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

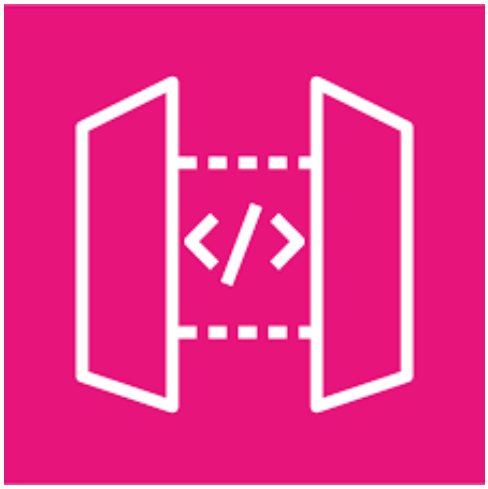
IES JUAN DE GARAY
C /Juan de Garay, 25 46017 VALENCIA
Tel.: 961206045 Fax: 961206046
46012963@gva.es juandegaray.com

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Práctica 8

API GATEWAY

Chat



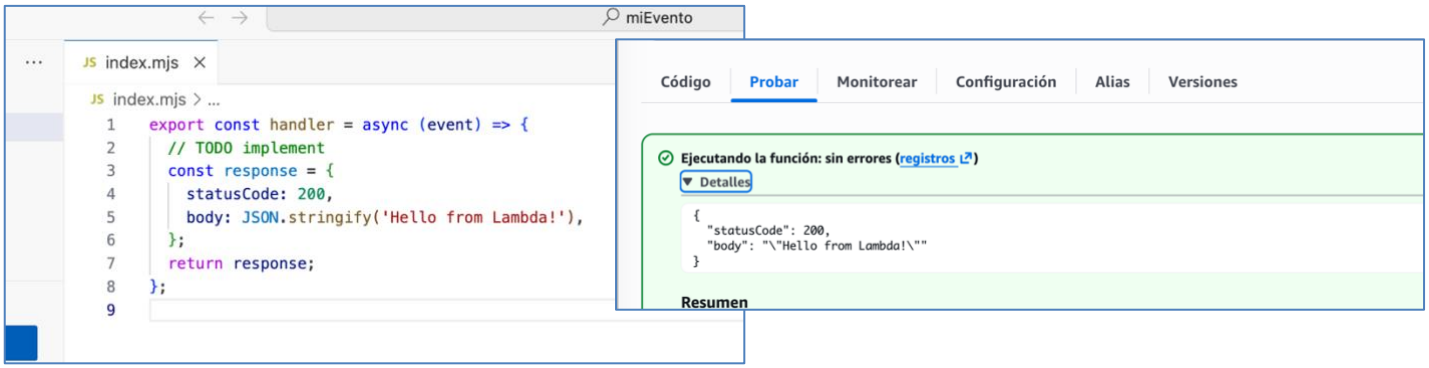
Introducción

Vamos a crear un chat donde una API envía un mensaje, éste se guarda en una BBDD DynamoDB y luego recuperaremos los mensajes, tanto la lectura como el almacenamiento usaremos el servicio Lambda, y para la parte del front integrando estas llamadas una página web alojada en un S3.

La visión final debería ser algo muy parecido a la imagen siguiente:

Lambda "Hola mundo" + API Gateway REST

En primer lugar, crearemos una lambda hola, donde haremos las pruebas para ver si funciona, de hecho, es código nos viene prácticamente dado y sólo hay que hacer la prueba.



Endpoint /send

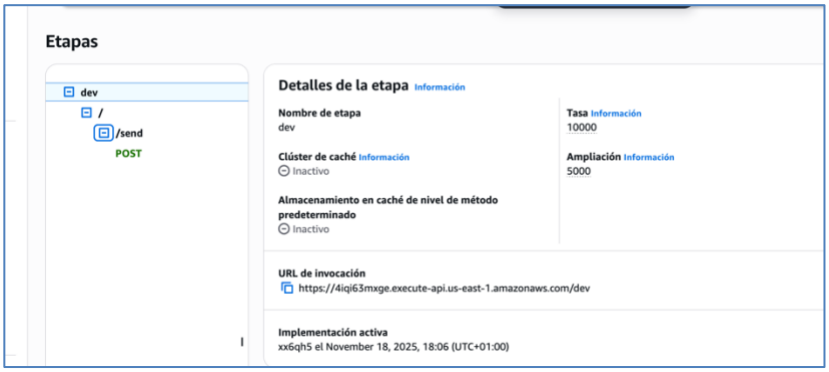
A continuación, crearemos otra lambda chat-send, la cual debería responder al siguiente contrato:

La primera versión de código para poder probarlo:

```
export const handler = async (event) => {
  let body;

  // Si viene como objeto, úsalo directo
  if (typeof event.body === "object") {
    body = event.body;
  } else {
    // Si viene como string (caso API Gateway o Lambda Test), parsear
    try {
      body = JSON.parse(event.body || "{}");
    } catch {
      return {
        statusCode: 400,
        body: JSON.stringify({ error: "JSON inválido" }),
      };
    }
  }

  const { user, text } = body;

  if (!user || !text) {
    return {
      statusCode: 400,
      body: JSON.stringify({ error: "user y text son obligatorios" }),
    };
  }

  const message = {
    user,
    text,
    timestamp: Date.now(),
  };

  return {
    statusCode: 200,
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ status: "ok", echo: message }),
  };
};
```

Cuando pruebo desde lambda con el JSON correcto debería salir algo como lo siguiente:

En el API Gateway crearemos algo con la misma estructura para poder probar lambda. (El post usa la función lambda chat-send), y la desplegamos.



Y para probar el endpoint con curl, nos devolverá un mensaje de error que podremos consultar en los eventos de Cloudwatch, una vez consultado retocaremos la lambda:

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

  let body = event.body;

  // Caso: API Gateway pasa user/text directamente
  if (!body && (event.user || event.text)) {
    body = event;
  }

  // Si body es string → parsear
  if (typeof body === "string") {
    try {
      body = JSON.parse(body);
    } catch {
      return {
        statusCode: 400,
        body: JSON.stringify({ error: "JSON inválido" }),
      };
    }
  }

  const { user, text } = body || {};

  if (!user || !text) {
    return {
      statusCode: 400,
      body: JSON.stringify({ error: "user y text son obligatorios" }),
    };
  }

  const message = {
    user,
    text,
    timestamp: Date.now(),
  };

  return {
    statusCode: 200,
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ status: "ok", echo: message }),
  };
};

```

Con esto ya podemos "enviar" mensajes, estos mensajes para hacerlos persistentes los guardaremos en una BBDD DynamoDB.

Base de datos DynamoDB

Se adjunta la configuración de la tabla creada:

Ahora pasaremos a modificar la lambda para que guarde estos mensajes en la tabla cada vez que hagamos una llamada curl.

```
jorge@mjolnir chat-aws % curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"user":"alice","text":"mensaje guardado en Dynamo"}' \
  https://4iqi63mxge.execute-api.us-east-1.amazonaws.com/dev/send

{"statusCode":200,"headers":{"Content-Type":"application/json"},"body":"{\"status\":\"saved\",\"message\":{\"room\":\"gen
ral\",\"timestamp\":1763491006530,\"user\":\"alice\",\"text\":\"mensaje guardado en Dynamo\"}}"}
jorge@mjolnir chat-aws %
```

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

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
const TABLE_NAME = process.env.TABLE_NAME || "ChatMessages";

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

  let { user, text, room } = event;

  if ((!user || !text) && event.body) {
    try {
      const body =
        typeof event.body === "string"
          ? JSON.parse(event.body)
          : event.body;
      user = body.user;
      text = body.text;
      room = body.room;
    } catch {
      return {
        statusCode: 400,
        body: JSON.stringify({ error: "JSON inválido" }),
      };
    }
  }

  if (!user || !text) {
    return {
      statusCode: 400,
      body: JSON.stringify({ error: "user y text son obligatorios" }),
    };
  }

  if (!room) room = "general"; // 👈 agregado

  const timestamp = Date.now();
  const message = { room, timestamp, user, text };

  await docClient.send(
    new PutCommand({
      TableName: TABLE_NAME,
      Item: message,
    })
  );

  return {
    statusCode: 200,
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ status: "saved", message }),
  };
};

```

Ahora llegados a este punto ya somos capaces de llamar a un endpoint de nuestra API y guardar datos, el siguiente paso será mostrar los mensajes.

Endpoint /mesages

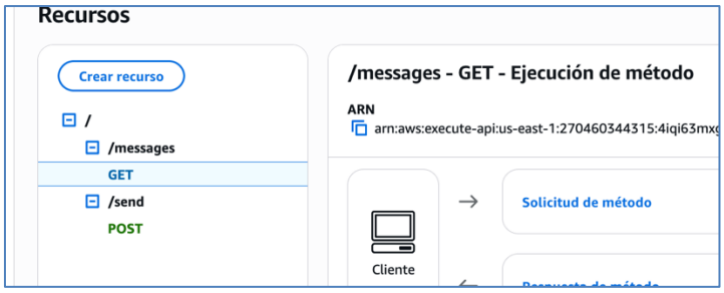
En este apartado primero crearemos una lambda para recuperar los mensajes y luego un endpoint en el AGW para poder llamarlo. Tendrá la forma "**GET /messages?room=general&limit=20".**

Al probar el siguiente código en la lambda:

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

const client = new DynamoDBClient({});

export const handler = async (event) => {
  const room = event.queryStringParameters?.room || "general";
  const limit = parseInt(event.queryStringParameters?.limit || "20", 10);

  try {
    const result = await client.send(
      new QueryCommand({
        TableName: "ChatMessages",
        KeyConditionExpression: "room = :room",
        ExpressionAttributeValues: {
          ":room": { S: room },
        },
        ScanIndexForward: false, // false = orden descendente (más recientes primero)
        Limit: limit,
      })
    );

    const messages = (result.Items || []).map((item) => ({
      room: item.room.S,
      timestamp: Number(item.timestamp.N),
      user: item.user.S,
      text: item.text.S,
    }));

    return {
      statusCode: 200,
      body: JSON.stringify({ messages }),
    };
  } catch (err) {
    console.error(err);
    return {
      statusCode: 500,
      body: JSON.stringify({ error: "Error leyendo mensajes" }),
    };
  }
};

```

Nos devolverá los registros:

Ahora solo falta integrarlo en el api gateway.



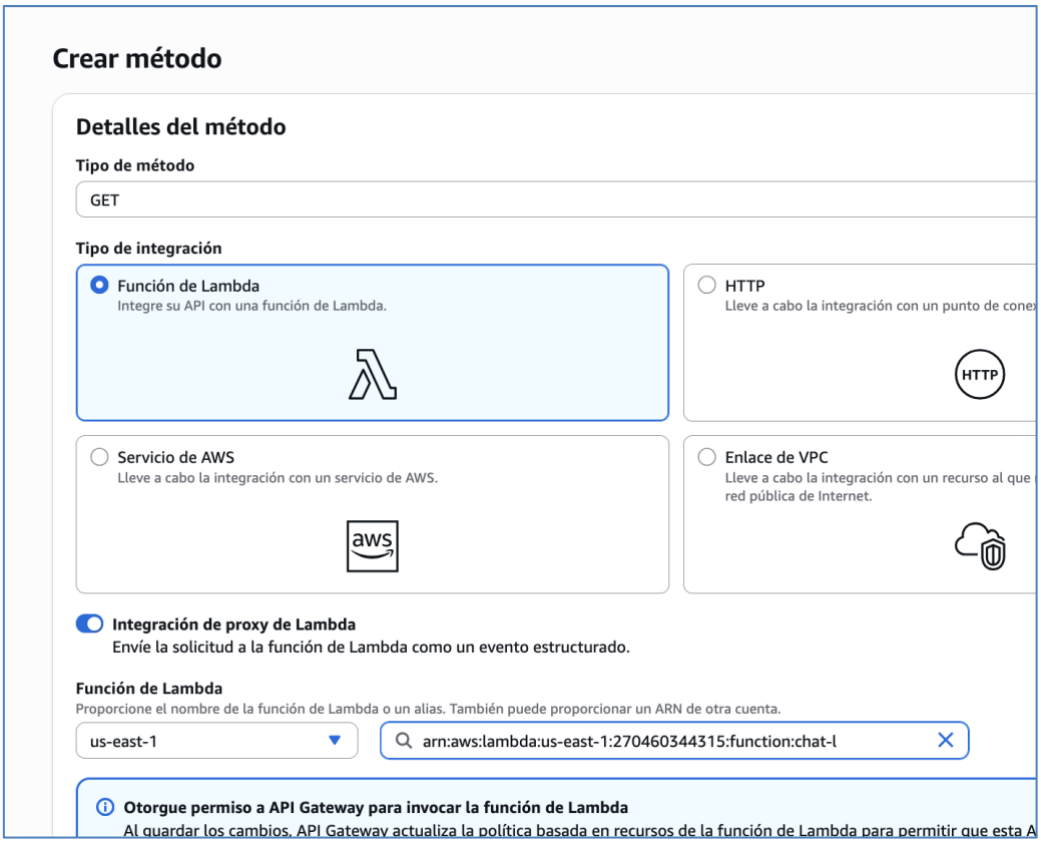
Y desplegarlo para obtener un endpoint. Para ello podremos probar en el navegador:

Llegados a este punto donde listamos y guardamos mensajes, ahora sólo nos faltaría el frontend.

Frontend

Cuando probemos a usar el navegador nos encontraremos con que nos da error, para ello tanto en las lambdas como en los recursos de nuestra API deberemos habilitar CROSS.

Aun así, veremos que la página SI que envía información y la guarda en la tabla, pero NO muestra los mensajes, para ello deberemos borrar el método GET y volverlo a crear con Lambda Proxy Integration, para esto analizaremos como recibe la información.



Ahora la UI quedará como se muestra en la imagen.

```
</head>
<body>
  <h1>Chat AWS 💬</h1>

  <div id="messages"></div>

  <label>Usuario:</label><br />
  <input id="user" type="text" placeholder="Tu nombre" /><br />

  <label>Mensaje:</label><br />
  <input id="text" type="text" placeholder="Escribe un mensaje..." /><br />

  <button onclick="sendMessage()">Enviar</button>
```

Y el script que llamará a las lambdas:

```
<script>
  const API_BASE = "https://4iqi63mxge.execute-api.us-east-1.amazonaws.com/dev"; // CAMBIAR

  async function sendMessage() {
    const user = document.getElementById("user").value;
    const text = document.getElementById("text").value;

    if (!user || !text) return alert("Usuario y mensaje son obligatorios");

    const res = await fetch(`${API_BASE}/send`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ user, text, room: "general" }),
    });

    const data = await res.json();
    console.log("Enviado:", data);
    document.getElementById("text").value = ""; // limpiar input
    loadMessages();
  }

  async function loadMessages() {
    try {
      const res = await fetch(`${API_BASE}/messages?room=general&limit=20`);
      const data = await res.json();

      const container = document.getElementById("messages");
      container.innerHTML = ""; // limpiar

      (data.messages || [])
        .sort((a, b) => a.timestamp - b.timestamp)
        .forEach((msg) => {
          const p = document.createElement("p");
          const date = new Date(msg.timestamp).toLocaleTimeString();
          p.textContent = `[${date}] ${msg.user}: ${msg.text}`;
          container.appendChild(p);
        });
    } catch (e) {
      console.error("Error cargando mensajes:", e);
    }
  }

  // refrescar cada 5 segundos
  setInterval(loadMessages, 5000);
  loadMessages(); // cargar al abrir
</script>
</body>
```

https://github.com/jorloque/chat-aws