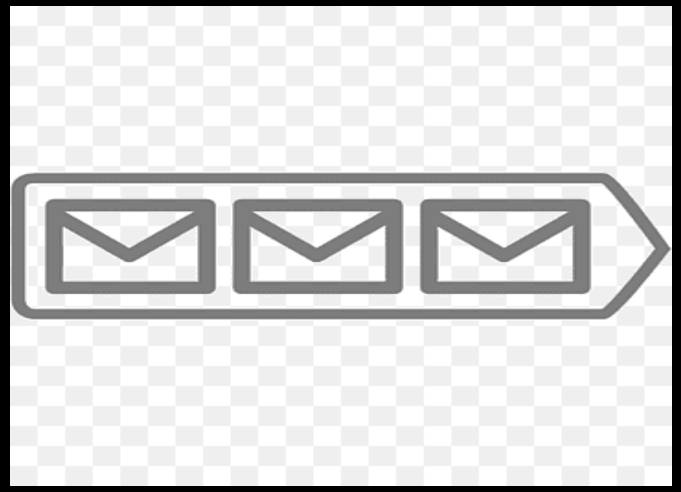
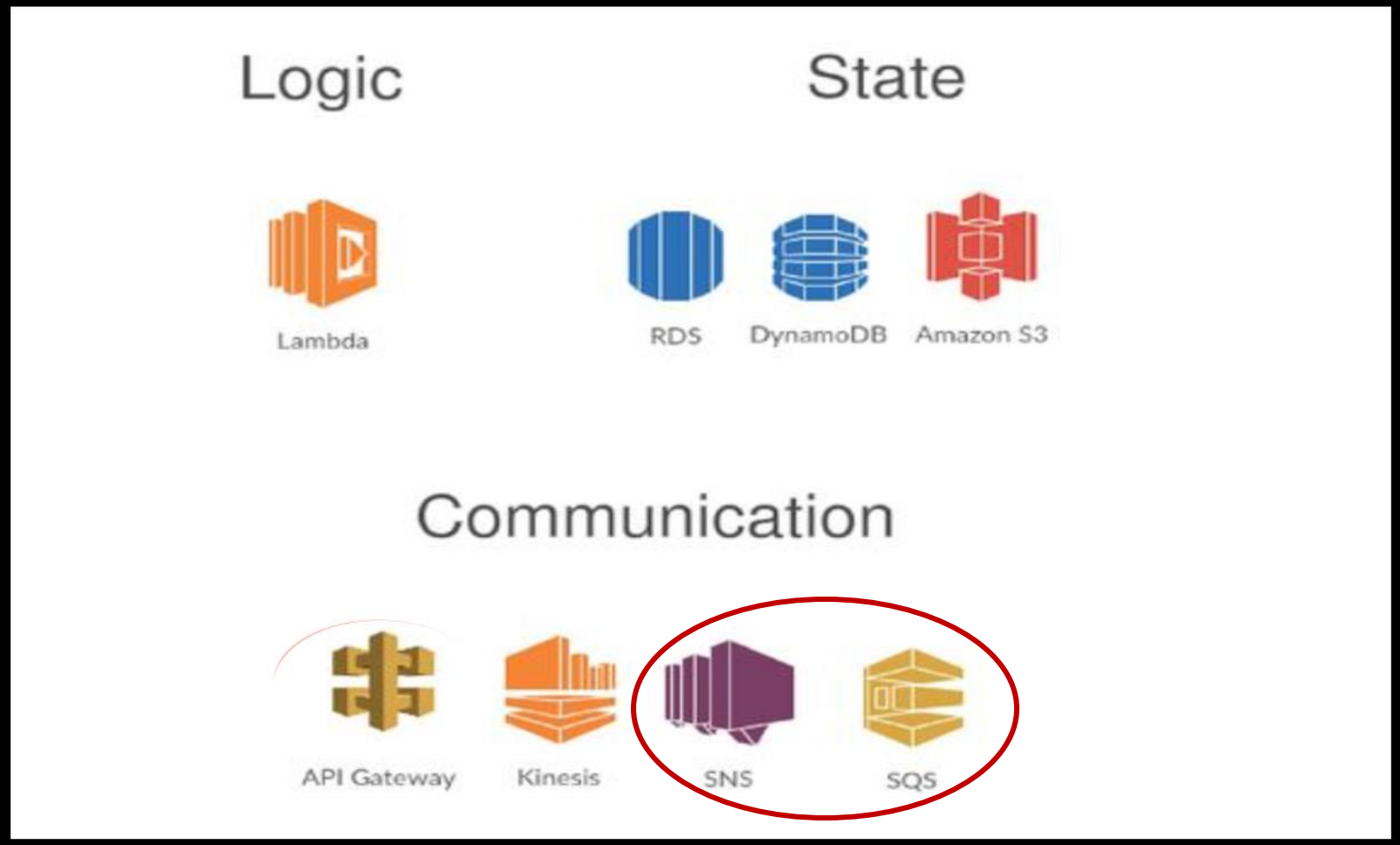


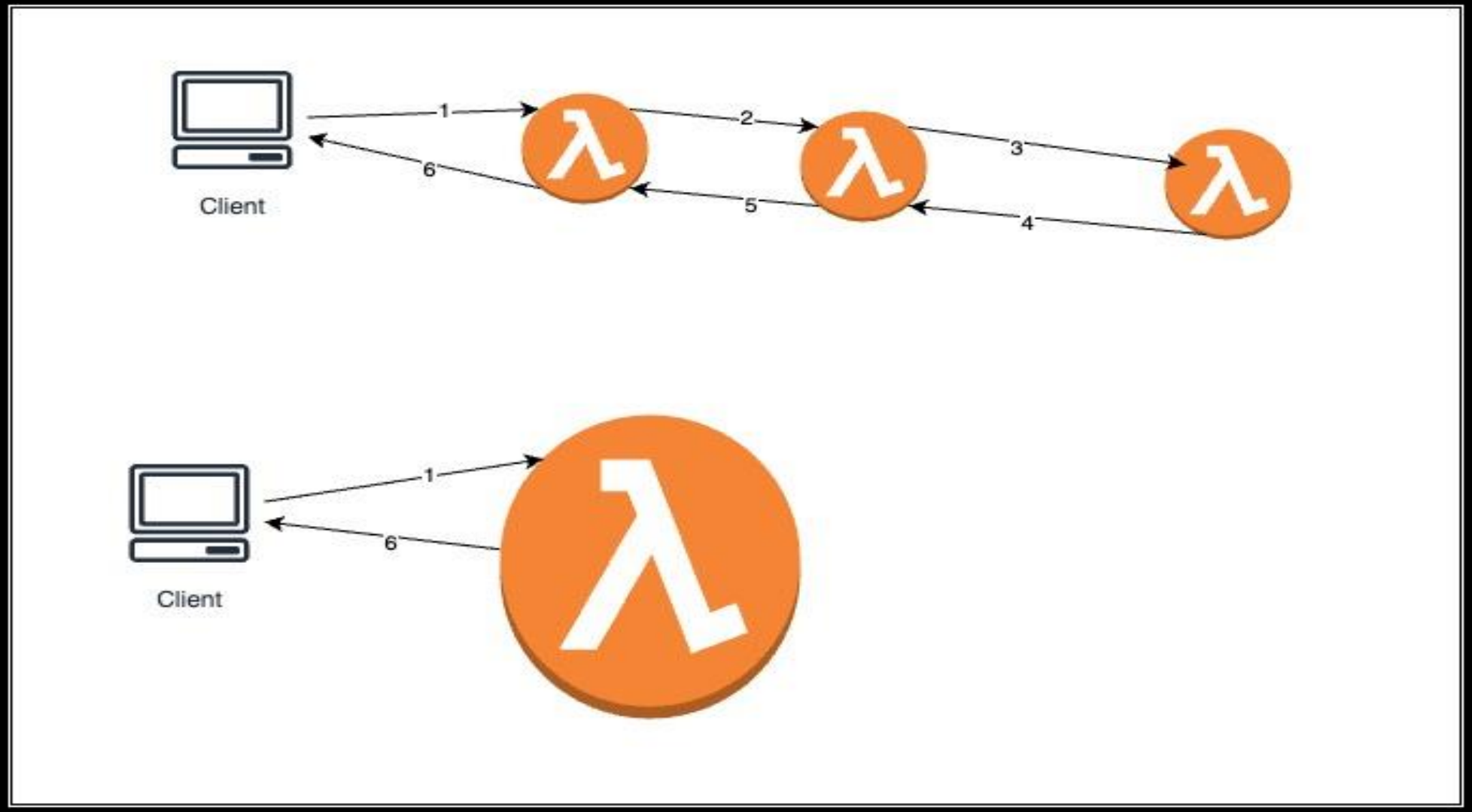
AWS Integration and Messaging Services.

Components of a Serverless, Message-Driven application

(aka Event Driven Architecture - EDA)



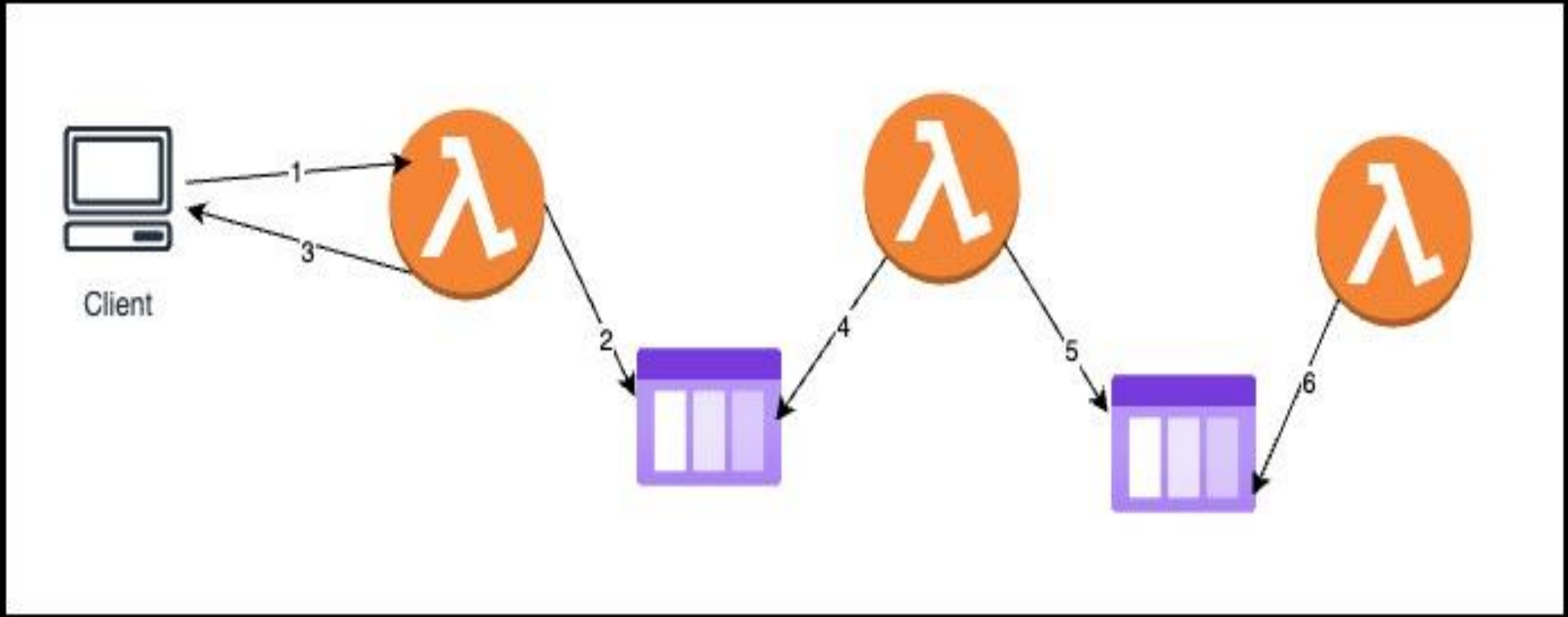
Why do we need Messaging Services?

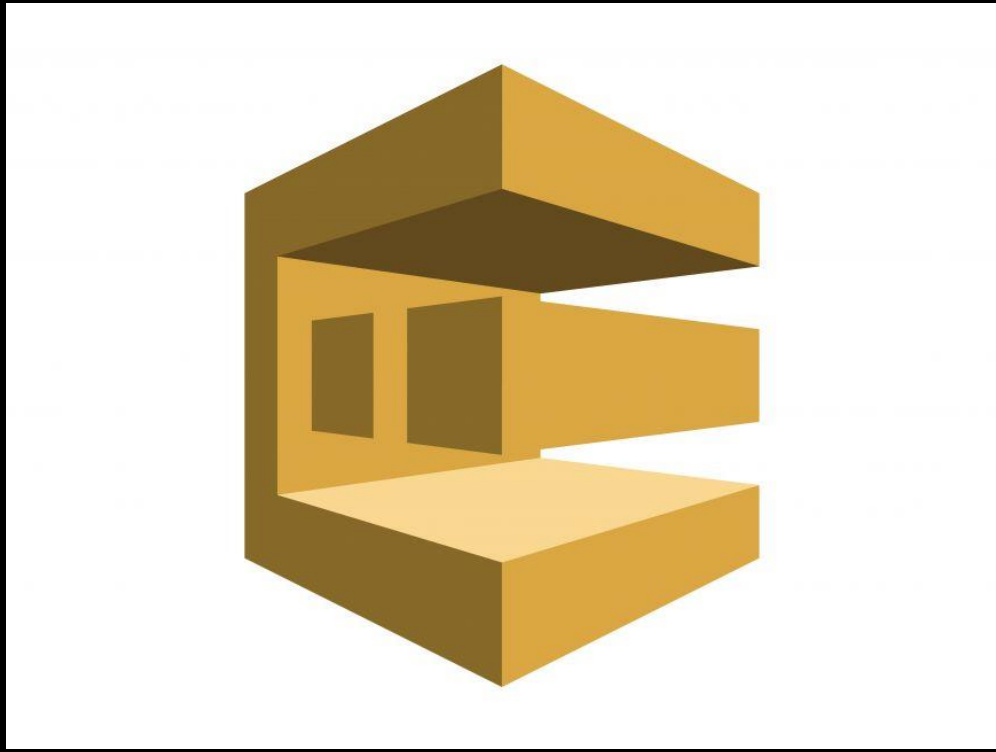


Why do we need Messaging Services?

- Synchronous communication between compute components (Lambdas, EC2 instance) can be problematic if there are sudden spikes in demand or gaps in availability.
  - E.g. 1000 parallel requests to encode video uploads, when usually the workload is a much smaller scale (10s).
- It's better to decouple compute components by using messaging intermediaries.
- AWS messaging services/techniques:
  - SQS: queueing model.
  - SNS: publisher-subscribe model.
  - Data streams.
- These techniques result in:
  - Reduced latency; Increased availability; Reduced complexity (by decreasing dependency).

How to use Messaging Services?

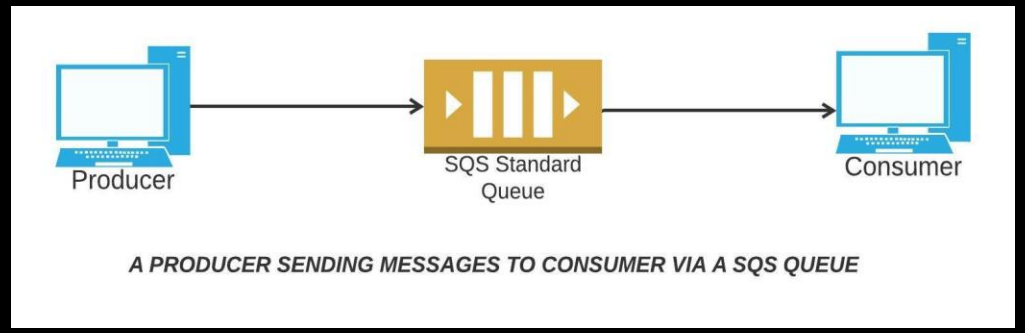


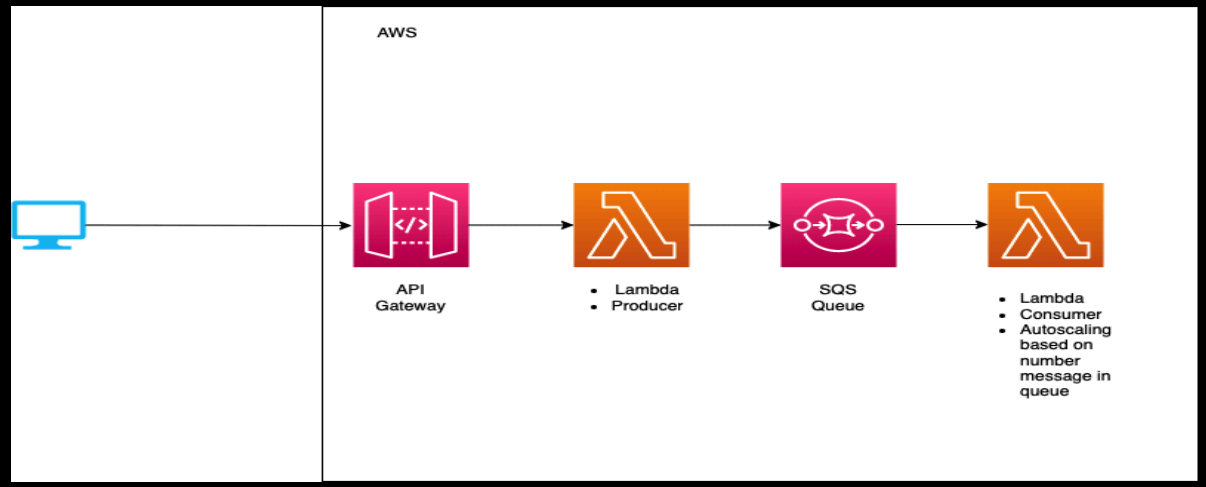


Simple Queue Service (SQS)

SQS - Overview

- Oldest AWS offering (2006).
- Fully managed, distributed queueing service, used to decouple applications/components.
- Compute component roles- Producers and Consumers.
-





SQS - Overview

- SQS Attributes:
  - Scalability - Unlimited throughput, unlimited number of messages in a queue.
  - Message Retention: 4 days (default), maximum of 14 days.
  - Low latency (< 10 msgs on publish and receive).
  - Limitation of 256KB per message.

- Caveats:
  - Duplicate messages may occur, occasionally.
    - So, consumer processing must be idempotent.
  - Message order not guaranteed (best-effort ordering).

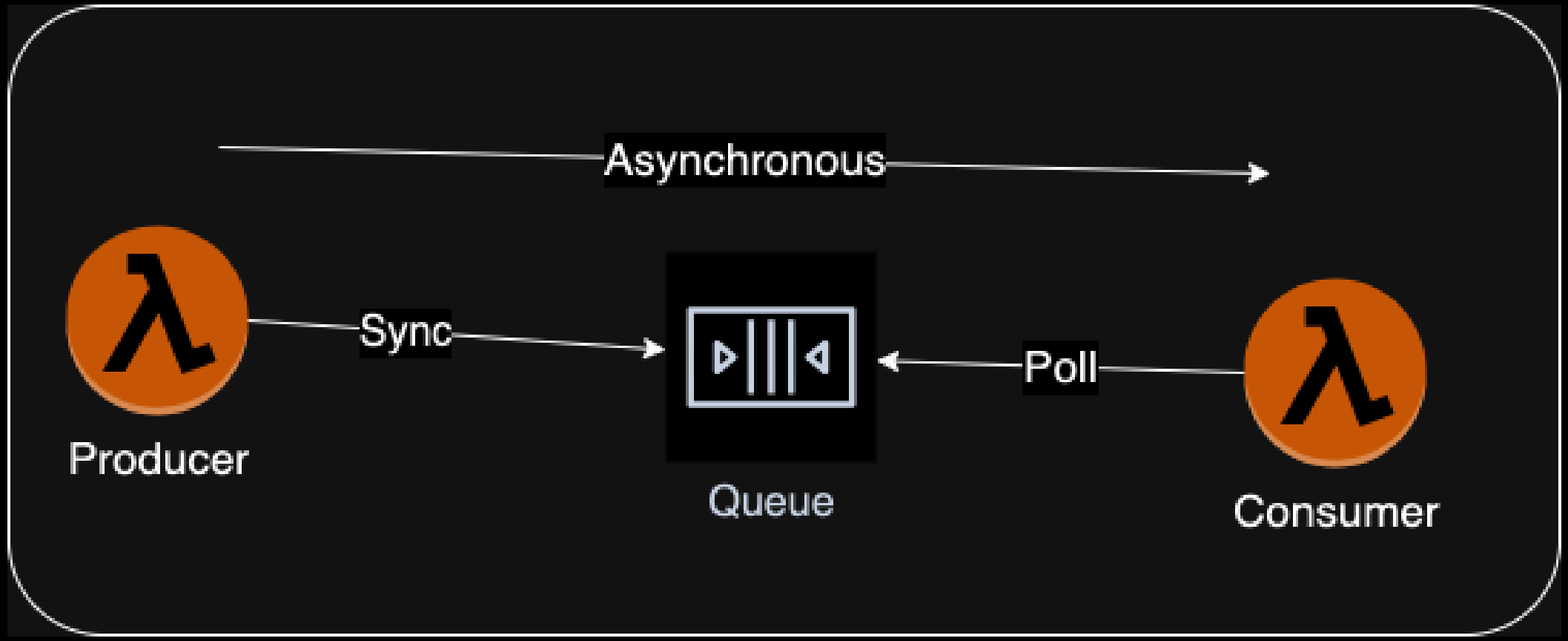
Basic Operations.

- Producer:
  - Publish/Write message to a queue using SQS SDK.
    - SQS persists messages until (a) a consumer deletes it, or (b) its TTL expires (default 4 days).
    - e.g. Publish an order to a queue for processing.

      Message = Order id + Customer id + Order details

- Consumer:
  1. Polls SQS for messages.
  2. Receives batch response (<= 10 messages).
  3. Process the message batch, e.g. validate & insert order into a d/b.
  4. Delete the message batch using the SQSSDK.

Communication styles.



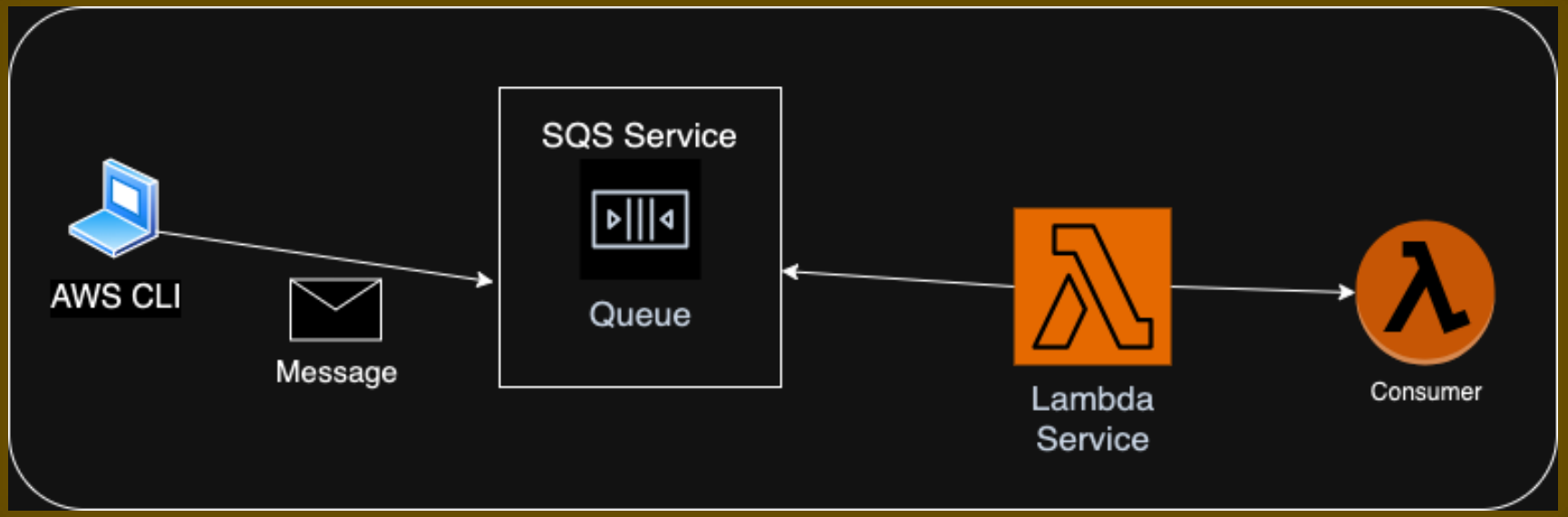
Security

- Encryption.
  - In-flight encryption using HTTPS.
  - At-rest encryption using KMS keys.

- Access Controls: IAM policies to regulate access to the SQS API.

- SQS Access Policies (similar to S3 bucket policies).
  - Useful for cross-account access to SQS queues.
  - Useful for allowing other services (SNS, S3...) to write to an SQS queue.

Demo.



- The Lambda service polls the SQS service for messages and calls the lambda function synchronously with a batch.
- If the function processes the batch without a failure/exception, the the Lambda service deletes the batch from the queue.
    - Otherwise, the entire batch remains in the queue for reprocessing by the function/consumer.

Demo - CDK Infrastructure.

```
254
255  const demoQueue = new Queue(this, "Demo Queue");
256
257  const qConsumerFn = new NodejsFunction(this, "SQSConsumerFn", {
258    architecture: Architecture.ARM_64,
259    runtime: Runtime.NODEJS_16_X,
260    entry: `${__dirname}/../lambdas/consumeQMessages.ts`,
261    timeout: Duration.seconds(10),
262    memorySize: 128,
263  });
264
265  const eventSource = new SqsEventSource(demoQueue);
266  qConsumerFn.addEventSource(eventSource)
267
268  new CfnOutput(this, "Queue Url", { value: demoQueue.queueUrl });
269
```

- Recall, lambda functions are triggered by an event.
- Here, the event source is a message queue polled by the Lambda service.

Demo - Producer & Consumer.

```
274
275  import { SQSHandler } from "aws-lambda";        You, 1 second ago • Ur
276
277  export const handler: SQSHandler = async (event) => {
278    try {
279      console.log("Event: ", JSON.stringify(event));
280      for (const record of event.Records) {
281        console.log("Message:  ", record.body);
282      }
283    } catch (error) {
284      console.log(JSON.stringify(error));
285    }
286  };
287
```

Batch

AWS CLI

```
$ aws sqs send-message
  --queue-url https://sqs.eu-west-1.amazonaws.com/517039770760/
  Demo-Stk-DemoQueueA7C0530A-FdRfEAcHwKCH
  --message-body "Hello world."
```

Demo – Lambda Consumer event structure.

2023-06-28T12:53:04.585+01:00 2023-06-28T11:53:04.585Z a42c47a1-f485-50e4

2023-06-28T11:53:04.585Z a4 INFO


```
{
    "Records": [
        {
            "messageId": "5ff8fa2a
            "receiptHandle":
"AQEBUDHPaBk73afcgiPcPuhPMREAduRbP sxf4s62e
Nl8d+6vomnM+vLO5NIcDadoSC8Ec9N0qfT K/IKnbbo
gejUk8cVdRTeivhXwJX8H7YZuWxlbGtxwQ L697dltZ
yXgpKAPxii2EvFMk8EfT2ORw9v0FISkl7m Jx3Lr4FK
XqxbriUBc2h57U3p0ycI=",
            "body": "Hello world."
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1687953184080",
                "SenderId": "AIDAXQYPYZSEFH75QIS7P",
                "ApproximateFirstReceiveTimestamp": "1687953184082"
            },
            "messageAttributes": {},
            "md5OfBody": "764569e58f53ea8b6404f6fa7fc0247f",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:eu-west-1:517039770760:Demo-Stk-Demo
FdRfEAcHwKCH",
            "awsRegion": "eu-west-1"
        }
    ]
}
```


CloudWatch log of lambda function event containing batch of message from queue.

Demo - JSON messages.

- SQS serialize JSON messages → Handler must parse it before processing.

```
2023-10-27T09:28:19.121Z        a2694ebd-51c6-530d-ad35-1308d52603b5    INFO    Even
{
    "Records": [
        {
            "messageId": "52c0079d-9f7f-406c-8584-bc9eec46e39f",
            "receiptHandle":
"AQEBBhJ2+J2W0pmbeb6aK5AvfKM8ERAW3P9bJCsCPK8DoIoMeGYjh+uWaXKtch/pD4/PQbbGwwy7k6S9Ifc
o2f1K5f9ojM51H3KrzwAFlHzMg87gAkgY0xnDjjGMrZd+Hdwk+Rd7HaQsquveUw2voJYe0+0abdwM6lEiEGc
0uxsBv29C+TOYvAWVA1LDf7GMFkb86OeMusWxJZLk+t+XTKrI3B9ghfrS3z/7tHxao+4GGn+nbmNBVv496HC
/c2zsFTkhggIgWwS56HFopf8JZyu+IcLMteheaPFJAhmjGUVfTVXwjLSSOFNpXvH8dOUz95SfItdY9MFI2qh
9joQWhFTW5uF/F4tf+LF=",
            "body": "{\n    \"name\" : \"Diarmuid O' Connor\",\n    \"address\" : \"1 Main Street\",\n
  \"email\": \"doconnor@wit.ie\"\n}",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1698398898745",
                "SenderId": "AIDAXQYPYZSEFH75QIS7P",
                "ApproximateFirstReceiveTimestamp": "1698398898750"
            },
            "messageAttributes": {},
            "md5OfBody": "85f8fd703039e25159f4268695f0cd5f",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:eu-west-1:517039770760:Demo-Stk-DemoQueueA7C0530A-
bQ8NgZV2f7bP",
            "awsRegion": "eu-west-1"
        }
    ]
}
```

```
{} message.json > ...
1  {
2      "name" : "Diarmuid O' Connor",
3      "address" : "1 Main Street",
4      "email": "doconnor@wit.ie"
5  }
```

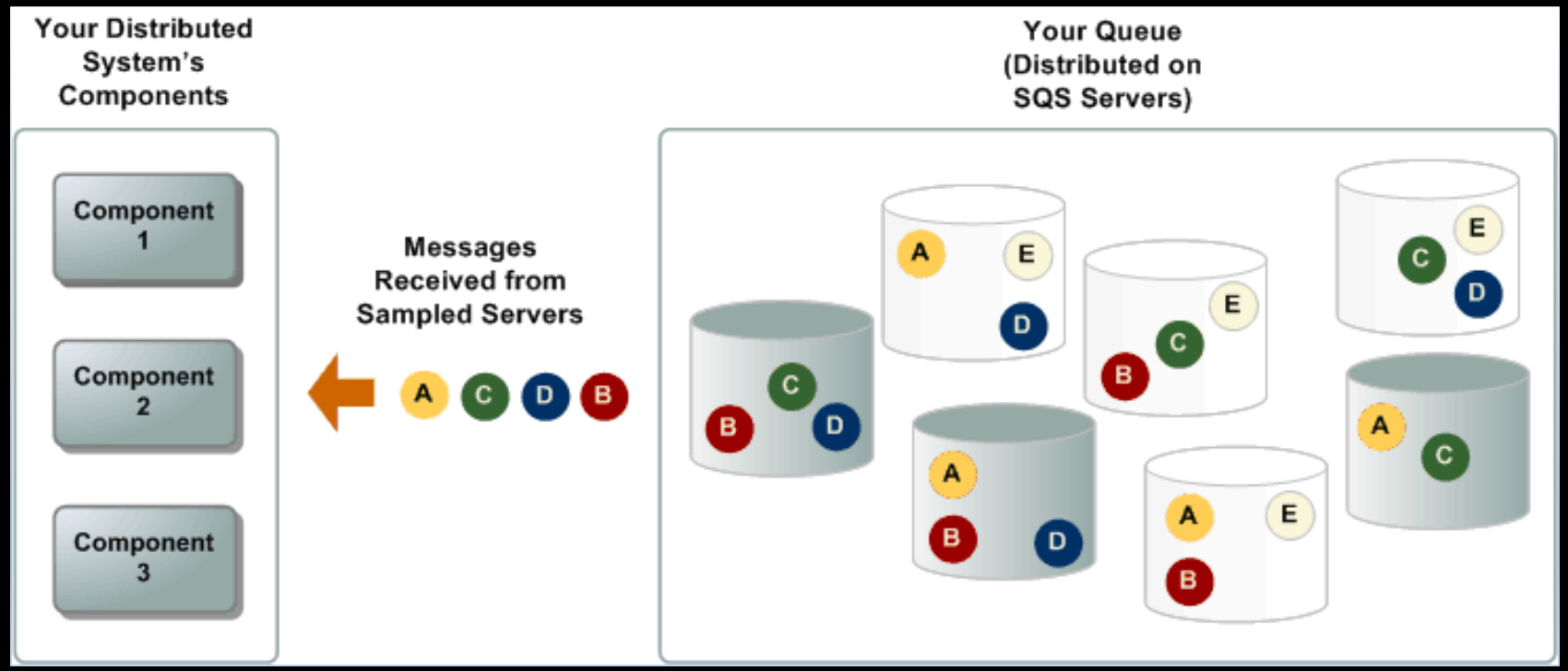
```
$ aws sqs send-message --queue-url <queue-url> --message-body file://./message.json
```

Demo - JSON messages.

- The lambda handler (Consumer)

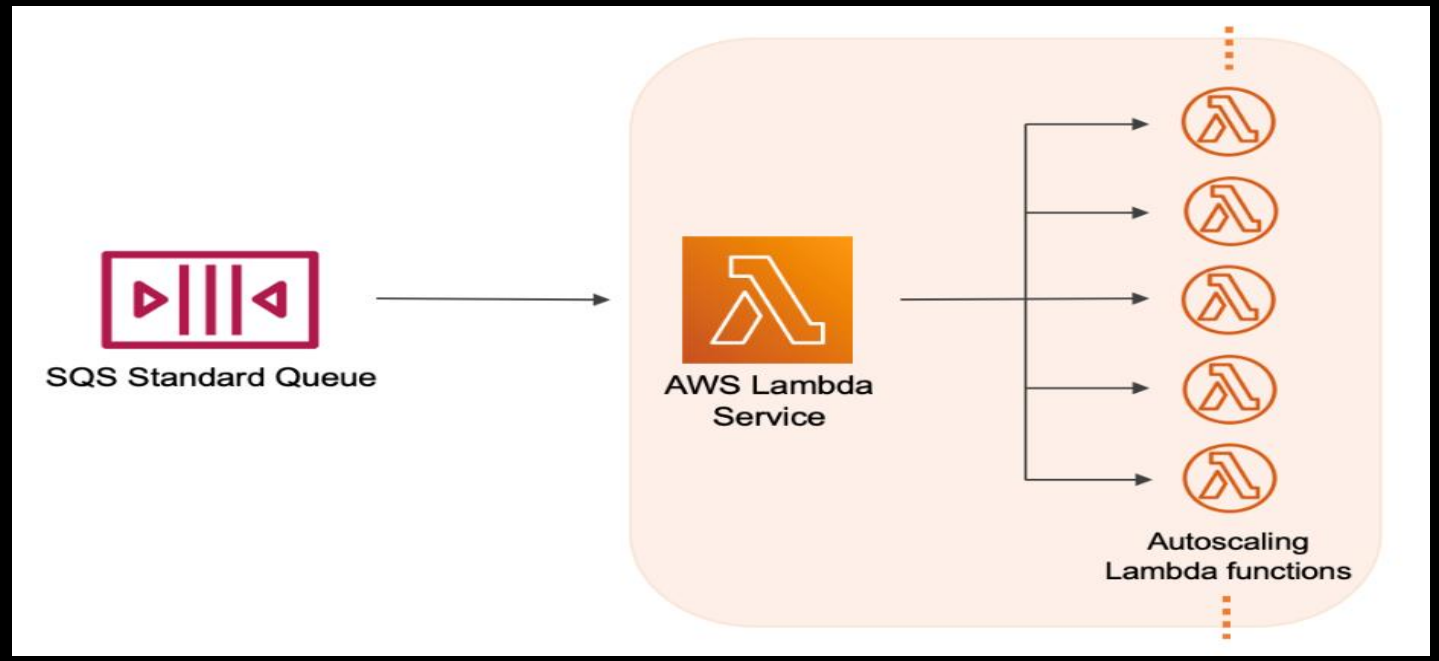
```
You, 16 seconds ago | 1 author (You)
import { SQSHandler } from "aws-lambda";

export const handler: SQSHandler = async (event) => {
  try {
    console.log("Event: ", event);
    for (const record of event.Records) {
      const message = JSON.parse(record.body)
      const {name, address } = message
      console.log(name,address);
    }
  } catch (error) {
    console.log(JSON.stringify(error));
  }
};
```

SQS is Resilient



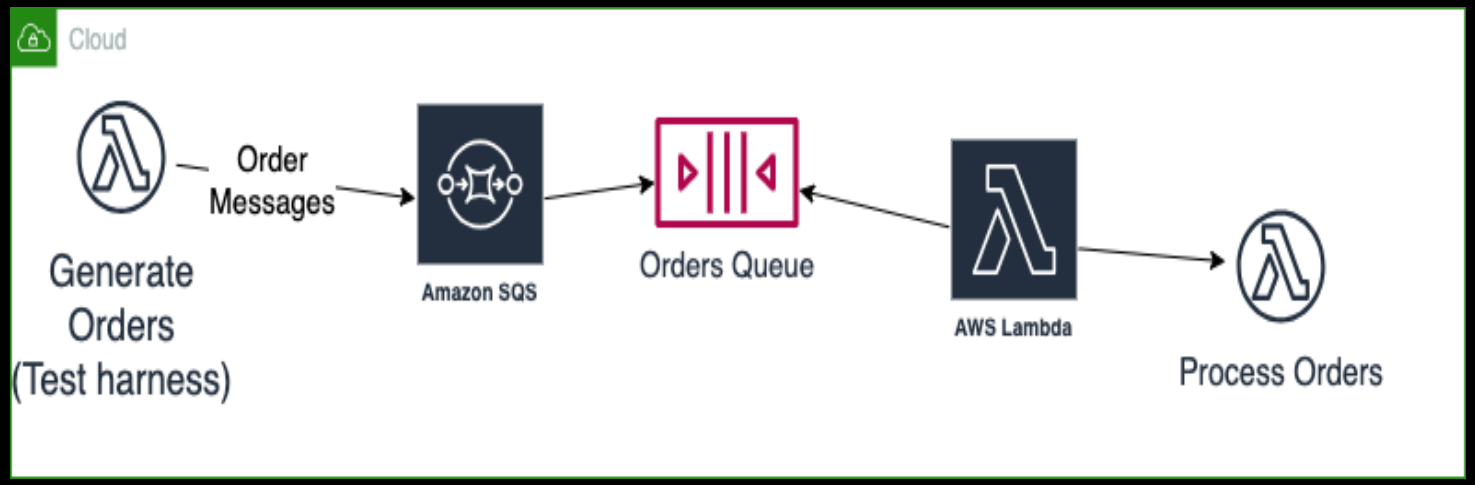
- When a consume polls a queue for messages, the SQS service samples a subset of its servers (based on a weighted random distribution) and returns messages from only those servers.

Lambda Consumer scaling



- Lambda service:
  1. Polls SQS and waits for a message batch response.
  2. Arbitrarily splits the batch into smaller sub-batches
  3. Instantiates a micro VM (function) for each sub-batch
- It adds up to 60 functions per minute, up to 1,000 functions, to consume large message volumes.

Demo



- Generate_Orders lambda function needs permission to send messages to a queue, i.e.
  ordersQueue.grantSendMessages(generateOrdersFn)

Demo - Generate Orders (Producer)

shared > TS types.d.ts > ...

```typescript

export type Order = {
  customerName: string;
  customerAddress: string;
  items: string[];
};

export type BadOrder = Partial<Order>;
export type OrderMix = Order | BadOrder;
```

```typescript
const orders: OrderMix[] = []
for (let i = 0 ; i < 10; i++) {
  orders.push({
    customerName: `User${i}`,
    customerAddress: "1 Main Street",
    items: [],

  })
}
```

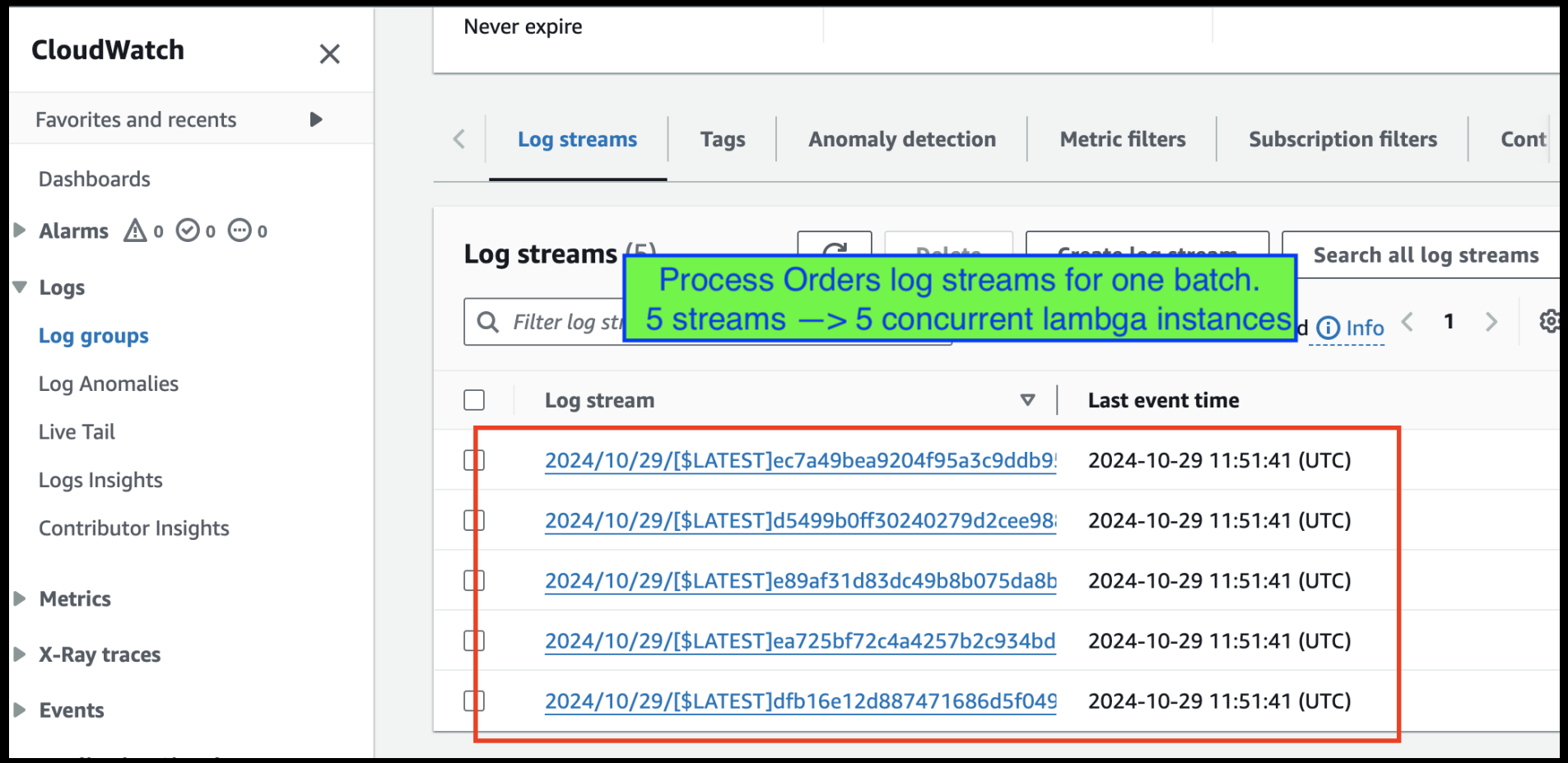
```typescript
const client =
          new SQSClient({ region: "eu-west-1" });
      You, 3 minutes ago • Uncommitted changes
export const handler: Handler = async (event) => {
  try {
    const entries: SendMessageBatchRequestEntry[] =
     orders.map((order) => {
       return {
         Id: v4(),
         MessageBody: JSON.stringify(order),
       };
    });
    const batchCommandInput: SendMessageBatchCommandInput = {
      QueueUrl: process.env.QUEUE_URL, Entries: entries,
    };
    const batchResult = await client.send(
      new SendMessageBatchCommand(batchCommandInput)
    );
    return {
      statusCode: 200,
      headers: {
        "content-type": "application/json",
      },
      body: "All orders queued for processing",
    };
```

Demo – Process Orders (Consumer)

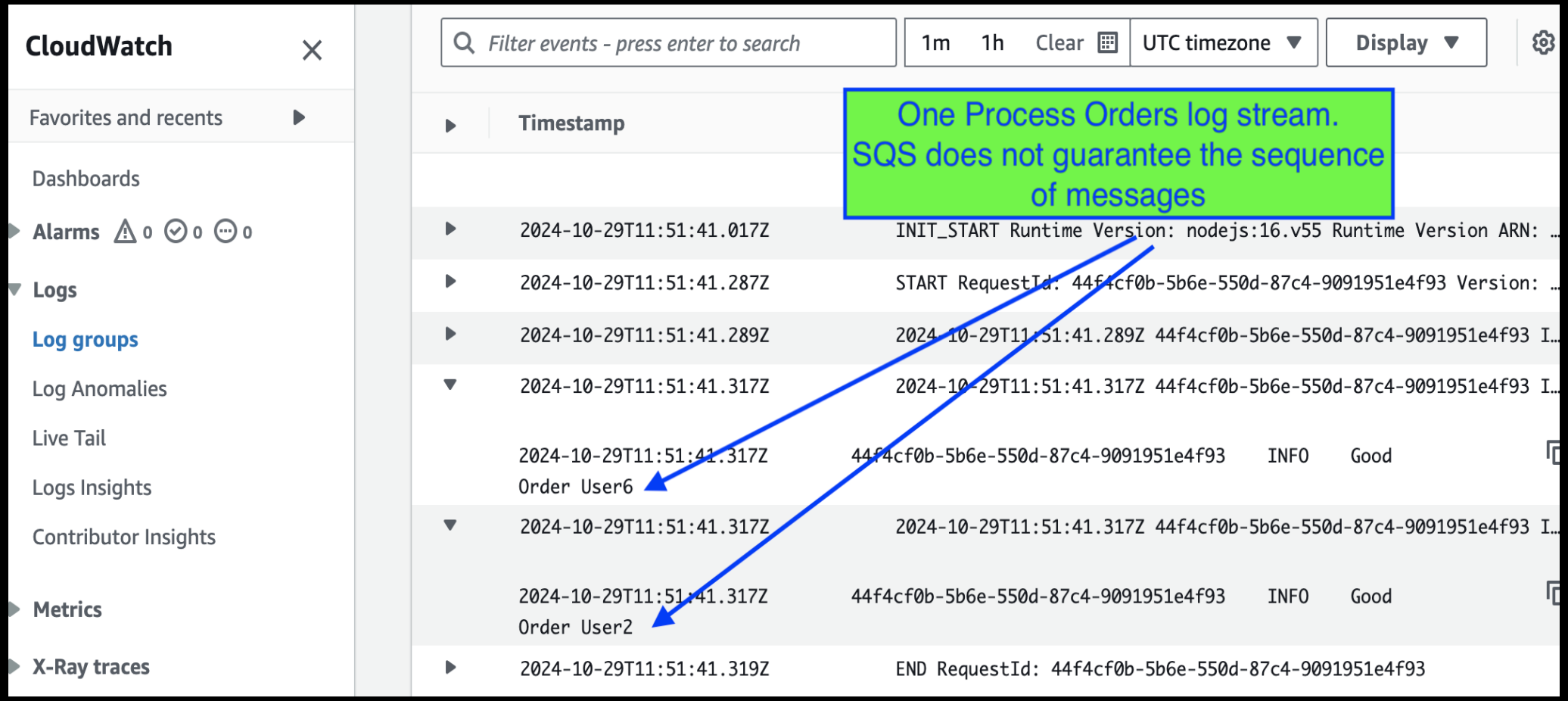
```
// Order Q processor

const ajv = new Ajv();
const isValidOrder = ajv.compile(schema.definitions["Order"] || {});
export const handler: SQSHandler = async (event) => {
  try {
    for (const record of event.Records) {
      const messageBody = JSON.parse(record.body);
      if (!isValidOrder(messageBody)  ) {
        throw new Error(" Bad Order");
      }
      // process good order
    }
  } catch (error) {
    throw new Error(JSON.stringify(error));
  }
};
```

Who handles the exception? (see later)

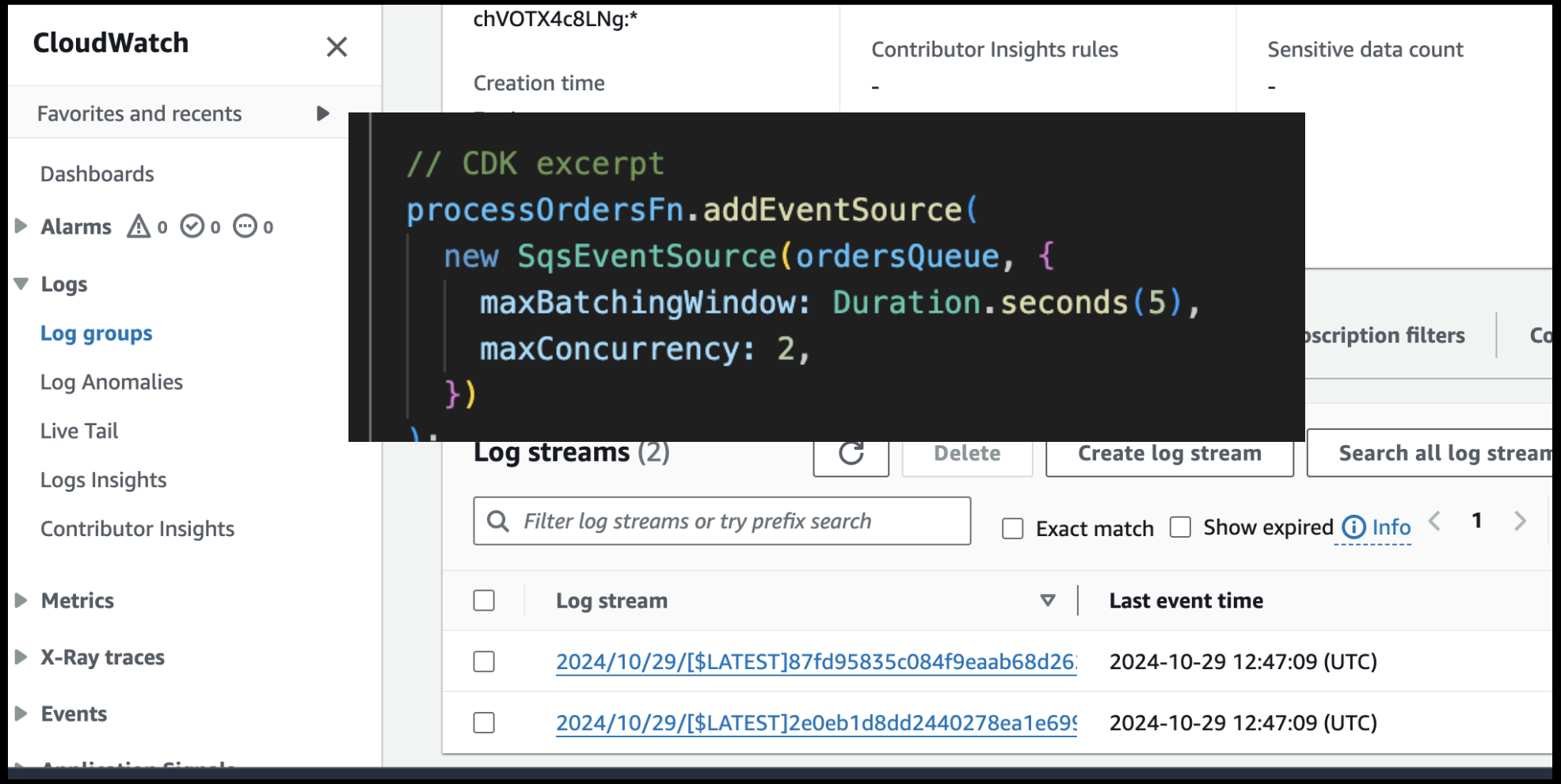
Demo – Lambda consumer scaling



Demo – No guarantee of message order

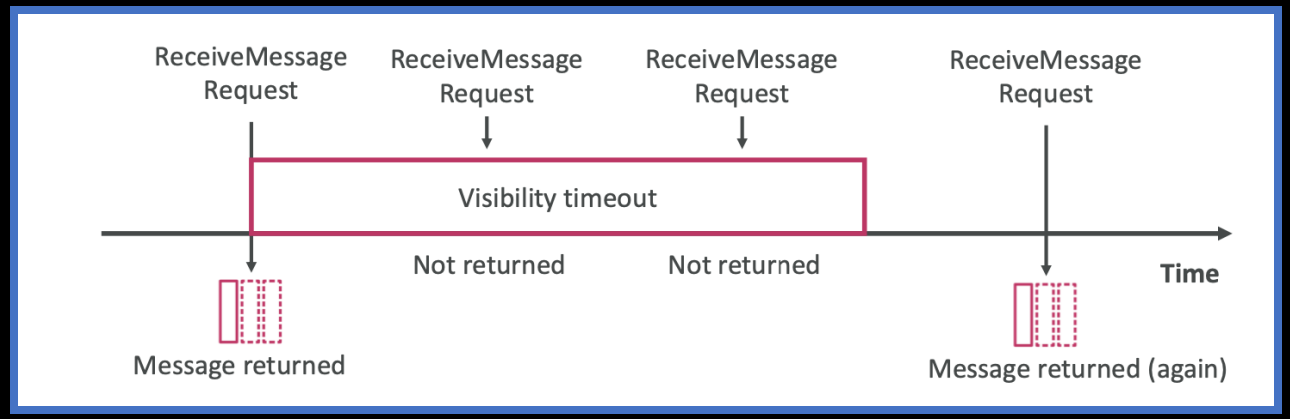


Demo – Controlling consumer concurrency.



Message Visibility.

- When a message is polled by a consumer, it remains in the queue but is invisible to other consumers.
  - The default “message visibility timeout” is 30 seconds.
- Consumer must process (and delete) a message within the timeout period. Otherwise, the message is “visible” again.

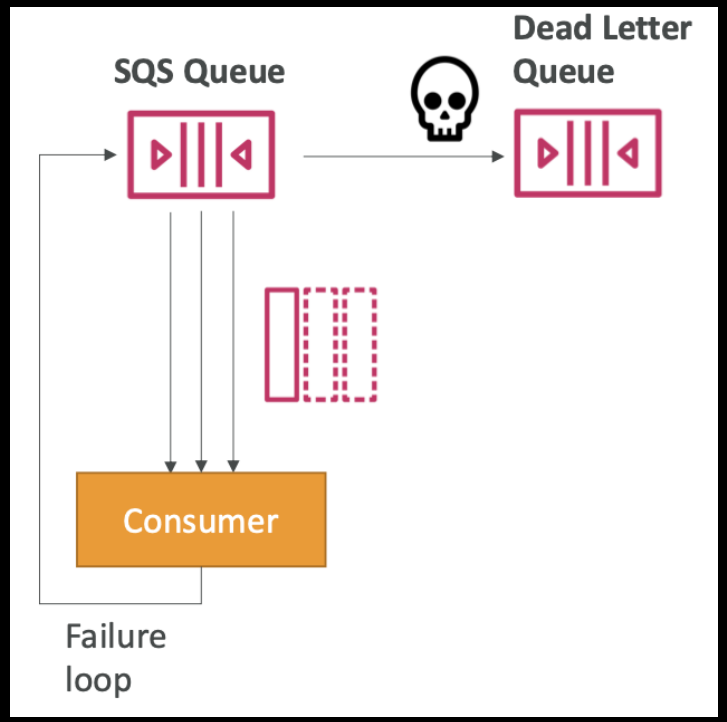


- Timeout too high (minutes/hours) => Re-processing delayed when consumer fails.
- Timeout too low => message may be processed by multiple consumer.

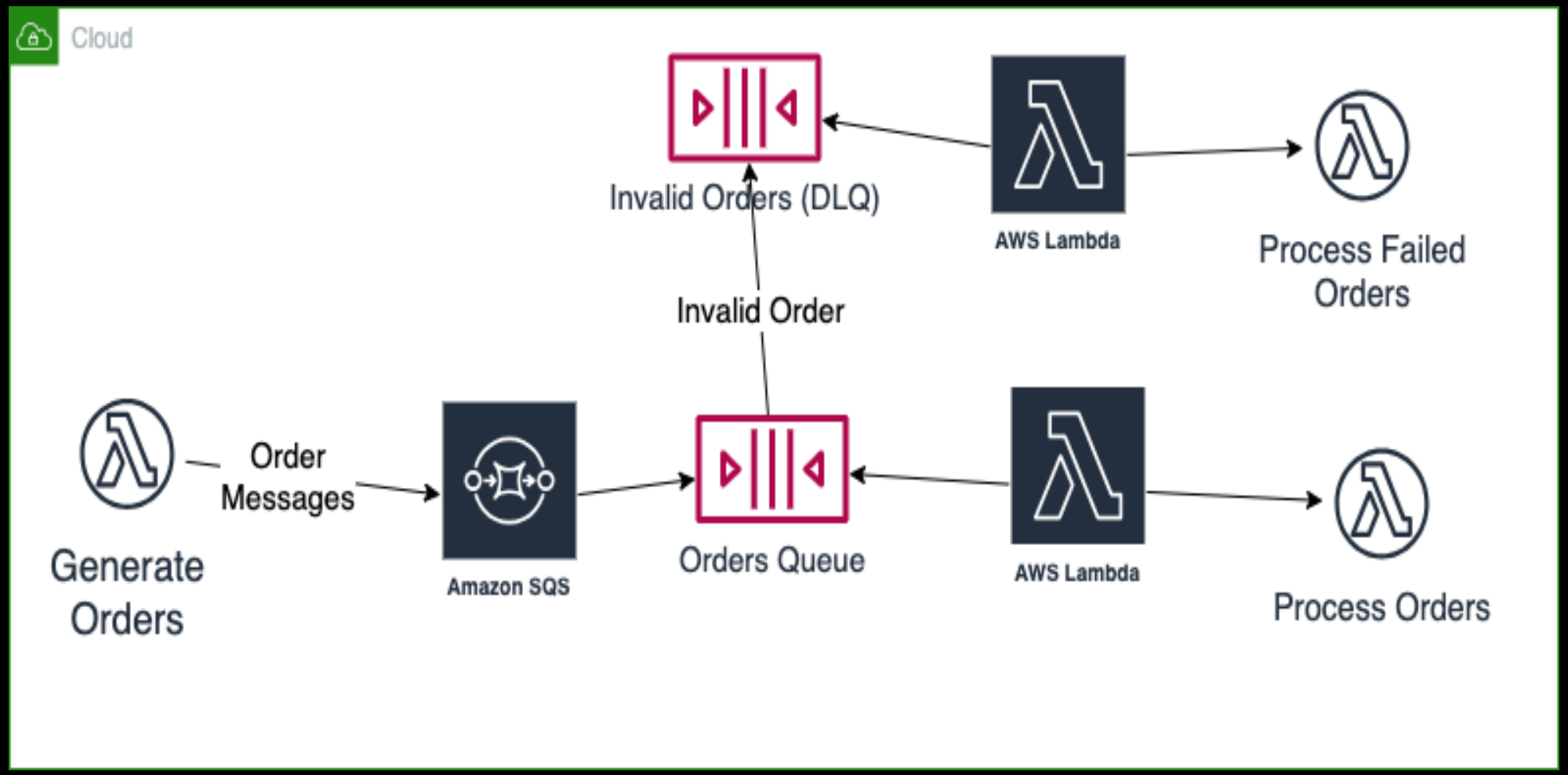
Dead Letter Queues (DLQs)

- If a consumer does not process a message batch within the visibility period (times out or throws exception), the batch is ‘returns to the queue’.
- Maximum Receives threshold - how many times a message is returned to the queue.
- After the threshold is exceeded, the message goes into a DLQ, if defined.

- Useful for debugging!
- Separate consumer required to process DLQ messages.



Demo – DLQ.


Cloud
Invalid Orders (DLQ)
AWS Lambda
Process Failed Orders
Invalid Order
Order Messages
Generate Orders
Amazon SQS
Orders Queue
AWS Lambda
Process Orders

To be continued ….