PARSHVANATH CHARITABLE TRUST'S

A.P. Shah Institute of Technology

Thane, 400615

Academic Year: 2022-23
Department of Computer Engineering

CSL605 SKILL BASED LAB COURSE: CLOUD COMPUTING

Mini Project Report

- **Title of Project**: Real Time Chat App
- **Year and Semester**: **T.E. (Sem VI)**
- **Group Members Roll No. & Name**:

88	**Chetan Kumbhar**
83	**Vaishnavi Kothari**
87	**Shivangi Kumar**

Table of Contents

Abstract

The project aims to build a real-time chat application using AWS Lambda, WebSocket, and API Gateway services. The chat application will allow users to communicate with each other instantly, and they can send text and images through the application.

The project's architecture will use API Gateway to create a WebSocket connection between the client-side and the server-side. AWS Lambda will handle the server-side code to manage the chatroom, messages, and user authentication. Additionally, the Amazon S3 service will be used to store the images and media files shared by the users.

The project will provide a secure and scalable solution, making use of AWS's serverless technologies to reduce infrastructure costs and complexity. Users will be able to communicate with each other in real-time without the need for any additional software or hardware.

The project will help developers understand how to leverage AWS serverless technologies to build scalable and secure chat applications. They will learn about the benefits of using WebSocket and API Gateway to create real-time communication channels and how AWS Lambda can handle the server-side code without the need for managing infrastructure.

Introduction

In today's world, the demand for real-time communication apps has increased significantly. One such app is a chat application that allows people to communicate with each other in real-time. Building a chat app can be a challenging task, but with the help of Amazon Web Services (AWS), it can become much easier. AWS provides various services that can be used to build a chat app quickly and efficiently.

This project aims to build a chat application using AWS Lambda, WebSocket, and API Gateway. AWS Lambda is a serverless computing service that allows developers to run code without the need for managing servers. WebSocket is a protocol that provides full-duplex communication channels over a single TCP connection. API Gateway is a fully managed service that makes it easy to create, deploy, and manage APIs at any scale.

The chat app will allow users to register and log in to the system. Once logged in, they can create chat rooms, join existing chat rooms, and communicate with other users in real-time. The app will also allow users to send and receive files, images, and emojis.

The project will involve creating a serverless backend using AWS Lambda and API Gateway, which will handle the WebSocket connections and messages. The frontend will be built using React and will interact with the backend using API Gateway. The application will be deployed on AWS using the Serverless Application Model (SAM).

The following sections of this project report will provide a detailed explanation of the various components used in building the chat app and how they are integrated to form a complete application.

Problem Definition

The problem that the project aims to solve is the need for a scalable and cost-effective chat application that can handle a large number of users and messages simultaneously. Traditional chat applications often face issues with scalability, as the number of users and messages increases, leading to increased server load and costs. Additionally, setting up and managing servers for a chat application can be time-consuming and require specialized knowledge.

The project aims to leverage the power of AWS Lambda, WebSocket, and API Gateway to create a serverless chat application that can scale easily and handle a large number of users and messages without incurring high costs. The application will provide users with real-time messaging capabilities, including the ability to create channels, join channels, and send messages. The serverless architecture of the application will also enable easy deployment and management, reducing the burden on developers.

Objective and Scope

Objective:

The objective of building a chat app using Lambda, WebSocket, and API Gateway on AWS is to create a scalable, reliable, and cost-effective solution for real-time communication between users. The chat app will allow users to create accounts, log in, and send messages to other users. The backend will be built using Lambda functions, which will handle user authentication, message storage, and communication between clients through WebSockets. The API Gateway will be used to manage the HTTP endpoints and WebSocket connections, while the frontend will be built using React and Hooks, providing a modern and responsive user interface. The project aims to demonstrate proficiency in AWS services, serverless computing, and full-stack development, as well as practical experience in building real-time applications.

Scope:

The scope of building a real-time chat application on AWS using Lambda, WebSocket and API gateway is quite extensive. It involves the creation of a real-time chat application that can be used for a variety of purposes, including team collaboration, customer support, and social networking. The chat app would leverage the AWS Lambda function, WebSocket, and API Gateway to enable real-time messaging and notification functionality.

The project scope includes the development of a backend serverless architecture using AWS Lambda, which would help in creating event-driven microservices. The chat app's user interface would be built using React and Hooks, providing a responsive and dynamic user experience. The app would also incorporate features like user authentication, message encryption, and notification alerts.

The project's scope would also involve the use of the AWS API Gateway service to create APIs for client applications to interact with the backend. This would include creating endpoints to handle WebSocket connections, user authentication, message sending, and retrieval. Additionally, the project scope may include integrating third-party services like AWS S3, SNS, or DynamoDB to store messages or handle notifications.

Description

- ### Cloud Services Used:

1. Amazon API Gateway: This service will be used to create and manage the APIs that will be used to handle WebSocket connections between clients and the Lambda functions.
2. AWS Lambda: This service will be used to run the backend logic for the chat app. It will handle the messages received from the clients, broadcast them to other clients, and store them in the database.
3. Amazon S3: This service will be used to store any files or images that users upload during the chat session.
4. AWS Identity and Access Management (IAM): This service will be used to manage the access and permissions for the various AWS services used in the chat app.

By using these cloud services, the chat app can be built with a highly scalable and reliable architecture that can handle large amounts of traffic and user activity.

- ### Methodology:

1. Providing IAM Access:

   When building a chat app using Lambda, WebSocket, and API Gateway on AWS, you will need to provide IAM (Identity and Access Management) access to ensure secure and controlled access to AWS services. By providing IAM access in this way, you can ensure that your chat app is secure and that access to AWS services is limited to only those who require it.

2. <u>Setting up API Gateway:</u>

   API Gateway is an important component in building a chat app using Lambda and WebSocket on AWS. It acts as a bridge between the client and backend services by providing a managed HTTP(S) endpoint for the WebSocket connection.

   The role of setting up API Gateway for the chat app is to provide a secure and scalable way of handling incoming requests from the clients. It also provides features like authentication, rate limiting, caching, and monitoring to enhance the overall performance of the app.

   By setting up API Gateway, developers can focus on building the core chat functionality without worrying about managing the infrastructure or the WebSocket connections. The API Gateway can handle the traffic seamlessly and route it to the appropriate Lambda function or WebSocket endpoint, based on the API configuration.

3. <u>Creating the Lambda Function:</u>

   The Lambda function is responsible for handling the WebSocket connections and receiving the messages from the clients. The messages received are then processed and sent to the appropriate clients.

   To create the Lambda function, the developer needs to write the code that will handle the WebSocket events, such as connecting, disconnecting, and receiving messages. This code can be written in any language that Lambda supports, including Node.js, Python, and Java.

Once the code is written, the Lambda function needs to be configured to use the appropriate WebSocket API Gateway. This involves setting up the permissions and configuring the environment variables.

After the Lambda function is created and configured, it can be deployed and tested. This is done by connecting to the WebSocket API Gateway using a WebSocket client and sending messages to the Lambda function. The Lambda function should then respond appropriately, sending messages back to the client or broadcasting messages to all connected clients.

4. Writing the Backend Code in Javascript/Nodejs:

   This code will be responsible for handling the incoming WebSocket connections and messages, processing the data, and interacting with the database to store and retrieve chat data.

   The backend code will typically use the AWS SDK to interact with the various AWS services, such as DynamoDB for storing chat data or S3 for storing media files. The code will also implement business logic such as user authentication, message filtering, and message delivery.

5. Testing the Websocket Using wscat:

   The role of testing the WebSocket using wscat for a project to build a chat app using Lambda, WebSocket and API Gateway on AWS is to ensure that the WebSocket connection is established and working correctly.

   wscat is a command-line tool that allows testing WebSocket connections by sending and receiving messages. It can be used to connect to the WebSocket endpoint provided by API Gateway, and send test messages to ensure that the backend Lambda function is processing the messages correctly.

6. Chat Client UI:

   The chat client UI provides a user-friendly interface for the chat application, which communicates with the backend using the WebSocket API provided by AWS API Gateway. The client-side UI is built using React and React Hooks, which provides an efficient way to manage the UI state.

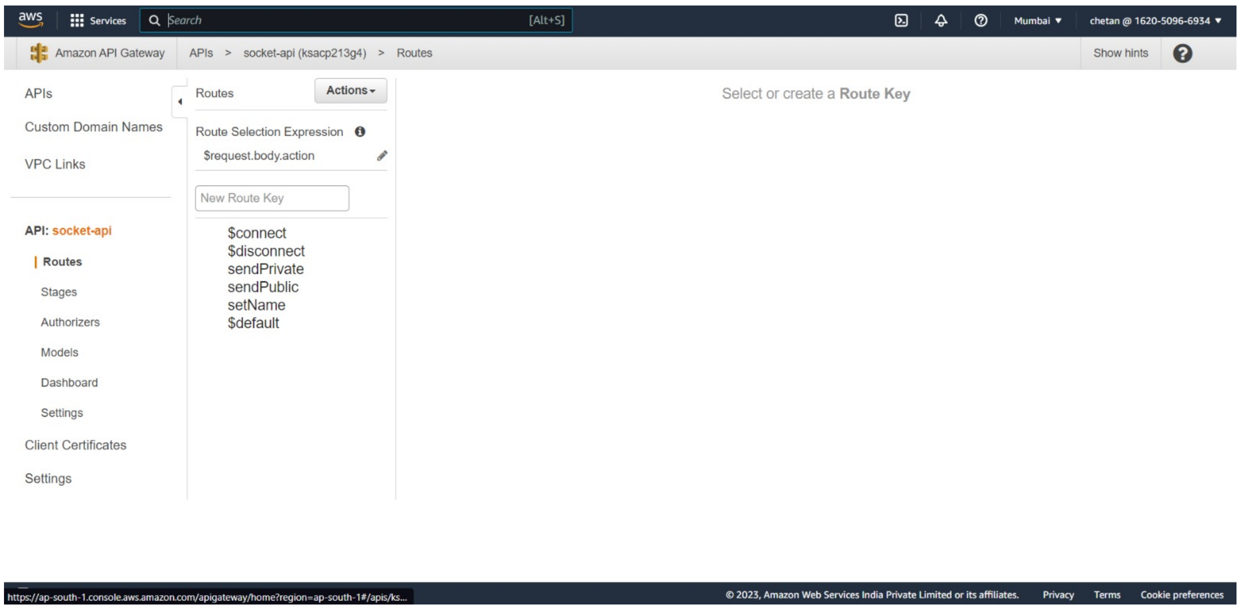
   The chat client UI allows users to send and receive messages in real-time, display messages in a chat window, and provide a way to select a user to send a message. Additionally, it provides features like user authentication, user profile management, and notification settings.

- **Software Requirement:**

1. AWS account: An AWS account is required to use the AWS services, including Lambda, API Gateway, and WebSocket.
2. Node.js: A JavaScript runtime environment that is used to execute the server-side code on Lambda.
3. AWS CLI: The AWS command-line interface (CLI) is a tool that enables developers to interact with AWS services from the command line.
4. Visual Studio Code: A code editor that provides features for writing and debugging code. It supports Node.js development and AWS development.
5. React: A JavaScript library for building user interfaces.
6. React Hooks: A feature in React that enables functional components to use state and other React features.

7. WebSocket Testing Tool: A WebSocket testing tool like wscat can be used to test the WebSocket connection.
8. Git: A version control system that is used to manage code changes.
9. NPM: A package manager for Node.js that is used to install and manage dependencies.
10. Serverless Framework: A framework that enables developers to build serverless applications using AWS Lambda and other cloud providers.

Implementation Details



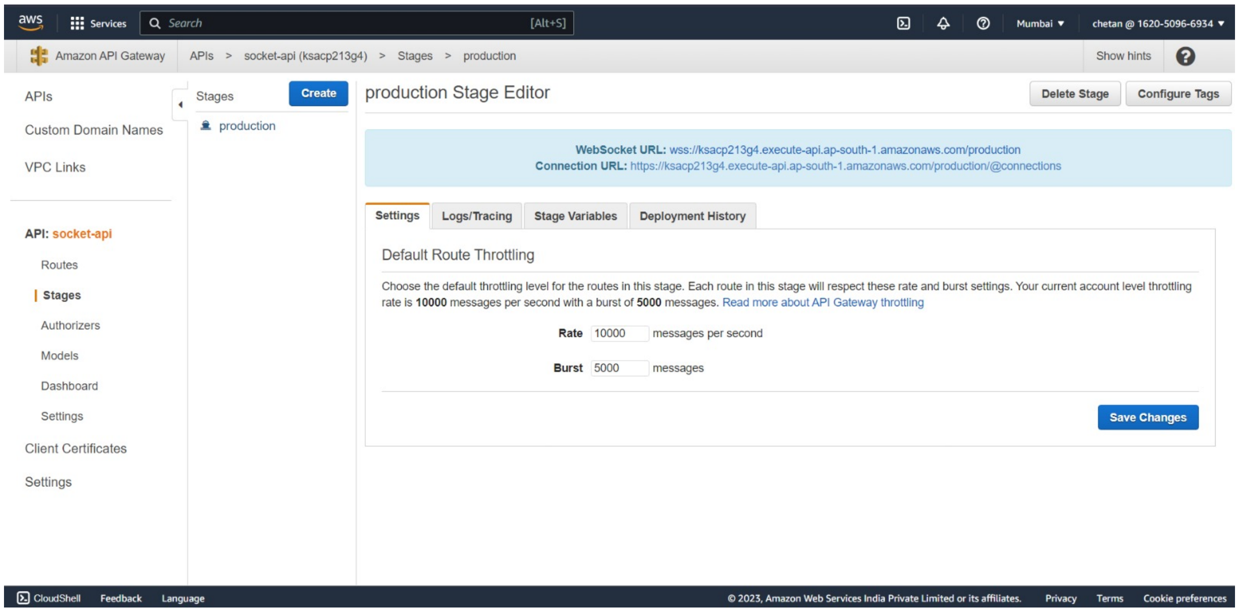
Routes

Route Selection Expression
$request.body.action

New Route Key

$connect
$disconnect
sendPrivate
sendPublic
setName
$default

Figure 1: API Gateway



production Stage Editor

WebSocket URL: wss://ksacp213g4.execute-api.ap-south-1.amazonaws.com/production
Connection URL: https://ksacp213g4.execute-api.ap-south-1.amazonaws.com/production/@connections

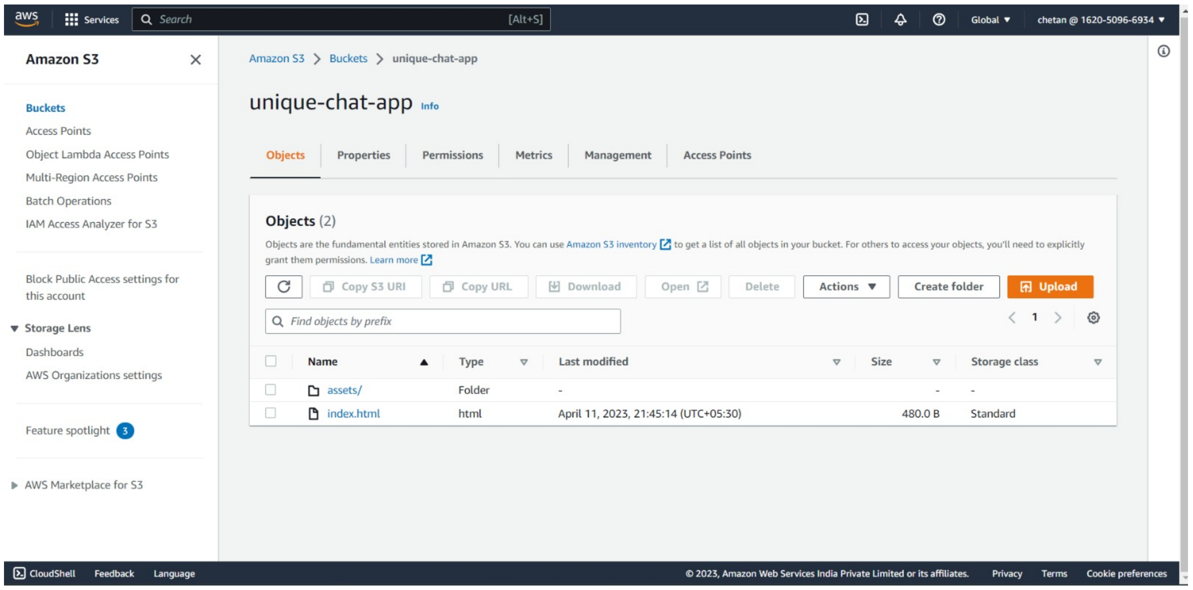
Default Route Throttling

Choose the default throttling level for the routes in this stage. Each route in this stage will respect these rate and burst settings. Your current account level throttling rate is **10000** messages per second with a burst of **5000** messages. Read more about API Gateway throttling

Rate 10000 messages per second

Burst 5000 messages

Figure 2: Production Stage



Amazon S3 > Buckets > unique-chat-app

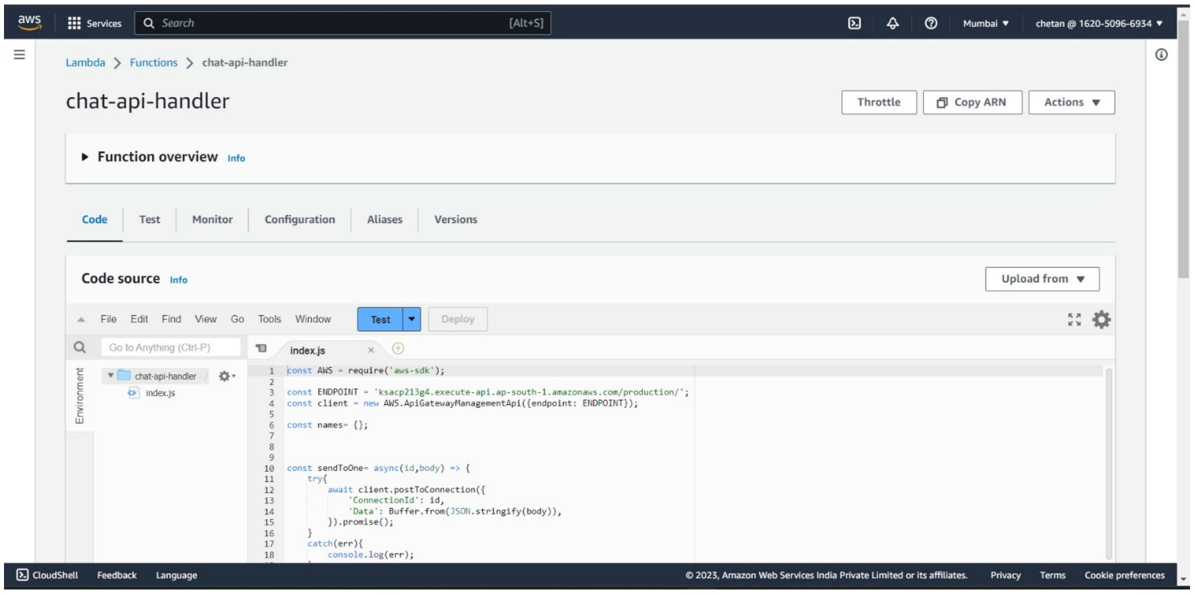
unique-chat-app

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
assets/	Folder	-	-	-
index.html	html	April 11, 2023, 21:45:14 (UTC+05:30)	480.0 B	Standard

Figure 3: Bucket to Deploy Application



Lambda > Functions > chat-api-handler

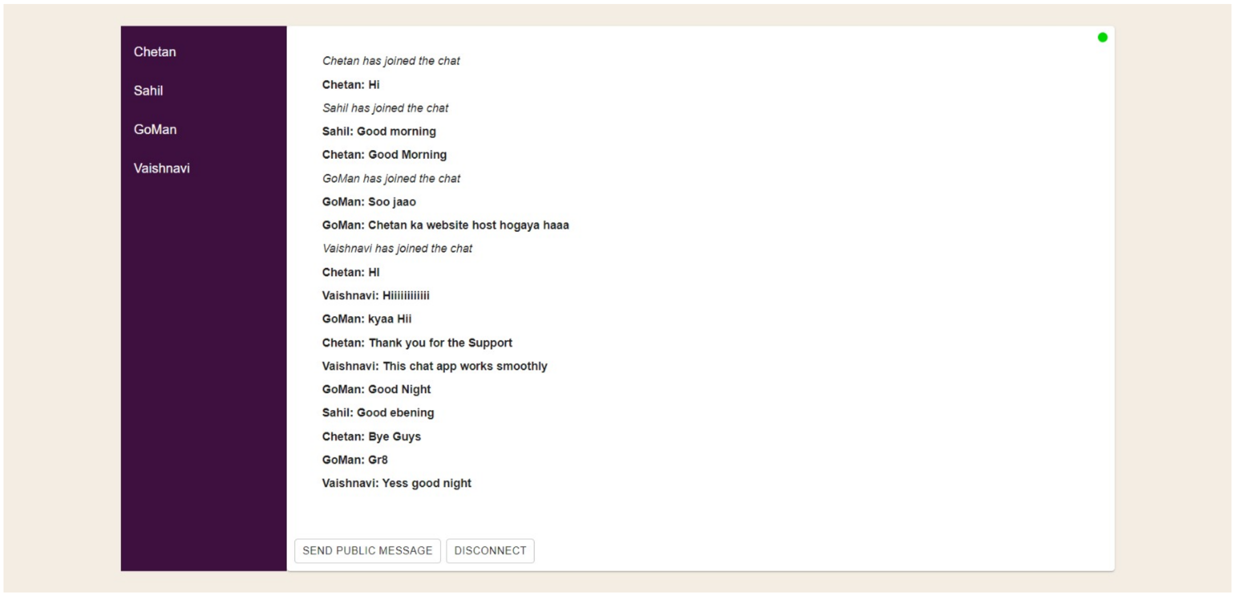
chat-api-handler

Code source

index.js

```
const AWS = require('aws-sdk');

const ENDPOINT = 'ksacp213g4.execute-api.ap-south-1.amazonaws.com/production/';
const client = new AWS.ApiGatewayManagementApi({endpoint: ENDPOINT});

const names= {};



const sendToOne= async(id,body) => {
    try{
        await client.postToConnection({
            'ConnectionId': id,
            'Data': Buffer.from(JSON.stringify(body)),
        }).promise();
    }
    catch(err){
        console.log(err);
```

Figure 4: Chat API Handler



Chetan
Sahil
GoMan
Vaishnavi

Chetan has joined the chat
Chetan: Hi
Sahil has joined the chat
Sahil: Good morning
Chetan: Good Morning
GoMan has joined the chat
GoMan: Soo jaao
GoMan: Chetan ka website host hogaya haaa
Vaishnavi has joined the chat
Chetan: HI
Vaishnavi: Hiiiiiiiiiii
GoMan: kyaa Hii
Chetan: Thank you for the Support
Vaishnavi: This chat app works smoothly
GoMan: Good Night
Sahil: Good ebening
Chetan: Bye Guys
GoMan: Gr8
Vaishnavi: Yess good night

Figure 5: Real-Time Chat App

Learning Outcome

Firstly, the project provides hands-on experience in using AWS services such as API Gateway, Lambda, and WebSocket. This helps in gaining a deeper understanding of how these services work together and how to integrate them into a serverless architecture.

Secondly, the project provides experience in building a real-time chat application using modern technologies such as React, Node.js, and JavaScript. This helps in developing proficiency in these technologies and gaining practical experience in building scalable and responsive web applications.

Thirdly, the project provides experience in writing efficient and optimized code for a serverless architecture. This helps in developing the ability to write code that can perform well in a distributed and scalable environment.

Fourthly, the project provides experience in testing and debugging a distributed application. This helps in developing the ability to troubleshoot issues that may arise in a complex distributed system.

Finally, the project helps in developing project management skills such as planning, organizing, and prioritizing tasks. It would also provide experience in working in a team and collaborating with others to achieve a common goal.