UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS2032 - Cloud Computing (Ciclo 2024-2)
Event-driven architecture
Semana 14 - Taller 3: SQS - Simple Queue Service

ELABORADO POR: GERALDO COLCHADO

Con apoyo de Asistentes de Cátedra y Laboratorio:
- Sofía García (sofia.garcia@utec.edu.pe)
- Rubén Aaron Coorahua (ruben.coorahua@utec.edu.pe)

Contenido

Event-driven architecture

1. **Objetivo del taller 3**
2. SQS - Simple Queue Service
3. Ejercicio 1: Despacho de pedidos
4. Ejercicio 2: Ejercicio propuesto
5. Cola de Mensajes Fallidos (DLQ)
6. Ejercicio 3: DLQ
7. Cierre

Event-driven architecture
Objetivo del Taller 3

- Diseño e implementación de una Arquitectura de Solución basada en eventos con el servicio “SQS - Simple Queue Service”.

Contenido

Event-driven architecture

1. Objetivo del taller 3
2. **SQS - Simple Queue Service**
3. Ejercicio 1: Despacho de pedidos
4. Ejercicio 2: Ejercicio propuesto
5. Cola de Mensajes Fallidos (DLQ)
6. Ejercicio 3: DLQ
7. Cierre

Event-driven architecture
SQS - Simple Queue Service

Amazon Simple Queue Service

Colas de mensajes completamente administradas para microservicios, sistemas distribuidos y aplicaciones sin servidor

Amazon Simple Queue Service (Amazon SQS) es un servicio de colas de mensajes completamente administrado que permite desacoplar y ajustar la escala de microservicios, sistemas distribuidos y aplicaciones sin servidor. SQS elimina la complejidad y los gastos generales asociados con la administración y el funcionamiento del middleware orientado a mensajes, y permite a los desarrolladores centrarse en la diferenciación del trabajo. Con SQS, puede enviar, almacenar y recibir mensajes entre componentes de software de cualquier volumen, sin pérdida de mensajes ni la necesidad de que otros servicios estén disponibles. Comience a usar SQS en minutos con la consola de administración de AWS, la interfaz de línea de comandos o el SDK de AWS de su elección, y tres comandos simples.

SQS ofrece dos tipos de colas de mensajes. Las colas estándar ofrecen una capacidad de procesamiento máxima, un ordenamiento de mejor esfuerzo y una entrega al menos una vez. Las colas FIFO de SQS están diseñadas para garantizar que los mensajes se procesen exactamente una vez, en el orden exacto en el que se enviaron.

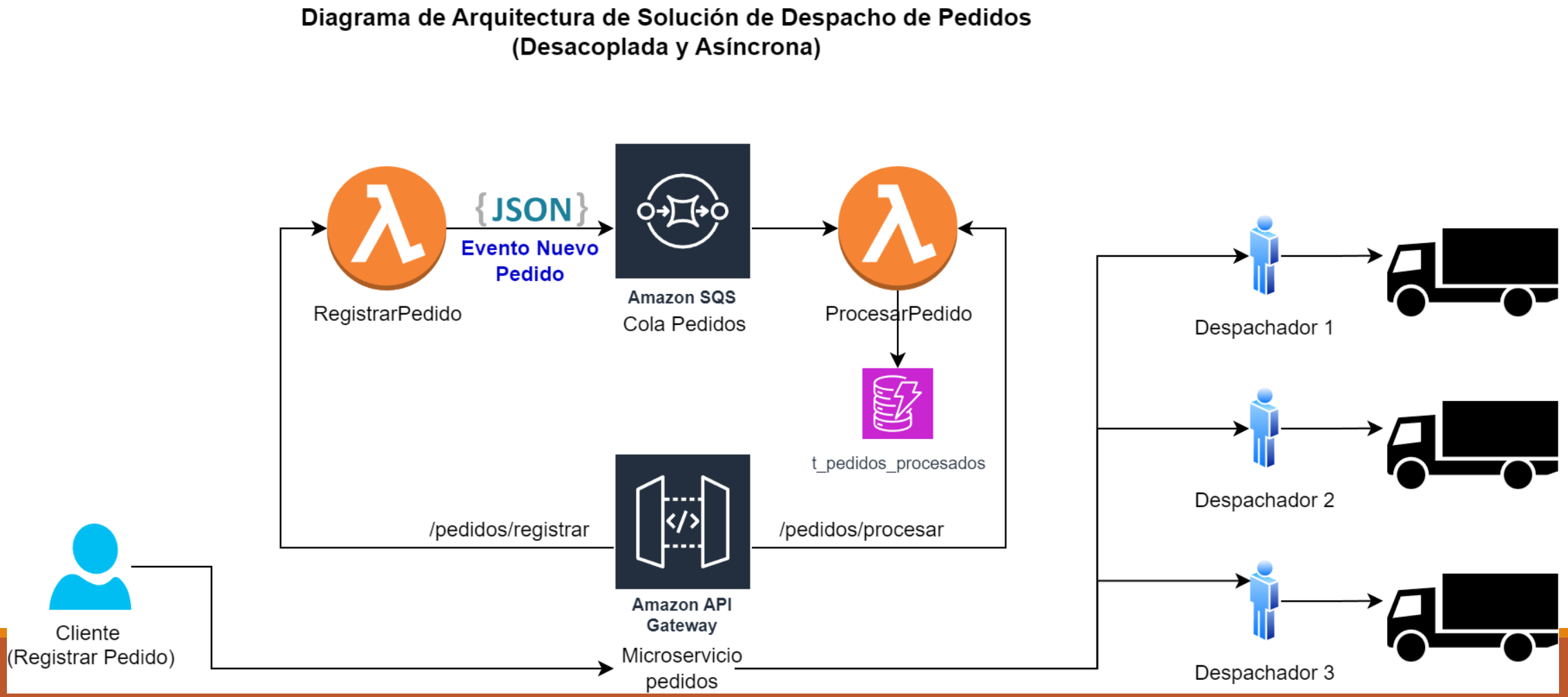
Fuente: https://aws.amazon.com/es/sqs/

Contenido

Event-driven architecture

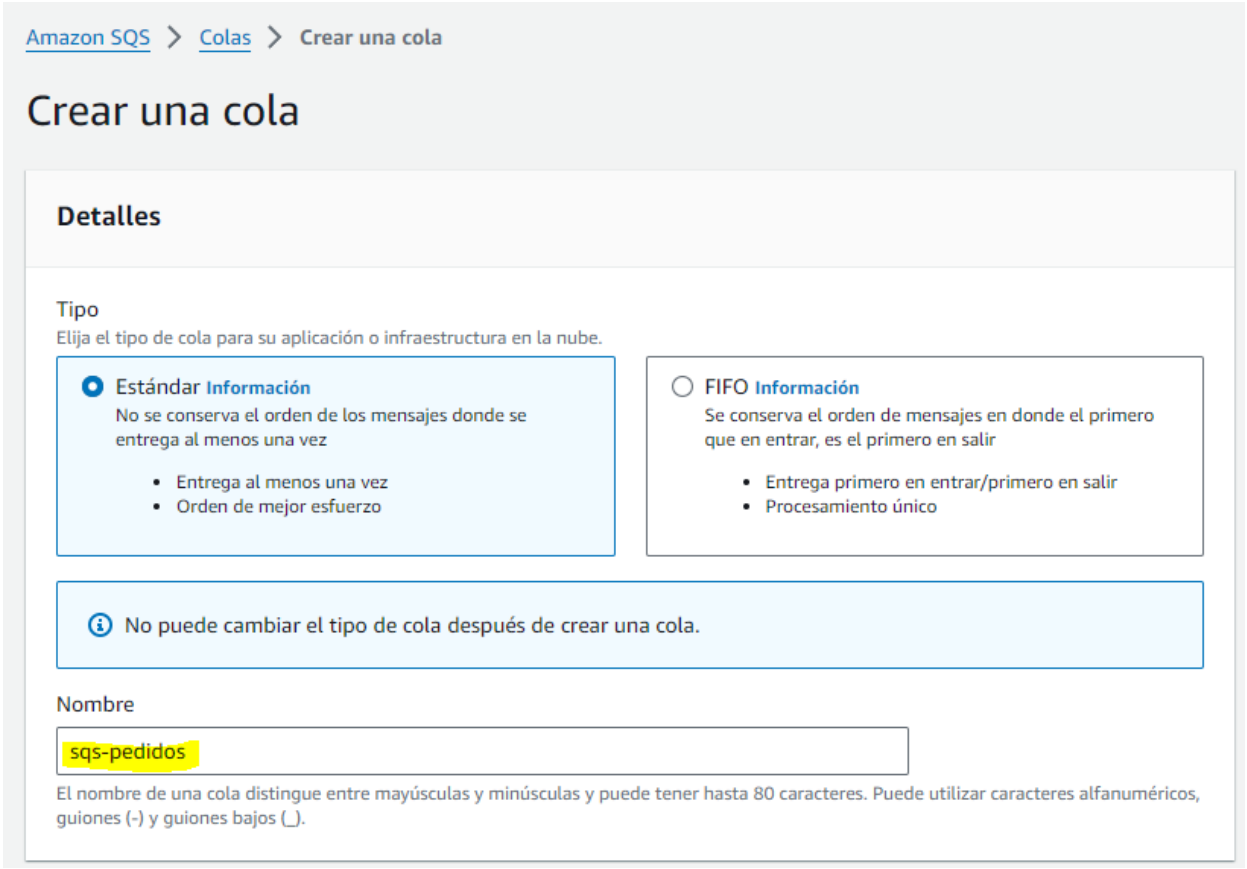
1. Objetivo del taller 3
2. SQS - Simple Queue Service
3. **Ejercicio 1: Despacho de pedidos**
4. Ejercicio 2: Ejercicio propuesto
5. Cola de Mensajes Fallidos (DLQ)
6. Ejercicio 3: DLQ
7. Cierre

Event-driven architecture
Ejercicio 1 - Despacho de Pedidos



Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

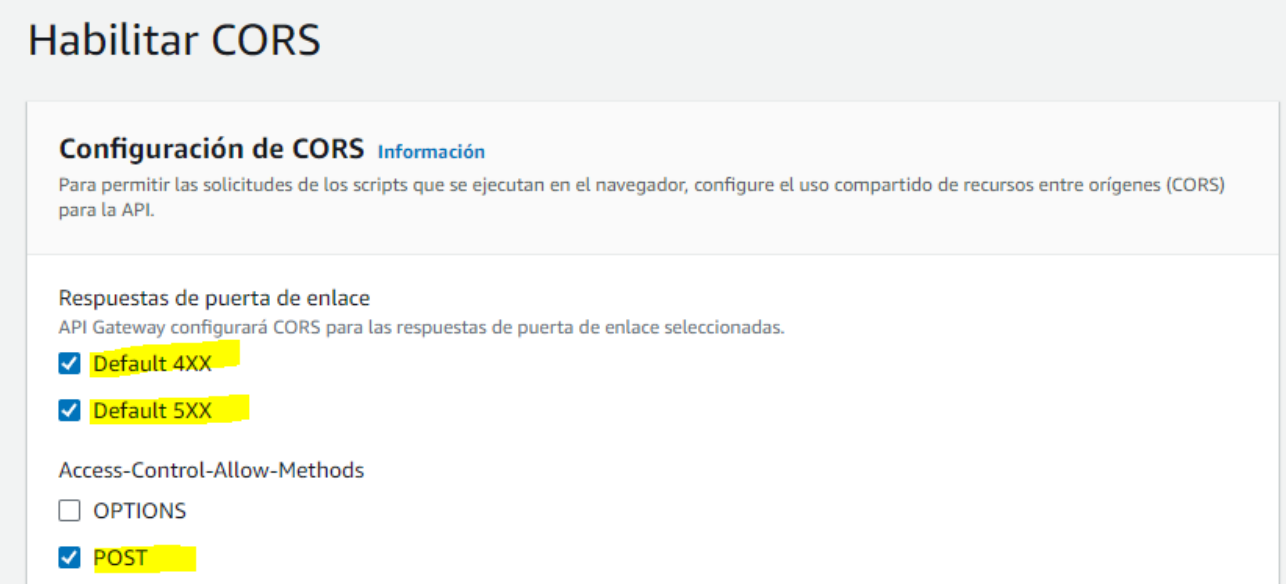
Paso 1: Crear Cola de Pedidos (sqs-pedidos)



Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

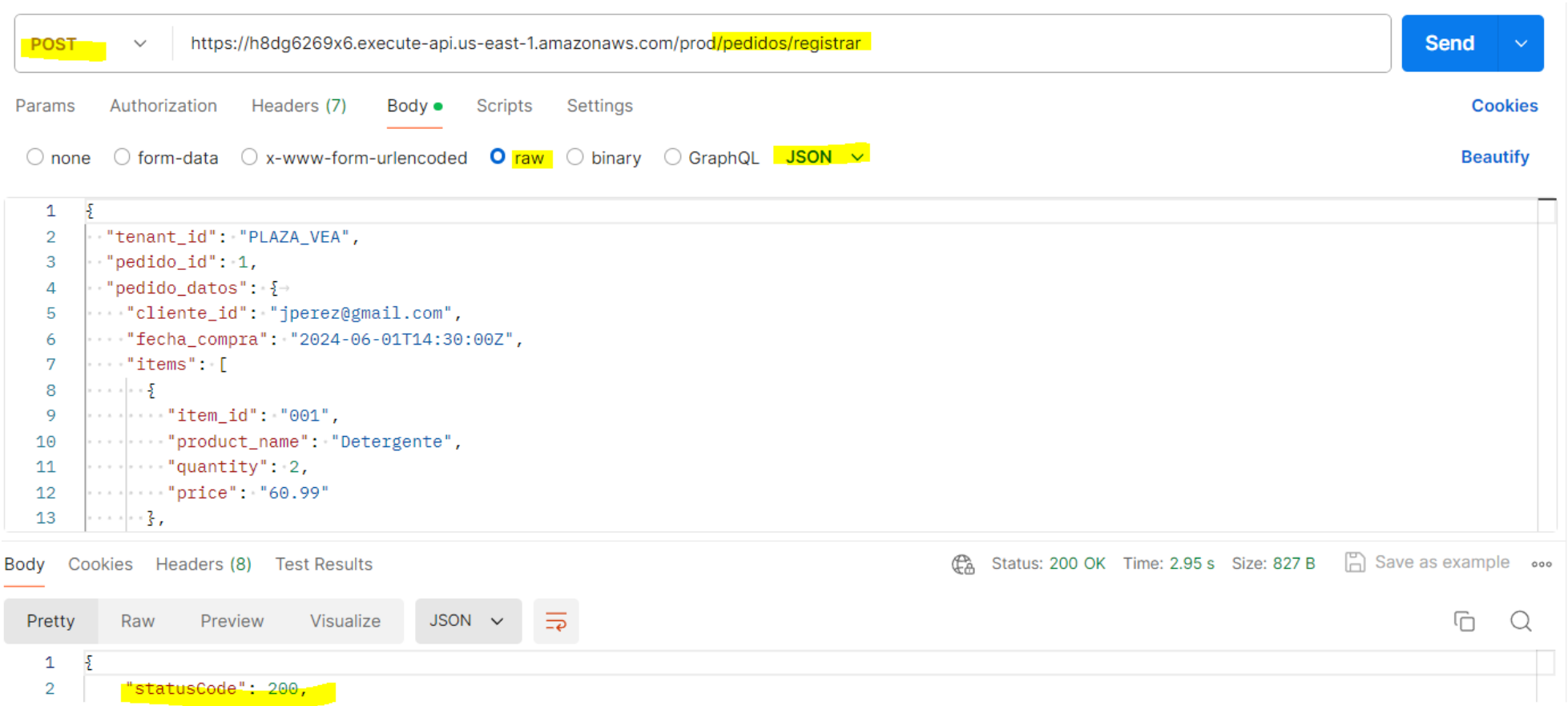
Paso 2: Crear Lambda RegistrarPedido con tiempo de espera de 30 segundos y reemplace el queue_url

Paso 3: Crear Api Gateway pedidos con recurso /pedidos/registrar con un método POST que ejecute el lambda anterior y habilite CORS



Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

Paso 4: Pruebe con postman registrar un pedido



Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

Paso 5: Verifique en la cola sqs-pedidos si llegó el mensaje

Nombre	Tipo	Creado	Mensajes disponibles	Mensajes en tránsito
sqs-pedidos	Estándar	2024-06-02T17:33-05:00	1	0

Mensaje: 78ce5045-ba71-4b9f-ac66-457bd2ca3474

Cuerpo | Atributos | Detalles

{"tenant_id": "PLAZA_VEA", "pedido_id": 1, "pedido_datos": {"cliente_id": "jperez@gmail.com", "fecha_compra": "2024-06-01T14:30:00Z", "items": [{"item_id": "001", "product_name": "Detergente", "quantity": 2, "price": "60.99"}, {"item_id": "002", "product_name": "Lavavajilla", "quantity": 1, "price": "10.35"}], "direccion_entrega": {"calle": "Avenida Los Frutales 123", "distrito": "La Molina", "provincia": "Lima", "pais": "Per\u00fa"}}}

Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

Paso 6: Registre los otros 4 mensajes por postman

Nombre	Tipo	Creado	Mensajes disponibles	Mensajes en tránsito
sqs-pedidos	Estándar	2024-06-02T17:33-05:00	5	0

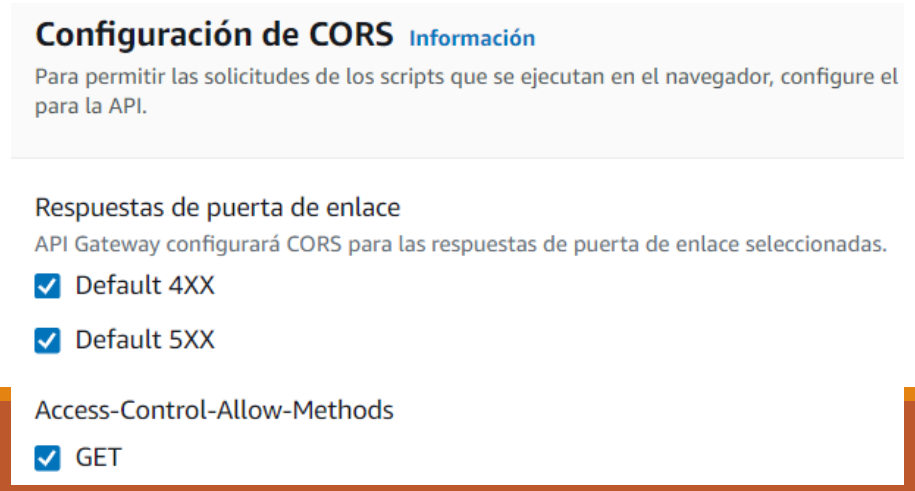
Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

Paso 7: Crear tabla DynamoDB t_pedidos_procesados

Nombre ▲	Estado	Clave de partición	Clave de ordenación
t_pedidos_procesados	Activo	tenant_id (S)	pedido_id (N)

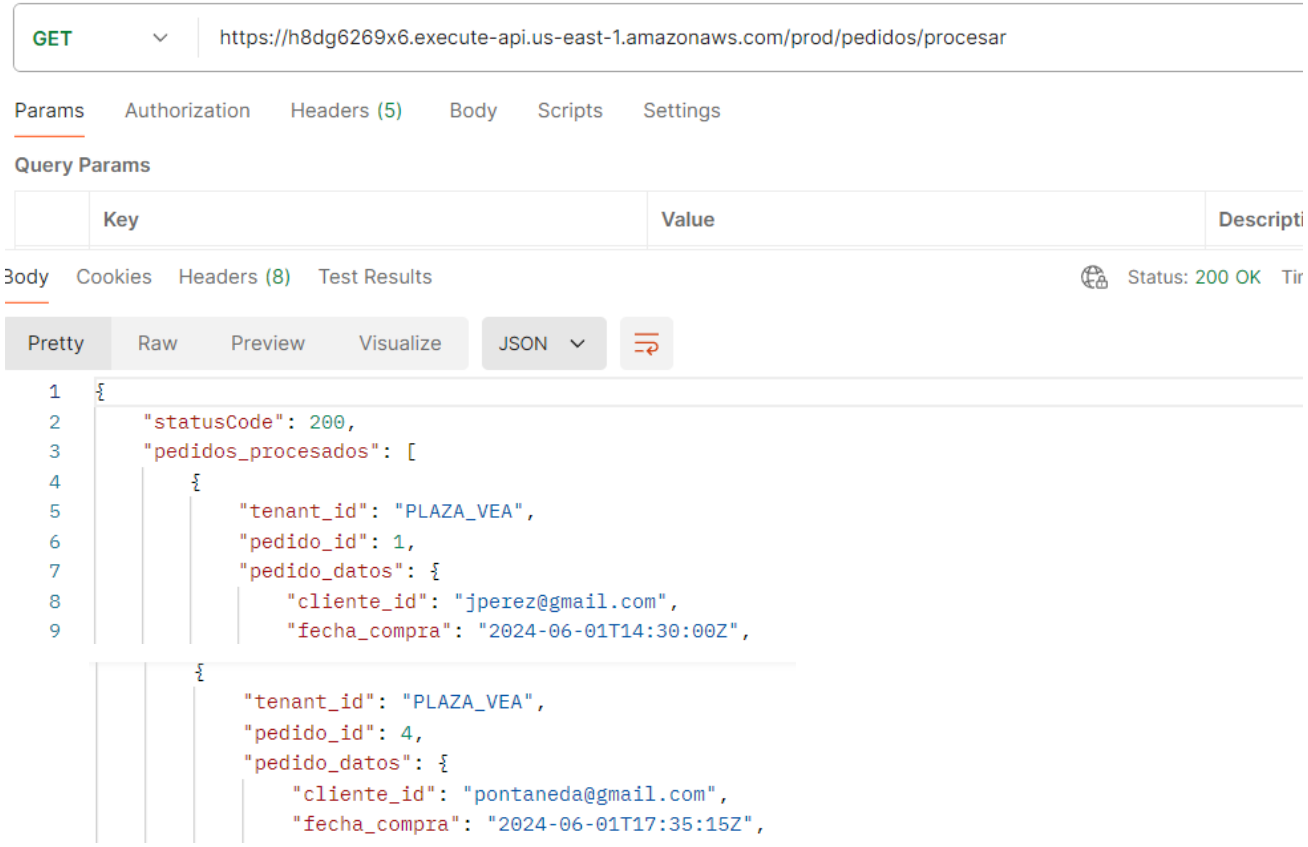
Paso 8: Crear lambda ProcesarPedido con tiempo de espera de 30 segundos

Paso 9: En Api Gateway pedidos crear recurso /pedidos/procesar con un método GET que ejecute el lambda anterior y habilite CORS



Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

Paso 10: Ejecute el api en postman simulando que es el Despachador 1



Event-driven architecture
Ejercicio 1 - Despacho de Pedidos

Paso 11: Verifique si se grabaron los pedidos en tabla

tenant_id *(Cadena)*	pedido_id *(Número)*	pedido_datos
PLAZA_VEA	1	{ "direccion_entrega" : { "M" : { "distrito" : { "S" : "La Molina" }, "provir
PLAZA_VEA	4	{ "direccion_entrega" : { "M" : { "distrito" : { "S" : "San Isidro" }, "provir

Paso 12: Ejecute el api hasta procesar todos los 5 pedidos simulando que es el Despachador 2 y Despachador 3 y analice

Contenido

Event-driven architecture

1. Objetivo del taller 3
2. SQS - Simple Queue Service
3. Ejercicio 1: Despacho de Pedidos
4. **Ejercicio 2: Ejercicio propuesto**
5. Cola de Mensajes Fallidos (DLQ)
6. Ejercicio 3: DLQ
7. Cierre

Event-driven architecture
Ejercicio 2 - Propuesto

- Modifique el lambda ProcesarPedido para que reciba como entrada un despachador_id (“DESP-01”, “DESP-02”, “DESP-03”) y este se grabe como campo en la tabla DynamoDB t_pedidos_procesados para identificar el despachador que procesó los pedidos. Debe modificar el método a POST en Api Gateway en pedidos/procesar.

- En la respuesta del lambda ProcesarPedido agregue un campo con la cantidad de pedidos procesados

```
{
        "statusCode": 200,
        "cantidad_pedidos_procesados": 3,
        "pedidos_procesados": [
        ...
```

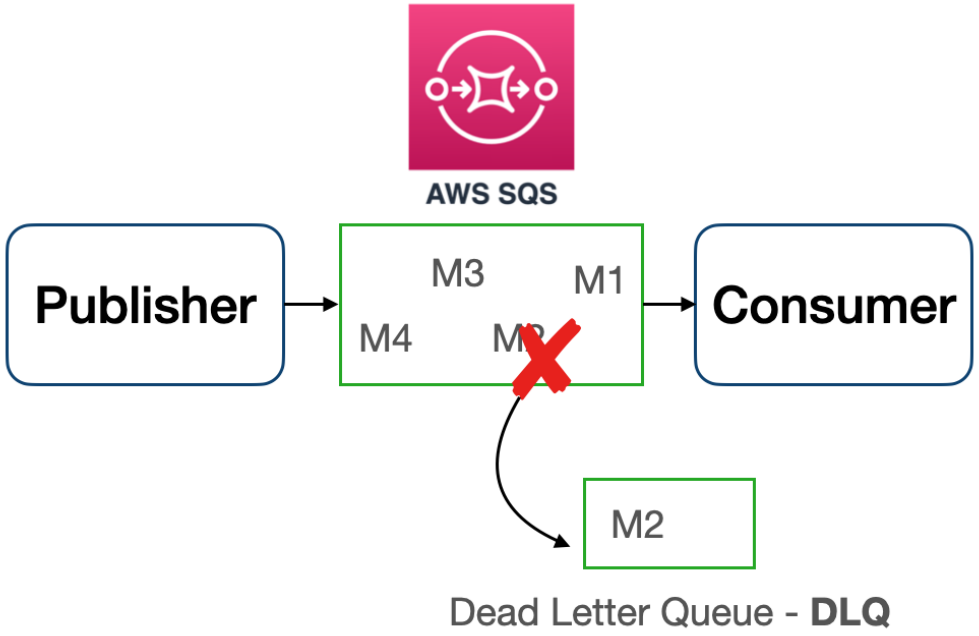
Contenido

Event-driven architecture

1. Objetivo del taller 3
2. SQS - Simple Queue Service
3. Ejercicio 1: Despacho de Pedidos
4. Ejercicio 2: Ejercicio propuesto
5. **Cola de Mensajes Fallidos (DLQ)**
6. Ejercicio 3: DLQ
7. Cierre

Event-driven architecture
Cola de Mensajes Fallidos (DLQ - Dead Letter Queue)

"Una cola de mensajes fallidos (DLQ) es un tipo especial de cola de mensajes que almacena temporalmente los mensajes que un sistema de software no puede procesar debido a errores."



Fuente: https://aws.amazon.com/es/what-is/dead-letter-queue/, https://siecola.com.br/blogs/aws_sqs_dlq.html

Contenido

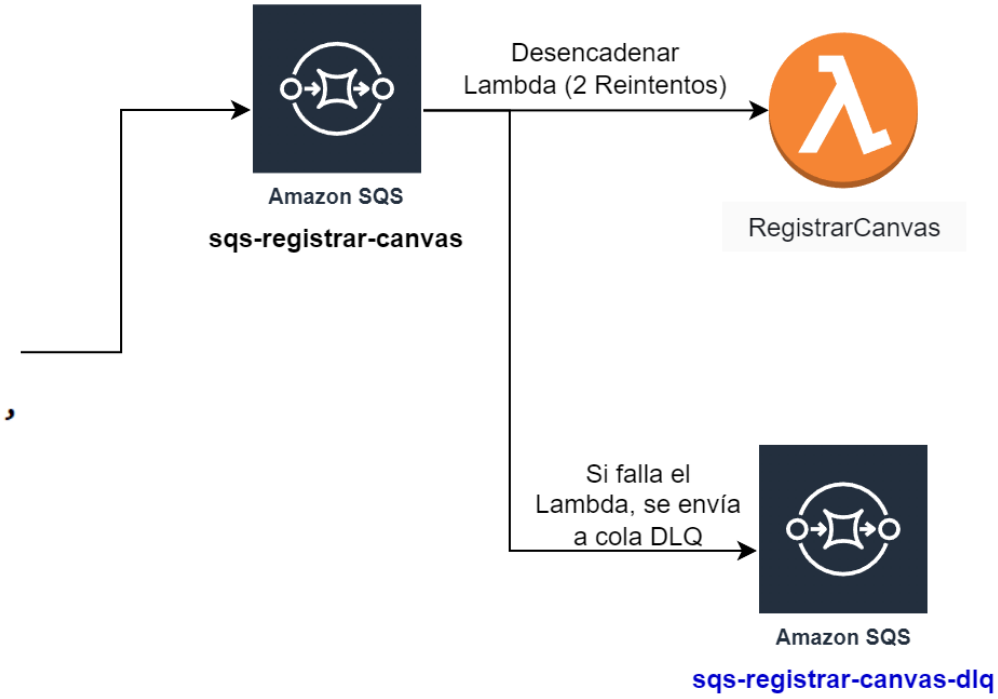
Event-driven architecture

1. Objetivo del taller 3
2. SQS - Simple Queue Service
3. Ejercicio 1: Despacho de Pedidos
4. Ejercicio 2: Ejercicio propuesto
5. Cola de Mensajes Fallidos (DLQ)
6. **Ejercicio 3: DLQ**
7. Cierre

Event-driven architecture
Ejercicio 3 - DLQ

Agregar una cola de mensajes fallidos (DLQ) en caso falle el registro de nuevo alumno en canvas

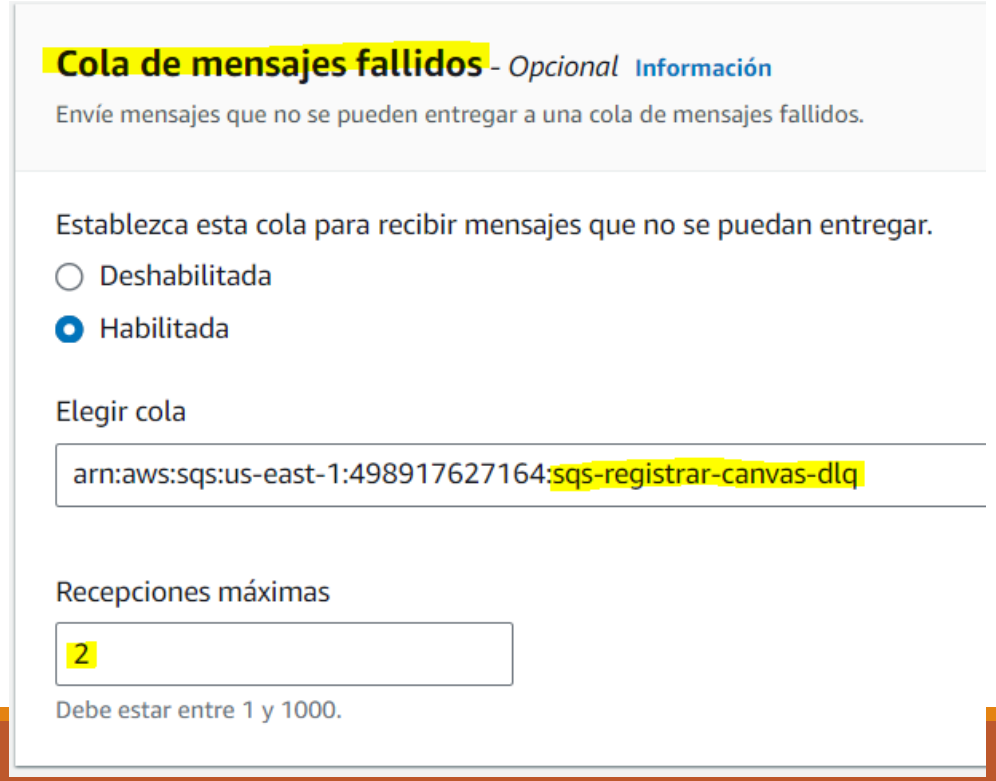
```
{
  "tenant_id": "UTEC",
  "alumno_id": "202310295",
  "alumno_datos": {
        "nombre": "Claudia Espinoza",
        "sexo": "M",
        "fecha_nac": "2004-12-04",
        "celular": "999736371",
        "domicilio": {
              "direcc": "Av. El Polo 1767",
              "distrito": "Monterrico",
              "provincia": "Lima",
              "departamento": "Lima",
              "pais": "Perú"
        }
  }
}
```



Event-driven architecture
Ejercicio 3 - DLQ

- ***Paso 1:*** *Cree una cola de mensajes fallidos sqs-registrar-canvas-dlq*

- ***Paso 2:*** *Cree una cola sqs-registrar-canvas y configure la cola de mensajes fallidos para que sea enviado el mensaje luego de 2 reintentos fallidos de procesarlo.*



Event-driven architecture
Ejercicio 3 - DLQ

- ***Paso 3:*** *Cree un lambda RegistrarCanvas que genere una excepción (simulando que falla)*

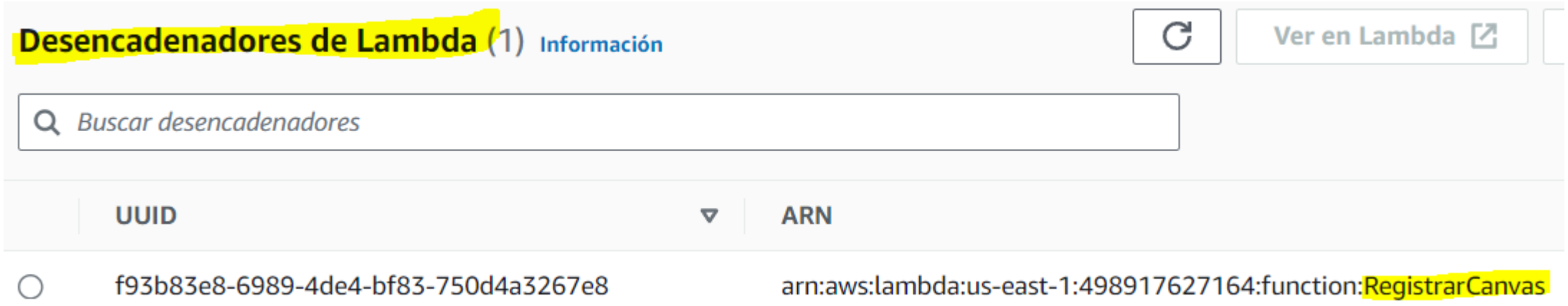
```python
import json

def lambda_handler(event, context):
    # TODO implement
    print(event)
    raise Exception("Error")

    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

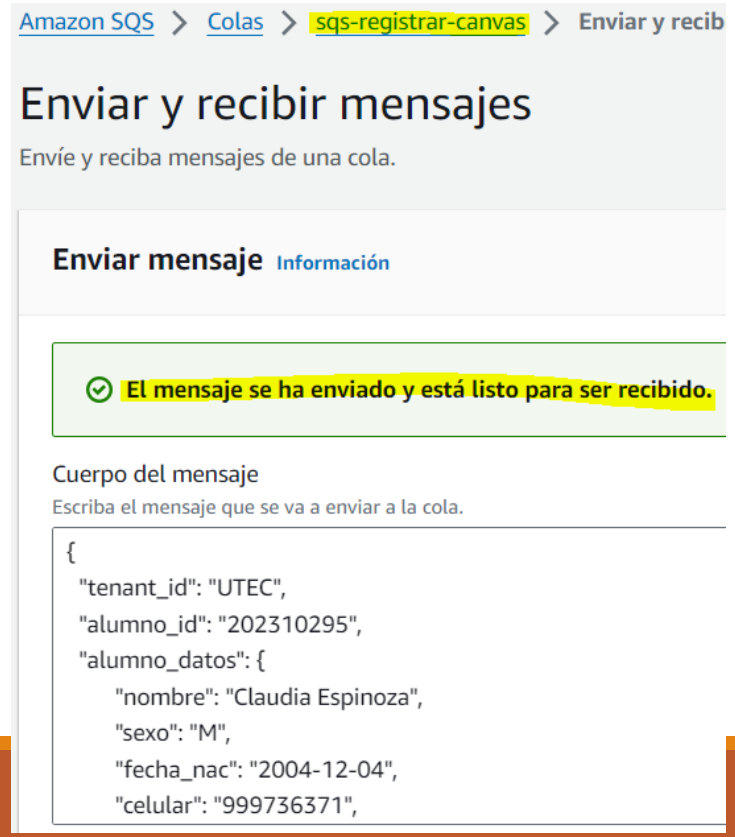
Event-driven architecture
Ejercicio 3 - DLQ

- ***Paso 4:*** *En la cola sqs-registrar-canvas configure un desencadenador de lambda a RegistrarCanvas para que procese el mensaje*



Event-driven architecture
Ejercicio 3 - DLQ

- ***Paso 5:*** *Envíe un mensaje en la cola sqs-registrar-canvas y valide los 2 reintentos con error en CloudWatch (cada 30 segundos) y que se genere un mensaje en cola sqs-registrar-canvas-dlq*



Nombre	Tipo	Creado	Mensajes disponibles	Mensajes en tránsito
sqs-registrar-canvas	Estándar	2024-06-09T17:25-05:00	0	1
sqs-registrar-canvas-dlq	Estándar	2024-06-09T17:25-05:00	0	0

Nombre	Tipo	Creado	Mensajes disponibles	Mensajes en tránsito
sqs-registrar-canvas	Estándar	2024-06-09T17:25-05:00	0	0
sqs-registrar-canvas-dlq	Estándar	2024-06-09T17:25-05:00	1	0

Contenido

Event-driven architecture

1. Objetivo del taller 3
2. SQS - Simple Queue Service
3. Ejercicio 1: Despacho de Pedidos
4. Ejercicio 2: Ejercicio propuesto
5. Cola de Mensajes Fallidos (DLQ)
6. Ejercicio 3: DLQ
7. **Cierre**

Cierre:
Event-driven architecture - Qué aprendimos?

- Diseño e implementación de una Arquitectura de Solución basada en eventos con el servicio “SQS - Simple Queue Service”.

Gracias

Elaborado por docente: Geraldo Colchado