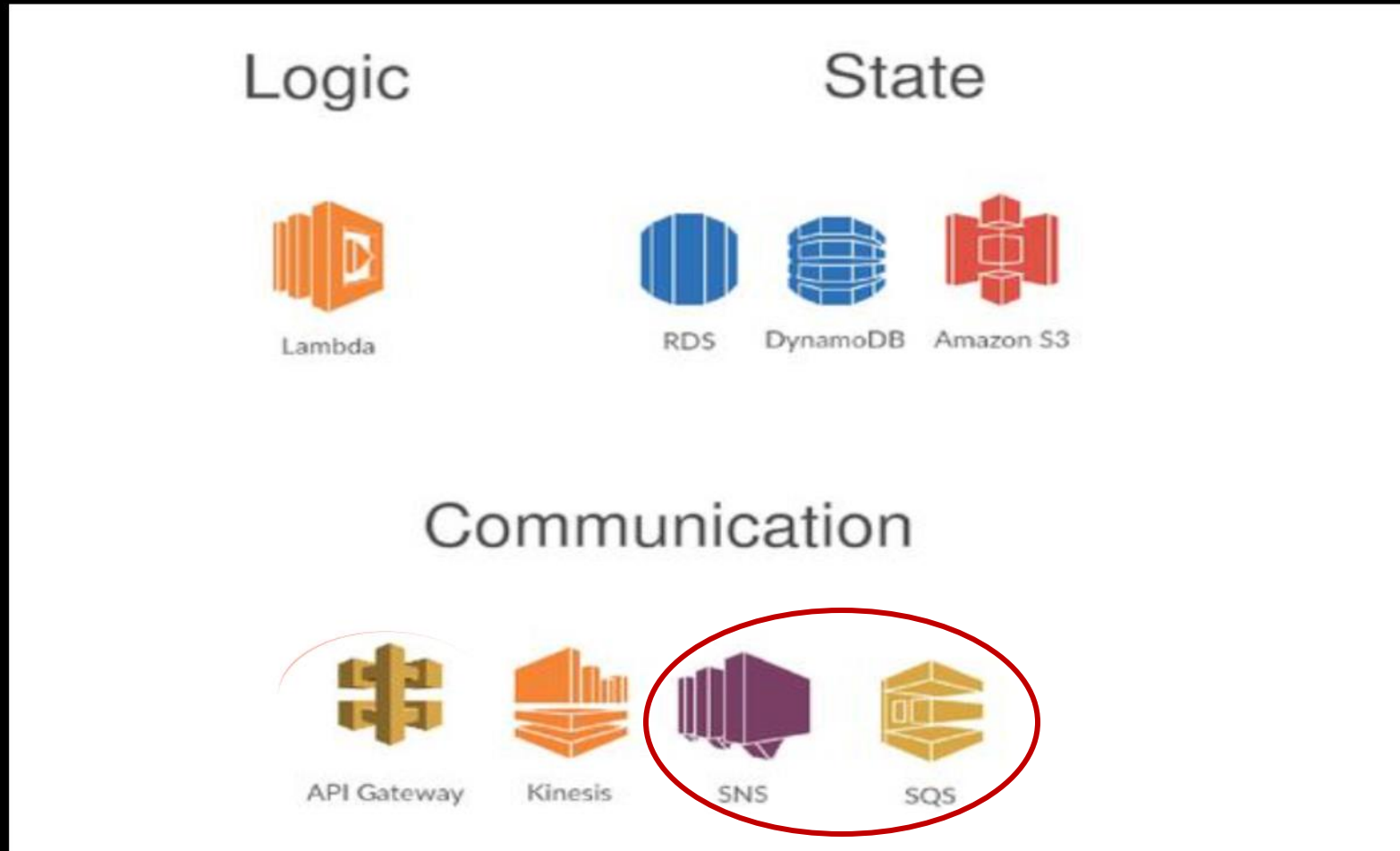
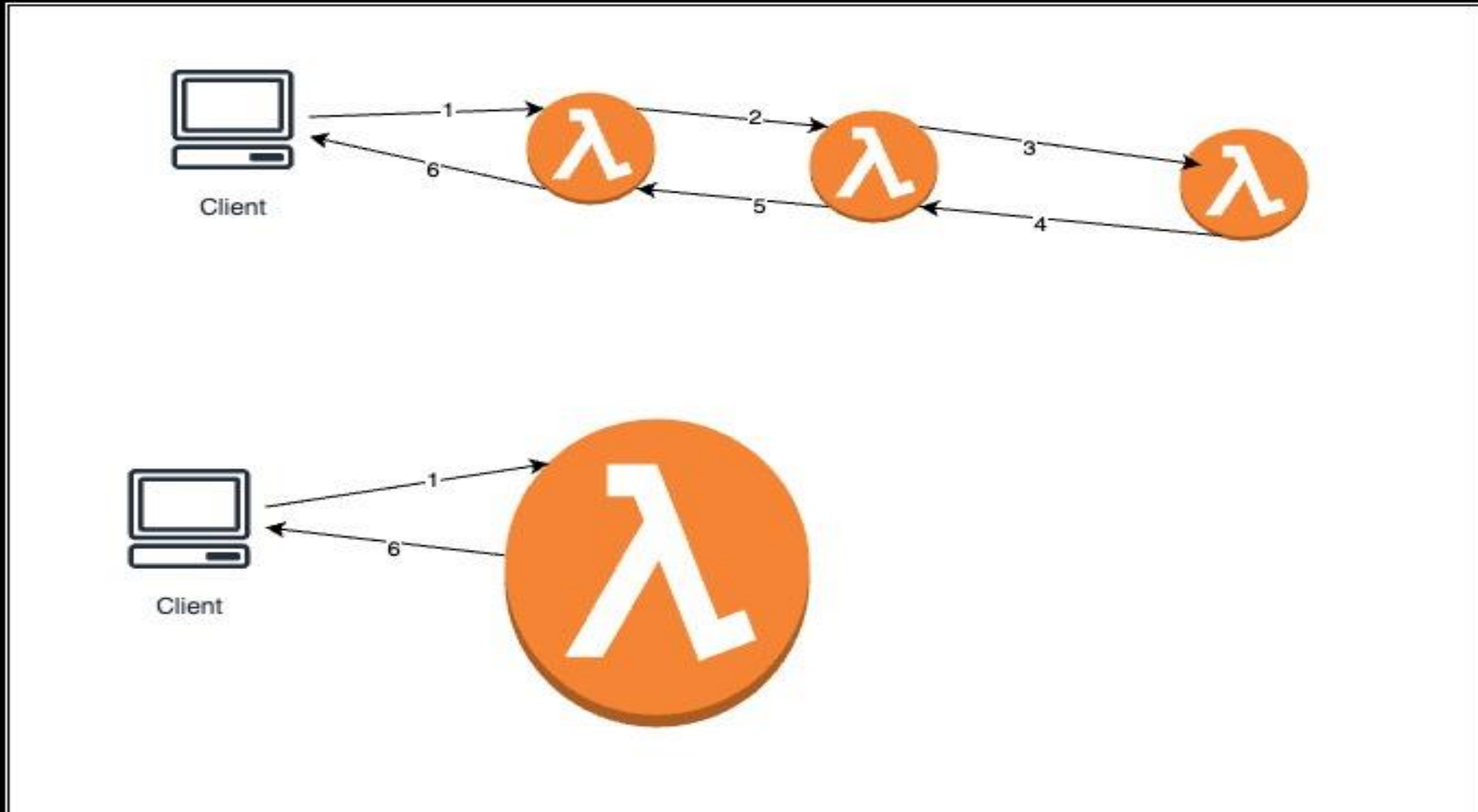


**AWS Integration and Messaging  
Services.**

# Components of a Serverless, Message-Driven application (aka Event Driven Architecture - EDA)



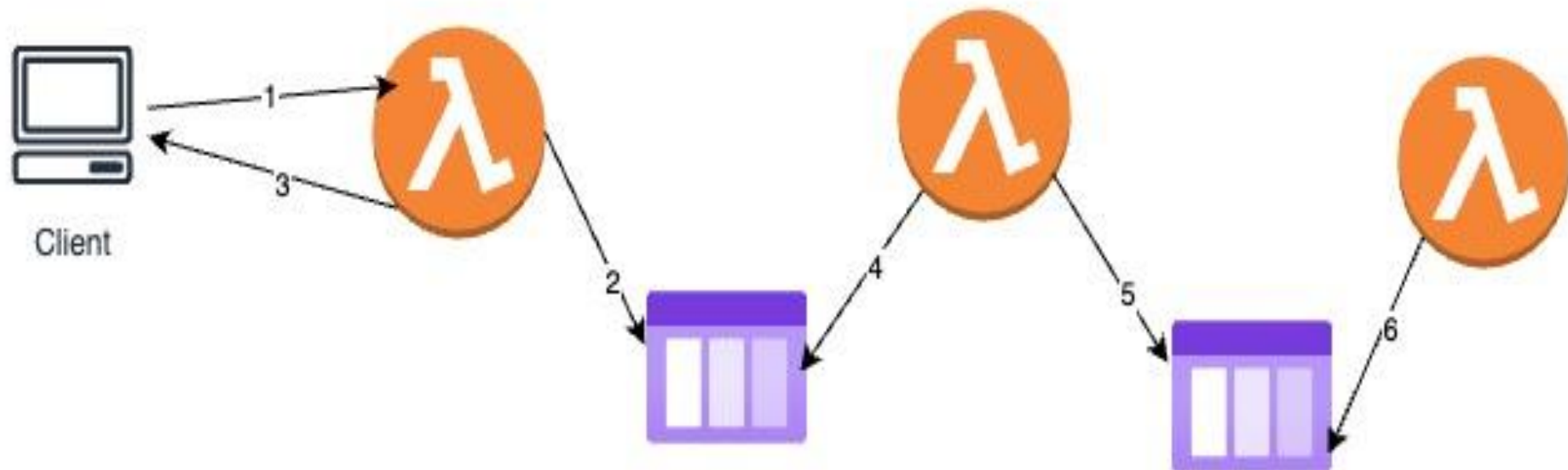
# Why do we need Messaging Services?



# 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?

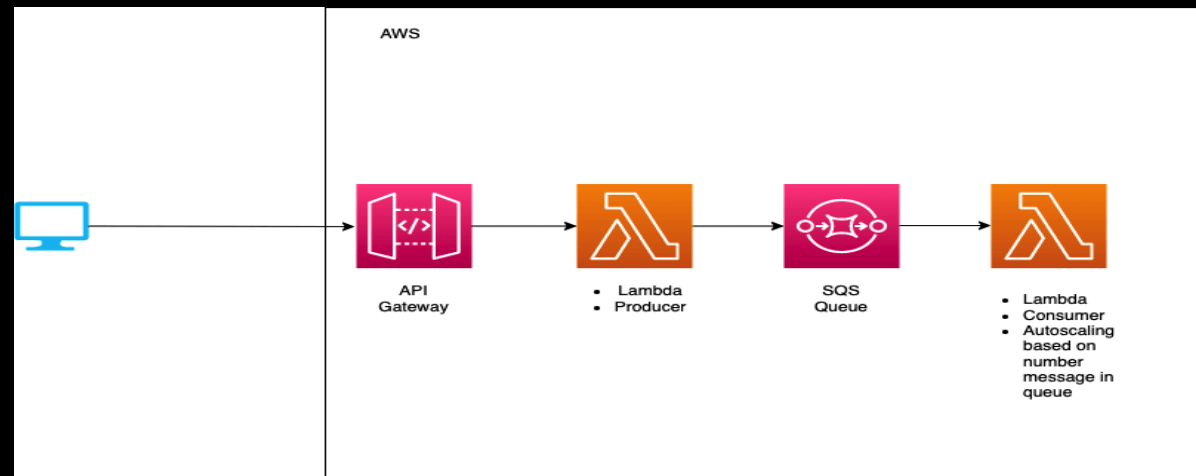
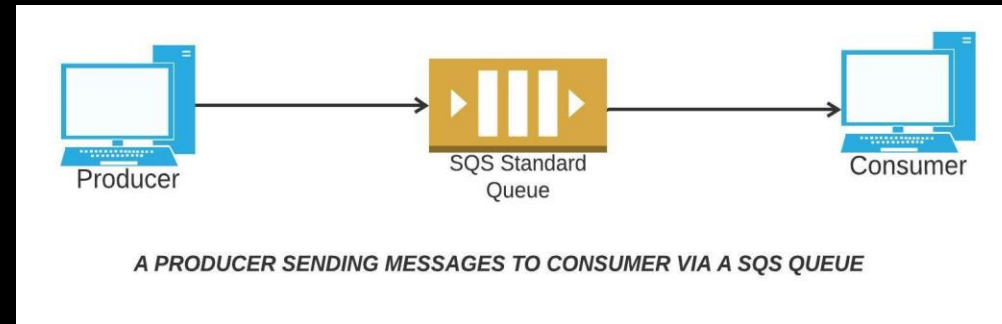




Simple Queue Service (SQS)

# SQS - Overview

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



# SQS - Overview

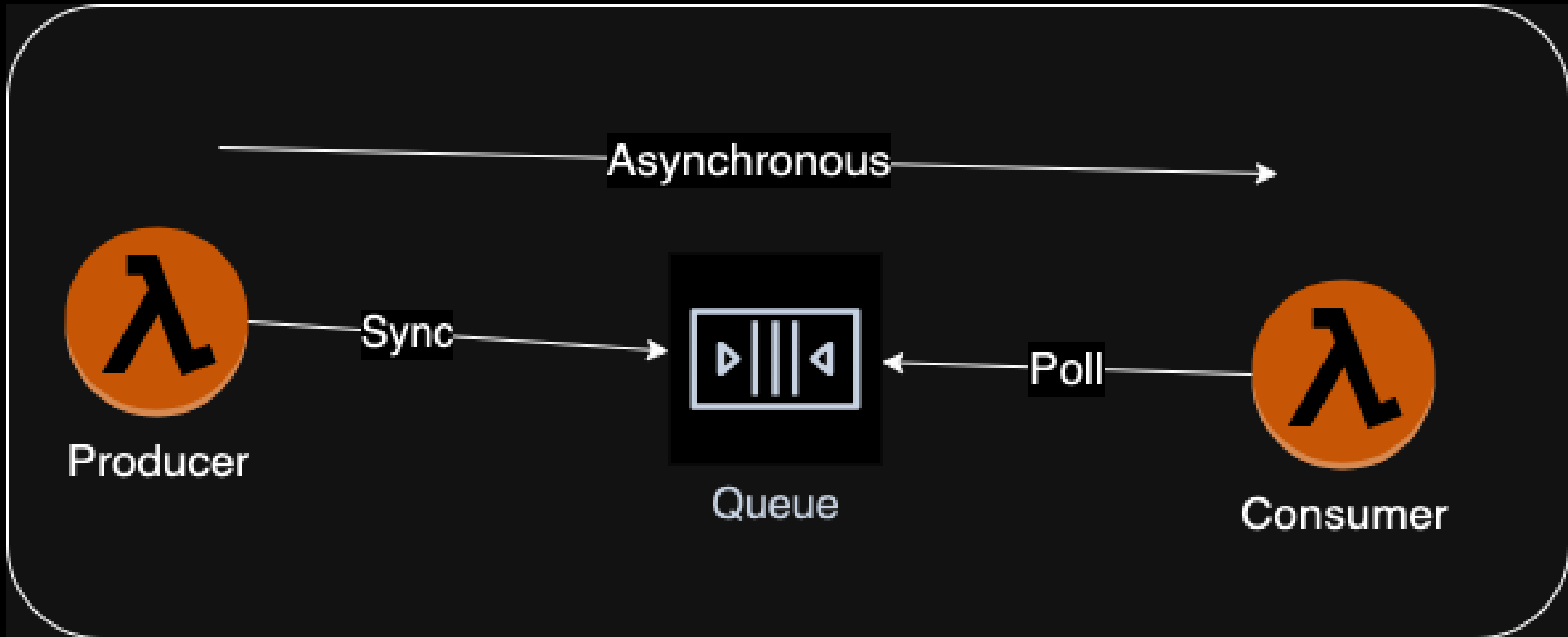
- SQS Attributes:
  - Scalability - Unlimited throughput, unlimited number of messages in a queue.
  - Message Retention: 4 days (default), maximum of 14 days.
  - Low latency (< 10 msgs on publish and receive).
  - Limitation of 256KB per message.
- Caveats:
  - Duplicate messages may occur, occasionally.
    - 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 ( $\leq 10$  messages).
  3. Process the message batch, e.g. validate & insert order into a d/b.
  4. Delete the message batch using the SQS SDK.

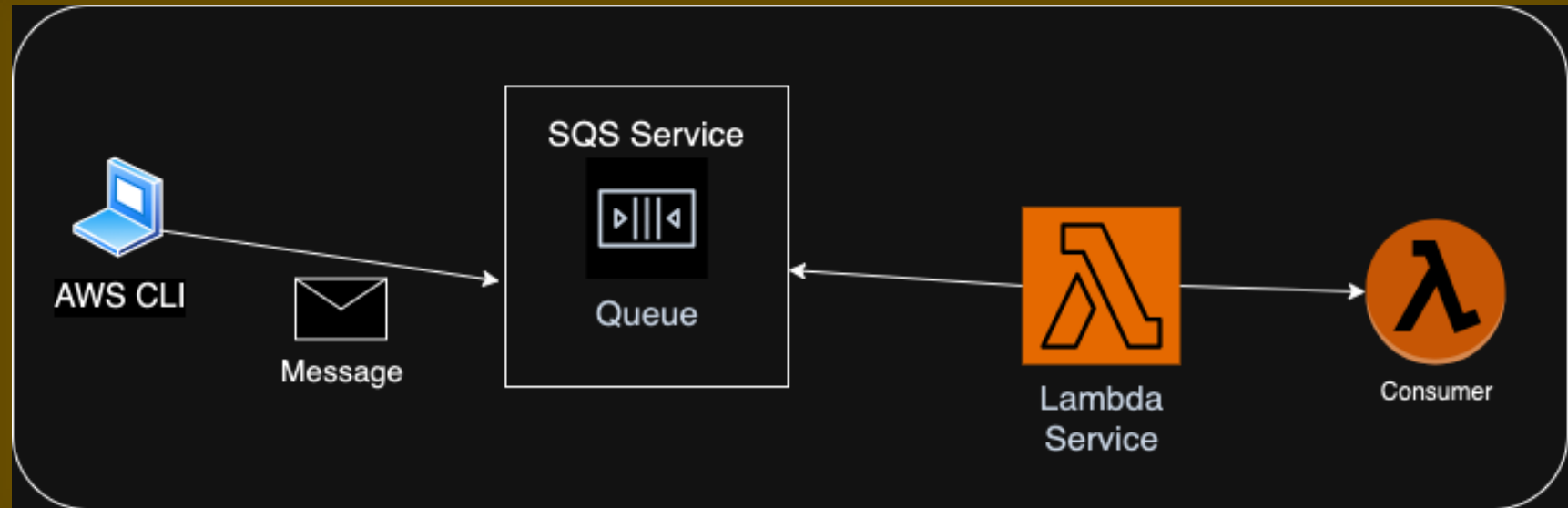
# Communication styles.



# 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...) write to a 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
270
```


- 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";
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   } catch (error) {
283     console.log(JSON.stringify(error));
284   }
285 };
286
287
```

You, 1 second ago • Un

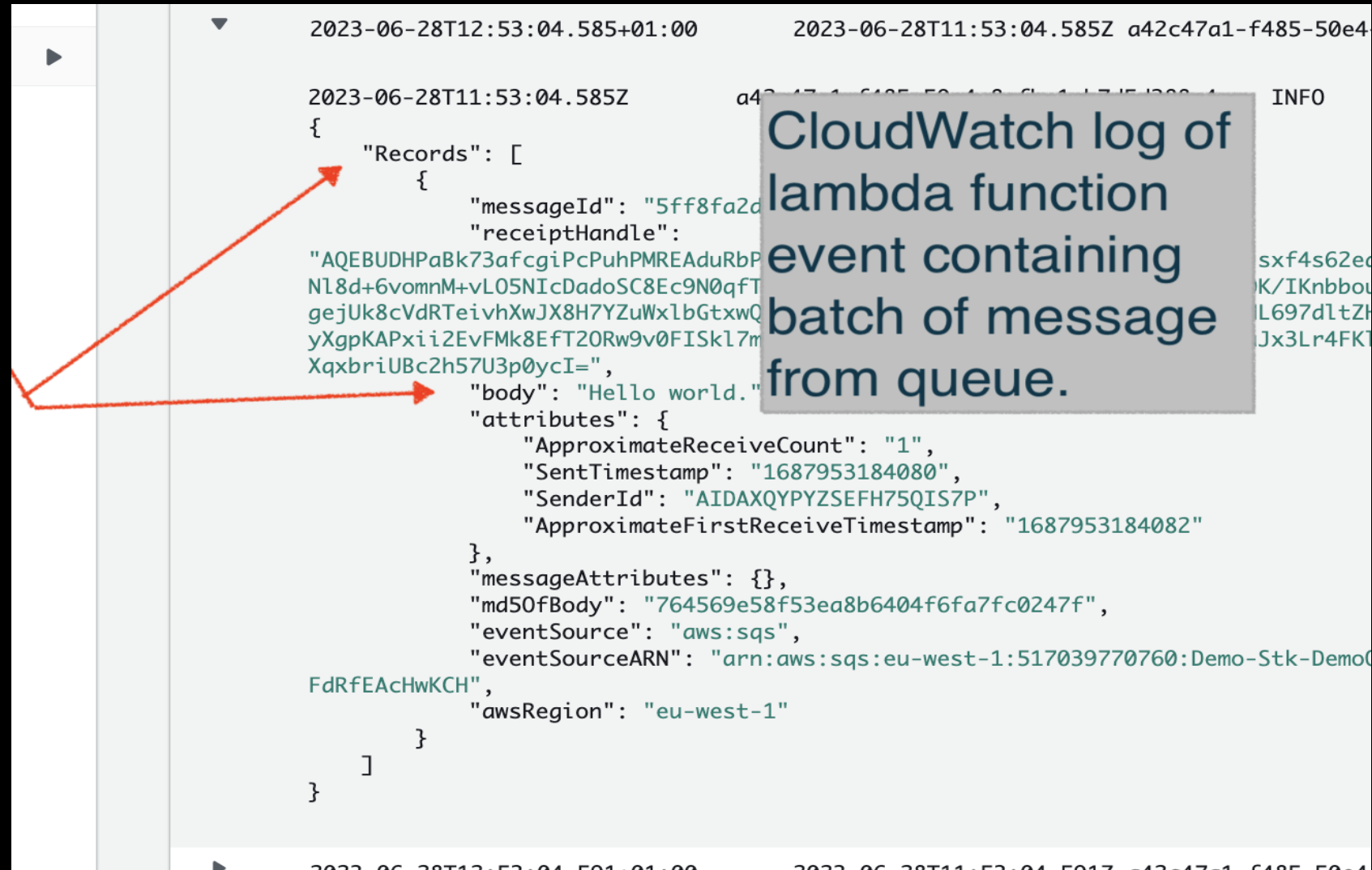
**Batch**



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

**AWS CLI**

# Demo – Lambda Consumer event structure.



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

```
2023-06-28T12:53:04.585+01:00 2023-06-28T11:53:04.585Z a42c47a1-f485-50e4-8000-000000000000 INFO
{
  "Records": [
    {
      "messageId": "5ff8fa2d-1b3e-4b3e-8000-000000000000",
      "receiptHandle": "AQEBUDHPaBk73afcgiPcPuhPMREAdurBPNl8d+6vomnM+vL05NIcDadoSC8Ec9N0qfTgejUk8cVdRTeivhXwJX8H7YZuWxlbGtxwQyXgpKAPxii2EvFMk8EFT20Rw9v0FISkl7mXqxbriUBc2h57U3p0ycI=",
      "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-Demo0FdrfEAChwKCH",
      "awsRegion": "eu-west-1"
    }
  ]
}
```

# 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      Ever
{
  "Records": [
    {
      "messageId": "52c0079d-9f7f-406c-8584-bc9eec46e39f",
      "receiptHandle": "AQEBBhJ2+J2W0pmb6aK5AvfKM8ERAW3P9bJCsCPK8DoIoMeGYjh+uWaXKtch/pD4/PQbbGwwy7k6S9Ifd
o2f1K5f9ojM51H3KrzWAF1HzMg87gAkgY0xnDjjGMrZd+Hdwk+Rd7HaQsqueUw2voJYe0+0abdwM6LEiEGd
0uxsBv29C+TOYvAWVA1LDf7GMFkb860eMusWxJZLk+t+XTKrI3B9ghfrS3z/7tHxao+4GGn+nbmNBVv496HC
/c2zsFTkhggIgWwS56HFopf8JZyu+IcLMteheaPFJAhmjGUVfTVXwjLSS0FNpXvH8d0Uz95SfItY9MFI2qk
9ioOWhETW5uE/F4tf+LE=",
      "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",
      "eventVersion": "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
```



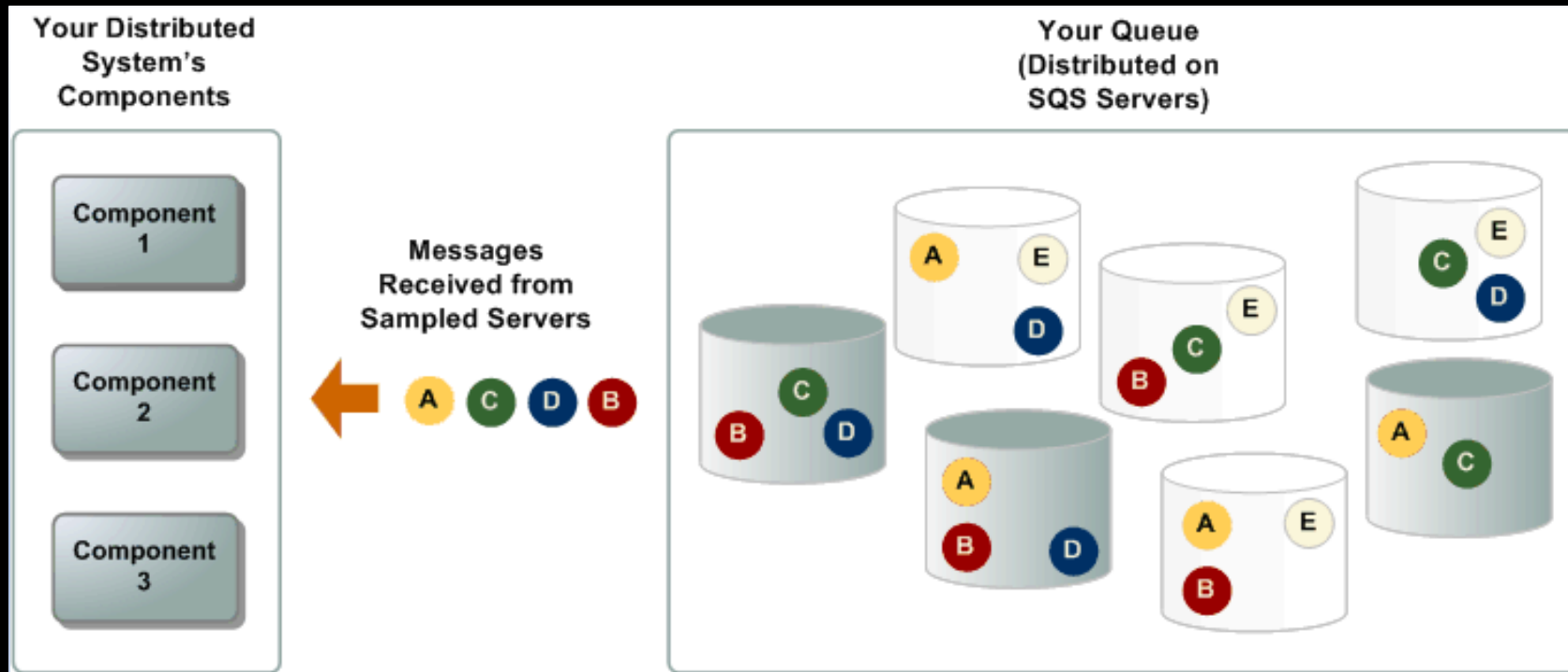
# Demo - JSON messages.

- The lambda handler (Consumer)

```
You, 16 seconds ago | 1 author (You)
1  import { SQSHandler } from "aws-lambda";
2
3  export const handler: SQSHandler = async (event) => {
4      try {
5          console.log("Event: ", event);
6          for (const record of event.Records) {
7              const message = JSON.parse(record.body)
8              const {name, address} = message
9              console.log(name, address);
10         }
11     } catch (error) {
12         console.log(JSON.stringify(error));
13     }
14 };
15
```

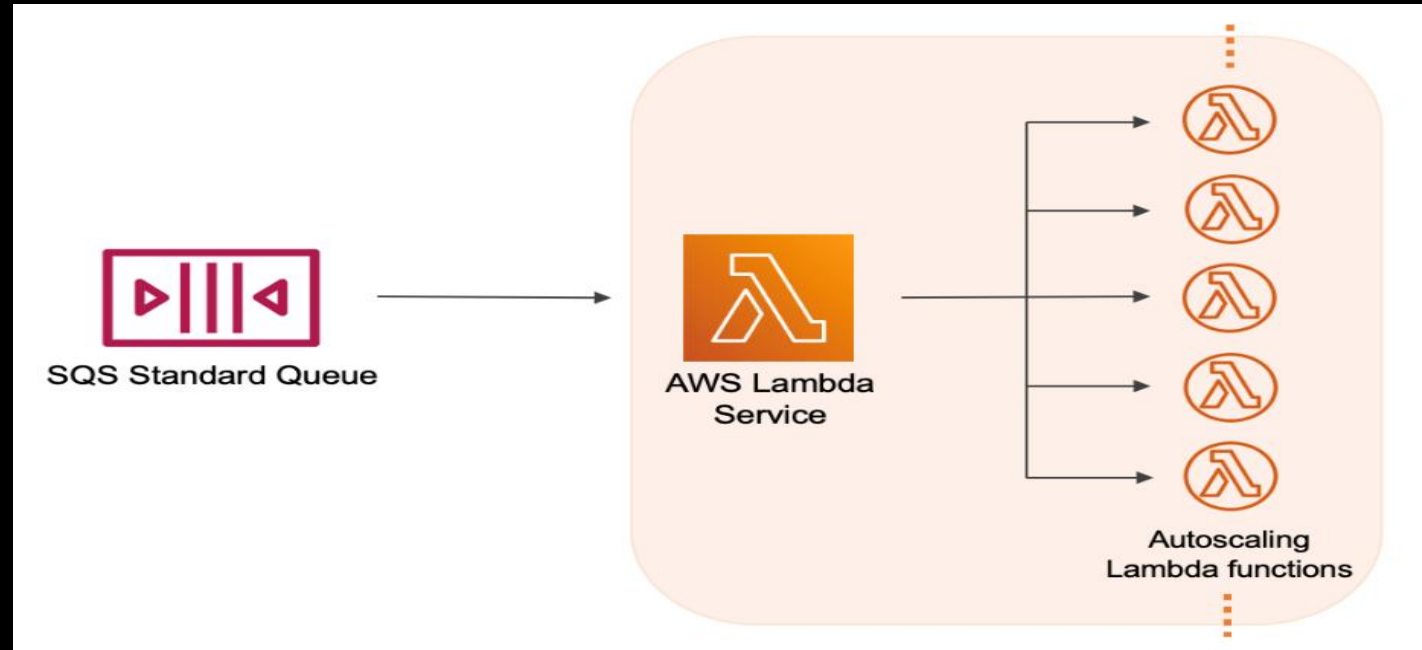


# SQS is Resilient



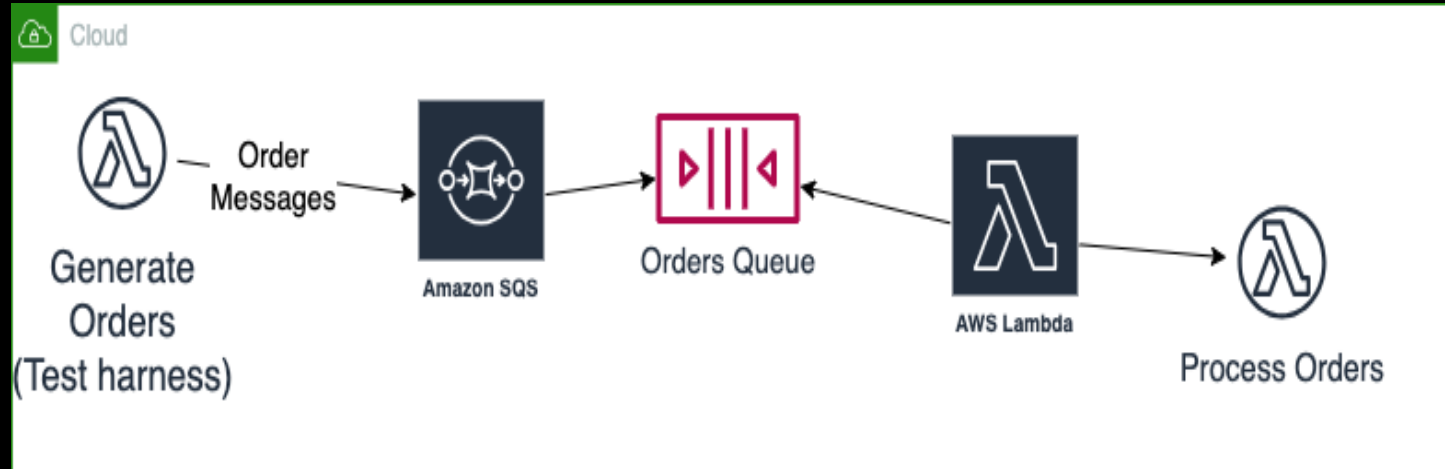
- When a consumer polls a queue for messages, the SQS service samples a subset of its servers (based on a weighted random distribution) and returns messages from only those servers.

# 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.grantSendMessage(generateOrdersFn)`

# Demo - Generate Orders (Producer)

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

```
1
2 export type Order = {
3   customerName: string;
4   customerAddress: string;
5   items: string[];
6 };
7
8 export type BadOrder = Partial<Order>;
9 export type OrderMix = Order | BadOrder;
10
```

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

```
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

CloudWatch

Never expire

Log streams

Tags

Anomaly detection

Metric filters

Subscription filters

Cont

Log streams (5)

Search all log streams

Filter log streams

Info

1

<input type="checkbox"/>	Log stream	Last event time
<input type="checkbox"/>	<a href="#">2024/10/29/[\$LATEST]ec7a49bea9204f95a3c9ddb9!</a>	2024-10-29 11:51:41 (UTC)
<input type="checkbox"/>	<a href="#">2024/10/29/[\$LATEST]d5499b0ff30240279d2cee98:</a>	2024-10-29 11:51:41 (UTC)
<input type="checkbox"/>	<a href="#">2024/10/29/[\$LATEST]e89af31d83dc49b8b075da8b</a>	2024-10-29 11:51:41 (UTC)
<input type="checkbox"/>	<a href="#">2024/10/29/[\$LATEST]ea725bf72c4a4257b2c934bd</a>	2024-10-29 11:51:41 (UTC)
<input type="checkbox"/>	<a href="#">2024/10/29/[\$LATEST]dfb16e12d887471686d5f049</a>	2024-10-29 11:51:41 (UTC)

Process Orders log streams for one batch.  
5 streams → 5 concurrent lambda instances

# Demo – No guarantee of message order

CloudWatch

Filter events - press enter to search

1m 1h Clear UTC timezone Display

Timestamp

One Process Orders log stream. SQS does not guarantee the sequence of messages

Timestamp	Log Stream	Message
2024-10-29T11:51:41.017Z	INIT_START Runtime Version: nodejs:16.v55 Runtime Version ARN: ...	
2024-10-29T11:51:41.287Z	START RequestId: 44f4cf0b-5b6e-550d-87c4-9091951e4f93 Version: ...	
2024-10-29T11:51:41.289Z	2024-10-29T11:51:41.289Z 44f4cf0b-5b6e-550d-87c4-9091951e4f93 I...	
2024-10-29T11:51:41.317Z	2024-10-29T11:51:41.317Z 44f4cf0b-5b6e-550d-87c4-9091951e4f93 I...	
2024-10-29T11:51:41.317Z	44f4cf0b-5b6e-550d-87c4-9091951e4f93 INFO Good	Order User6
2024-10-29T11:51:41.317Z	2024-10-29T11:51:41.317Z 44f4cf0b-5b6e-550d-87c4-9091951e4f93 I...	
2024-10-29T11:51:41.317Z	44f4cf0b-5b6e-550d-87c4-9091951e4f93 INFO Good	Order User2
2024-10-29T11:51:41.319Z	END RequestId: 44f4cf0b-5b6e-550d-87c4-9091951e4f93	



# Demo – Controlling consumer concurrency.

The screenshot shows the AWS CloudWatch console interface. On the left is a navigation sidebar with options like Alarms, Logs, Metrics, and X-Ray traces. The main area displays a list of log streams under the heading "Log streams (2)". A dark overlay with a code snippet is positioned over the log streams list.

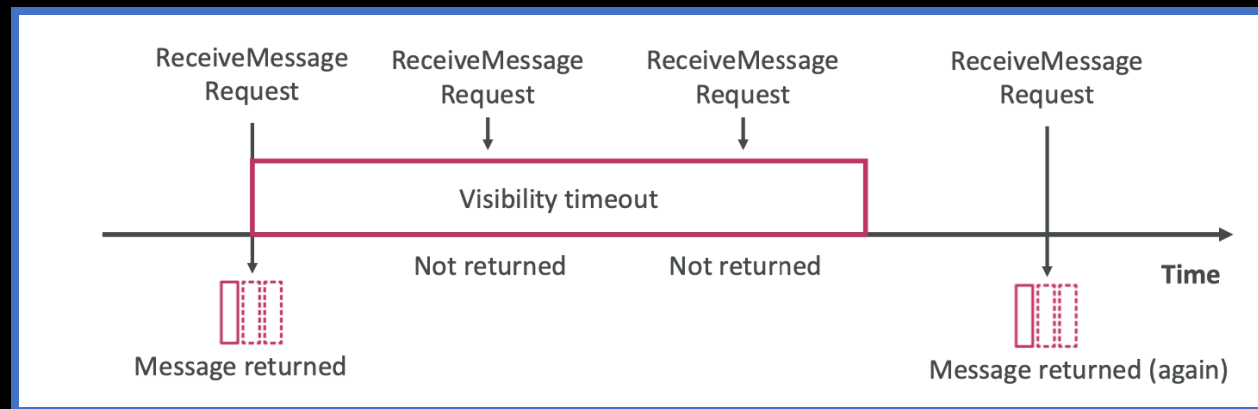
```
// CDK excerpt
processOrdersFn.addEventSource(
  new SqsEventSource(ordersQueue, {
    maxBatchingWindow: Duration.seconds(5),
    maxConcurrency: 2,
  })
);
```

Below the code overlay, the log streams table is visible. It includes a search bar, checkboxes for each log stream, and a column for the last event time.

<input type="checkbox"/>	Log stream	Last event time
<input type="checkbox"/>	<a href="#">2024/10/29/[\$LATEST]87fd95835c084f9eaab68d26</a>	2024-10-29 12:47:09 (UTC)
<input type="checkbox"/>	<a href="#">2024/10/29/[\$LATEST]2e0eb1d8dd2440278ea1e69</a>	2024-10-29 12:47:09 (UTC)

# Message Visibility.

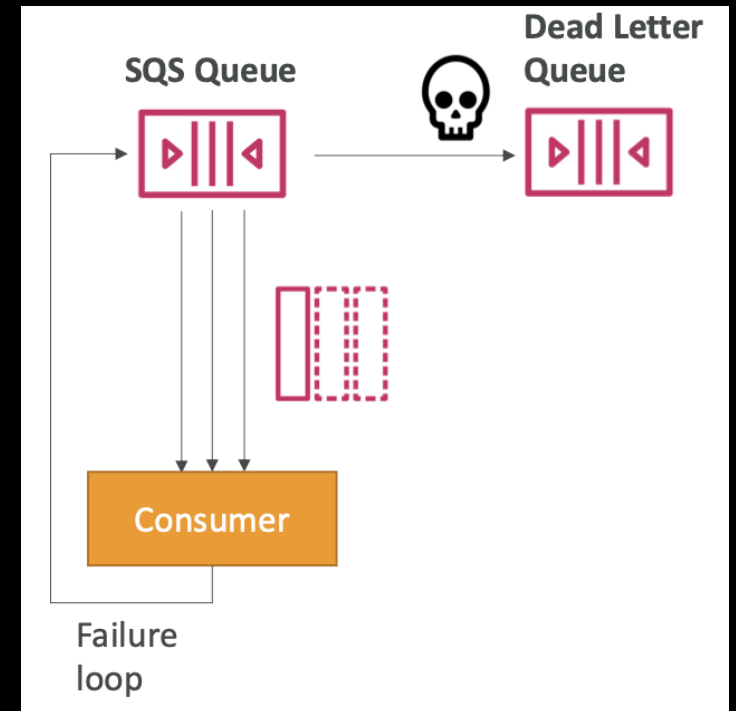
- When a message is polled by a consumer, it remains in the queue but is invisible to other consumers.
  - The default “message visibility timeout” is 30 seconds.
- Consumer must process (and delete) a message within the timeout period. Otherwise, the message is “visible” again.



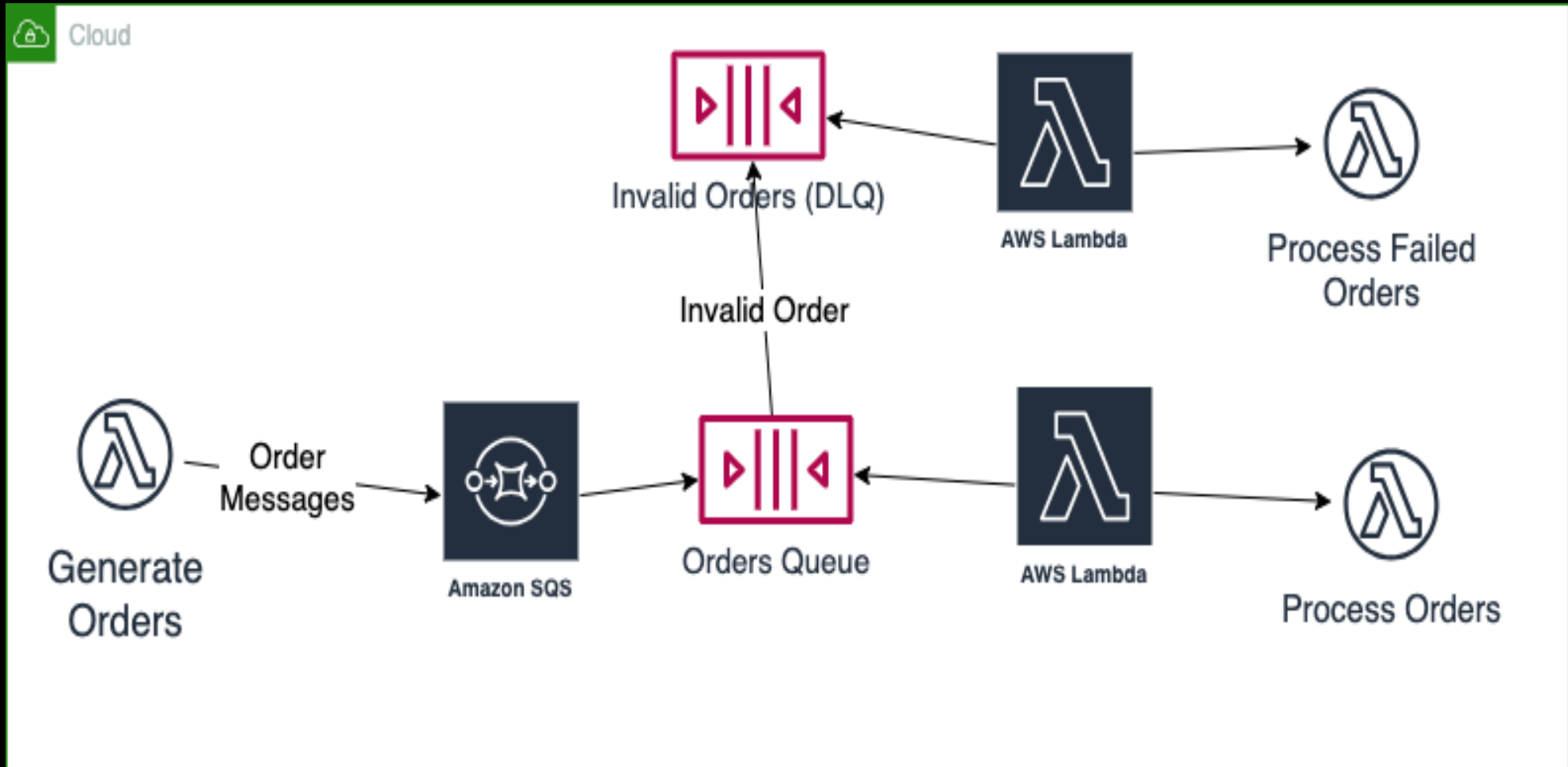
- Timeout too high (minutes/hours) => Re-processing delayed when consumer fails.
- Timeout too low => message may be processed by multiple consumer.

# Dead Letter Queue (DLQ)

- If a consumer does not process a message batch within the visibility period (due to timeout or an exception), the batch is 'returns to the queue'.
- Maximum Receives threshold – the number of times a message is returned to the queue.
- After the threshold is exceeded, the batch goes into a DLQ, if defined.
  - Useful for debugging!
  - DLQ may have a separate consumer.



# Demo – DLQ.



# Demo – Provision the DLQ.

```
const badOrdersQueue = new Queue(this, 'bad-orders-q');

const ordersQueue = new Queue(this, 'orders-queue', {
  deadLetterQueue: {
    queue: badOrdersQueue,
    // # of rejections by consumer (lambda function) before
    // message is transferred to DLQ
    maxReceiveCount: 1,
  },
});

// .... declare Lambda function resources .....

// Set SQS queues as Event sources for lambda functions
processOrdersFn.addEventSource(new SqsEventSource(ordersQueue))
failedOrdersFn.addEventSource(new SqsEventSource(badOrdersQueue));
```

## Demo – Generate Orders (Producer).

```
const orders: OrderMix[] = [];  
for (let i = 0; i < 10; i++) {  
  orders.push({  
    customerName: `User${i}`,  
    customerAddress: "1 Main Street",  
    items: [],  
  });  
}  
  
orders.splice(6, 0, {  
  // No address property – Bad.  
  customerName: "UserX",  
  items: [],  
});
```

# Demo – Sample Execution

- Scenario:
  - Set the maxConcurrency of the Process Orders handler to 2.
  - Set the maximum receive count of the Orders q to 1 – batches containing a bad order are sent to DLQ after first failed processing attempt.
- Outcome:
  - Process Orders instance 1 → User0, User4, User7, User9.
  - Process Orders instance 2 → User1, User2, User3, User5, UserX, User8.
  - Bad Orders instance 1 → User1, User3, User8.
  - Bad Orders instance 2 → User2, User5, UserX.

# Demo – Sample Execution

CloudWatch

Favorites and recents

Dashboards

Alarms 0 0 0

Logs

Log groups

Log Anomalies

Live Tail

Logs Insights

Contributor Insights

Metrics

X-Ray traces

Events

Process OrdersFn log stream 1  
User0, User4, User7, User9

No older events at this moment. [Retry](#)

2024-10-29T15:11:33.366Z

START RequestId: 71566f78-734d-5696-8f91-30e28e08bfeb Version: ...

2024-10-29T15:11:33.424Z

2024-10-29T15:11:33.424Z 71566f78-734d-5696-8f91-30e28e08bfeb I...

2024-10-29T15:11:33.425Z

2024-10-29T15:11:33.425Z 71566f78-734d-5696-8f91-30e28e08bfeb I...

2024-10-29T15:11:33.425Z

71566f78-734d-5696-8f91-30e28e08bfeb INFO Good

2024-10-29T15:11:33.425Z

2024-10-29T15:11:33.425Z 71566f78-734d-5696-8f91-30e28e08bfeb I...

2024-10-29T15:11:33.425Z

71566f78-734d-5696-8f91-30e28e08bfeb INFO Good

2024-10-29T15:11:33.444Z

2024-10-29T15:11:33.444Z 71566f78-734d-5696-8f91-30e28e08bfeb I...

2024-10-29T15:11:33.444Z

71566f78-734d-5696-8f91-30e28e08bfeb INFO Good

2024-10-29T15:11:33.464Z

2024-10-29T15:11:33.464Z 71566f78-734d-5696-8f91-30e28e08bfeb I...

2024-10-29T15:11:33.464Z

71566f78-734d-5696-8f91-30e28e08bfeb INFO Good

Order User9

Order User4

Order User7

Order User0



# Demo – Sample Execution

CloudWatch

Favorites and recents

Dashboards

Alarms 0 0 0

Logs

Log groups

Log Anomalies

Live Tail

Logs Insights

Contributor Insights

Metrics

Process Orders log stream 2

2024-10-29T15:11:33.478Z

INIT\_START Runtime Version: nodejs:18.v48 Runtime Version ARN: ...

2024-10-29T15:11:33.750Z

START RequestId: ee6c5f9e-546c-5bff-865e-0004a926f0dd Version: ...

2024-10-29T15:11:33.752Z

2024-10-29T15:11:33.752Z ee6c5f9e-546c-5bff-865e-0004a926f0dd I...

2024-10-29T15:11:33.753Z

2024-10-29T15:11:33.753Z ee6c5f9e-546c-5bff-865e-0004a926f0dd I...

2024-10-29T15:11:33.753Z

ee6c5f9e-546c-5bff-865e-0004a926f0dd INFO Bad

2024-10-29T15:11:33.755Z

2024-10-29T15:11:33.755Z ee6c5f9e-546c-5bff-865e-0004a926f0dd E...

2024-10-29T15:11:33.755Z

ee6c5f9e-546c-5bff-865e-0004a926f0dd ERROR Invoke

Error

{

"errorType": "Error",

"errorMessage": "{}",




"stack": [

"Error: {}",

"    at Runtime.handler (/var/task/index.js:6538:11)",

"    at Runtime.handleOnceNonStreaming (file:///var/runtime/index.mjs:1173:29)"

# Demo – Sample Execution

CloudWatch	Timestamp	Message
		No older events at this moment. <a href="#">Retry</a>
Favorites and recents	2024-10-29T15:12:18.510Z	INIT_START Runtime Version: nodejs:16.v55 Runtime Version ARN: ...
Dashboards	2024-10-29T15:12:18.661Z	START RequestId: 0f5df9fb-edf1-5887-b3f4-a6611083a2cb Version: ...
Alarms  0  0  0	2024-10-29T15:12:18.663Z	2024-10-29T15:12:18.663Z 0f5df9fb-edf1-5887-b3f4-a6611083a2cb I...
Logs	2024-10-29T15:12:18.664Z	2024-10-29T15:12:18.664Z 0f5df9fb-edf1-5887-b3f4-a6611083a2cb I...
Log groups	2024-10-29T15:12:18.664Z	0f5df9fb-edf1-5887-b3f4-a6611083a2cb INFO User8
Log Anomalies	2024-10-29T15:12:18.664Z	2024-10-29T15:12:18.664Z 0f5df9fb-edf1-5887-b3f4-a6611083a2cb I...
Live Tail	2024-10-29T15:12:18.664Z	0f5df9fb-edf1-5887-b3f4-a6611083a2cb INFO User1
Logs Insights	2024-10-29T15:12:18.664Z	2024-10-29T15:12:18.664Z 0f5df9fb-edf1-5887-b3f4-a6611083a2cb I...
Contributor Insights	2024-10-29T15:12:18.664Z	0f5df9fb-edf1-5887-b3f4-a6611083a2cb INFO User3
Metrics	2024-10-29T15:12:18.693Z	END RequestId: 0f5df9fb-edf1-5887-b3f4-a6611083a2cb
X-Ray traces		
Events		

# Demo – Sample Execution

CloudWatch

Favorites and recents

Dashboards

Alarms 0 0 0

▼ Logs

Log groups

Log Anomalies

Live Tail

Logs Insights

Contributor Insights

Metrics

X-Ray traces

Events

Timestamp

Message

No older events at this moment. [Retry](#)

▶ 2024-10-29T15:12:18.532Z

INIT\_START Runtime Version: nodejs:16.v55 Runtime Version ARN: ...

▶ 2024-10-29T15:12:18.691Z

START RequestId: 048036ce-d6cc-56ef-9cd1-69542bbe0f2f Version: ...

▶ 2024-10-29T15:12:18.693Z

2024-10-29T15:12:18.693Z 048036ce-d6cc-56ef-9cd1-69542bbe0f2f I...

▼ 2024-10-29T15:12:18.694Z

2024-10-29T15:12:18.694Z 048036ce-d6cc-56ef-9cd1-69542bbe0f2f I...

2024-10-29T15:12:18.694Z

048036ce-d6cc-56ef-9cd1-69542bbe0f2f INFO UserX

▼ 2024-10-29T15:12:18.699Z

2024-10-29T15:12:18.699Z 048036ce-d6cc-56ef-9cd1-69542bbe0f2f I...

2024-10-29T15:12:18.699Z

048036ce-d6cc-56ef-9cd1-69542bbe0f2f INFO User2

▼ 2024-10-29T15:12:18.700Z

2024-10-29T15:12:18.700Z 048036ce-d6cc-56ef-9cd1-69542bbe0f2f I...

2024-10-29T15:12:18.700Z

048036ce-d6cc-56ef-9cd1-69542bbe0f2f INFO User5

▶ 2024-10-29T15:12:18.720Z

END RequestId: 048036ce-d6cc-56ef-9cd1-69542bbe0f2f

Back to top ^

BadOrdersFn log stream 1  
UserX, User2, User5

# SQS - Summary

- What:
  - Messaging service
  - Decoupling app compute components
- Why:
  - Decrease response time to client; Improve scalability.
- Actors: Producer and Consumer.
- Consumer polls the queue.
- Lambda function consumer.
  - Lambda service polls SQS; Scales handler instances
- Dealing with error cases:
  - Retries (Infinite by default)
  - Dead Letter Queue (DLQ)