[Amazon Simple Queue Service (Amazon SQS)](https://aws.amazon.com/sqs/) is a fully managed message queuing service. It enables you to decouple and scale microservices, distributed systems, and serverless applications. A commonly used feature of Amazon SQS is dead-letter queues. The DLQ (dead letter queue) is used to store messages that can't be processed (consumed) successfully.

This post describes how to add automated resilience to an existing SQS queue. It monitors the dead letter queue and moves a message back to the main queue to see if it can be processed again. It also uses a specific algorithm to make sure this is not repeated ad infinitum. Each time it attempts to reprocess the message, the replay time increases until the message is finally considered dead.

I use [Amazon SQS dead-letter queues](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html), [AWS Lambda](https://aws.amazon.com/lambda/), and a specific algorithm to decrease the rate of retries for failed messages. I then package and publish this serverless solution in the [AWS Serverless Application Repository](https://aws.amazon.com/serverless/serverlessrepo/).

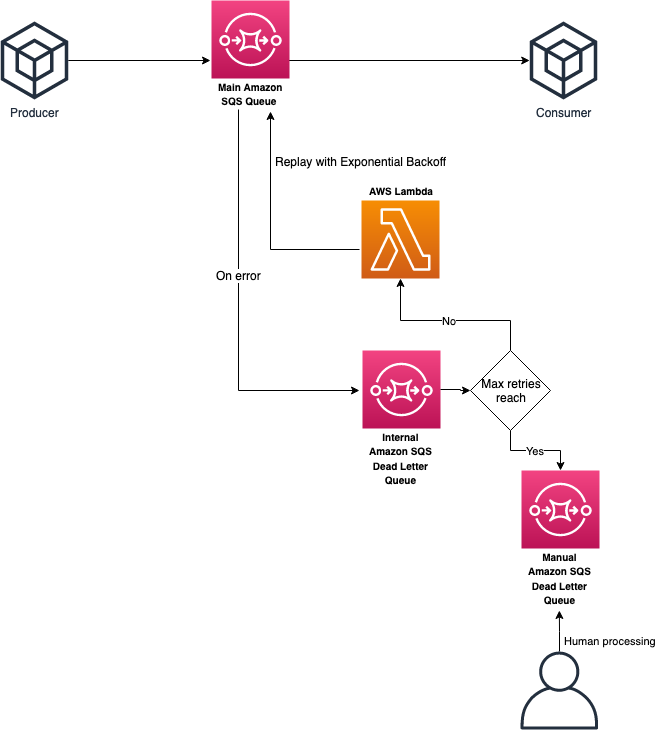
**DEAD-LETTER QUEUES AND MESSAGE REPLAY**

The main task of a dead letter queue (DLQ) is to handle message failure. It allows you to set aside and isolate non processed messages, in order to determine why their processing failed. Often these failed messages are caused by application errors. For example, a consumer application fails to parse a message correctly and throws an unhandled exception. This exception then triggers an error response that sends the message to the DLQ. The AWS documentation contains a [tutorial](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-configure-dead-letter-queue.html) detailing the configuration of an Amazon SQS Dead-Letter Queue.

To process the failed messages, the usual technique is to implement retries. I build this retry mechanism by implementing an exponential backoff algorithm.

The idea behind exponential backoff is to use progressively longer waits between retries for consecutive error responses. Most exponential backoff algorithms use jitter (randomized delay) to prevent successive collisions. This has the effect of spreading the message retries more evenly across time, allowing them to be processed more efficiently.

### Solution overview



The flow of the message sent by the producer to Amazon SQS is as follows:

1. The consumer application fails to process the message.
2. The message is moved from the main Amazon SQS queue to the default dead letter queue as per the component settings.
3. An AWS Lambda function is configured with the Amazon SQS main dead letter queue as an event source. It receives and send back the message to the original queue adding a [message timer](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-message-timers.html).
4. The message timer is defined by the exponential backoff and jitter algorithm.
5. You can limit the number of retries. If the message exceeds this limit, the message is moved to a second DLQ where an operator will process it manually.

### HOW DOES THE REPLAY FUNCTION WORK?

Each time the Amazon SQS main dead letter queue receives a message, it triggers AWS Lambda to run the replay function. The replay code uses an SQS message attribute `sqs-dlq-replay-nb` as a persistent counter for the current number of retries attempted. The number of retries is compared to the maximum number (defined in the application configuration file). If it exceeds the maximum, the message is moved to the human operated queue. If not, the function uses the AWS Lambda event data to build a new message for the Amazon SQS main queue. Finally it updates the retry counter, adds a new message timer to the message, and it sends the message back (replays) to the main queue.

def handler(event, context):  
    """Lambda function handler."""  
    for record in event['Records']:  
        nbReplay = 0  
        # number of replay  
        if 'sqs-dlq-replay-nb' in record['messageAttributes']:  
            nbReplay = int(record['messageAttributes']['sqs-dlq-replay-nb']["stringValue"])  
  
        nbReplay += 1  
        if nbReplay > config.MAX\_ATTEMPS:  
            raise MaxAttempsError(replay=nbReplay, max=config.MAX\_ATTEMPS)  
  
        # SQS attributes  
        attributes = record['messageAttributes']  
        attributes.update({'sqs-dlq-replay-nb': {'StringValue': str(nbReplay), 'DataType': 'Number'}})  
  
        \_sqs\_attributes\_cleaner(attributes)  
  
        # Backoff  
        b = backoff.ExpoBackoffFullJitter(base=config.BACKOFF\_RATE, cap=config.MESSAGE\_RETENTION\_PERIOD)  
        delaySeconds = b.backoff(n=int(nbReplay))  
  
        # SQS  
        SQS.send\_message(  
            QueueUrl=config.SQS\_MAIN\_URL,  
            MessageBody=record['body'],  
            DelaySeconds=int(delaySeconds),  
            MessageAttributes=record['messageAttributes']  
        )

**BUILDING AND PACKAGING THE SERVERLESS APPLICATION**

Our solution is available and shared as reusable application to the [AWS Serverless Application Repository (AWS SAR).](https://aws.amazon.com/serverless/serverlessrepo/)

A serverless application is a combination of Lambda functions, event sources, and other resources that work together to perform tasks. For this application, we use the open-source [AWS Serverless Application Model (AWS SAM)](https://aws.amazon.com/serverless/sam/) to declare the parameters, resources, and metadata of the *SQS-DLQ-Replay* solution.

We declare several sections in our AWS SAM template:

* “Metadata” section specifies application information that we want to publish in the repository AWS SAR.
* “Parameters” section contains default Amazon SQS settings, as well as the configuration for our exponential backoff algorithm.
* “Resources” section defines our three Amazon SQS queues and where to host the above AWS Lambda function.
* “Outputs” section lists arn of our three Amazon SQS queues and AWS Lambda we want to output.

See the full yaml here. (github link)

AWS SAM is backed by a command line interface (AWS SAM CLI). AWS SAM CLI helps to build, package, and publish our solution to AWS SAR.

# Build and compile dependencies for Lambda functions

sam build

# Package for deployment

same package --template-file template.yaml --output-template-file packaged.yaml --s3-bucket <your-bucket-name>

# Publish to AWS SAR

sam publish --template packaged.yaml

### How to use the application

You can use this serverless application via:

* The AWS Management Console: Inside the Lambda service, choose the “Browse serverless app repository” option to create your function. Select “sqs-dlq-replay" application in the public applications repository. Then, configure the application with the default SQS parameters and the replay feature parameters.
* The Serverless Framework, as described by Yan Cui in [this blog post](https://medium.com/theburningmonk-com/how-to-include-serverless-repository-apps-in-serverless-yml-6d8233c5d684).
* Your own CloudFormation template with the use of `AWS::ServerlessRepo::Application` resource, as described in the [documentation](https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#awsserverlessapplication).

Here is an example of an AWS CloudFormation template using the SAR application:

AWSTemplateFormatVersion: '2010-09-09'

Transform: AWS::Serverless-2016-10-31

Resources:

ReplaySqsQueue:

Type: AWS::Serverless::Application

Properties:

Location:

ApplicationId: arn:aws:serverlessrepo:eu-west-1:862440218923:applications~sqs-dlq-replay

SemanticVersion: 1.0.0

Parameters:

BackoffRate: "2"

MaxAttempts: "3"

### CONCLUSION

I describe how an exponential backoff algorithm (with jitter) enhances the message processing capabilities of an Amazon SQS queue. You can now find the *SQS-DLQ-Replay* application in the AWS Serverless Application Repository.

To get started with dead-letter queues in Amazon SQS, see the following topics in the Amazon SQS Developer Guide:

* [Using Amazon SQS Dead-Letter Queues](http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html)
* [Monitoring Amazon SQS Using CloudWatch](http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/MonitorSQSwithCloudWatch.html)

To go further and implement your replay mechanism as described in this article:

* [Increase your knowledge on the backoff algorithm reading this blog post by Marc Brooker.](https://aws.amazon.com/blogs/architecture/exponential-backoff-and-jitter/)
* [Leverage SQS Message Timers feature to manage the message visibility in the queue.](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-message-timers.html)