Real-Time Chat Application using WebSockets and Serverless

Project Overview

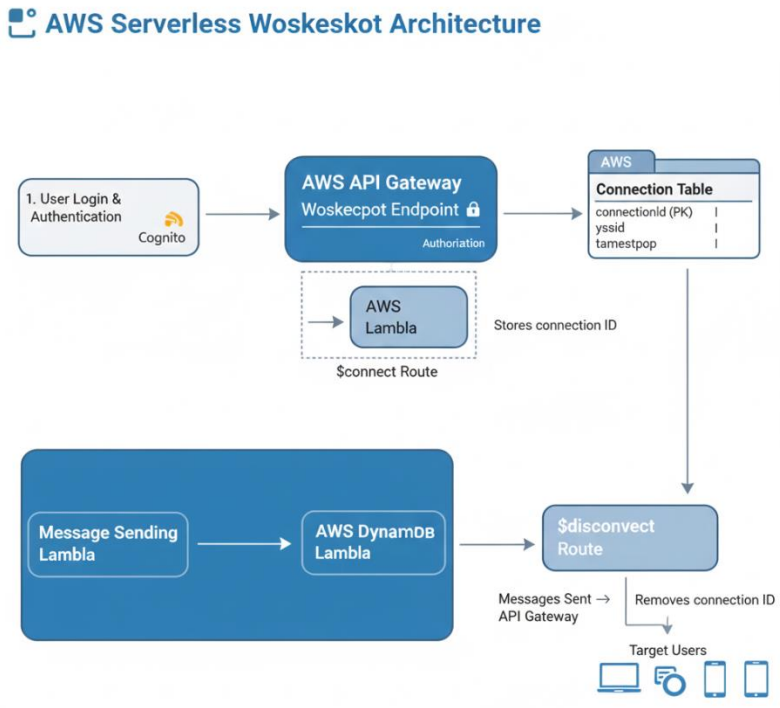
Project Name: Real-Time Chat App using WebSockets and Serverless
Goal: Build a scalable real-time messaging platform without managing servers
Architecture Type: Fully Serverless, Event-Driven
AWS Region: Choose one region (example: ap-south-1)

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.



STEP-BY-STEP AWS CONSOLE INSTRUCTIONS

STEP 1: Create DynamoDB Table (Store WebSocket Connections)

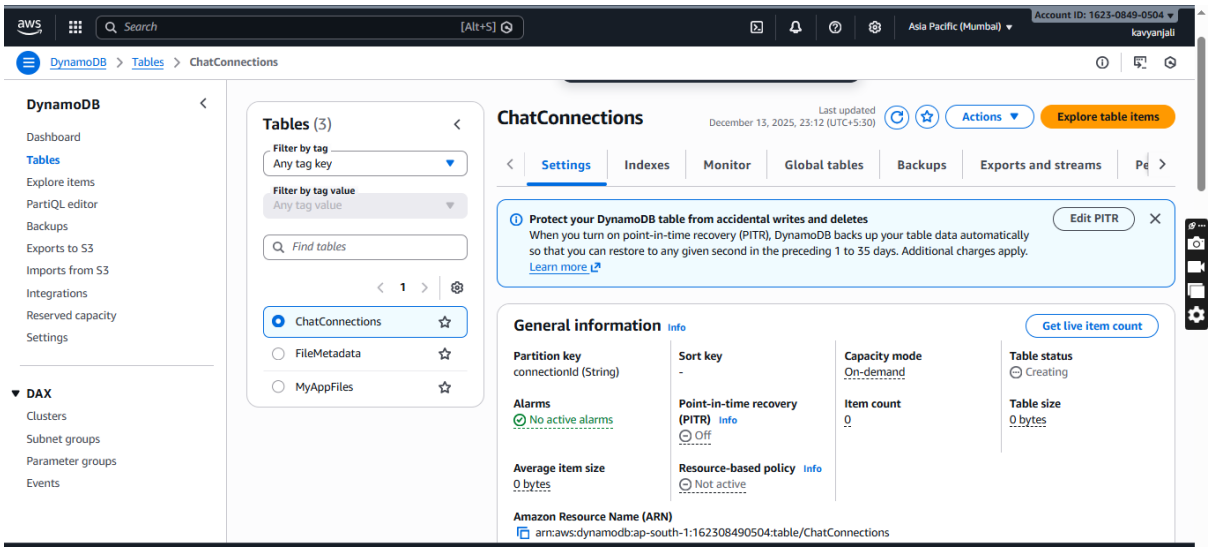
Console Steps:

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

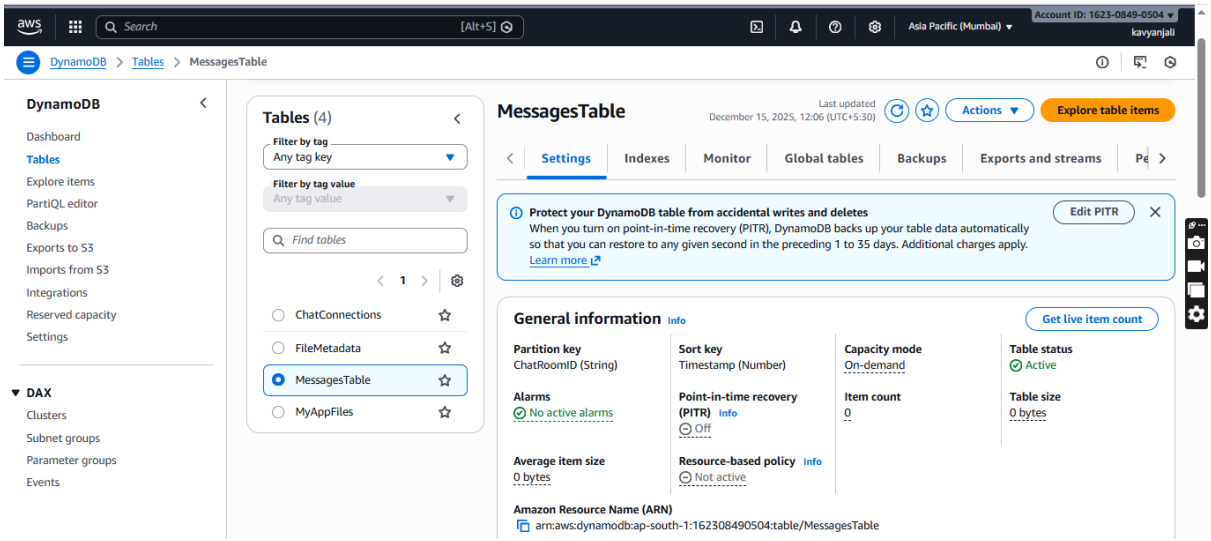
Configuration:

- **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:

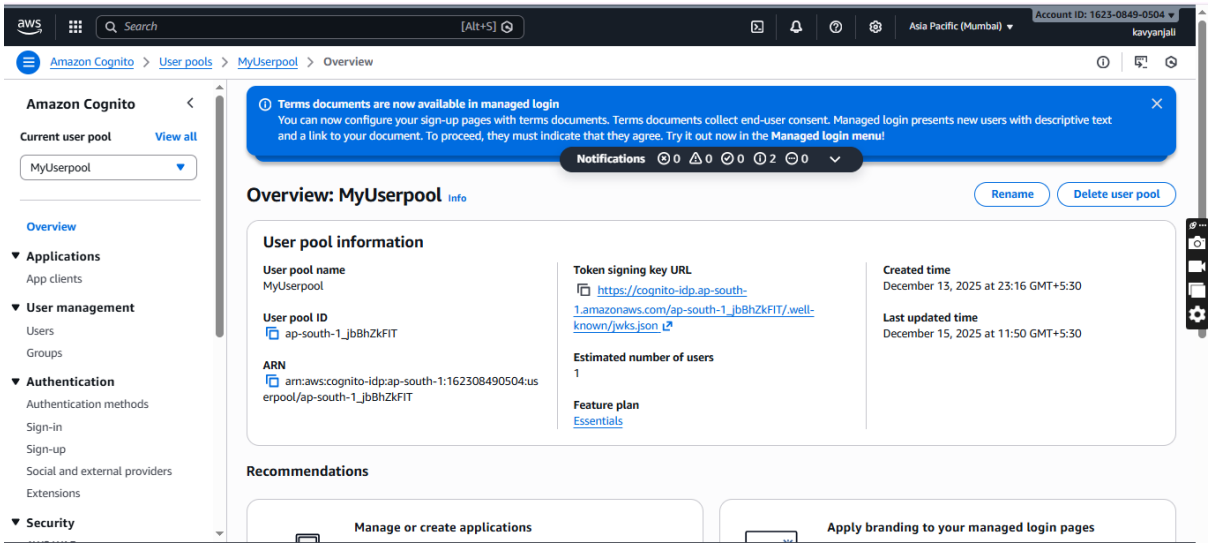
- Sign-in options: **Email**
- Password policy: Default
- MFA: Optional (disable for demo)

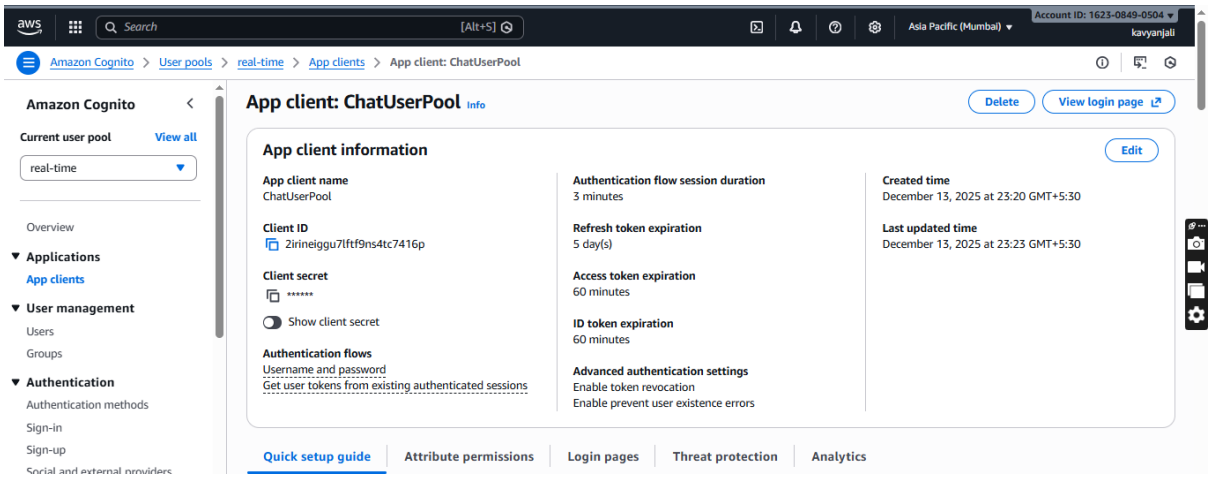
App Client:

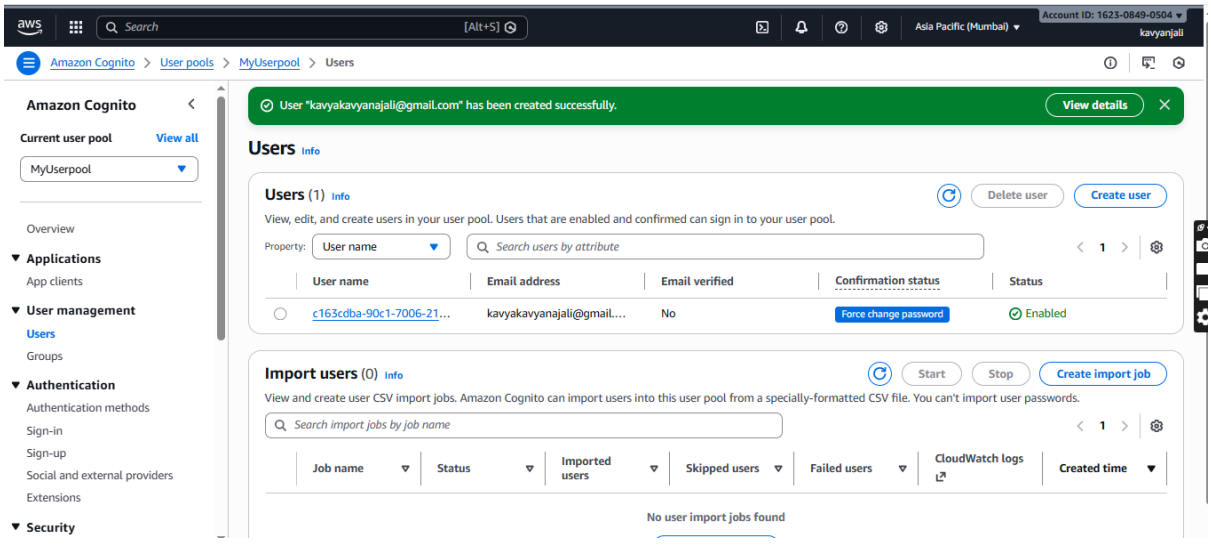
- App client name: ChatAppClient
- Disable client secret

Click **Create user pool**

Purpose: Authenticate users before chat access







STEP 3: Create IAM Role for Lambda

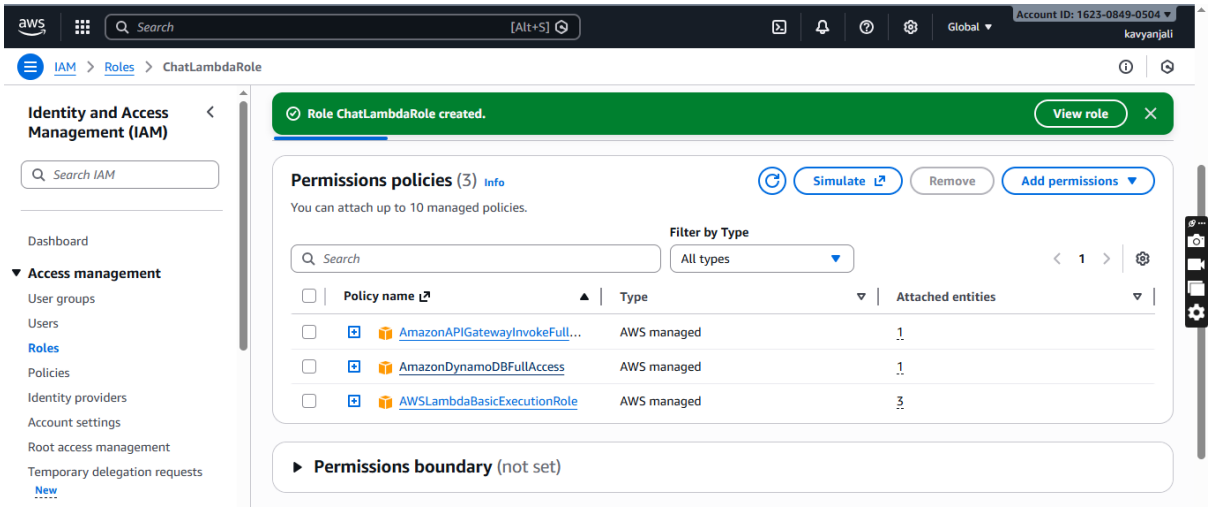
Console Steps:

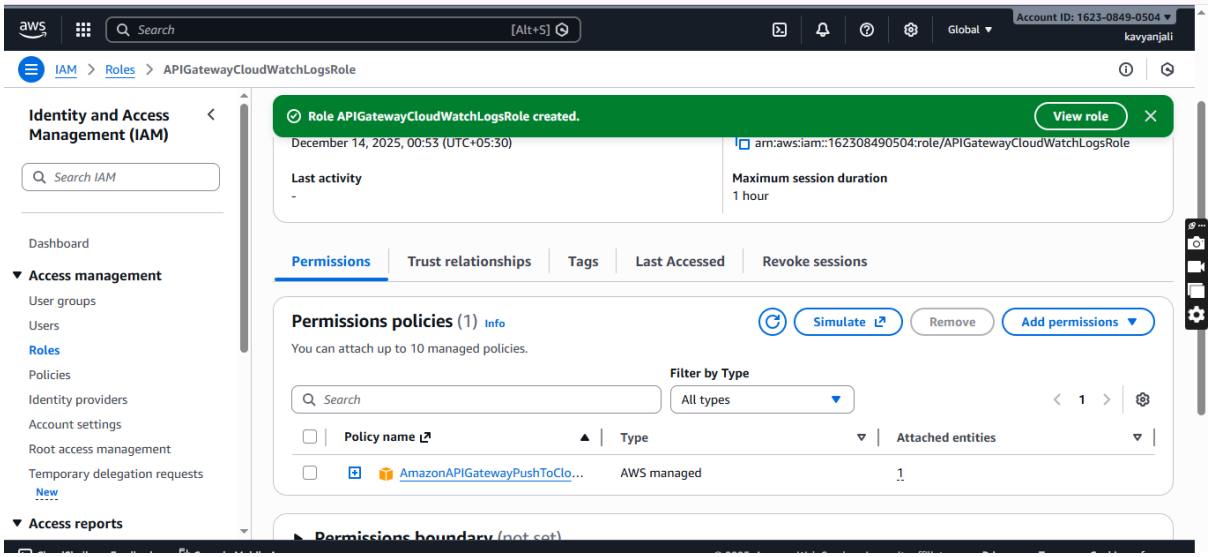
1. Go to **IAM → Roles**
2. Click **Create role**
3. Trusted entity: **AWS service**
4. Service: **Lambda**
5. Click **Next**

Attach Policies:

- AWSLambdaBasicExecutionRole
- AmazonDynamoDBFullAccess
- AmazonAPIGatewayInvokeFullAccess

6. Role name: ChatLambdaRole
7. Click **Create role**





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**

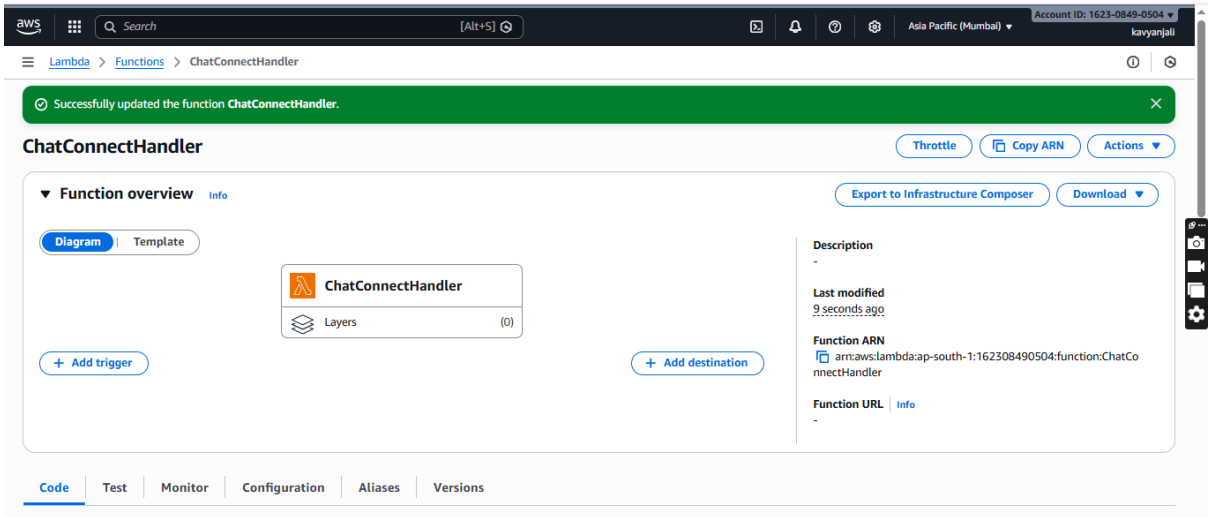
Code :

```
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.

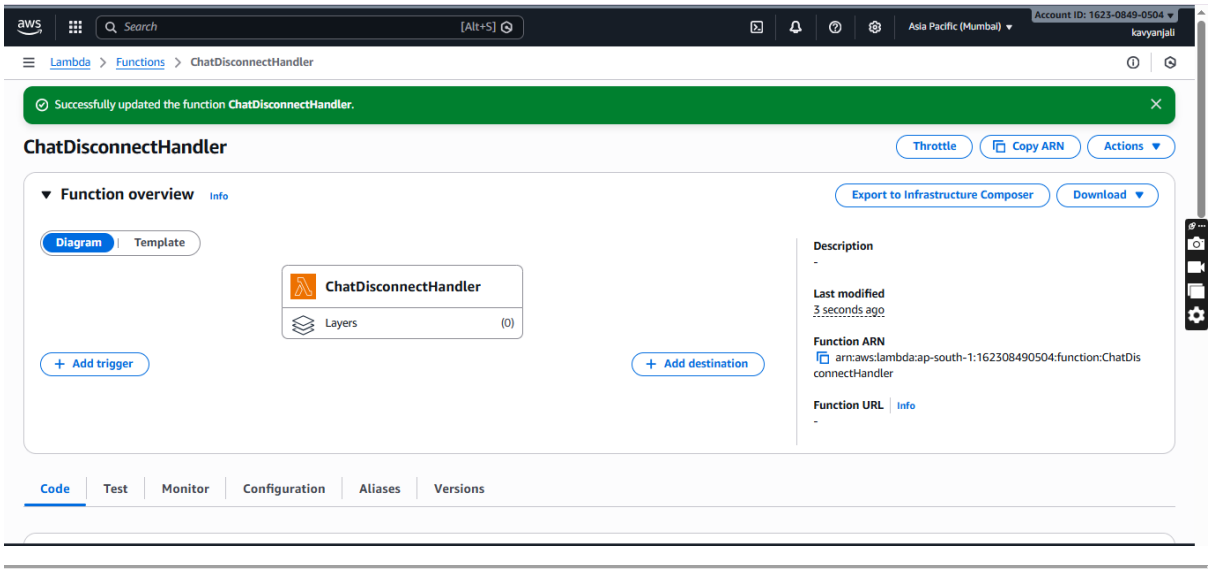
- **Function name:** ChatDisconnectHandler

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:

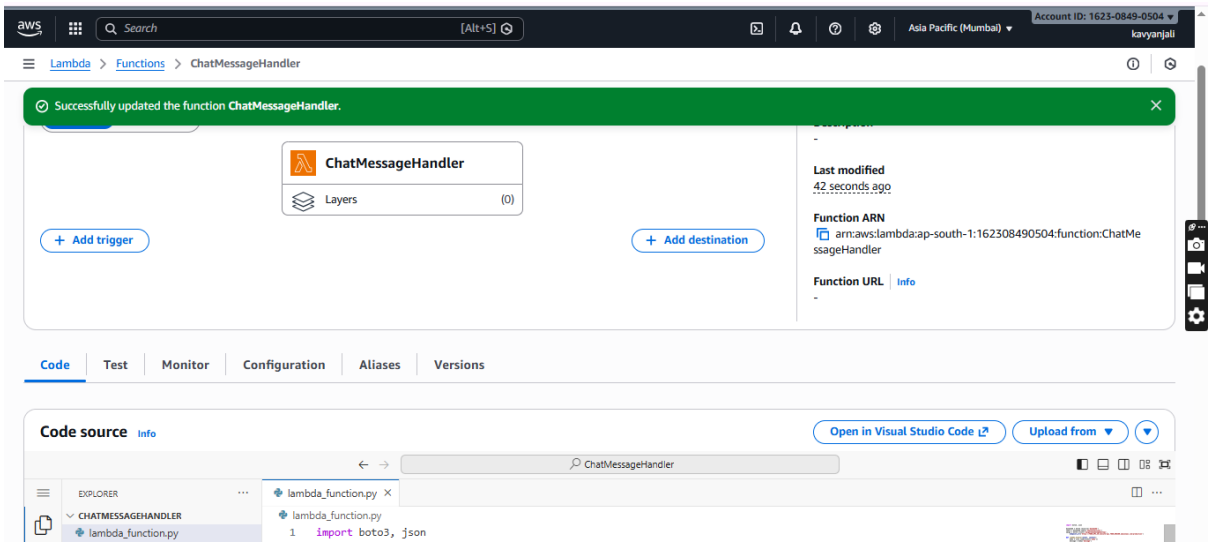
```
import boto3, json

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

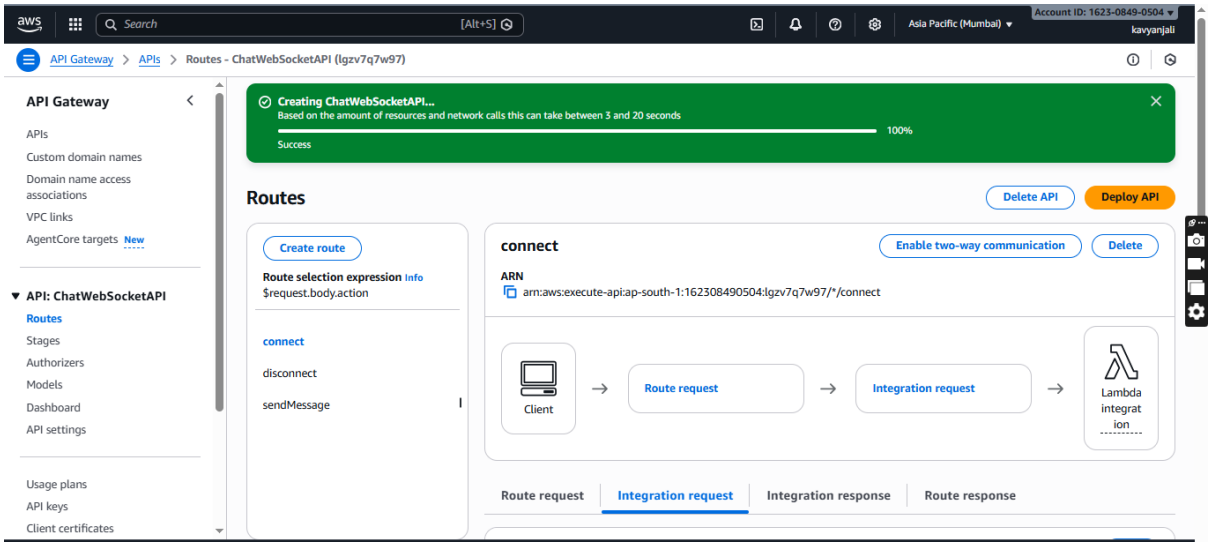
Routes to Create:

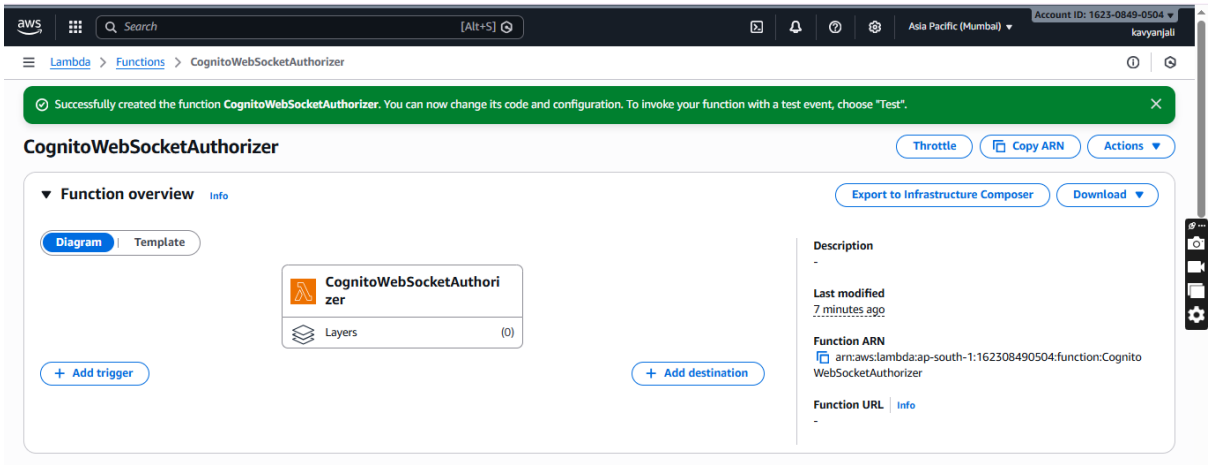
Route	Lambda
$connect	ChatConnectHandler
$disconnect	ChatDisconnectHandler
sendMessage	ChatMessageHandler

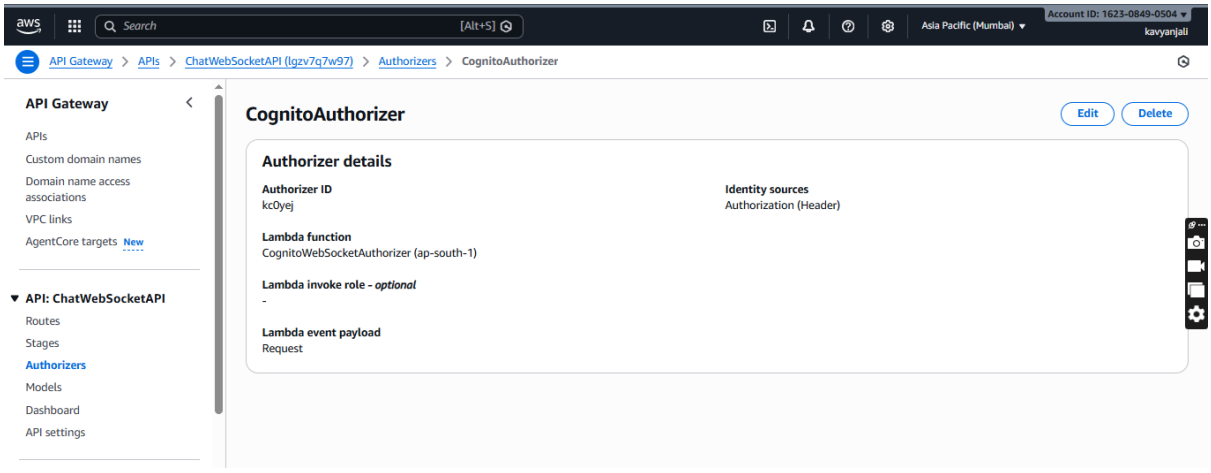
Steps:

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

4. Attach Lambda integration





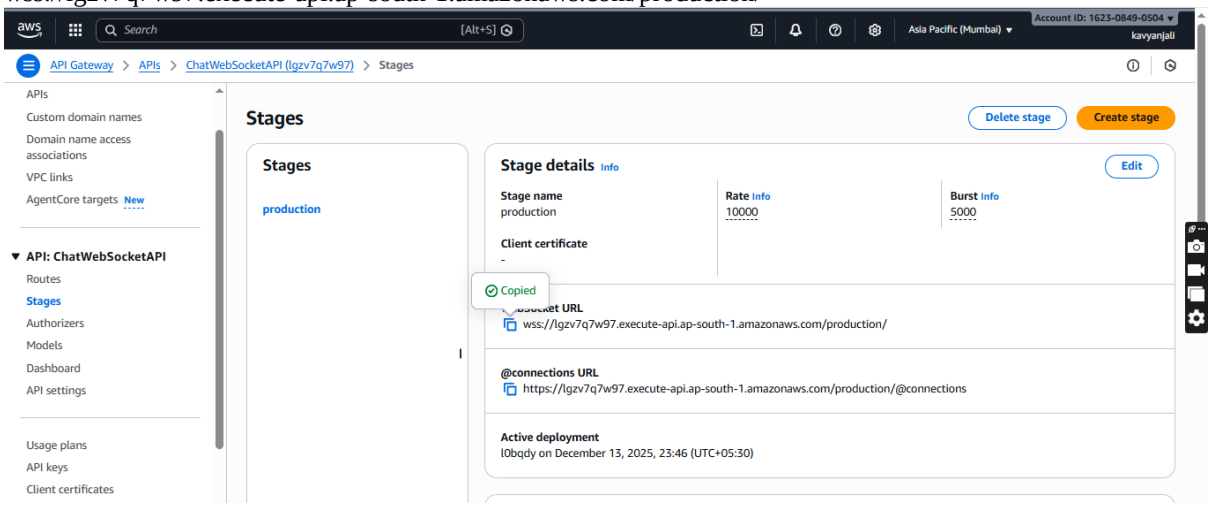


STEP 7: Deploy WebSocket API

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/

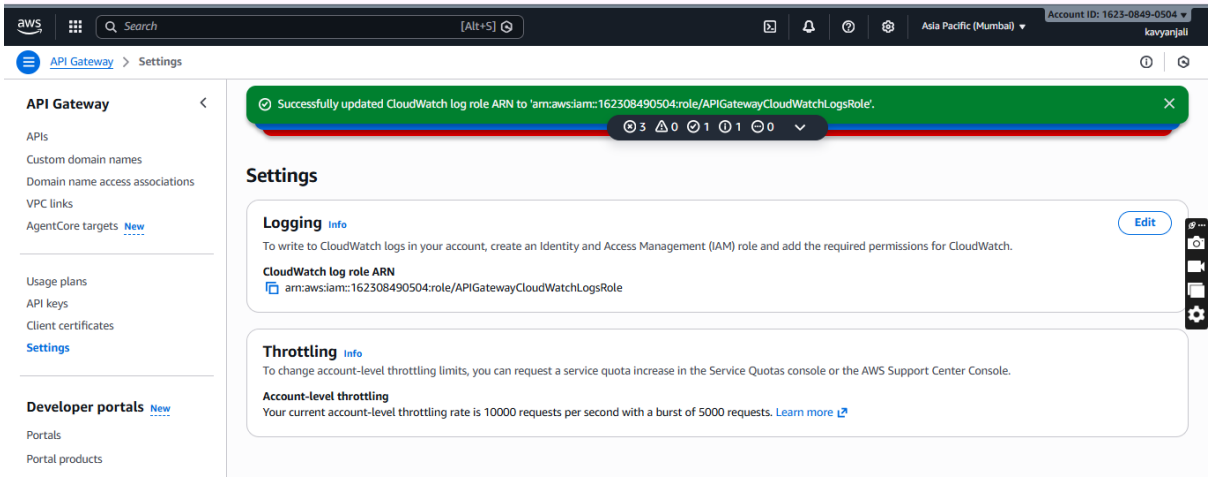


STEP 8: Enable Cognito Authorizer

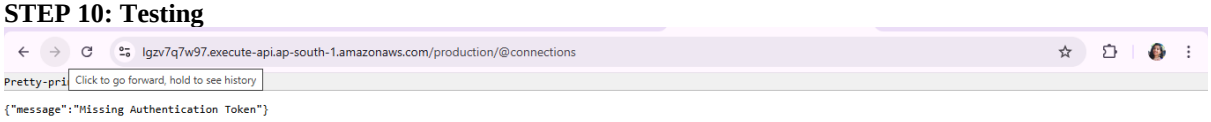
1. Go to **API Gateway → Authorizers**
2. Create **JWT Authorizer**
3. Identity source: Authorization
4. Issuer URL: Cognito User Pool URL
5. Audience: App Client ID
6. Attach to $connect route

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.
