GENERALITAT VALENCIANA
CONSELLERIA D'EDUCACIÓ, INVESTIGACIÓ, CULTURA I ESPORT

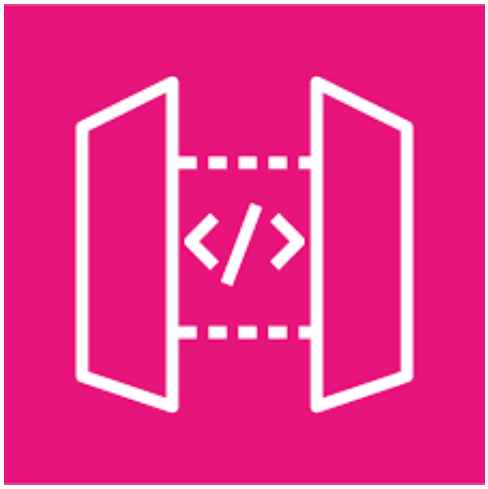
IES JUAN DE GARAY
C /Juan de Garay, 25 46017 VALENCIA
Tel.: 961206045 Fax: 961206046
46012963@gva.es juandegaray.com

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Práctica 9

API GATEWAY

Web Socket Chat



WS-1

Vamos a crear un chat donde se envían un mensaje, éste se guarda en una BBDD DynamoDB y luego recuperaremos los mensajes o haremos un eco de ellos.

Para empezar, vamos a montar un mini-chat en tiempo real usando:

- API Gateway WebSocket
- AWS Lambda
- DynamoDB (para gestionar conexiones)
- wscat para probar desde consola

Esquema lógico:

- El cliente (wscat o navegador) abre un WebSocket a API Gateway.
- API Gateway:
- En connect → llama a Lambda A (registra conexión).
- En disconnect → llama a Lambda B (borra conexión).
- En el routeKey sendMessage → llama a Lambda C (envía mensajes al resto). Lambda C usa la Management API de API Gateway para mandar mensajes a todos los connectionId guardados en DynamoDB.

Primero, Crear la tabla DynamoDB para conexiones
Objetivo: Guardar los connectionId de los clientes conectados.

- Nombre: ChatConnections
- Primary key:
- Partition key: connectionId (String)

Crear las Lambdas

Vamos a crear 4 funciones Lambda (todas Node.js 20.x, runtime en región, p.ej. eu-west-1):

- onConnect → maneja $connect
- onDisconnect → maneja $disconnect
- defaultRoute → maneja $default
- sendMessage → maneja el route sendMessage

Las lambdas las crearemos con el código que hay por defecto para poder desplegarlas en el api. Ahora podemos hacer una primera prueba de la función connect:

```
jorge@mjolnir chat-aws % wscat -c wss://mzjfmrdu86.execute-api.us-east-1.amazonaws.com/production/
Connected (press CTRL+C to quit)
>
```

Comprobad que en CloudWatch aparece la función lambda onConnect y onDisconnect

Ahora vamos a guardar la conexión en una tabla de DynamoDB que ya hemos creado previamente y la borraremos cuando se desconecte, para ello el código de la lambda onConnect:

index.mjs

```
import { DynamoDBClient, DeleteItemCommand } from "@aws-sdk/client-dynamodb";
const dynamo = new DynamoDBClient({});

export const handler = async (event) => {
  const connectionId = event.requestContext.connectionId;

  await dynamo.send(
    new DeleteItemCommand({
      TableName: "ChatConnections",
      Key: { connectionId: { S: connectionId } }
    })
  );

  return { statusCode: 200 };
};
```

index.mjs > handler > Item > connectionId

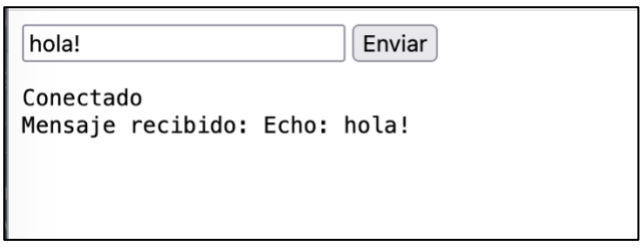
```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

const dynamo = new DynamoDBClient({});

export const handler = async (event) => {
  const connectionId = event.requestContext.connectionId;

  await dynamo.send(
    new PutItemCommand({
      TableName: "ChatConnections",
      Item: { connectionId: { S: connectionId } }
    })
  );

  return { statusCode: 200 };
};
```

Nos volveremos a conectar y entonces podremos ver como aparece el id de la conexión en la tabla. Y cuando se desconecte se borrará. Ahora es el momento de empezar a configurar el envío de mensajes. Para ello ejecutaremos el siguiente comando en la terminal:

```
jorge@mjolnir chat-aws % wscat -c wss://mzjfmrdu86.execute-api.us-east-1.amazonaws.com/production/
Connected (press CTRL+C to quit)
> {"action":"sendMessage","data":"hola desde wscat"}
< Echo: hola desde wscat
>
```

```
import { ApiGatewayManagementApi } from "@aws-sdk/client-apigatewaymanagementapi";

export const handler = async (event) => {
  console.log("Evento recibido:", JSON.stringify(event, null, 2));

  const { requestContext, body } = event;

  let parsedBody;
  try {
    parsedBody = JSON.parse(body);
  } catch (err) {
    console.error("Error al parsear body:", err);
    return { statusCode: 400 };
  }

  const message = parsedBody.data || "mensaje vacío";

  const domain = requestContext.domainName;
  const stage = requestContext.stage;

  const endpoint = `https://${domain}/${stage}`;

  const apiGw = new ApiGatewayManagementApi({
    endpoint,
  });

  try {
    await apiGw.postToConnection({
      ConnectionId: requestContext.connectionId,
      Data: Buffer.from(`Echo: ${message}`),
    });

    return { statusCode: 200 };
  } catch (err) {
    console.error("ERROR al enviar mensaje:", err);
    return {
      statusCode: 500,
      body: JSON.stringify({ error: "Failed to send message" }),
    };
  }
};
```

Ahora vamos a crear un front desde el cual podamos interactuar:



Y el código será:

```html
<!DOCTYPE html>
<html>
<body>
<input id="msg" placeholder="Mensaje..." />
<button onclick="enviar()">Enviar</button>

<pre id="log"></pre>

<script>
    const ws = new WebSocket("wss://mzjfmrdu86.execute-api.us-east-1.amazonaws.com/production");

    ws.onopen = () => log("Conectado");

    ws.onmessage = (msg) => {
        log("Mensaje recibido: " + msg.data);
    };

    function enviar() {
        const texto = document.getElementById("msg").value;

        ws.send(JSON.stringify({
            action: "sendMessage",
            data: texto
        }));
    }

    function log(msg) {
        document.getElementById("log").textContent += msg + "\n";
    }
</script>
</body>
</html>
```

Ahora queda como mejora realizar un chat en el cual recupere los mensajes y muestre el usuario que está conectado, para ello también mostraremos los últimos mensajes, y usaremos diferentes pestañas para tal fin.