Project Name – Real-time Chat Application using API Gateway, Lambda and Dynamo DB.

Summary- In these Project we need to create Real-Time Chat Application Using API Gateway, Lamda and Dynamo DB.

Websocket API Gateway - A WebSocket API in API Gateway is a collection of WebSocket routes that are integrated with backend HTTP endpoints, Lambda functions, or other AWS services to make two way interactive communication between server and client.

In our project we used api gateway to setup application it is connect to three lamda to perform three action i.e established connection ,send message and disconnect connection

Lamda- AWS Lambda is a serverless compute service that runs your code in response to events and it act as bridge between api gateway and dymano db whenever there is request to api , lamda get trigger and the data automatically put into dynamo db.

STEP 1-CREATE DYNAMO DB.

Enter Table name – Websocket-connection

Id- connectionid

Step 2- Now Create Three Lambda for three action – Connection . Delete and Send Message.

PROCEDURE –STEP 1-CREATE FIRST CONNECTION LAMDA FUNCTION

STEP 2-ENTER python code for an AWS Lambda function interacts with a DynamoDB table to store a connectionid from an incoming WebSocket event.

```
import json

import boto3

import os


# Initialize DynamoDB client

dynamodb = boto3.client('dynamodb')
```

```
def lambda_handler(event, context):
    try:
        # Extract the connection ID from the event
        connection_id = event['requestContext']['connectionId']

        # Get the table name from environment variables
        table_name = os.environ['WEBSOCKET_TABLE']

        # Add the connection ID to the DynamoDB table
        dynamodb.put_item(
            TableName=table_name,
            Item={'connectionid': {'S': connection_id}}
        )

        # Return a success response
        return {
            "statusCode": 200,
            "body": json.dumps({"message": "Connection ID saved successfully"})
        }
    except Exception as e:
        # Log the error and return a failure response
        print(f"Error: {str(e)}")
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)})
        }
```

STEP 2-CREATE ENVIRONMENTAL VARIABLE and assihning role for using dynamo db.

CREATE ENVIRONMENTAL VARIABLE

GO TO CONFIGURATION AND SET ENVIRONMENTAL VARIABLE .

ID NAME – WEBSOCKET_TABLE

VALUE –(DYNAMO DB TABLE NAME)

NOW ASSIGN ROLE TO LAMBDA

GO TO PERMISSION

CLICK ON ROLE

IN VISUAl ADD MORE POLICY

-

SELECT SERVICE- DYNAMO DB

ACTION- PUT ITEM

Set SPECIFIC ARN

NOW REPEATED THE SAME PROCEDURE TO CREATE 2 LAMBDA FUNCTION FOR DISCONNECT AND SEND MESSAGE.

Python code for delete action lambda function

This code is used for deleting a WebSocket connection ID from a DynamoDB table in an AWS Lambda function. Specifically, it removes a connection ID from the DynamoDB table that stores active WebSocket connections.

```
import json

import boto3

import os
```

```
# Initialize DynamoDB client
dynamodb = boto3.client('dynamodb')  # Corrected boto2 to boto3

def lambda_handler(event, context):  # Fixed spelling of 'context'
    try:
        # Extract the connection ID from the event
        connection_id = event['requestContext']['connectionId']  # Fixed 'connectionoid' to
'connectionId'

        # Get the table name from environment variables
        table_name = os.environ['WEBSOCKET_TABLE']

        # Delete the item from DynamoDB
        dynamodb.delete_item(
            TableName=table_name,
            Key={'connectionid': {'S': connection_id}}  # Fixed syntax error with `key` (use
Key, not key)
        )

        # Return a success response
        return {
            "statusCode": 200,
            "body": json.dumps({"message": "Connection ID deleted successfully"})
        }
    except Exception as e:
        # Handle errors and return a failure response
        print(f"Error: {str(e)}")
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)})
        }
```

Python code for send action lambda function

This code is used for managing WebSocket connections and broadcasting messages to multiple connected WebSocket clients using AWS API Gateway, DynamoDB, and Lambda

```
import json
import boto3
import os

# Initialize DynamoDB client
dynamodb = boto3.client('dynamodb')

def lambda_handler(event, context):
    try:
        # Validate if 'body' exists in the event payload
        if 'body' not in event:
            raise KeyError("Missing 'body' in the event payload")

        # Parse the incoming message from the request body
        body = json.loads(event['body'])
        if 'message' not in body:
            raise KeyError("Missing 'message' in the body")
        message = body['message']

        # Initialize a paginator for scanning the DynamoDB table
        paginator = dynamodb.get_paginator('scan')

        # List to hold all connection IDs
        connection_ids = []

        # Get the API Gateway Management API client
        apigatewaymanagementapi = boto3.client(
            'apigatewaymanagementapi',
            endpoint_url="https://" + event["requestContext"]["domainName"] + "/" +
event["requestContext"]["stage"]
        )

        # Retrieve all connection IDs from the DynamoDB table
        for page in paginator.paginate(TableName=os.environ['WEBSOCKET_TABLE']):
            connection_ids.extend(page['Items'])

        # Post the message to each connection ID
        for connection_id in connection_ids:
            try:
                apigatewaymanagementapi.post_to_connection(
                    Data=message,
                    ConnectionId=connection_id['connectionid']['S']  # Correct key structure
                )
            except Exception as e:
                print(f"Failed to post to connection {connection_id['connectionid']['S']}: {str(e)}")

        return {
            "statusCode": 200,
            "body": json.dumps({"message": "Message sent to all connections successfully"})
        }
    except KeyError as ke:
        # Handle missing keys
        print(f"KeyError: {str(ke)}")
        return {
            "statusCode": 400,
            "body": json.dumps({"error": str(ke)})
        }
    except Exception as e:
        # Handle unexpected errors
        print(f"Error: {str(e)}")
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)})
        }
```

But here we will assign two role since its main action so to access dynamo db for scan data and additionally we need to assign get execute api role to allow lambda to get post message in all connection.

Add service- dynamo db

Action- scan

Add service –execute api

Action – Manage connection.

Save

This code should not work as intended in your AWS Lambda environment. Ensure that:

- **The environment variable WEBSOCKET_TABLE is correctly set.**
- **The Lambda execution role has the necessary permissions to perform the dynamodb:PutItem action on the specified DynamoDB table.**

Step 3- create Api gateway now

Select WEBSOCKET API GATEWAY

Create 3 route

connect , disconnect and default route and integrate lamba

Add route and integrate lambda here now used postman to test our api gateway

Use postman install postman to test api gateway

First create 2 or 3 websocket api gateway

Then try to send message by using these form at

{“action”: “default” , “message”: “yo yo honey singh”

INTERVIEW QUESTION :-

What is the role of API Gateway in a serverless architecture?

API Gateway acts as a fully managed service that enables developers to create, publish, maintain, and secure APIs. It serves as the entry point for client requests and routes them to the backend services, such as AWS Lambda functions.

How do you set up WebSocket API in API Gateway for real-time communication?

Create a WebSocket API in API Gateway. Define routes like $connect, $disconnect, and $default. Associate each route with an integration, such as an AWS Lambda function, to handle events. Deploy the API and share the endpoint with clients.

What are the key differences between REST API and WebSocket API in API Gateway?

REST APIs

Request-Response Communication:

How it works: The client sends a request to the server, and the server processes it and sends back a response.

Example:

Client: "What's the weather like today?"

Server: "The weather is sunny."

This type of communication is one-directional at a time.

FOR EG - ONLINE GOOGLE FORM SUBMISSION.

WebSocket APIs

Full-Duplex Communication:

How it works: Both the client and the server can send messages to each other independently once the connection is established. This enables real-time, two-way communication.

Example:

After connecting to a chat server:

Client: "Hi, how are you?"

Server: "I’m good, thanks for asking!"

Server (proactively): "Don’t forget the team meeting at 2 PM."

EG - REAL-TIME CHAT APPLICATION.

How do you secure an API Gateway WebSocket endpoint?

Use IAM authentication, custom authorizers, or Amazon Cognito to validate and authorize connections.

What is the purpose of the $connect, $disconnect, and $default routes in WebSocket APIs?

$connect: Triggered when a client connects to the WebSocket API.

$disconnect: Triggered when a client disconnects.

$default: Triggered for any messages sent by the client that do not match other defined routes.

Postman: Postman is an API(application programming interface) development tool that helps to build, test and modify APIs.

AWS Lambda

How does AWS Lambda integrate with API Gateway for WebSocket APIs?

Lambda functions are associated with WebSocket routes ($connect, $disconnect, $default). When a route is triggered, API Gateway invokes the respective Lambda function to handle the event.

What are the key differences between REST API and WebSocket API in API Gateway?

REST APIs

Request-Response Communication:

How it works: The client sends a request to the server, and the server processes it and sends back a response.

Example:

Client: "What's the weather like today?"

Server: "The weather is sunny."

This type of communication is one-directional at a time.

FOR EG - ONLINE GOOGLE FORM SUBMISSION.

WebSocket APIs

Full-Duplex Communication:

How it works: Both the client and the server can send messages to each other independently once the connection is established. This enables real-time, two-way communication.

Example:

After connecting to a chat server:

Client: "Hi, how are you?"

Server: "I'm good, thanks for asking!"

Server (proactively): "Don't forget the team meeting at 2 PM."

EG - REAL-TIME CHAT APPLICATION.

How do you secure an API Gateway WebSocket endpoint?

Use IAM authentication, custom authorizers, or Amazon Cognito to validate and authorize connections.

What is the purpose of the $connect, $disconnect, and $default routes in WebSocket APIs?

$connect: Triggered when a client connects to the WebSocket API.

$disconnect: Triggered when a client disconnects.

$default: Triggered for any messages sent by the client that do not match other defined routes.

Postman: Postman is an API(application programming interface) development tool that helps to build, test and modify APIs.

AWS Lambda

How does AWS Lambda integrate with API Gateway for WebSocket APIs?

Lambda functions are associated with WebSocket routes ($connect, $disconnect, $default). When a route is triggered, API Gateway invokes the respective Lambda function to handle the event.

How do you manage scaling issues in Lambda for high-volume chat applications?

Use reserved concurrency to ensure enough execution capacity.

In a chat application, each WebSocket message from a client triggers a Lambda invocation. If you expect peak traffic of 500 concurrent users, you can reserve 500 concurrent executions to handle the load.

What is the best practice for handling cold starts in Lambda functions?

Use provisioned concurrency to reduce cold start delays, especially for critical real-time use cases.

Provisioned concurrency pre-warms Lambda instances, ensuring that a specified number of environments are always ready to handle requests. These pre-initialized environments reduce the latency caused by cold starts.

How can you maintain a client connection across multiple Lambda invocations?

Store the connection ID in DynamoDB or a similar database. Use the connection ID to send messages across different Lambda invocations.

How do you debug AWS Lambda functions in production?

Use CloudWatch Logs and X-Ray to analyze and debug Lambda execution

dynamo db question-

Why is DynamoDB suitable for a real-time chat application?

DynamoDB offers low-latency, high-throughput performance, and supports scalability and serverless architecture, making it ideal for chat applications.

Explain the concept of partition key and sort key in DynamoDB.

The partition key determines data distribution across partitions. The sort key allows sorting and querying of related items within the same partition.

How do you use DynamoDB Streams to trigger actions in a chat application?

we can simply used lambda to trigger in which we run our python bash code for puting and deleting item in dynamo db.

How can you handle DynamoDB limits and optimize performance in a high-traffic application?

Use on-demand capacity mode for unpredictable traffic

What challenges might you face when scaling a real-time chat application, and how can you overcome them?

Challenges: High connection volume, message ordering, and latency.

Solutions: Use DynamoDB auto-scaling, design partition keys carefully, and optimize Lambda concurrency

How can you monitor and log user activity in a real-time chat application?

Use CloudWatch Logs and metrics for Lambda and API Gateway.