

# Real-Time Chat Application using WebSockets and Serverless

## Project Overview

**Project Name:** Real-Time Chat App using WebSockets and Serverless  
**Goal:** Build a scalable real-time messaging platform without managing servers  
**Architecture Type:** Fully Serverless, Event-Driven  
**AWS Region:** Choose one region (example: ap-south-1)

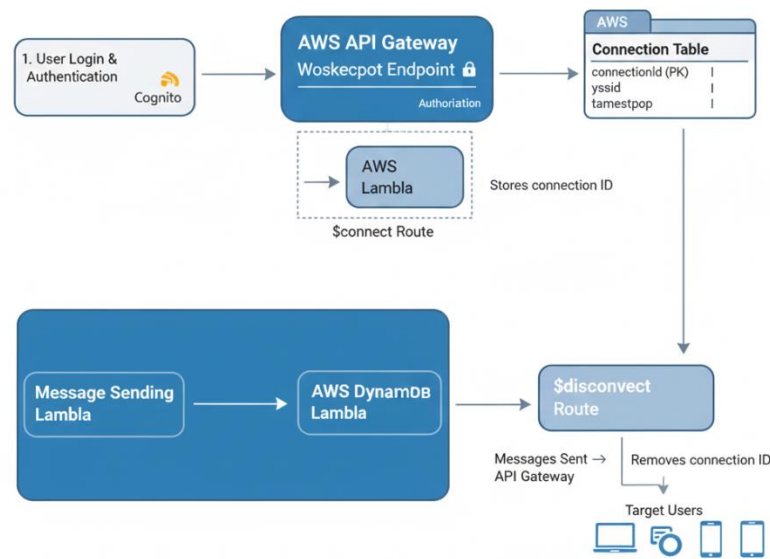
## AWS Services Used

Service	Purpose
API Gateway (WebSocket API)	Real-time bidirectional communication
AWS Lambda	Handle connect, disconnect, and messages
DynamoDB	Store active WebSocket connections
Amazon Cognito	User authentication & identity federation
IAM	Secure permissions between services
CloudWatch	Logs and monitoring

## High-Level Architecture Flow

1. User logs in via **Cognito**
2. Client connects to **WebSocket API**
3. \$connect Lambda stores connection ID in DynamoDB
4. Messages sent → Lambda → API Gateway → Target users
5. \$disconnect Lambda removes connection ID
- 6.

## AWS Serverless Woskeskot Architecture



# STEP-BY-STEP AWS CONSOLE INSTRUCTIONS

## STEP 1: Create DynamoDB Table (Store WebSocket Connections)

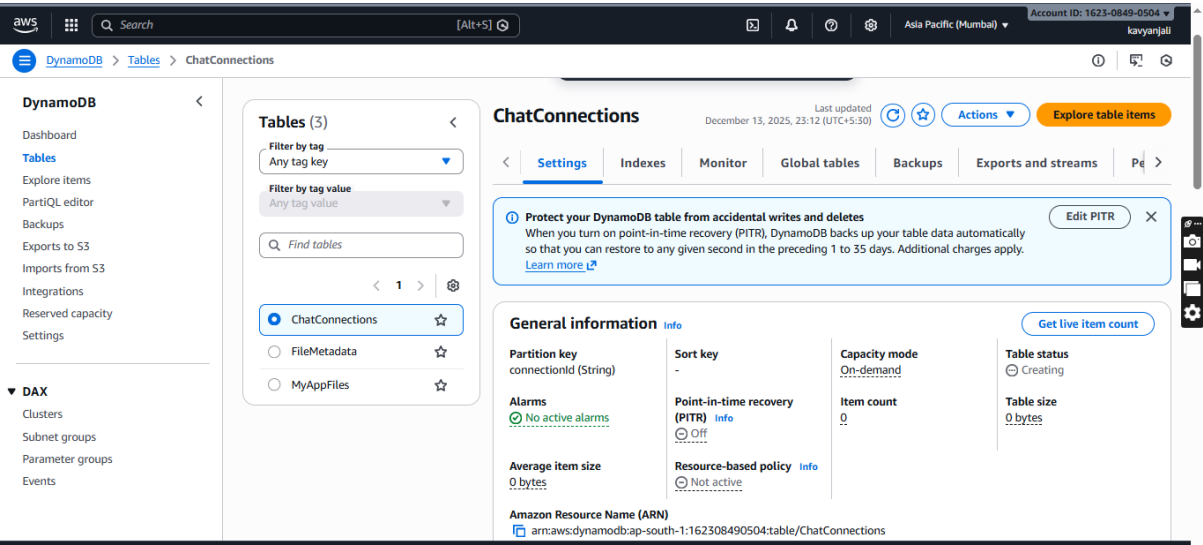
### Console Steps:

- 1. Open **AWS Console**
- 2. Go to **Services → DynamoDB**
- 3. Click **Create table**

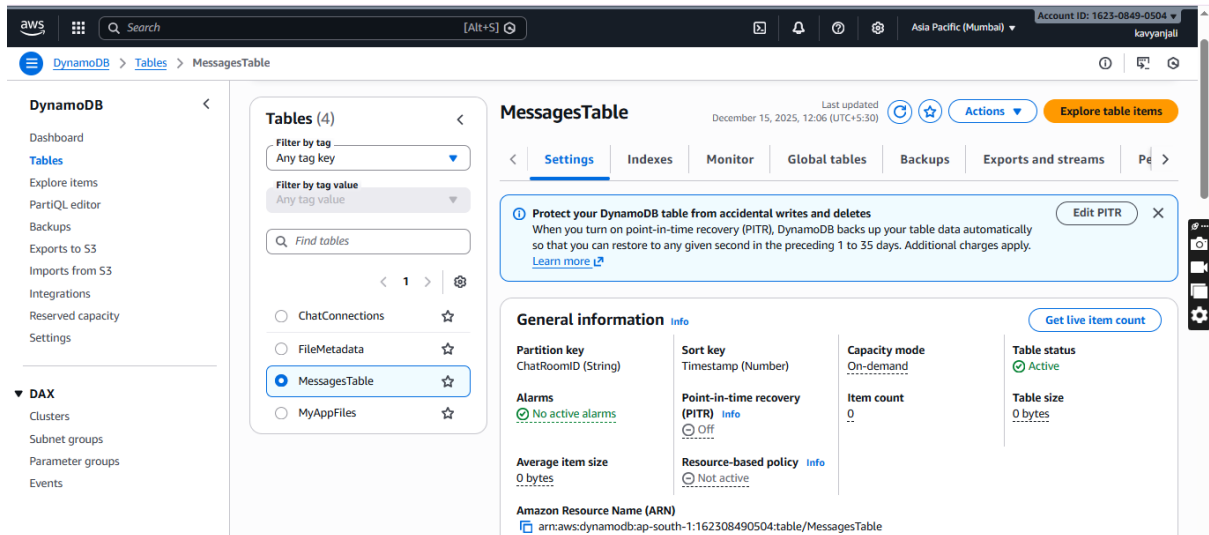
### Configuration:

- **Table name:** ChatConnections
- **Partition key:**
  - Name: connectionId
  - Type: String
- **Table settings:** Default
- Click **Create table**

Purpose: Store active WebSocket connection IDs



## Step 1.2 Create Messages Table



## STEP 2: Create Cognito User Pool (Authentication)

### Console Steps:

1. Go to **Services** → **Cognito**
2. Click **Create user pool**
3. Choose **Cognito User Pool**
4. Click **Next**

### Configure Sign-in:

- Sign-in options: **Email**
- Password policy: Default
- MFA: Optional (disable for demo)

### App Client:

- App client name: ChatAppClient
- Disable client secret

### Click **Create user pool**

Purpose: Authenticate users before chat access

Amazon Cognito > User pools > MyUserpool > Overview

Current user pool: MyUserpool

**Overview: MyUserpool**

**User pool information**

- User pool name: MyUserpool
- User pool ID: ap-south-1\_jbBhZkFIT
- ARN: arn:aws:cognito-idp:ap-south-1:162308490504:userpool/ap-south-1\_jbBhZkFIT
- Token signing key URL: [https://cognito-idp.ap-south-1.amazonaws.com/ap-south-1\\_jbBhZkFIT/well-known/jwks.json](https://cognito-idp.ap-south-1.amazonaws.com/ap-south-1_jbBhZkFIT/well-known/jwks.json)
- Created time: December 13, 2025 at 23:16 GMT+5:30
- Last updated time: December 15, 2025 at 11:50 GMT+5:30
- Estimated number of users: 1
- Feature plan: Essentials

**Recommendations**

Manage or create applications | Apply branding to your managed login pages

Amazon Cognito > User pools > real-time > App clients > App client: ChatUserPool

Current user pool: real-time

**App client: ChatUserPool**

**App client information**

- App client name: ChatUserPool
- Client ID: 2irneiggu7lft9ns4tc7416p
- Client secret: \*\*\*\*\*
- Show client secret: ☐
- Authentication flows: Username and password, Get user tokens from existing authenticated sessions
- Authentication flow session duration: 3 minutes
- Refresh token expiration: 5 day(s)
- Access token expiration: 60 minutes
- ID token expiration: 60 minutes
- Advanced authentication settings: Enable token revocation, Enable prevent user existence errors
- Created time: December 13, 2025 at 23:20 GMT+5:30
- Last updated time: December 13, 2025 at 23:23 GMT+5:30

[Quick setup guide](#) | Attribute permissions | Login pages | Threat protection | Analytics

Amazon Cognito > User pools > MyUserpool > Users

Current user pool: MyUserpool

**Users**

User "kavyakavyanajali@gmail.com" has been created successfully. [View details](#)

**Users (1)**

View, edit, and create users in your user pool. Users that are enabled and confirmed can sign in to your user pool.

Property: User name | Search users by attribute

User name	Email address	Email verified	Confirmation status	Status
c163cd8a-90c1-7006-21...	kavyakavyanajali@gmail...	No	<a href="#">Force change password</a>	Enabled

**Import users (0)**

View and create user CSV import jobs. Amazon Cognito can import users into this user pool from a specially-formatted CSV file. You can't import user passwords.

Search import jobs by job name

Job name	Status	Imported users	Skipped users	Failed users	CloudWatch logs	Created time
No user import jobs found						

## STEP 3: Create IAM Role for Lambda

### Console Steps:

1. Go to **IAM** → **Roles**
2. Click **Create role**
3. Trusted entity: **AWS service**
4. Service: **Lambda**
5. Click **Next**

#### Attach Policies:

- AWSLambdaBasicExecutionRole
  - AmazonDynamoDBFullAccess
  - AmazonAPIGatewayInvokeFullAccess
6. Role name: ChatLambdaRole
  7. Click **Create role**

**Role ChatLambdaRole created.** [View role]

**Permissions policies (3)** Info [Simulate] [Remove] [Add permissions]

You can attach up to 10 managed policies.

Filter by Type: All types

<input type="checkbox"/>	Policy name	Type	Attached entities
<input type="checkbox"/>	AmazonAPIGatewayInvokeFull...	AWS managed	1
<input type="checkbox"/>	AmazonDynamoDBFullAccess	AWS managed	1
<input type="checkbox"/>	AWSLambdaBasicExecutionRole	AWS managed	3

► **Permissions boundary** (not set)

**Role APIGatewayCloudWatchLogsRole created.** [View role]

December 14, 2025, 00:53 (UTC+05:30) [am:aws:iam::162308490504:role/APIGatewayCloudWatchLogsRole]

**Last activity** - **Maximum session duration** 1 hour

**Permissions** | Trust relationships | Tags | Last Accessed | Revoke sessions

**Permissions policies (1)** Info [Simulate] [Remove] [Add permissions]

You can attach up to 10 managed policies.

Filter by Type: All types

<input type="checkbox"/>	Policy name	Type	Attached entities
<input type="checkbox"/>	AmazonAPIGatewayPushToClo...	AWS managed	1

► **Permissions boundary** (not set)

## STEP 4: Create Lambda Functions

You need **3 Lambda functions**.

## 4.1 Lambda: Connect Handler

### Console Steps:

1. Go to **Services** → **Lambda**
2. Click **Create function**
3. Author from scratch

### Configuration:

- Function name: ChatConnectHandler
- Runtime: **Python 3.12**
- Execution role: **Use existing role**
- Role: ChatLambdaRole
- Click **Create function**

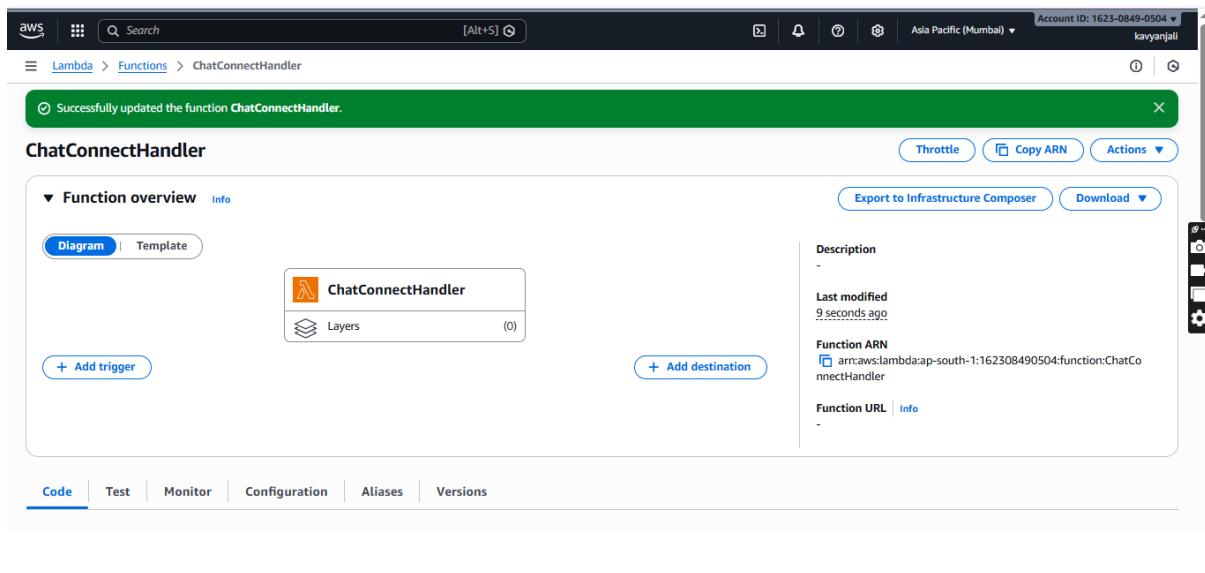
### Code :

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('ChatConnections')

def lambda_handler(event, context):
    connection_id = event['requestContext']['connectionId']
    table.put_item(Item={'connectionId': connection_id})
    return {'statusCode': 200}
```

### Click **Deploy**



## 4.2 Lambda: Disconnect Handler

Repeat steps above.

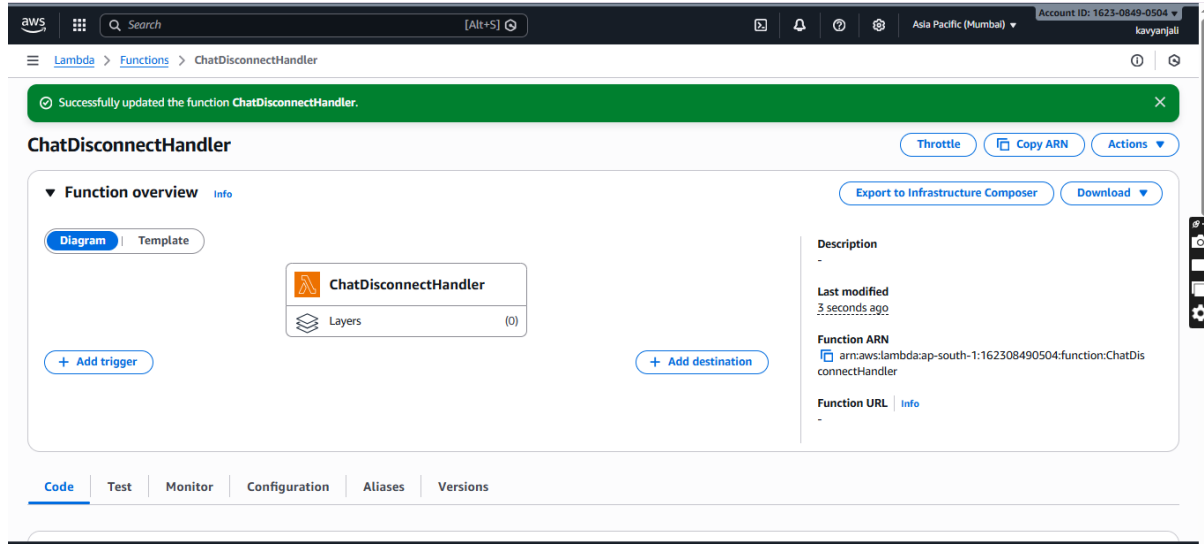
- **Function name:** ChatDisconnectHandler

### Code:

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('ChatConnections')

def lambda_handler(event, context):
    connection_id = event['requestContext']['connectionId']
    table.delete_item(Key={'connectionId': connection_id})
    return {'statusCode': 200}
```



### 4.3 Lambda: Message Handler

- **Function name:** ChatMessageHandler

#### Code:

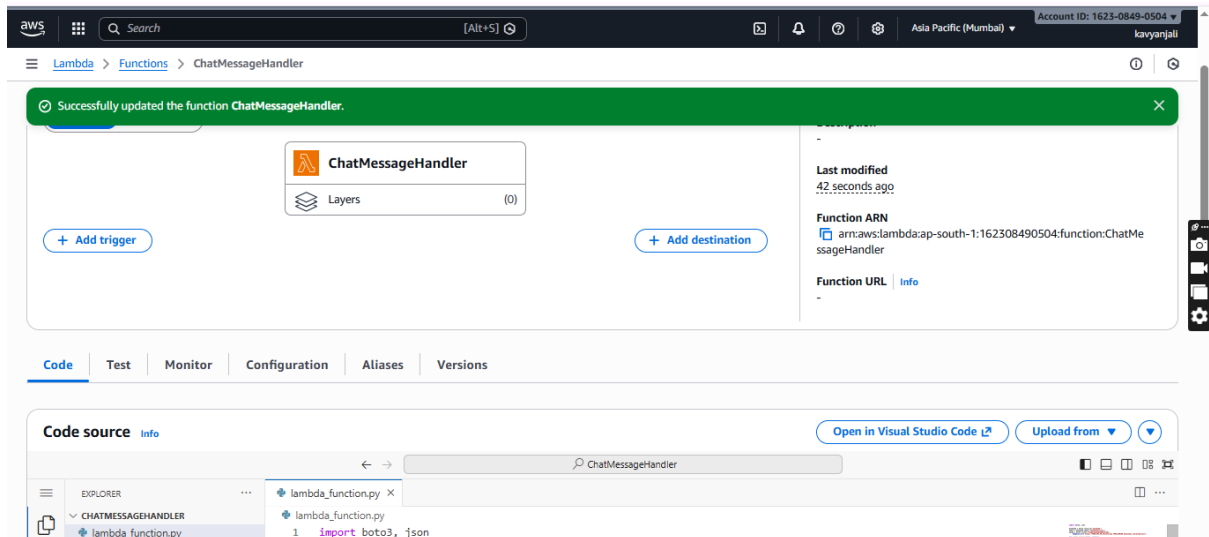
```
import boto3, json

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('ChatConnections')
apigw = boto3.client('apigatewaymanagementapi',
    endpoint_url='wss://lgzv7q7w97.execute-api.ap-south-1.amazonaws.com/production/')

def lambda_handler(event, context):
    body = json.loads(event['body'])
    message = body['message']

    connections = table.scan()['Items']
    for conn in connections:
        apigw.post_to_connection(
            ConnectionId=conn['connectionId'],
            Data=message.encode('utf-8')
        )

    return {'statusCode': 200}
```



## STEP 5: Create WebSocket API (API Gateway)

### Console Steps:

1. Go to **Services** → **API Gateway**
2. Click **Create API**
3. Choose **WebSocket API**
4. Click **Build**

### WebSocket Settings:

- API name: ChatWebSocketAPI
- Route selection expression: \$request.body.action
- Click **Create API**

**WebSocket URL:** wss://lgzv7q7w97.execute-api.ap-south-1.amazonaws.com/production/

## STEP 6: Configure Routes

### Routes to Create:

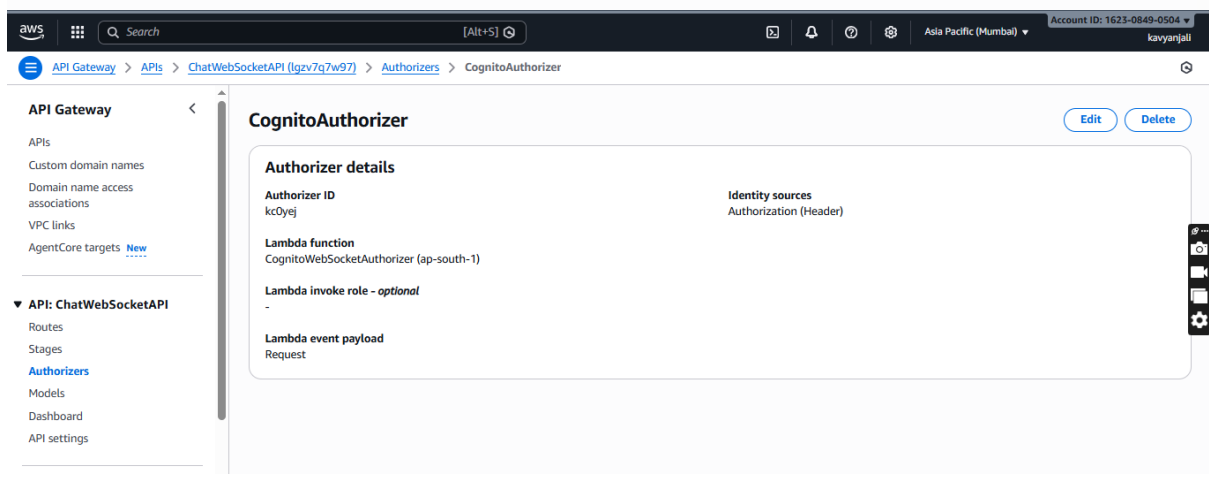
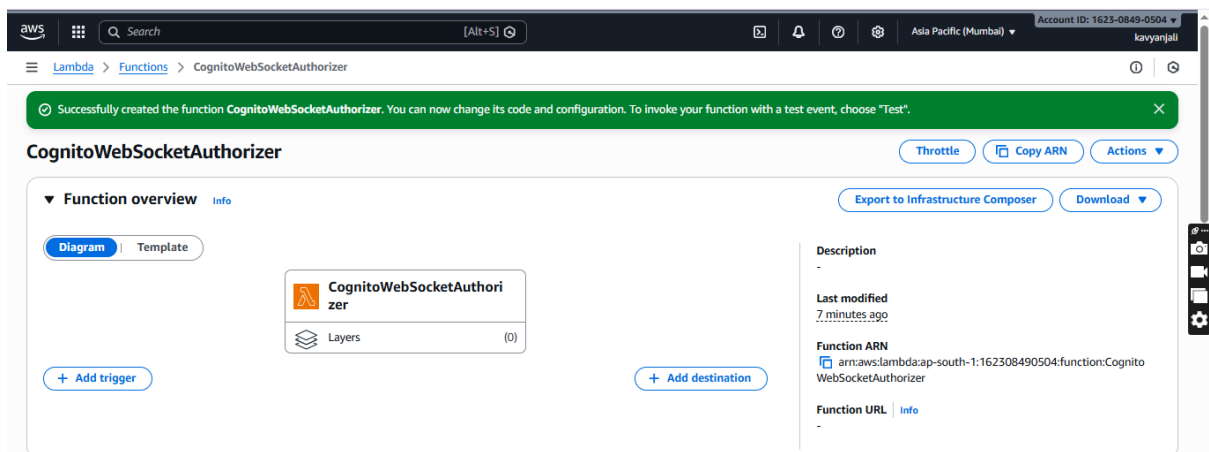
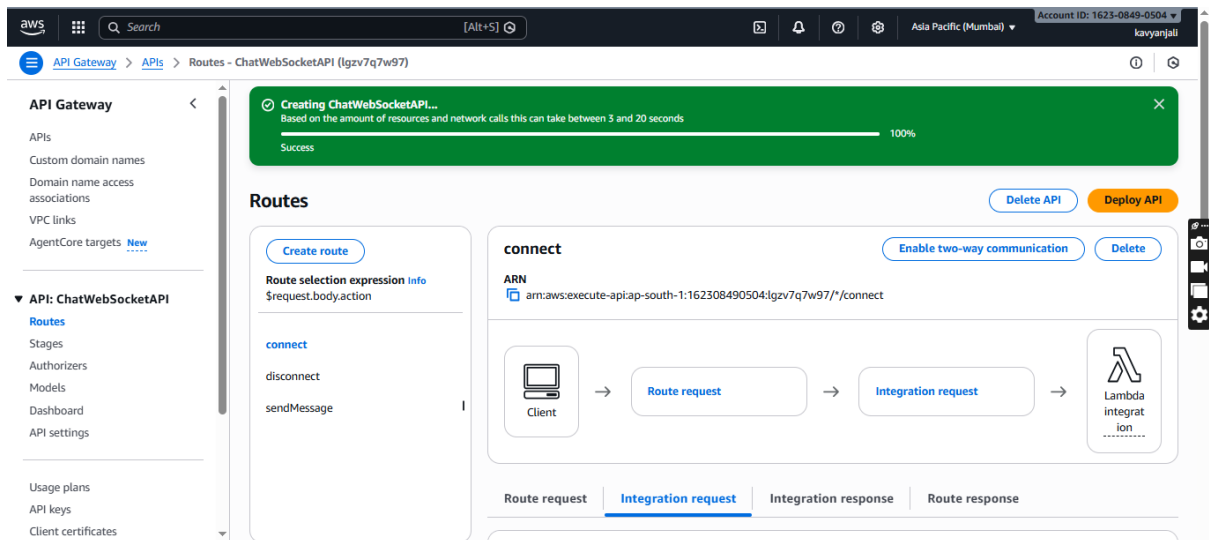
Route	Lambda
\$connect	ChatConnectHandler
\$disconnect	ChatDisconnectHandler
sendMessage	ChatMessageHandler

### Steps:

1. Open **Routes**
2. Click **Create route**
3. Enter route key → Save



#### 4. Attach Lambda integration

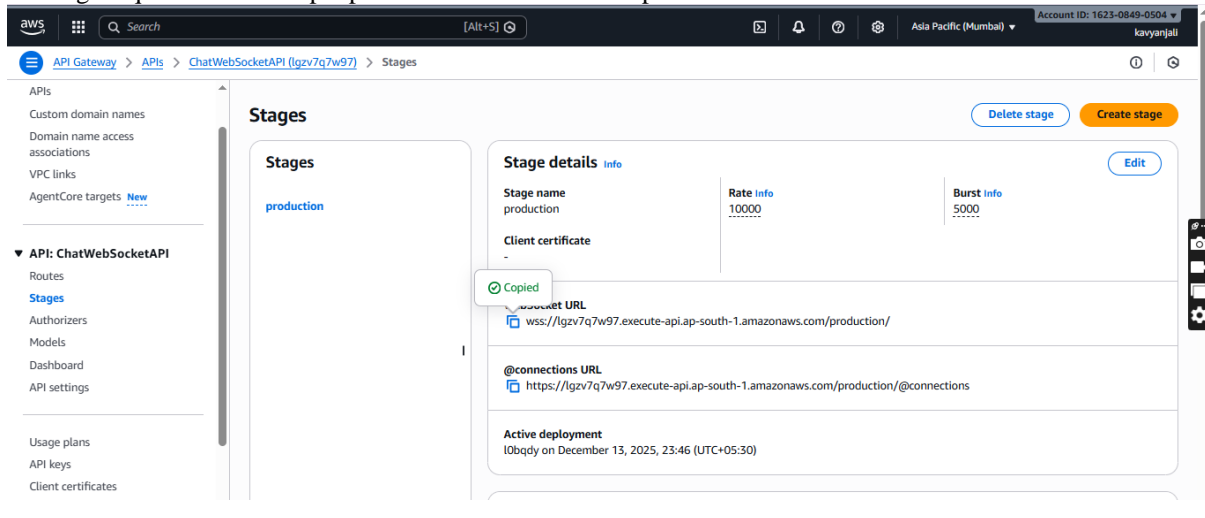


#### STEP 7: Deploy WebSocket API

1. Go to **Deployments**
2. Click **Create deployment**
3. Stage name: **production**
4. Click **Deploy**

Copy **WebSocket URL**:

wss://lgzv7q7w97.execute-api.ap-south-1.amazonaws.com/production/

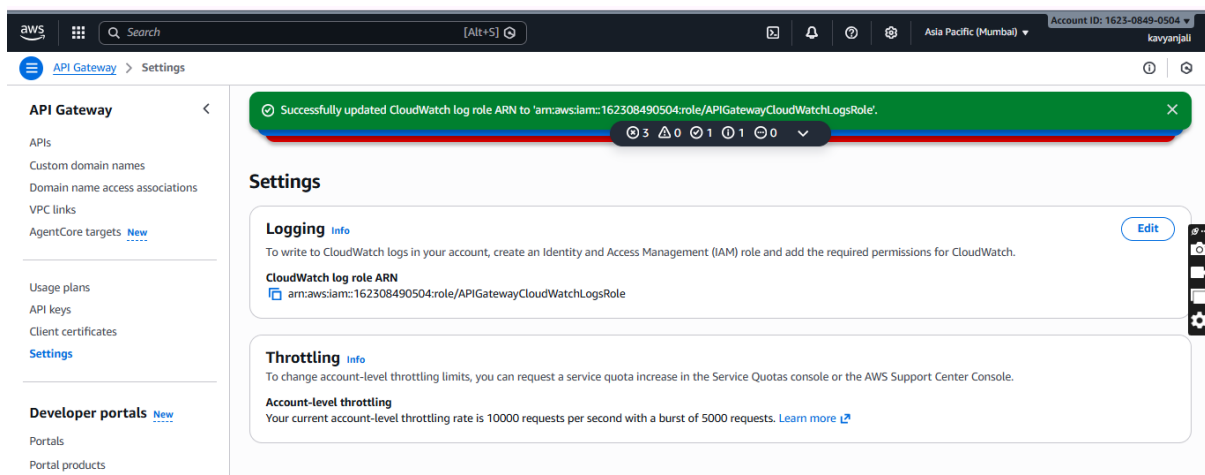


## STEP 8: Enable Cognito Authorizer

1. Go to **API Gateway** → **Authorizers**
2. Create **JWT Authorizer**
3. Identity source: Authorization
4. Issuer URL: Cognito User Pool URL
5. Audience: App Client ID
6. Attach to \$connect route

## STEP 9: Enable Logging (CloudWatch)

1. Go to **API Gateway** → **Stages**
2. Select **production**
3. Enable **CloudWatch Logs**
4. Log level: INFO



## STEP 10: Testing



### Security & Scalability Notes

- Serverless auto-scales
  - No server management
  - Cognito secures users
  - DynamoDB handles millions of connections
- 

### Skills Demonstrated

- WebSocket lifecycle management
  - Event-driven architecture
  - Identity federation with Cognito
  - Stateless Lambda design
  - Real-time serverless communication
- 

### Conclusion

This project implements a fully serverless real-time chat application using AWS WebSocket APIs, Lambda, DynamoDB, and Cognito. It demonstrates scalable event-driven design, secure identity management, and efficient state handling without maintaining servers.

---