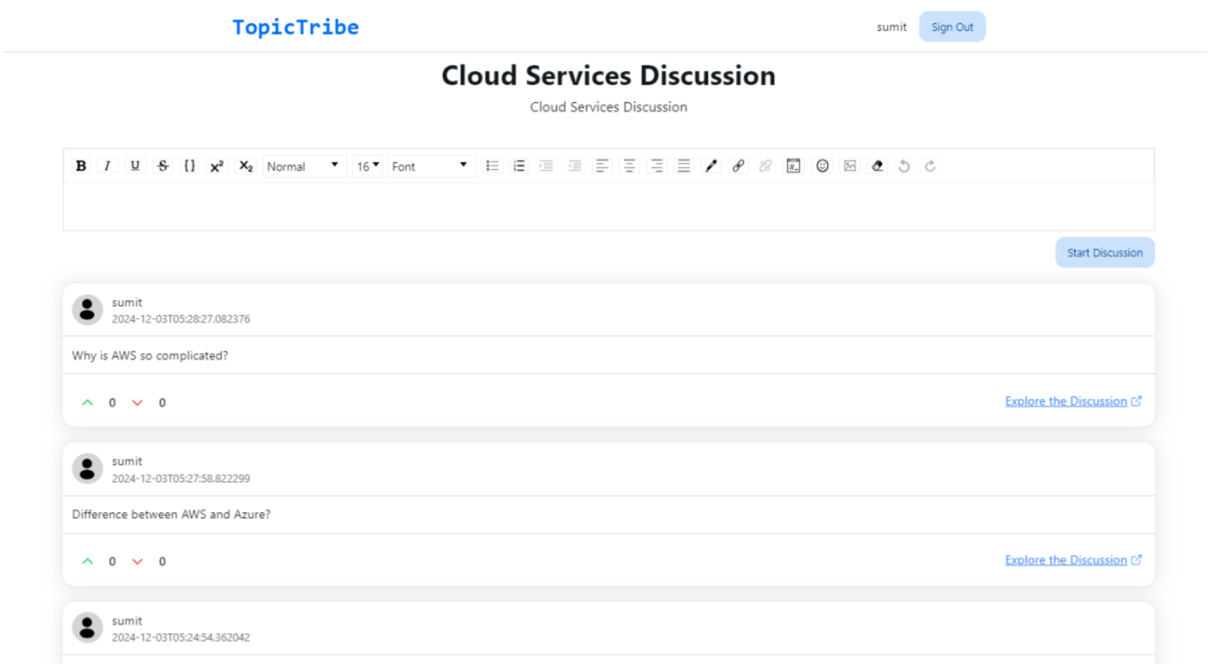
vidhi
2024-12-03T01:55:31.926525
All About Devops
This a forum for discussing about current and upcoming DevOps technologies.
Explore the Forum

Kiri
2024-12-02T23:11:05.221267
Projects for Subjects
List of ideas for each category
Explore the Forum

Kiri
2024-12-02T23:11:05.212995
Projects for Subjects
List of ideas for each category
Explore the Forum

rahulkrish28
2024-12-02T22:51:39.952025
CPSC 465 forum
Class Homework Help
Explore the Forum

Figure 22: List of Forum (Home Page)


TopicTribe

sumit Sign Out

Cloud Services Discussion

Cloud Services Discussion

Start Discussion

sumit
2024-12-03T05:28:27.082376

Why is AWS so complicated?

0 0 Explore the Discussion

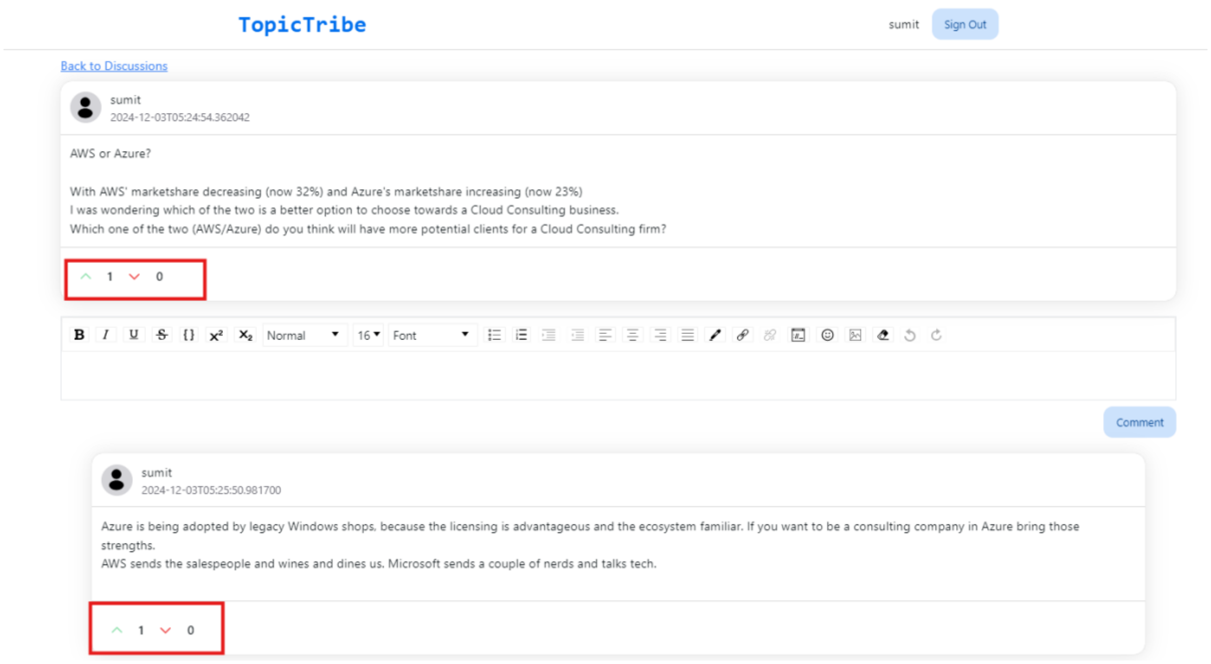
sumit
2024-12-03T05:27:58.822299

Difference between AWS and Azure?

0 0 Explore the Discussion

sumit
2024-12-03T05:24:54.362042

Figure 23: Discussion of Forum


TopicTribe

sumit Sign Out

Back to Discussions

sumit
2024-12-03T05:24:54.362042

AWS or Azure?

With AWS' marketshare decreasing (now 32%) and Azure's marketshare increasing (now 23%)
I was wondering which of the two is a better option to choose towards a Cloud Consulting business.
Which one of the two (AWS/Azure) do you think will have more potential clients for a Cloud Consulting firm?

1 0

Comment

sumit
2024-12-03T05:25:50.981700

Azure is being adopted by legacy Windows shops, because the licensing is advantageous and the ecosystem familiar. If you want to be a consulting company in Azure bring those strengths.
AWS sends the salespeople and wines and dines us. Microsoft sends a couple of nerds and talks tech.

1 0

Figure 24: Topics on Discussion with vote and comments

6. Results and Discussions

6.1 Performance Metrics

- **Latency, Response Times, and Throughput:** The AWS Cloud-Based Discussion Forum ensures low latency and fast response times by using AWS Lambda for serverless processing and Amazon DynamoDB for scalable data storage. This serverless architecture automatically scales based on demand, minimizing delays during traffic spikes. API Gateway efficiently handles requests, ensuring smooth interactions, real-time updates, and notifications.

- **Cost Comparison with and without Serverless Architecture:** With AWS Lambda's serverless architecture, the platform incurs costs based on actual usage, reducing expenses during low-traffic periods. In contrast, traditional server-based solutions require fixed costs for maintaining infrastructure, even during periods of low activity. Serverless architecture eliminates this overhead, offering a more cost-effective and scalable solution.

6.2 Challenges Encountered

1. **Difficulty in Technology Selection:** We initially faced challenges selecting technologies, starting with EC2 and later transitioning to serverless Lambda, while also moving from RDS to DynamoDB for a better fit with our project requirements.

2. **User Pool Configuration Issue:** The initial Amazon Cognito user pool was configured to accept just the user emails and passwords, but later we needed to include users' names. However, for this change, the existing configuration could not be modified.

3. **Converting Relational Schema to DynamoDB Schema:** As we plan to add more features to our project, modifying a relational database would have been challenging. Therefore, we transitioned to DynamoDB, which offers a schema less structure and greater scalability.

4. **User Verification Email Issue:** We initially planned to send only one verification email to users using Amazon Cognito and Amazon SES. To achieve this, we attempted to use an AWS Lambda function with a Post Confirmation trigger to send a custom email. However, we encountered an issue where SES's default confirmation email could not be bypassed, and the user would always receive both the SES-generated email and our custom email.

5. **Mindful Use of AWS Free Tier:** Resource management was critical to avoid exceeding the AWS free tier limits.

6.3 Benefits

- **Scalability:** The serverless architecture using AWS Lambda and DynamoDB ensures automatic scaling to handle growing user traffic without manual intervention, maintaining performance during high demand.

- **Security:** Amazon Cognito provides robust user authentication and authorization, ensuring secure access to the platform while protecting user data.

- **User Experience:** Real-time interactions, instant notifications via Amazon SES, and a responsive Next.js front end create an engaging, seamless experience for users across devices.

7. Conclusions and Future Work

7.1 Conclusions

The AWS Cloud-Based Discussion Forum is a scalable, secure platform designed for real-time user engagement. Built on a serverless architecture, it uses AWS Lambda for backend processing, Amazon Cognito for user authentication, and Amazon DynamoDB for fast, reliable data storage. The forum supports dynamic interactions, allowing users to create posts, comment, and vote, with real-time notifications via Amazon SES to keep users engaged.

The system's microservices architecture ensures flexibility and easy maintenance, allowing each component (forums, posts, votes) to scale independently. Despite challenges like migrating to a serverless setup and handling schema changes, the team successfully navigated these issues. The result is a robust, responsive forum offering a seamless, scalable experience for users.

7.2 Future Work

- Implement AI-based content moderation using tools like Amazon Comprehend for sentiment analysis and automated flagging of inappropriate content.

- Add multi-language support to allow users from different regions to engage in discussions in their native languages.

- Develop a mobile app for iOS and Android to provide a more seamless user experience with features like push notifications and offline access.

8. References

- AWS Documentation, AWS Documentation
- AWS Cognito Documentation, AWS Cognito
- AWS Lambda Documentation, AWS Lambda
- AWS DynamoDB Documentation, DynamoDB

9. Repository Link: https://github.com/Komal-7/topictribe