AWS News Blog

A New Set of APIs for Amazon SQS Dead-Letter Queue Redrive

by Sébastien Stormacq | on 08 JUN 2023 | in Amazon Simple Queue Service (SQS), Announcements, Developer Tools, Launch, News | Permalink | Comments | Share

Today, we launch a new set of APIs for Amazon SQS. These new APIs allow you to manage dead-letter queue (DLQ) redrive programmatically. You can now use the AWS SDKs or the AWS Command Line Interface (AWS CLI) to programmatically move messages from the DLQ to their original queue, or to a custom queue destination, to attempt to process them again. A DLQ is a queue where Amazon SQS automatically moves messages that are not correctly processed by your consumer application.

To fully appreciate how this new API might help you, let's have a quick look back at history.

Message queues are an integral part of modern application architectures. They allow developers to decouple services by allowing asynchronous and message-based communications between message producers and consumers. In most systems, messages are persisted in shared storage (the queue) until the consumer processes them. Message queues allow developers to build applications that are resilient to temporary service failure. They help prioritize message processing and scale their fleet of worker nodes that process the messages. Message queues are also popular in event-driven architectures.

Asynchronous message exchange is not new in application architectures. The concept of exchanging messages asynchronously between applications appeared in the 1960s and was first made popular when IBM launched TCAM for OS/360 in 1972. The general adoption came 20 years later with IBM MQ Series in 1993 (now IBM MQ) and when Sun Microsystems released Java Messaging Service (JMS) in 1998, a standard API for Java applications to interact with message queues.

AWS launched Amazon SQS on July 12, 2006. Amazon SQS is a highly scalable, reliable, and elastic queuing service that "just works." As Werner wrote at the time: "*We have chosen a concurrency model where the process working on a message automatically acquires a leased lock on that message; if the message is not deleted before the lease expires, it becomes available for processing again. Makes failure handling very simple.*"

On January 29, 2014, we introduced dead-letter queues (DLQ). DLQs help you avoid a message that failed to be processed from staying forever on top of the queue, possibly preventing other messages in the queue from processing. With DLQs, each queue has an associated property telling Amazon SQS how many times a message may be presented for processing (`maxReceiveCount`). Each message also has an associated receive counter (`ReceiveCount`). Each time a consumer application picks up a message for processing, the message receive count is incremented by 1. When `ReceiveCount` > `maxReceiveCount` , Amazon SQS moves the message to your designated DLQ for human analysis and debugging. You generally associate alarms with the DLQ to send notifications when such events happen. Typical reasons to move a message to the DLQ are because they are incorrectly formatted, there are bugs in the consumer application, or it takes too long to process the message.

At AWS re:Invent 2021, AWS announced dead-letter queue redrive on the Amazon SQS console. The redrive addresses the second part of the failed message lifecycle. It allows you to reinject the message in its original queue

to attempt processing it again. After the consumer application is fixed and ready to consume the failed messages, you can redrive the messages from the DLQ back in the source queue or a customized queue destination. It just requires a couple of clicks on the console.

Today, we are adding APIs allowing you to write applications and scripts that handle the redrive programmatically. There is no longer a need to have a human clicking on the console. Using the API increases the scalability of your processes and reduces the risk of human error.

Let's See It in Action

To try out this new API, I open a terminal for a command-line only demo. Before I get started, I make sure I have the latest version of the AWS CLI. On macOS I enter `brew upgrade awscli`.

I first create two queues. One is the dead-letter queue, and the other is my application queue:

```
Bash

5678900/awsnewsblog-dlq \


g-dlq"


 post messages in the DLQ after three delivery attempts


rn\":\"arn:aws:sqs:us-east-2:012345678900:awsnewsblog-dlq\",\"maxReceiveCount\":\"3\"

awsnewsblog"
```

Now that the two queues are ready, I post a message to the application queue:

```
Bash
➜ ~ aws sqs send-message \
            --queue-url https://sqs.us-east-2.amazonaws.com/012345678900/awsnewsblog \
            --message-body "Hello World"
{
"MD5OfMessageBody": "b10a8db164e0754105b7a99be72e3fe5",
"MessageId": "fdc26778-ce9a-4782-9e33-ae73877cfcb2"
}
```

Next, I consume the message, but I don't delete it from the queue. This simulates a crash in the message consumer application. Message consumers are supposed to delete the message after successful processing. I set the `maxReceivedCount` property to 3 when I entered the `redrivePolicy`. I therefore repeat this operation three times to force Amazon SQS to move the message to the dead-letter queue after three delivery attempts. The default visibility timeout is 30 seconds, so I have to wait 30 seconds or more between the retries.

Bash

```
➜ ~ aws sqs receive-message \
            --queue-url https://sqs.us-east-2.amazonaws.com/012345678900/awsnewsblog
{
"Messages": [
{
"MessageId": "fdc26778-ce9a-4782-9e33-ae73877cfcb2",
"ReceiptHandle": "AQEBP8yOfgBlnjlkGXjyeLROiY7xg7cZ6Znq8Aoa0d3Ar4uvTLPrHZptNotNfKRK25xm
"MD5OfBody": "b10a8db164e0754105b7a99be72e3fe5",
"Body": "Hello World"
}
]
}

# wait 30 seconds,
# then repeat two times (for a total of three receive-message API calls)
```

After three processing attempts, the message is not in the queue anymore:

Bash

```
➜  ~ aws sqs receive-message \
            --queue-url  https://sqs.us-east-2.amazonaws.com/012345678900/awsnewsblog
{
    "Messages": []
}
```

The message has been moved to the dead-letter queue. I check the DLQ to confirm (notice the queue URL ending with `-dlq`):

Bash

```
➜  ~ aws sqs receive-message \
            --queue-url  https://sqs.us-east-2.amazonaws.com/012345678900/awsnewsblog
{
    "Messages": [
        {
            "MessageId": "fdc26778-ce9a-4782-9e33-ae73877cfcb2",
```

```
            "ReceiptHandle": "AQEBCLtBMoZYVMMq7fUGNHeCliqE3mFXnkuJ+nOXLK1++uoXWBG31nDe
            "MD5OfBody": "b10a8db164e0754105b7a99be72e3fe5",
            "Body": "Hello World"
        }
    ]
}
```

Now that the setup is ready, let's programmatically redrive the message to its original queue. Let's assume I understand why the consumer didn't correctly process the message and that I fixed the consumer application code. I use `start-message-move-task` on the DLQ to start the asynchronous redrive. There is an optional attribute (`MaxNumberOfMessagesPerSecond`) to control the velocity of the redrive:

Bash

```
➜ ~ aws sqs start-message-move-task \
            --source-arn arn:aws:sqs:us-east-2:012345678900:awsnewsblog-dlq
{
    "TaskHandle": "eyJ0YXNrSWQiOiI4ZGJmNjBiMy00MmUwLTQzYTYtYjg4Zi1iMTZjYWRjY2FkNmEiLCJ
}
```

I can list and check status the of the move tasks I initiated with `list-message-move-tasks` or cancel a running task by calling the `cancel-message-move-task` API:

Bash

```
➜ ~ aws sqs list-message-move-tasks \
            --source-arn arn:aws:sqs:us-east-2:012345678900:awsnewsblog-dlq
{
    "Results": [
        {
            "Status": "COMPLETED",
            "SourceArn": "arn:aws:sqs:us-east-2:012345678900:awsnewsblog-dlq",
            "ApproximateNumberOfMessagesMoved": 1,
            "ApproximateNumberOfMessagesToMove": 1,
            "StartedTimestamp": 1684135792239
        }
    ]
}
```

Now my application can consume the message again from the application queue:

Bash

```
➜  ~ aws sqs receive-message \
            --queue-url  https://sqs.us-east-2.amazonaws.com/012345678900/awsnewsblog
{
    "Messages": [
        {
            "MessageId": "a7ae83ca-cde4-48bf-b822-3d4bc1f4dcae",
            "ReceiptHandle": "AQEB9a+Dm2nvb3VUn9+46j9UsDidU/W6qFwJtXtNWTyfoSDOKT7h73e6
            "MD5OfBody": "b10a8db164e0754105b7a99be72e3fe5",
            "Body": "Hello World"
        }
    ]
}
```

Availability

DLQ redrive APIs are available today in all commercial Regions where Amazon SQS is available.

Redriving the messages from the dead-letter queue to the source queue or a custom destination queue generates additional API calls billed based on existing pricing (starting at $0.40 per million API calls, after the first million, which is free every month). Amazon SQS batches the messages while redriving them from one queue to another. This makes moving messages from one queue to another a simple and low-cost option.

To learn more about DLQ and DLQ redrive, check our documentation.

Remember that we live in an asynchronous world—so should your applications. **Get started today and write your first redrive application**.

-- seb

Comments

What do you think?

22 Responses


Upvote


Funny


Love


Surprised


Angry


Sad

1 Comment Prashanth ▾

Join the discussion...

♡ **Share** **Best** **Newest** **Oldest**

P **Peter Woods**

2 months ago

This is really good news. Unfortunately, the new functions are not supported in Step Functions. A workaround for this is to create a custom lambda that calls the new function and call that from your Step Function (but that's rather messy in my opinion).

o o Reply • Share ›

Subscribe **Privacy** **Do Not Sell My Data**