개발자 가이드

Amazon Simple Queue Service

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Simple Queue Service: 개발자 가이드

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께 사용되어서는 안되며, 고객에게 혼동을 일으키거나 Amazon 브랜드 이미지를 떨어뜨리고 폄하하는 방식으로 이용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

Table of Contents

Amazon SQS란 무엇인가요? 1
Amazon SQS 사용의 이점 1
기본 아키텍처 1
분산 대기열 2
메시지 수명 주기 2
Amazon SQS, Amazon MQ 및 Amazon SNS 간의 차이점 4
시작 6
설정 6
1단계: AWS 계정 및 IAM 사용자 생성 6
2단계: 프로그래밍 방식 액세스 권한 부여 8
3단계: 예제 코드 사용 준비 10
다음 단계 10
Amazon SQS 콘솔 이해 10
대기열 유형 12
Amazon SQS에서 요청-응답 시스템 구현 14
표준 대기열 생성 15
대기열 만들기 15
표준 대기열을 사용하여 메시지 전송 17
FIFO 대기열 만들기 17
대기열 생성 17
FIFO 대기열을 사용하여 메시지 전송 20
일반적인 작업 20
대기열 관리 22
대기열 편집 22
메시지 수신 및 삭제 22
대기열이 비어 있는지 확인 24
대기열 삭제 25
대기열 제거 26
표준 대기열 27
Amazon SQS 최소 1회 전송 27
대기열 및 메시지 식별자 28
표준 대기열의 식별자 28
FIFO 대기열 30
FIFO 대기열 주요 용어 31

FIFO 전송 로직 ... 32
메시지 보내기 ... 32
메시지 수신 ... 33
다수의 재시도 횟수 ... 34
FIFO 동작에 대한 추가 참고 사항 ... 34
이해를 돕기 위한 예 ... 35
정확히 1회 처리 ... 35
표준 대기열에서 FIFO 대기열로 이동 ... 36
FIFO 대기열 및 Lambda 동시성 동작 ... 37
FIFO 대기열 메시지 그룹화 ... 37
FIFO 대기열을 사용한 Lambda 동시성 ... 38
사용 사례 예제 ... 38
FIFO 대기열의 높은 처리량 ... 38
사용 사례 ... 39
파티션 및 데이터 배포 ... 39
FIFO 대기열의 높은 처리량 활성화 ... 42
대기열 및 메시지 식별자 ... 43
FIFO 대기열의 식별자 ... 28
FIFO 대기열의 추가 식별자 ... 44
할당량 ... 46
FIFO 대기열 할당량 ... 46
Amazon SQS 할당량 ... 46
표준 대기열 할당량 ... 47
메시지 할당량 ... 49
정책 할당량 ... 54
특징 및 기능 ... 56
배달 못한 편지 대기열 ... 56
배달 못한 편지 대기열에 대한 정책 사용 ... 56
배달 못한 편지 대기열의 메시지 보존 기간 이해 ... 57
배달 못한 편지 대기열 구성 ... 57
배달 못한 편지 대기열 리드라이브 구성 ... 58
CloudTrail 업데이트 및 권한 요구 사항 ... 68
Amazon CloudWatch를 사용하여 배달 못한 편지 대기열에 대한 경보 생성 ... 72
Amazon SQS의 메시지 메타데이터 ... 72
메시지 속성 ... 72
메시지 시스템 속성 ... 76

메시지를 처리하는 데 필요한 리소스 ... 76
목록 대기열 페이지 매김 ... 77
비용 할당 태그 ... 77
짧은 폴링 및 긴 폴링 ... 78
짧은 폴링을 사용한 메시지 사용 ... 79
긴 폴링을 사용한 메시지 사용 ... 79
긴 폴링과 짧은 폴링의 차이점 ... 80
제한 시간 초과 ... 80
가시성 제한 시간 사용 사례 ... 81
표시 제한 시간 설정 및 조정 ... 81
비행 중 메시지 및 할당량 ... 82
표준 및 FIFO 대기열의 표시 제한 시간 이해 ... 82
실패 처리 ... 83
표시 제한 시간 변경 및 종료 ... 83
모범 사례 ... 83
공정 대기열 ... 84
FIFO 대기열과의 차이점 ... 86
공정 대기열 사용 ... 86
공정 대기열 CloudWatch 지표 ... 86
지연 대기열 ... 87
임시 대기열 ... 88
가상 대기열 ... 89
요청-응답 메시징 패턴(가상 대기열) ... 90
예제 시나리오: 로그인 요청 처리 ... 91
대기열 정리 ... 93
메시지 타이머 ... 94
EventBridge 파이프에 액세스 ... 94
대규모 메시지 관리 ... 96
Java용 확장 클라이언트 라이브러리 사용 ... 96
Python용 확장 클라이언트 라이브러리 사용 ... 102
Amazon SQS 구성 ... 106
Amazon SQS에 대한 ABAC ... 106
ABAC란 무엇입니까? ... 106
Amazon SQS에 ABAC를 사용해야 하는 이유는 무엇입니까? ... 107
액세스 제어용 태그 지정 ... 107
IAM 사용자 및 Amazon SQS 대기열 생성 ... 108

속성 기반 액세스 제어 테스트 ... 111
대기열 파라미터 구성 ... 113
액세스 정책 구성 ... 114
대기열에 대해 SSE-SQS 구성 ... 115
대기열에 대해 SSE-KMS 구성 ... 116
대기열에 대해 태그 구성 ... 118
대기열에서 주제 구독 ... 118
교차 계정 구독 ... 119
리전 간 구독 ... 120
Lambda 트리거 구성 ... 120
사전 조건 ... 121
EventBridge를 사용하여 알림 자동화 ... 122
메시지 속성 ... 123
모범 사례 ... 125
오류 처리 및 문제가 있는 메시지 ... 125
Amazon SQS에서 요청 오류 처리 ... 125
Amazon SQS에서 문제가 있는 메시지 캡처 ... 126
Amazon SQS에서 Dead Letter Queue(DLQ) 보존 설정 ... 126
메시지 중복 제거 및 그룹화 ... 126
Amazon SQS에서 일관되지 않은 메시지 처리 방지 ... 127
메시지 중복 제거 ID 사용 ... 127
메시지 그룹 ID 사용 ... 129
수신 요청 시도 ID 사용 ... 131
메시지 처리 및 타이밍 ... 131
Amazon SQS에서 적시에 메시지 처리 ... 131
Amazon SQS에서 긴 폴링 설정 ... 132
Amazon SQS에서 적절한 폴링 모드 사용 ... 133
Java SDK 예제 ... 134
서버 측 암호화 사용 ... 134
기존 대기열에 SSE 추가 ... 134
대기열에 SSE 비활성화 ... 135
SSE를 사용하여 대기열 생성 ... 136
SSE 속성 검색 ... 136
태그 구성 ... 137
태그 나열 ... 137
태그 추가 또는 업데이트 ... 137

태그 제거 ... 138
메시지 속성 보내기 ... 139
속성 정의 ... 139
속성을 포함하는 메시지 전송 ... 141
API 사용 ... 142
AWS JSON 프로토콜을 사용하여 쿼리 API 요청 수행 ... 143
엔드포인트 구성 ... 143
POST 요청 만들기 ... 144
아마존 SQS JSON API 응답 해석 ... 145
Amazon SQS AWS JSON 프로토콜 FAQs ... 146
쿼리 프로토콜을 사용하여 AWS 쿼리 API 요청 수행 ... 149
엔드포인트 구성 ... 149
GET 요청 만들기 ... 150
POST 요청 만들기 ... 144
Amazon SQS XML API 응답 해석 ... 151
요청 인증 ... 153
HMAC-SHA를 사용한 기본 인증 프로세스 ... 153
1부: 사용자의 요청 ... 155
2부:의 응답 AWS ... 155
배치 작업 ... 156
메시지 작업 일괄 처리 ... 157
Amazon SQS로 클라이언트 측 버퍼링 및 요청 배치 처리 활성화 ... 158
Amazon SQS로 수평적 조정과 작업 배치 처리를 사용하여 처리량 증대 ... 167
AWS SDKs 작업 ... 180
JMS 사용 ... 182
사전 조건 ... 182
Java 메시징 라이브러리 사용 ... 183
JMS 연결 생성 ... 183
Amazon SQS 대기열 생성 ... 184
메시지 동기식 전송 ... 185
메시지 동기식 수신 ... 187
메시지 비동기식 수신 ... 188
클라이언트 승인 모드 사용 ... 189
정렬되어 있지 않은 승인 모드 사용 ... 190
JMS 클라이언트를 다른 Amazon SQS 클라이언트와 함께 사용 ... 191
표준 대기열에 JMS를 사용하는 Java 작업 예제 ... 192

ExampleConfiguration.java ... 193
TextMessageSender.java ... 195
SyncMessageReceiver.java ... 197
AsyncMessageReceiver.java ... 199
SyncMessageReceiverClientAcknowledge.java ... 201
SyncMessageReceiverUnorderedAcknowledge.java ... 205
SpringExampleConfiguration.xml ... 208
SpringExample.java ... 210
ExampleCommon.java ... 212
지원되는 JMS 1.1 구현 ... 214
지원되는 공통 인터페이스 ... 214
지원되는 메시지 유형 ... 214
지원되는 메시지 승인 모드 ... 214
JMS 정의된 헤더 및 예약된 속성 ... 215
자습서 ... 216
를 사용하여 Amazon SQS 대기열 생성 AWS CloudFormation ... 216
VPC에서 메시지 전송 ... 218
1단계: Amazon EC2 키 페어 생성 ... 219
2단계: AWS 리소스 생성 ... 219
3단계: EC2 인스턴스에 공개적으로 액세스할 수 없는지 확인 ... 220
4단계: Amazon SQS용 Amazon VPC 엔드포인트 생성 ... 221
5단계: Amazon SQS 대기열에 메시지 전송 ... 222
코드 예제 ... 224
기본 사항 ... 237
Hello Amazon SNS ... 238
작업 ... 250
시나리오 ... 411
메시징 애플리케이션 생성 ... 411
메신저 애플리케이션 생성 ... 412
Amazon Textract 탐색기 애플리케이션 생성 ... 413
FIFO 주제에 생성 및 게시 ... 415
동영상에서 사람과 객체 감지 ... 427
S3를 사용하여 대용량 메시지 관리 ... 428
S3 이벤트 알림 처리 ... 432
대기열에 메시지 게시 ... 436
메시지 일괄 전송 및 수신 ... 551

Amazon SQS에서 .NET용 AWS 메시지 처리 프레임워크 사용 582
Amazon SQS Java 메시징 라이브러리를 사용하여 JMS 인터페이스 작업 582
대기열 태그 작업 604
서버리스 예제 608
Amazon SQS 트리거에서 간접적으로 Lambda 함수 간접 호출 608
Amazon SQS 트리거로 Lambda 함수에 대한 배치 항목 실패 보고 617
문제 해결 628
액세스 거부 오류 628
Amazon SQS 대기열 정책 및 IAM 정책 629
AWS Key Management Service (AWS KMS) 권한 629
VPC 엔드포인트 정책 631
조직 서비스 제어 정책 631
API 오류 632
QueueDoesNotExist 오류 632
InvalidAttributeValue 오류 632
ReceiptHandle 오류 633
DLQ 및 DLQ 리드라이브 문제 634
DLQ 문제 634
DLQ 리드라이브 문제 635
FIFO 제한 문제 637
ReceiveMessage API 직접 호출에 대해 반환되지 않은 메시지 638
빈 대기열 638
처리 중 한도에 도달 638
메시지 지연 638
메시지 처리 중 638
폴링 메서드 639
네트워크 오류 639
ETIMEOUT error 639
UnknownHostException error 640
X-Ray를 사용하여 대기열 문제 해결 641
보안 642
데이터 보호 642
데이터 암호화 643
인터네트워크 트래픽 개인 정보 654
연결을 위한 듀얼 스택 엔드포인트 사용 656
자격 증명 및 액세스 관리 657

대상 657
ID를 통한 인증 657
정책을 사용하여 액세스 관리 660
개요 663
Amazon Simple Queue Service가 IAM으로 작동하는 방식 669
AWS 관리형 정책 676
문제 해결 678
정책 사용 680
로깅 및 모니터링 725
API 직접 호출 로깅 728
대기열 모니터링 731
규정 준수 확인 747
복원성 748
분산 대기열 749
인프라 보안 749
모범 사례 750
대기열에 대한 공개적 액세스 방지 750
최소 권한 액세스 구현 750
Amazon SQS 액세스가 필요한 애플리케이션 및 AWS 서비스에 IAM 역할 사용 751
서버 측 암호화 구현 751
전송 중인 데이터의 암호화 강제 시행 752
VPC 엔드포인트를 사용한 Amazon SQS 액세스 고려 752
관련 리소스 753
문서 기록 754
..... dcclxii

Amazon Simple Queue Service란 무엇인가요?

Amazon Simple Queue Service(Amazon SQS)는 내구력 있고 가용성이 뛰어난 보안 호스팅 대기열을 제공하며 이를 통해 분산 소프트웨어 시스템과 구성 요소를 통합 및 분리할 수 있습니다. Amazon SQS는 배달 못한 편지 대기열 및 비용 할당 태그와 같은 공용 구성을 제공합니다. AWS SDK가 지원하는 모든 프로그래밍 언어를 사용하여 액세스할 수 있는 일반 웹 서비스 API를 제공합니다.

Amazon SQS 사용의 이점

- 보안 - Amazon SQS 대기열에 메시지를 보내고 받을 수 있는 사람을 제어합니다. 기본 Amazon SQS 관리형 서버 측 암호화(SSE)를 사용하거나 AWS Key Management Service (AWS KMS)에서 관리되는 사용자 지정 SSE 키를 사용하여 대기열에 있는 메시지의 콘텐츠를 보호함으로써 민감한 데이터를 전송하도록 선택할 수 있습니다.
- 내구성 – 메시지의 안전을 위해 Amazon SQS는 메시지를 여러 서버에 저장합니다. 표준 대기열은 최소 1회 메시지 전송을 지원하고 FIFO 대기열은 정확히 1회 메시지 처리 및 높은 처리량 모드를 지원합니다.
- 가용성 – Amazon SQS는 중복 인프라를 사용하여 메시지에 대한 고도의 동시 액세스와 메시지 생성 및 소비에 대한 고가용성을 제공합니다.
- 확장성 – Amazon SQS는 버퍼링된 요청을 각각 독립적으로 처리하여 프로비저닝 지침 없이도 로드 증가 또는 급증을 처리하기 위해 투명하게 확장할 수 있습니다.
- 신뢰성 – Amazon SQS는 처리 중에 메시지를 잠그므로 여러 생산자와 소비자가 동시에 메시지를 전송 및 수신할 수 있습니다.
- 사용자 지정 – 대기열이 똑같을 필요는 없습니다. 예를 들어 대기열에서 기본 지연 시간을 설정할 수 있습니다. Amazon Simple Storage Service(Amazon S3) 또는 Amazon DynamoDB를 사용하여 1MiB보다 큰 메시지의 내용을 저장할 수 있으며, Amazon SQS는 Amazon S3 객체에 대한 포인터를 보유하거나 큰 메시지를 더 작은 메시지로 분할할 수 있습니다. DynamoDB

Amazon SQS 기본 아키텍처

이 섹션에서는 분산 메시징 시스템의 구성 요소를 설명하고 Amazon SQS 메시지의 수명 주기를 설명합니다.

분산 대기열

분산 메시징 시스템에는 분산 시스템의 구성 요소, 대기열(Amazon SQS 서버에 배포됨) 및 대기열의 메시지라는 세 가지 주요 부분이 있습니다.

다음 시나리오에서 시스템에는 여러 생산자(메시지를 대기열로 전송하는 구성 요소) 및 소비자(대기열의 메시지를 수신하는 구성 요소)가 있습니다. 대기열(메시지 A~E 유지)은 여러 Amazon SQS 서버에서 메시지를 중복으로 저장합니다.

메시지 수명 주기

다음 시나리오에서는 대기열에서 생성부터 삭제까지 Amazon SQS 메시지의 수명 주기를 설명합니다.

1

생산자(구성 요소 1)는 메시지 A를 대기열로 전송하고 메시지는 Amazon SQS 서버에 중복으로 배포됩니다.

2

소비자(구성 요소 2)가 메시지를 처리할 준비가 되면 대기열의 메시지를 소비하고 메시지 A가 반환됩니다. 메시지 A는 처리되는 동안 대기열에 그대로 남아 있고 제한 시간 초과가 지속되는 동안 후속 수신 요청으로 반환되지 않습니다.

3

소비자(구성 요소 2)는 가시성 제한 시간이 만료될 때 메시지가 수신되고 다시 처리되지 않도록 대기열에서 메시지 A를 삭제합니다.

> **Note**
>
> Amazon SQS는 최대 메시지 보존 기간 넘게 대기열에 유지된 메시지를 자동으로 삭제합니다. 기본 메시지 보존 기간은 4일입니다. 그러나 `SetQueueAttributes` 작업을 사용하면 메시지 보존 기간을 60초에서 1,209,600초(14일) 사이로 설정할 수 있습니다.

Amazon SQS, Amazon MQ 및 Amazon SNS 간의 차이점

Amazon SQS, Amazon SNS, Amazon MQ는 확장성이 뛰어나고 사용하기 쉬운 관리형 메시징 서비스를 제공하며, 각 서비스는 분산 시스템 내의 특정 역할에 맞게 설계되었습니다. 다음은 이러한 서비스 간의 차이점에 대한 확장된 개요입니다.

Amazon SQS는 분산 소프트웨어 시스템 및 구성 요소를 대기열 서비스로 분리 및 확장합니다. 일반적으로 단일 구독자를 통해 메시지를 처리하므로 순서 및 손실 방지가 중요한 워크플로에 적합합니다. 더 광범위한 배포를 위해 Amazon SQS를 Amazon SNS와 통합하면 팬아웃 메시징 패턴이 활성화되어 한 번에 여러 구독자에게 메시지를 효과적으로 푸시할 수 있습니다.

Amazon SNS를 사용하면 게시자가 커뮤니케이션 채널 역할을 하는 주제를 통해 여러 구독자에게 메시지를 보낼 수 있습니다. 구독자는 Amazon Data Firehose, Amazon SQS, Lambda, HTTP, 이메일, 모바일 푸시 알림 및 모바일 문자 메시지(SMS)와 같이 지원되는 엔드포인트 유형을 사용하여 게시된 메시지를 수신할 수 있습니다. 이 서비스는 실시간 사용자 참여 또는 경보 시스템과 같이 즉각적인 알림이 필요한 시나리오에 적합합니다. 구독자가 오프라인 상태일 때 메시지 손실을 방지하기 위해 Amazon SNS를 Amazon SQS 대기열 메시지와 통합하면 일관된 전송이 보장됩니다.

Amazon MQ는 Apache ActiveMQ 및 RabbitMQ와 함께 AMQP 및 MQTT와 같은 표준 메시징 프로토콜을 지원하여 기존 메시지 브로커에서 마이그레이션하려는 기업에 가장 적합합니다. 중요한 재구성 없이 안정적이고 신뢰할 수 있는 메시징이 필요한 레거시 시스템과 호환됩니다.

다음 차트는 각 서비스의 리소스 유형에 대한 개요를 제공합니다.

리소스 유형	Amazon SNS	Amazon SQS	Amazon MQ
동기식	아니요	아니요	예

리소스 유형	Amazon SNS	Amazon SQS	Amazon MQ
비동기식	예	예	예
대기열	아니요	예	예
게시자-구독자 메시징	예	아니요	예
메시지 브로커	아니요	아니요	예

Amazon SQS와 Amazon SNS는 무제한에 가까운 확장성과 간편한 API를 활용할 수 있는 새로운 애플리케이션에 사용하면 좋습니다. 일반적으로 종량제 요금으로 대용량 애플리케이션에 더욱 비용 효과적인 솔루션을 제공합니다. JMS 등과 같은 API 또는 AMQP(Advanced Message Queuing Protocol), MQTT, OpenWire 및 STOMP(Simple Text Oriented Message Protocol)와 같은 프로토콜과 호환되는 기존 메시지 브로커의 애플리케이션을 마이그레이션할 때에는 Amazon MQ를 사용하면 좋습니다.

Amazon SQS 시작하기

이 주제에서는 Amazon SQS 콘솔을 사용하여 표준 대기열과 FIFO 대기열을 생성하고 관리하는 방법을 안내합니다. 콘솔을 탐색하고, 대기열 속성을 보고, 대기열 유형을 구분하는 방법을 알아봅니다. 주요 작업에는 메시지 전송, 수신 및 구성, 표시 제한 시간 및 메시지 보존과 같은 파라미터 조정, 정책을 통한 대기열 액세스 관리가 포함됩니다.

주제

- Amazon SQS 설정
- Amazon SQS 콘솔 이해
- Amazon SQS 대기열 유형
- Amazon SQS 표준 대기열 생성 및 메시지 전송
- Amazon SQS FIFO 대기열 생성 및 메시지 전송
- Amazon SQS 시작을 위한 일반적인 작업

Amazon SQS 설정

Amazon SQS를 처음 사용하려면 먼저 다음 단계를 완료해야 합니다.

1단계: AWS 계정 및 IAM 사용자 생성

AWS 서비스에 액세스하려면 먼저 AWS 제품을 사용할 수 있는 Amazon.com 계정인 AWS 계정를 생성해야 합니다. AWS 계정 를 사용하여 활동 및 사용 보고서를 보고 인증 및 액세스를 관리할 수 있습니다.

Amazon SQS 작업에 AWS 계정 루트 사용자를 사용하지 않으려면 Amazon SQS에 대한 관리 액세스 권한이 필요한 각 사용자에 대해 IAM 사용자를 생성하는 것이 좋습니다.

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

2단계: 프로그래밍 방식 액세스 권한 부여

Amazon SQS 작업(예: Java 사용 또는를 통해 AWS Command Line Interface)을 사용하려면 액세스 키 ID와 보안 액세스 키가 필요합니다.

> **Note**
> 액세스 키 ID 및 보안 액세스 키는에 고유합니다 AWS Identity and Access Management. Amazon EC2 키 페어와 같은 다른 AWS 서비스의 자격 증명과 혼동하지 마세요.

사용자는 AWS 외부에서와 상호 작용하려는 경우 프로그래밍 방식으로 액세스해야 합니다 AWS Management Console. 프로그래밍 방식 액세스를 부여하는 방법은에 액세스하는 사용자 유형에 따라 다릅니다 AWS.

사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 를 사용하도록 구성을 AWS IAM Identity Center 참조하세요. • AWS SDKs, 도구 및 AWS APIs의 경우 SDK 및 도구 참조 안내서의 IAM Identity Center 인증을 참조하세요. AWS SDKs
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	IAM 사용 설명서의 AWS 리소스에서 임시 자격 증명 사용의 지침을 따릅니다.
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 IAM 사용자 자격 증명을 사용하여 인증을 참조하세요. • AWS SDKs 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용하여 인증을 참조하세요. AWS SDKs

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		• AWS APIs 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하세요.

3단계: 예제 코드 사용 준비

이 가이드에는 Java용 AWS SDK를 사용하는 예제가 포함되어 있습니다. 예제 코드를 실행하려면 Java 2.0용 AWS SDK 시작하기에 나와 있는 설정 지침을 따르세요.

Go, JavaScript, Python 및 Ruby와 같은 다른 프로그래밍 언어로 AWS 애플리케이션을 개발할 수 있습니다. 자세한 내용은 빌드 기반 도구를 참조하세요 AWS.

> **Note**
>
> AWS Command Line Interface (AWS CLI) 또는 Windows PowerShell과 같은 도구로 코드를 작성하지 않고도 Amazon SQS를 탐색할 수 있습니다. AWS CLI 명령 참조의 Amazon SQS 섹션에서 AWS CLI 예제를 찾을 수 있습니다. Windows PowerShell 예제는 AWS Tools for PowerShell Cmdlet 참조의 Amazon Simple Queue Service 섹션에서 찾을 수 있습니다.

다음 단계

이제 AWS Management Console을 사용하여 Amazon SQS 대기열 및 메시지 관리를 시작할 준비가 되었습니다.

Amazon SQS 콘솔 이해

Amazon SQS 콘솔을 열 때 탐색 창에서 대기열을 선택합니다. 대기열 페이지는 활성 리전의 모든 대기열에 대한 정보를 제공합니다.

각 대기열 항목은 유형 및 키 속성을 포함하여 대기열에 대한 필수 정보를 제공합니다. 최대 처리량과 최상의 메시지 순서에 최적화된 표준 대기열은 엄격한 메시지 시퀀싱이 필요한 애플리케이션의 메시지 순서와 고유성을 우선시하는 선입선출(FIFO) 대기열과 구별됩니다.

대화형 요소 및 작업

대기열 페이지에는 대기열을 관리하기 위한 여러 옵션이 있습니다.

1. 빠른 작업 - 각 대기열 이름에 인접한 드롭다운 메뉴를 사용하면 메시지 전송, 메시지 보기 또는 삭제, 트리거 구성, 대기열 자체 삭제와 같은 일반적인 작업에 빠르게 액세스할 수 있습니다.
2. 세부 보기 및 구성 - 대기열 이름을 클릭하면 대기열 설정 및 구성에 대해 자세히 알아볼 수 있는 세부 정보 페이지가 열립니다. 여기서 메시지 보존 기간, 표시 제한 시간 및 최대 메시지 크기와 같은 파라미터를 조정하여 애플리케이션의 요구 사항에 맞게 대기열을 조정할 수 있습니다.

리전 선택 및 리소스 태그

대기열에 효과적으로 액세스하고 관리 AWS 리전 하려면 올바른에 있어야 합니다. 또한 리소스 태그를 활용하여 대기열을 구성하고 분류하여 AWS 공유 환경 내에서 리소스 관리, 비용 할당 및 액세스 제어를 개선하는 것이 좋습니다.

Amazon SQS 콘솔 내에서 제공되는 특성과 기능을 활용하여 메시징 인프라를 효율적으로 관리하고 대기열 성능을 최적화하며 애플리케이션에 안정적인 메시지 전송을 보장할 수 있습니다.

Amazon SQS 대기열 유형

Amazon SQS는 표준 대기열과 FIFO 대기열이라는 두 가지 유형의 대기열을 지원합니다. 다음 표를 사용하여 필요에 가장 적합한 대기열을 결정합니다.

표준 대기열	FIFO 대기열
무제한 처리량 - 표준 대기열은 작업(`SendMessage` , `ReceiveMessage` 또는 `DeleteMessage`)별 초당 거의 무제한의 매우 높은 API 직접 호출 수를 지원합니다. 이렇게 처리량이 높기 때문에 실시간 데이터 스트리밍 또는 대규모 애플리케이션과 같이 대량의 메시지를 빠르게 처리해야 하는 사용 사례에 적합합니다. 표준 대기열은 수요에 따라 자동으로 확장되지만 특히 워크로드가 많은 리전에서는 최적의 성능을 보장하기 위해 사용 패턴을 모니터링하는 것이 중요합니다. 최소 1회 전송 - 최소 1회 전송을 보장합니다. 즉, 모든 메시지가 최소 1회 전송되지만 경우에 따라 재시도 또는 네트워크 지연으로 인해 메시지가 두 번 이상 전송될 수 있습니다. 동일한 메시지를 여러 번 처리해도 시스템 상태에 영향을 주지 않도록 멱등성 작업을 사용하여 잠재적인 중복 메시지를 처리하도록 애플리케이션을 설계해야 합니다. 최선의 순서 지정 - 최선의 순서 지정을 제공합니다. 즉, Amazon SQS는 전송된 순서대로 메시지를 전송하려고 시도하지만 이를 보장하지는 않습니다. 경우에 따라 특히 처리량이 많거나 장애 복구가 필요한 조건에서 메시지가 잘못 도착할 수 있습니다. 메시지 처리 순서가 중요한 애플리케이션의 경우, 애플리케이션 내에서 재주문 로직을 처리하거나 엄격한 순서 보장을 위해서 FIFO 대기열을 사용해야 합니다.	높은 처리량 - 배치 처리를 사용하는 경우 FIFO 대기열은 API 메서드(`SendMessageBatch` , `ReceiveMessage` 또는 `DeleteMessageBatch`)별 초당 최대 3,000개의 메시지를 처리합니다. 이 처리량은 초당 300개의 API 직접 호출에 의존하며, 각 API 직접 호출은 10개의 메시지를 배치 처리합니다. 높은 처리량 모드를 활성화하면 메시지 그룹 내에서 순서를 완화하여 30,000건의 초당 트랜잭션(TPS)까지 확장할 수 있습니다. 배치 처리가 없으면 FIFO 대기열은 API 메서드(`SendMessage` , `ReceiveMessage` 또는 `DeleteMessage`)별 초당 최대 300개의 API 직접 호출을 지원합니다. 처리량이 더 필요한 경우 AWS Support Center를 통해 할당량 증가를 요청할 수 있습니다. 높은 처리량 모드를 활성화하려면 Amazon SQS에서 FIFO 대기열의 높은 처리량 활성화 섹션을 참조하세요. 정확히 1회 처리 - FIFO 대기열은 각 메시지를 한 번 전송하고 처리 및 삭제할 때까지 사용 가능한 상태로 유지합니다. `MessageDeduplicationId` 또는 콘텐츠 기반 중복 제거와 같은 기능을 사용하면 네트워크 문제 또는 제한 시간으로 인해 재시도할 때에도 중복 메시지를 방지할 수 있습니다. 선입선출 전송 - FIFO 대기열은 각 메시지 그룹 내에서 전송되는 순서대로 메시지를 수신하도록 합니다. 여러 그룹에 메시지를 배포하여 각 그룹

표준 대기열	FIFO 대기열
내구성 및 중복성 - 표준 대기열은 여러 AWS 가용 영역에 걸쳐 각 메시지의 여러 복사본을 저장하여 높은 내구성을 보장합니다. 이렇게 하면 인프라 장애가 발생하더라도 메시지가 손실되지 않습니다. 표시 제한 시간 - Amazon SQS를 사용하면 메시지 수신 후 숨겨진 상태로 유지되는 기간을 제어하도록 표시 제한 시간을 구성할 수 있으므로, 메시지가 완전히 처리되거나 제한 시간이 만료될 때까지 다른 소비자가 메시지를 처리하지 않도록 할 수 있습니다.	내에서 순서를 유지하면서 병렬로 처리할 수 있습니다.

표준 대기열	FIFO 대기열
처리량이 중요한 경우 표준 대기열을 사용하여 애플리케이션 간에 데이터를 전송합니다. 다음 예를 참조하세요. • 실시간 사용자 요청을 폭넓은 배경 작업과 분리합니다. 백그라운드에서 크기 조정 또는 인코딩과 같은 작업을 처리하는 동안 사용자가 미디어를 빠르게 업로드하도록 허용하여 시스템에 과부하를 가하지 않고 빠른 응답 시간을 보장합니다. • 여러 워커 노드에 작업을 할당합니다. 여러 워커 노드에 대량의 신용카드 확인 요청을 배포하고 처리 오류를 방지하기 위해 멱등성 작업으로 중복 메시지를 처리합니다. • 이후의 처리를 위해 메시지를 배치 처리합니다. 데이터베이스에 배치를 추가하기 위해 여러 항목을 대기열에 추가합니다. 메시지 순서가 보장되지 않으므로 필요한 경우 순서에 맞지 않는 처리를 처리하도록 시스템을 설계합니다.	다음 예와 같이 FIFO 대기열을 사용하여 이벤트 순서가 중요할 때 애플리케이션 간에 데이터를 전송합니다. • 사용자가 입력한 명령이 올바른 순서로 실행되도록 보장합니다. 이는 명령 순서가 중요한 FIFO 대기열의 주요 사용 사례입니다. 예를 들어 사용자가 애플리케이션에서 일련의 작업을 수행하는 경우 FIFO 대기열은 작업이 입력된 순서와 동일한 순서로 처리되도록 합니다. • 가격 수정을 올바른 순서로 전송하여 올바른 제품 가격이 표시됩니다. FIFO 대기열은 제품 가격에 대한 여러 업데이트가 순차적으로 도착하고 처리되도록 합니다. FIFO가 없으면 가격 인상 후 가격 인하가 처리되어 잘못된 데이터가 표시될 수 있습니다. • 학생이 계정 등록 전에 과정에 등록하지 못하도록 차단합니다. FIFO 대기열을 사용하면 등록 프로세스가 올바른 순서로 이루어지도록 할 수 있습니다. 시스템은 계정 등록을 먼저 처리한 다음 과정 등록을 처리하여 등록 요청이 조기에 실행되지 않도록 합니다.

Amazon SQS에서 요청-응답 시스템 구현

요청 응답 또는 RPC(원격 프로시저 호출) 시스템을 구현할 경우에는 다음 모범 사례에 유의하십시오.

- 시작 시 응답 대기열 생성 - 메시지당 응답 대기열을 생성하는 대신 시작할 때 생산자별로 응답 대기열을 생성합니다. 상관관계 ID 메시지 속성을 사용하여 요청에 대한 응답을 효율적으로 매핑합니다.
- 생산자 간의 응답 대기열 공유 방지 - 각 생산자에게 자체 응답 대기열이 있도록 보장합니다. 응답 대기열을 공유하면 한 생산자가 다른 생산자를 위한 응답 메시지를 수신할 수도 있습니다.

임시 대기열 클라이언트를 사용하여 요청-응답 패턴을 구현하는 방법에 대한 자세한 내용은 요청-응답 메시징 패턴(가상 대기열) 단원을 참조하십시오.

Amazon SQS 표준 대기열 생성 및 메시지 전송

Amazon SQS 콘솔을 사용하여 표준 대기열을 생성하고 메시지를 보낼 수 있습니다. 또한이 주제에서는 대기열 이름에서 민감한 정보를 피하고 관리형 서버 측 암호화를 활용하는 등 모범 사례를 강조합니다.

Amazon SQS 콘솔을 사용하여 표준 대기열 생성

> **Important**
>
> 2022년 8월 17일에 기본 서버 측 암호화(SSE)가 모든 Amazon SQS 대기열에 적용되었습니다.
> 개인 식별 정보(PII)나 기타 기밀 정보 또는 민감한 정보를 대기열 이름에 추가하지 마세요. 대기열 이름을 통해 결제 및 CloudWatch 로그를 포함하여 많은 Amazon Web Services에 액세스할 수 있습니다. 대기열 이름은 개인 데이터나 민감한 데이터에 사용하기 위한 것이 아닙니다.

Amazon SQS 대기열을 생성하려면

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 대기열 생성을 선택합니다.
3. 유형의 경우 표준 대기열 유형이 기본적으로 설정됩니다.

   > **Note**
   >
   > 대기열을 생성한 후에는 대기열 유형을 변경할 수 없습니다.

4. 대기열의 이름을 입력합니다.
5. (선택 사항) 콘솔은 대기열 구성 파라미터의 기본값을 설정합니다. 구성에서 다음 파라미터에 새 값을 설정할 수 있습니다.

   a. 제한 시간 초과에는 기간과 단위를 입력합니다. 범위는 0초~12시간입니다. 기본값은 30초입니다.

   b. 메시지 보존 기간에는 기간과 단위를 입력합니다. 범위는 1분~14일입니다. 기본값은 4일입니다.

   c. 전송 지연에는 기간과 단위를 입력합니다. 범위는 0초~15분입니다. 기본값은 0초입니다.

d. 최대 메시지 크기에는 값을 입력합니다. 범위는 1KiB~1024KiB입니다. 기본값은 1024KiB입니다.

e. 메시지 수신 대기 시간은 값을 입력합니다. 범위는 0~20초입니다. 기본값은 0초이며 짧은 폴링을 설정합니다. 0이 아닌 값은 모두 긴 폴링을 설정합니다.

6. (선택 사항) 액세스 정책을 정의합니다. 액세스 정책은 대기열에 액세스할 수 있는 계정, 사용자 및 역할을 정의합니다. 또한 액세스 정책은 사용자가 액세스할 수 있는 작업(예: `SendMessage`, `ReceiveMessage` 또는`DeleteMessage`)을 정의합니다. 기본 정책에서는 대기열 소유자만 메시지를 보내고 받을 수 있도록 허용합니다.

   액세스 정책을 정의하려면 다음 중 하나를 수행합니다.

   - 대기열에 메시지를 보낼 수 있는 사람과 대기열에서 메시지를 받을 수 있는 사람을 구성하려면 기본을 선택합니다. 콘솔은 사용자의 선택에 따라 정책을 생성하고 결과 액세스 정책을 읽기 전용 JSON 창에 표시합니다.
   - JSON 액세스 정책을 직접 수정하려면 고급을 선택합니다. 이를 통해 각 주체(계정, 사용자 또는 역할)가 수행할 수 있는 사용자 지정 작업 집합을 지정할 수 있습니다.

7. 리드라이브 허용 정책의 경우 활성화를 선택합니다. 모두 허용, 대기열 기준 또는 모두 거부 중 하나를 선택합니다. 대기열 기준 선택 시 Amazon 리소스 이름(ARN)으로 최대 10개 소스 대기열의 목록을 지정합니다.

8. Amazon SQS는 기본적으로 관리형 서버 측 암호화를 제공합니다. 암호화 키 유형을 선택하거나 Amazon SQS 관리형 서버 측 암호화를 비활성화하려면 암호화를 확장합니다. 암호화 키 유형에 대한 자세한 내용은 SQS 관리형 암호화 키를 사용하여 대기열에 대해 서버 측 암호화 구성 및 Amazon SQS 콘솔을 사용하여 대기열에 대한 서버 측 암호화 구성 섹션을 참조하세요.

   > **Note**
   >
   > SSE를 활성화하면 암호화된 대기열에 대한 익명 `SendMessage` 및 `ReceiveMessage` 요청이 거부됩니다. Amazon SQS 보안 모범 사례에서는 익명 요청을 사용하지 말 것을 권장합니다. Amazon SQS 대기열로 익명 요청을 보내려면 SSE를 비활성화해야 합니다.

9. (선택 사항) 배달되지 않은 메시지를 수신하도록 DLQ(Dead Letter Queue)를 구성하려면 DLQ(Dead Letter Queue)를 펼칩니다.

10. (선택 사항) 대기열에 태그를 추가하려면 태그를 펼칩니다.

11. 대기열 생성을 선택합니다. Amazon SQS가 대기열을 생성하고 대기열의 세부 정보 페이지를 표시합니다.

Amazon SQS는 새 대기열에 대한 정보를 시스템 전체에 전파합니다. Amazon SQS는 분산 시스템이므로 콘솔이 대기열 페이지에 대기열을 표시할 때까지 약간의 지연이 발생할 수 있습니다.

표준 대기열을 사용하여 메시지 전송

대기열이 생성된 후 대기열에 메시지를 보낼 수 있습니다.

1. 왼쪽 탐색 창에서 대기열을 선택합니다. 대기열 목록에서 자신이 생성한 대기열을 선택합니다.
2. 작업에서 메시지 전송 및 수신을 선택합니다.

   콘솔에 메시지 전송 및 수신 페이지가 표시됩니다.
3. 메시지 본문에 메시지 텍스트를 입력합니다.
4. 표준 대기열의 경우 전송 지연 시간 값을 입력하고 단위를 선택할 수 있습니다. 예를 들어 `60` 입력 후 초를 선택합니다. 자세한 내용은 Amazon SQS 메시지 타이머 단원을 참조하십시오.
5. 메시지 전송을 선택합니다.

   메시지를 전송하면 콘솔에 성공 메시지가 표시됩니다. 전송한 메시지에 대한 정보를 표시하려면 세부 정보 보기를 선택합니다.

Amazon SQS FIFO 대기열 생성 및 메시지 전송

콘솔을 사용하여 Amazon SQS FIFO 대기열을 생성하고 메시지를 보낼 수 있습니다. 이 주제에서는 대기열 이름의 민감한 정보를 피하고 서버 측 암호화를 활성화하는 등의 보안 모범 사례를 따르면서 가시성 제한 시간, 메시지 보존 및 중복 제거를 포함한 대기열 파라미터를 설정하는 방법을 설명합니다. 또한 액세스 정책 정의, 배달 못한 편지 대기열 구성, 메시지 그룹 ID 및 중복 제거 ID와 같은 FIFO별 속성으로 메시지 전송도 다룹니다.

Amazon SQS 콘솔을 사용하여 FIFO 대기열 생성

Amazon SQS 콘솔을 사용하여 FIFO 대기열을 만들 수 있습니다. 콘솔은 대기열 이름을 제외한 모든 설정의 기본값을 제공합니다.

> **Important**
>
> 2022년 8월 17일에 기본 서버 측 암호화(SSE)가 모든 Amazon SQS 대기열에 적용되었습니다.

개인 식별 정보(PII)나 기타 기밀 정보 또는 민감한 정보를 대기열 이름에 추가하지 마세요. 대기열 이름을 통해 결제 및 CloudWatch 로그를 포함하여 많은 Amazon Web Services에 액세스할 수 있습니다. 대기열 이름은 개인 데이터나 민감한 데이터에 사용하기 위한 것이 아닙니다.

Amazon SQS FIFO 대기열 생성

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 대기열 생성을 선택합니다.
3. 유형의 경우 표준 대기열 유형이 기본적으로 설정됩니다. FIFO 대기열을 만들려면 FIFO를 선택합니다.

   > **Note**
   > 대기열을 생성한 후에는 대기열 유형을 변경할 수 없습니다.

4. 대기열의 이름을 입력합니다.

   FIFO 대기열의 이름은 `.fifo` 접미사로 끝나야 합니다. 접미사는 문자 80개의 대기열 이름 할당량에 포함됩니다. 해당 대기열이 접미사로 끝나는지 확인하여 FIFO 대기열인지 여부를 알 수 있습니다.

5. (선택 사항) 콘솔은 대기열 구성 파라미터의 기본값을 설정합니다. 구성에서 다음 파라미터에 새 값을 설정할 수 있습니다.

   a. 제한 시간 초과에는 기간과 단위를 입력합니다. 범위는 0초~12시간입니다. 기본값은 30초입니다.

   b. 메시지 보존 기간에는 기간과 단위를 입력합니다. 범위는 1분~14일입니다. 기본값은 4일입니다.

   c. 전송 지연에는 기간과 단위를 입력합니다. 범위는 0초~15분입니다. 기본값은 0초입니다.

   d. 최대 메시지 크기에는 값을 입력합니다. 범위는 1KiB~1024KiB입니다. 기본값은 1024KiB입니다.

   e. 메시지 수신 대기 시간은 값을 입력합니다. 범위는 0~20초입니다. 기본값은 0초이며 짧은 폴링을 설정합니다. 0이 아닌 값은 모두 긴 폴링을 설정합니다.

   f. FIFO 대기열의 경우 콘텐츠 기반 중복 제거를 선택하여 콘텐츠 기반 중복 제거를 활성화합니다. 기본값은 비활성화입니다.

g. (선택 사항) FIFO 대기열에서 메시지를 보내고 받을 때 처리량을 높이려면 높은 처리량 FIFO 활성화를 선택합니다.

   이 옵션을 선택하면 관련 옵션(중복 제거 범위 및 FIFO 처리량 한도)이 FIFO 대기열의 높은 처리량을 활성화하는 데 필요한 설정으로 변경됩니다. 높은 처리량 FIFO를 사용하는 데 필요한 설정을 변경하면 대기열에 일반 처리량이 적용되고 지정된 대로 중복 제거가 수행됩니다. 자세한 내용은 Amazon SQS에서 FIFO 대기열의 높은 처리량 및 Amazon SQS 메시지 할당량 섹션을 참조하세요.

6. (선택 사항) 액세스 정책을 정의합니다. 액세스 정책은 대기열에 액세스할 수 있는 계정, 사용자 및 역할을 정의합니다. 또한 액세스 정책은 사용자가 액세스할 수 있는 작업(예: `SendMessage`, `ReceiveMessage` 또는`DeleteMessage`)을 정의합니다. 기본 정책에서는 대기열 소유자만 메시지를 보내고 받을 수 있도록 허용합니다.

   액세스 정책을 정의하려면 다음 중 하나를 수행합니다.

   - 대기열에 메시지를 보낼 수 있는 사람과 대기열에서 메시지를 받을 수 있는 사람을 구성하려면 기본을 선택합니다. 콘솔은 사용자의 선택에 따라 정책을 생성하고 결과 액세스 정책을 읽기 전용 JSON 창에 표시합니다.
   - JSON 액세스 정책을 직접 수정하려면 고급을 선택합니다. 이를 통해 각 주체(계정, 사용자 또는 역할)가 수행할 수 있는 사용자 지정 작업 집합을 지정할 수 있습니다.

7. 리드라이브 허용 정책의 경우 활성화를 선택합니다. 모두 허용, 대기열 기준 또는 모두 거부 중 하나를 선택합니다. 대기열 기준 선택 시 Amazon 리소스 이름(ARN)으로 최대 10개 소스 대기열의 목록을 지정합니다.

8. Amazon SQS는 기본적으로 관리형 서버 측 암호화를 제공합니다. 암호화 키 유형을 선택하거나 Amazon SQS 관리형 서버 측 암호화를 비활성화하려면 암호화를 확장합니다. 암호화 키 유형에 대한 자세한 내용은 SQS 관리형 암호화 키를 사용하여 대기열에 대해 서버 측 암호화 구성 및 Amazon SQS 콘솔을 사용하여 대기열에 대한 서버 측 암호화 구성 섹션을 참조하세요.

   > **Note**
   >
   > SSE를 활성화하면 암호화된 대기열에 대한 익명 `SendMessage` 및 `ReceiveMessage` 요청이 거부됩니다. Amazon SQS 보안 모범 사례에서는 익명 요청을 사용하지 말 것을 권장합니다. Amazon SQS 대기열로 익명 요청을 보내려면 SSE를 비활성화해야 합니다.

9. (선택 사항) 배달되지 않은 메시지를 수신하도록 DLQ(Dead Letter Queue)를 구성하려면 DLQ(Dead Letter Queue)를 펼칩니다.

10. (선택 사항) 대기열에 태그를 추가하려면 태그를 펼칩니다.

11. 대기열 생성을 선택합니다. Amazon SQS가 대기열을 생성하고 대기열의 세부 정보 페이지를 표시합니다.

Amazon SQS는 새 대기열에 대한 정보를 시스템 전체에 전파합니다. Amazon SQS는 분산 시스템이므로 콘솔이 대기열 페이지에 대기열을 표시할 때까지 약간의 지연이 발생할 수 있습니다.

대기열을 생성한 후에는 대기열에 메시지를 보내고 메시지를 수신 및 삭제할 수 있습니다. 대기열 유형을 제외한 모든 대기열 구성 설정을 편집할 수도 있습니다.

FIFO 대기열을 사용하여 메시지 전송

대기열을 생성한 후 그 대기열에 메시지를 보낼 수 있습니다.

1. 왼쪽 탐색 창에서 대기열을 선택합니다. 대기열 목록에서 자신이 생성한 대기열을 선택합니다.
2. 작업에서 메시지 전송 및 수신을 선택합니다.

   콘솔에 메시지 전송 및 수신 페이지가 표시됩니다.
3. 메시지 본문에 메시지 텍스트를 입력합니다.
4. 선입선출(FIFO) 대기열의 경우 메시지 그룹 ID를 입력합니다. 자세한 내용은 Amazon SQS의 FIFO 대기열 전송 로직 단원을 참조하십시오.
5. (선택 사항) FIFO 대기열의 경우 메시지 중복 제거 ID를 입력할 수 있습니다. 대기열에 콘텐츠 기반 중복 제거를 활성화한 경우에는 메시지 중복 제거 ID가 필요하지 않습니다. 자세한 내용은 Amazon SQS의 FIFO 대기열 전송 로직 단원을 참조하십시오.
6. FIFO 대기열은 개별 메시지의 타이머를 지원하지 않습니다. 자세한 내용은 Amazon SQS 메시지 타이머 단원을 참조하십시오.
7. 메시지 전송을 선택합니다.

   메시지를 전송하면 콘솔에 성공 메시지가 표시됩니다. 전송한 메시지에 대한 정보를 표시하려면 세부 정보 보기를 선택합니다.

Amazon SQS 시작을 위한 일반적인 작업

대기열을 생성하고 메시지를 전송, 수신 및 삭제하는 방법을 배웠으면 다음을 시도해 볼 수 있습니다.

- Lambda 함수를 트리거하여 수신 메시지를 자동으로 처리하므로 지속적인 폴링 없이 이벤트 기반 워크플로를 사용할 수 있습니다.
- SSE 및 기타 기능을 포함하여 대기열을 구성합니다.

- 속성이 있는 메시지를 전송합니다.
- VPC로부터 메시지를 전송합니다.
- Amazon SQS의 기능과 아키텍처를 알아봅니다.
- Amazon SQS를 최대한 활용하는 데 도움이 되는 지침과 주의 사항을 알아봅니다.
- AWS SDK for Java 2.x 개발자 안내서와 같은 AWS SDK의 Amazon SQS 예제를 살펴봅니다.
- Amazon SQS AWS CLI 명령에 대해 알아봅니다.
- Amazon SQS API 작업에 대해 알아봅니다.
- 프로그래밍 방식으로 Amazon SQS와 상호 작용하는 방법을 알아봅니다. APIs하고 AWS 개발 센터를 살펴보세요.
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows 및 .NET

- 비용 및 리소스를 모니터링하는 방법을 알아봅니다.
- 데이터를 보호하는 방법을 알아봅니다.
- Amazon SQS 워크플로에 대해 자세히 알아봅니다.

Amazon SQS 대기열 관리

대기열 설정 편집, 메시지 수신 및 삭제, 대기열 비우기 확인, 대기열 삭제 또는 제거 등 콘솔을 사용하여 Amazon SQS 대기열을 관리하는 방법을 알아봅니다. 긴 폴링 사용, 가시성 제한 시간 관리, 대시보드 또는 모니터링을 통한 지표 확인과 같은 효율적인 메시지 처리를 위한 모범 사례를 이해합니다 AWS CLI. 실제 단계에 따라 중단을 최소화하면서 대기열을 유지하고 메시지를 효과적으로 처리합니다.

콘솔을 사용하여 Amazon SQS 대기열 편집

Amazon SQS 콘솔을 사용하여 대기열 구성 파라미터(대기열 유형 제외)를 편집하고 필요에 따라 기능을 수정하거나 제거할 수 있습니다.

Amazon SQS 대기열을 편집하려면(콘솔)

1. Amazon SQS 콘솔의 대기열 페이지를 엽니다.
2. 대기열을 선택한 다음 편집을 선택합니다.
3. (선택 사항) 구성에서 대기열의 구성 파라미터를 업데이트합니다.
4. (선택 사항) 액세스 정책을 업데이트하려면 액세스 정책에서 JSON 정책을 수정합니다.
5. (선택 사항) 배달 못한 편지 대기열 리드라이브 허용 정책을 업데이트하려면 리드라이브 허용 정책을 확장합니다.
6. (선택 사항) 암호화를 업데이트하거나 제거하려면 암호화를 확장합니다.
7. (선택 사항) 배달 못한 편지 대기열을 추가, 업데이트 또는 제거하려면(전달할 수 없는 메시지를 수신할 수 있음) 배달 못한 편지 대기열을 확장합니다.
8. (선택 사항) 대기열에 태그를 추가, 업데이트 또는 제거하려면 태그를 확장합니다.
9. 저장을 선택합니다.

   - 콘솔에 대기열에 대한 세부 정보 페이지가 표시됩니다.

Amazon SQS에서 메시지 수신 및 삭제

Amazon SQS 대기열로 메시지를 전송한 후 해당 메시지를 검색하고 삭제하여 애플리케이션 워크플로를 처리할 수 있습니다. 이 프로세스는 안전하고 신뢰할 수 있는 메시지 처리를 보장합니다. 이 주제에서는 Amazon SQS 콘솔을 사용하여 메시지를 검색하고 삭제하는 방법을 안내하고이 작업을 최적화하기 위한 주요 설정을 설명합니다. 다음은 메시지를 수신하고 삭제하기 위한 주요 개념입니다.

1. 메시지 수신
   - Amazon SQS 대기열에서 메시지를 검색할 때는 특정 메시지를 대상으로 지정할 수 없습니다. 대신 단일 요청에서 검색할 최대 메시지 수(최대 10개)를 지정합니다.
   - Amazon SQS의 분산 특성으로 인해 메시지가 거의 없는 대기열에서 검색하면 빈 응답이 반환될 수 있습니다. 이를 완화하려면:
     - 메시지를 사용할 수 있거나 폴링 시간이 초과될 때까지 기다리는 긴 폴링을 사용합니다. 이 접근 방식은 불필요한 폴링 비용을 줄이고 효율성을 개선합니다.
     - 필요한 경우 요청을 다시 발행합니다.
2. 메시지 가시성 및 삭제
   - 검색 후에는 메시지가 자동으로 삭제되지 않습니다. 이 기능을 사용하면 애플리케이션 장애 또는 네트워크 중단 시 메시지를 재처리할 수 있습니다.
   - 처리 후 메시지를 영구적으로 제거하려면 삭제 요청을 명시적으로 전송해야 합니다. 이 작업은 성공적인 처리를 확인합니다.
   - Amazon SQS 콘솔을 사용하여 검색된 메시지는 다시 검색할 수 있도록 계속 표시됩니다. 자동 환경의 표시 제한 시간 설정을 조정하여 처리 중인 다른 소비자의 메시지를 일시적으로 숨깁니다.
3. 가시성 제한 시간
   - 이 설정은 검색 후 메시지가 숨겨진 상태로 유지되는 기간을 결정합니다. 메시지가 한 번만 처리되고 분산 처리 중에 중복이 발생하지 않도록 적절한 제한 시간을 설정합니다.

콘솔을 사용하여 메시지를 수신 및 삭제하려면

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열 페이지에서 메시지를 수신할 대기열을 선택한 다음 메시지 전송 및 수신을 선택합니다.
4. 메시지 전송 및 수신 페이지에서 메시지 폴링을 선택합니다.

   Amazon SQS는 폴링 기간을 나타내는 진행률 표시줄을 표시합니다. 검색된 메시지가 메시지 섹션에 나타나 다음과 같이 표시됩니다.

   - 메시지 ID
   - 전송 날짜
   - 크기
   - 수신 개수
5. 메시지를 삭제하려면 제거할 메시지를 선택하고 삭제를 선택합니다.

삭제를 선택하여 메시지 삭제 대화 상자에서 삭제를 확인합니다.

API 기반 메시지 검색 및 삭제를 포함한 고급 작업에 대한 자세한 내용은 Amazon SQS API 참조 가이드를 참조하세요.

Amazon SQS 대기열이 비어 있는지 확인

대부분의 경우 긴 폴링을 사용하여 대기열이 비어 있는지 확인할 수 있습니다. 드문 경우지만, 대기열에 여전히 메시지가 있는 경우에도 빈 응답을 받을 수 있습니다. 특히 큐를 만들 때 메시지 수신 대기 시간 값을 낮게 지정한 경우 더욱 그렇습니다. 이 섹션에서는 대기열이 비어 있는지 확인하는 방법을 설명합니다.

대기열이 비어 있는지 확인하는 방법(콘솔)

1. 모든 생산자가 메시지를 보내지 못하게 합니다.
2. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
3. 탐색 창에서 대기열을 선택합니다.
4. 대기열 페이지에서 대기열을 선택합니다.
5. 모니터링 탭을 선택합니다.
6. 모니터링 대시보드의 오른쪽 상단에서 새로 고침 기호 옆에 있는 아래쪽 화살표를 선택합니다. 드롭다운 메뉴에서 자동 새로 고침을 선택합니다. 새로 고침 간격을 1분으로 유지합니다.
7. 다음 대시보드를 살펴봅니다.

   - 지연된 메시지의 대략적인 수
   - 표시되지 않은 메시지의 대략적인 수
   - 표시된 메시지의 대략적인 수

   몇 분 동안 모든 항목의 값이 `0`으로 표시되면 대기열이 비어 있는 것입니다.

대기열이 비어 있는지 확인하려면(AWS CLI, AWS API)

1. 모든 생산자가 메시지를 보내지 못하게 합니다.
2. 다음 명령 중 하나를 반복적으로 실행합니다.

   - AWS CLI: `get-queue-attributes`

- AWS API: `GetQueueAttributes`

3. 지표를 관찰하여 다음 속성이 있는지 확인합니다.

   - `ApproximateNumberOfMessagesDelayed`
   - `ApproximateNumberOfMessagesNotVisible`
   - `ApproximateNumberOfMessagesVisible`

   몇 분 동안 모든 항목이 `0`으로 표시되면 대기열이 비어 있는 것입니다.

Amazon CloudWatch 지표에 의존하는 경우, 대기열이 비어 있다고 간주하기 전에 여러 개의 연속된 0 데이터 포인트가 표시되는지 확인하세요. CloudWatch 지표에 대한 자세한 내용은 Amazon SQS의 CloudWatch 지표 액세스 섹션을 참조하세요.

Amazon SQS 대기열 삭제

Amazon SQS 대기열을 더 이상 사용하지 않고 가까운 시일 내에 사용할 계획이 없는 경우 대기열을 삭제합니다.

> **Tip**
> 대기열을 삭제하기 전에 대기열이 비어 있는지 확인하려면 Amazon SQS 대기열이 비어 있는지 확인 섹션을 참조하세요.

대기열은 비어있지 않은 경우에도 삭제할 수 있습니다. 대기열의 메시지는 삭제하지만 대기열 자체는 삭제하지 않을 경우 대기열을 제거합니다.

대기열을 삭제하려면(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열 페이지에서 삭제할 대기열을 선택합니다.
4. 삭제를 선택합니다.
5. 대기열 삭제 대화 상자에 **`delete`**를 입력하여 삭제를 확인합니다.
6. 삭제를 선택합니다.

대기열(AWS CLI 및 API)을 삭제하려면

필요에 따라 대기열을 삭제할 적절한 방법을 선택합니다.

- AWS CLI: `aws sqs delete-queue`
- AWS API: `DeleteQueue`

Amazon SQS 콘솔을 사용하여 대기열에서 메시지 제거

Amazon SQS 대기열을 유지하고 모든 메시지를 제거하려면 대기열을 제거하면 됩니다. 이렇게 하면 현재 보이지 않는 메시지(비행 중)를 포함한 모든 메시지가 삭제됩니다. 제거 프로세스는 최대 60초까지 걸릴 수 있으므로 대기열 크기에 관계없이 60초 동안 기다립니다.

> **Important**
>
> 대기열을 제거할 경우 삭제된 메시지를 검색할 수 없습니다.

대기열을 제거하려면(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열 페이지에서 제거할 대기열을 선택합니다.
4. 작업에서 제거를 선택합니다.
5. 대기열 제거 대화 상자에서 **`purge`**라고 입력하고 제거를 선택하여 제거를 확인합니다.
   - 대기열에서 모든 메시지가 제거됩니다. 콘솔에 확인 배너가 표시됩니다.

Amazon SQS 표준 대기열

Amazon SQS는 표준 대기열을 기본 대기열 유형으로 제공하여 `SendMessage`, `ReceiveMessage` 및 `DeleteMessage`와 같은 작업에 대해 초당 거의 무제한의 API 직접 호출 수를 지원합니다. 표준 대기열은 최소 1회 메시지 전송을 보장하지만, 고도로 분산된 아키텍처로 인해 두 개 이상의 메시지 사본이 전송될 수 있으며 메시지가 가끔 순서에 맞지 않게 도착할 수 있습니다. 그럼에도 불구하고 표준 대기열은 메시지가 전송되는 순서를 유지하기 위해 최선의 노력을 기울입니다.

`SendMessage`를 사용하여 메시지를 보내면 Amazon SQS는 메시지를 승인하기 전에 여러 가용 영역(AZ)에 메시지를 중복 저장합니다. 이러한 이중화를 통해 단일 컴퓨터, 네트워크 또는 AZ 장애로 인해 메시지에 액세스할 수 없는 상황이 발생하지 않도록 합니다.

Amazon SQS 콘솔을 사용하여 대기열을 생성하고 구성할 수 있습니다. 자세한 지침은 Amazon SQS 콘솔을 사용하여 표준 대기열 생성 섹션을 참조하세요. Java 관련 예는 Amazon SQS Java SDK 예제 섹션을 참조하세요.

표준 대기열 사용 사례

표준 메시지 대기열은 애플리케이션에서 두 번 이상 또는 순서에 맞지 않게 도착하는 메시지를 처리할 수 있는 한 다양한 시나리오에 적합합니다. 그러한 예는 다음과 같습니다.

- 실시간 사용자 요청을 폭넓은 배경 작업과 분리 - 사용자는 시스템이 백그라운드에서 미디어 크기를 조정하거나 인코딩하는 동안 미디어를 업로드할 수 있습니다.
- 작업을 여러 워커 노드에 할당 - 예를 들면 대량의 신용카드 확인 요청을 처리하는 것입니다.
- 이후의 처리를 위해 메시지를 배치 처리 - 다수의 항목이 나중에 데이터베이스에 추가되도록 예약합니다.

표준 대기열 관련 할당량에 대한 정보는 Amazon SQS 표준 대기열 할당량 섹션을 참조하세요.

표준 대기열 작업에 대한 모범 사례는 Amazon SQS 모범 사례 섹션을 참조하세요.

Amazon SQS 최소 1회 전송

Amazon SQS는 중복성과 고가용성을 위해 여러 대의 서버에 메시지 사본을 저장합니다. 드물게는 메시지 사본을 받거나 삭제할 때 메시지 사본을 저장하는 서버 중 하나를 사용할 수 없을 수도 있습니다.

이 문제가 발생할 경우 사용 불가능한 해당 서버에서 메시지의 사본이 삭제되지 않으며, 메시지를 받을 때 해당 메시지 사본을 다시 가져올 수 있습니다. 따라서 애플리케이션이 idempotent가 되도록 설계해야 합니다(다시 말해 동일한 메시지를 두 번 이상 처리할 경우 부정적인 영향을 받지 않아야 함).

Amazon SQS 대기열 및 메시지 식별자

이 주제에서는 표준 및 FIFO 대기열의 식별자를 설명합니다. 이러한 식별자는 특정 대기열과 메시지를 찾고 조작하는 데 도움이 될 수 있습니다.

Amazon SQS 표준 대기열의 식별자

다음 식별자에 대한 자세한 내용은 Amazon Simple Queue Service API 참조를 참조하세요.

대기열 이름 및 URL

새 대기열을 생성할 때는 AWS 계정 및 리전에 고유한 대기열 이름을 지정해야 합니다. Amazon SQS는 사용자가 생성한 각 대기열에 대기열 이름과 기타 Amazon SQS 구성 요소가 포함된 대기열 URL이라는 식별자를 할당합니다. 대기열에서 작업을 실시할 때마다 대기열 URL을 지정합니다.

다음은 AWS 계정 번호 `123456789012`를 갖는 사용자가 소유하는 `MyQueue`라는 이름의 대기열의 대기열 URL입니다.

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
```

대기열을 나열하고 계정 번호 뒤에 오는 문자열을 구문 분석하여 프로그래밍 방식으로 대기열의 URL을 검색할 수 있습니다. 자세한 내용은 `ListQueues` 단원을 참조하십시오.

메시지 ID

각 메시지는 Amazon SQS가 `SendMessage` 응답으로 반환하는 시스템 할당 메시지 ID를 수신합니다. 이 식별자는 메시지를 식별하는 데 유용합니다. 메시지 ID의 최대 길이는 100자입니다.

수신 핸들

대기열에서 메시지를 수신할 때마다 해당 메시지의 수신 핸들을 수신합니다. 이 핸들은 메시지 자체가 아닌 메시지 수신 작업과 관련되어 있습니다. 메시지를 삭제하거나 메시지 표시를 변경하려면, (메시지 ID가 아닌) 수신 핸들을 지정해야 합니다. 또한, 메시지를 수신한 후 언제든지 삭제할 수 있습니다(메시지를 대기열에 배치한 후 회수할 수 없음). 수신 핸들의 최대 길이는 1024자입니다.

> **Important**
>
> 메시지를 두 번 이상 받으면, 받을 때마다 다른 수신 핸들을 받습니다. 메시지 삭제를 요청할 때에는 가장 최근에 받은 수신 핸들을 제공해야 합니다(그렇지 않으면 메시지가 삭제될 수 있음).

다음은 세 줄로 구분된 수신 핸들의 예입니다.

```
MbZj6wDWli+JvwwJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon SQS FIFO 대기열

FIFO(First-In-First-Out) 대기열에는 표준 대기열의 모든 기능이 있지만 작업 및 이벤트 순서가 중요하거나 중복을 허용할 수 없는 경우 애플리케이션 간의 메시징을 개선하도록 설계되었습니다.

FIFO 대기열의 가장 중요한 기능은 FIFO(선입선출) 전송과 정확히 1회 처리입니다.

- 메시지가 전송 및 수신되는 순서가 엄격히 유지되며 메시지가 한 번 전송되어 소비자가 이 메시지를 처리하고 삭제할 때까지 계속 사용할 수 없습니다.
- 중복 항목을 대기열에 삽입하지 않습니다.

또한 FIFO 대기열은 단일 대기열에 정렬된 여러 메시지 그룹을 허용하는 메시지 그룹을 지원합니다. FIFO 대기열 내 메시지 그룹의 수에는 할당량이 없습니다.

FIFO 대기열을 사용할 수 있는 상황의 예는 다음과 같습니다.

1. 주문이 중요한 전자 상거래 주문 관리 시스템
2. 이벤트를 순서대로 처리해야 하는 타사 시스템과의 통합
3. 사용자가 입력한 내용을 입력한 순서대로 처리
4. 통신 및 네트워킹 - 데이터와 정보를 동일한 순서로 전송 및 수신
5. 컴퓨터 시스템 - 사용자가 입력한 명령이 올바른 순서로 실행되도록 보장
6. 교육 기관 - 학생이 계정 등록 전에 과정에 등록하지 못하도록 차단
7. 온라인 티켓팅 시스템 - 티켓이 선착순으로 배포됨

> **Note**
>
> 또한 FIFO 대기열은 정확히 1회 처리를 제공하지만 초당 트랜잭션(TPS) 수가 제한적입니다. FIFO 대기열에서 Amazon SQS 높은 처리량 모드를 사용하면 트랜잭션 한도를 늘릴 수 있습니다. 높은 처리량 모드 사용에 대한 자세한 내용은 Amazon SQS에서 FIFO 대기열의 높은 처리량 섹션을 참조하세요. 처리량 할당량에 대한 자세한 내용은 the section called “메시지 할당량” 섹션을 참조하세요.

Amazon SQS를 사용할 수 있는 모든 리전에서 Amazon SQS FIFO 대기열을 사용할 수 있습니다.

순서가 복잡한 FIFO 대기열 사용에 대한 자세한 내용은 Solving Complex Ordering Challenges with Amazon SQS FIFO Queues 항목을 참조하세요.

Amazon SQS 콘솔을 사용하여 대기열을 생성 및 구성하는 방법에 대한 자세한 내용은 Amazon SQS 콘솔을 사용하여 표준 대기열 생성 섹션을 참조하세요. Java 예제는 Amazon SQS Java SDK 예제 섹션을 참조하세요.

FIFO 대기열 작업의 모범 사례는 Amazon SQS 모범 사례 섹션을 참조하세요.

Amazon SQS FIFO 대기열 주요 용어

다음 주요 용어 설명은 FIFO 대기열 기능을 보다 정확하게 이해하는 데 도움이 될 수 있습니다. 자세한 내용은 Amazon Simple Queue Service API 참조를 참조하세요.

클라이언트

Amazon SQS의 버퍼링된 비동기식 클라이언트는 현재 FIFO 대기열을 지원하지 않습니다.

메시지 중복 제거 ID

Amazon SQS FIFO 대기열에 사용되는 토큰은 메시지를 고유하게 식별하고 중복을 방지합니다. 중복 제거 ID가 동일한 여러 메시지가 5분 중복 제거 간격 내에 전송되는 경우 중복으로 취급되고 복사본 하나만 전송됩니다. 중복 제거 ID를 지정하지 않고 콘텐츠 기반 중복 제거가 활성화된 경우 Amazon SQS는 메시지 본문을 해싱하여 중복 제거 ID를 생성합니다. 이 메커니즘은 지정된 기간 내에 중복 메시지를 제거하여 정확히 한 번 전송되도록 합니다.

> **Note**
>
> Amazon SQS는 메시지를 수신하고 삭제한 후에도 중복 제거 ID를 계속 추적합니다.

메시지 그룹 ID

FIFO(First-In-First-Out) 대기열에서 `MessageGroupId`는 메시지를 고유한 그룹으로 구성하는 속성입니다. 동일한 메시지 그룹 내의 메시지는 항상 한 번에 하나씩 엄격한 순서로 처리되므로 동일한 그룹의 두 메시지가 동시에 처리되지 않습니다. 표준 대기열에서를 사용하면 공정 대기열이 `MessageGroupId` 활성화됩니다. 엄격한 주문이 필요한 경우 FIFO 대기열을 사용합니다.

수신 요청 시도 ID

수신 요청 시도 ID는 Amazon SQS에서 `ReceiveMessage` 호출을 중복 제거하는 데 사용되는 고유한 토큰입니다.

시퀀스 번호

Amazon SQS가 각 메시지에 할당하는 연속되지 않은 큰 숫자입니다.

서비스

애플리케이션이 여러 AWS 서비스를 사용하거나 AWS 및 외부 서비스를 혼합하는 경우 FIFO 대기열을 지원하지 않는 서비스 기능을 이해하는 것이 중요합니다.

Amazon SQS에 알림을 보내는 일부 AWS 또는 외부 서비스는 FIFO 대기열을 대상으로 설정할 수 있음에도 불구하고 FIFO 대기열과 호환되지 않을 수 있습니다.

다음 AWS 서비스 기능은 현재 FIFO 대기열과 호환되지 않습니다.

- Amazon S3 이벤트 알림
- Auto Scaling 수명 주기 후크
- AWS IoT 규칙 작업
- AWS Lambda 배달 못한 편지 대기열

기타 서비스와 FIFO 대기열의 호환성에 관한 자세한 내용은 서비스 설명서를 참조하세요.

Amazon SQS의 FIFO 대기열 전송 로직

다음 개념은 특히 메시지 순서 및 메시지 그룹 ID를 처리할 때 Amazon SQS FIFO 대기열이 메시지 전송 및 수신을 처리하는 방법을 명확히 합니다. IDs

메시지 보내기

Amazon SQS FIFO 대기열은 고유한 중복 제거 IDs 및 메시지 그룹 IDs. 이 주제에서는 그룹 내에서 엄격한 순서를 유지하기 위한 메시지 그룹 IDs의 중요성을 강조하고 여러 생산자 간에 안정적이고 순서가 지정된 메시지 전송을 보장하기 위한 모범 사례를 강조합니다.

1. 주문 보존
   - 여러 메시지가 고유한 메시지 중복 제거 IDs를 사용하여 FIFO 대기열로 연속으로 전송되는 경우 Amazon SQS는 메시지를 저장하고 전송을 승인합니다. 그런 다음 이러한 메시지는 전송된 정확한 순서로 수신되고 처리됩니다.
2. 메시지 그룹 ID
   - FIFO 대기열에서 메시지는 메시지 그룹 ID를 기준으로 정렬됩니다. 여러 생산자 또는 스레드가 동일한 메시지 그룹 ID로 메시지를 전송하는 경우 Amazon SQS는 메시지가 도착한 순서대로 저장되고 처리되도록 합니다.

- 모범 사례: 여러 생산자 간에 엄격한 메시지 순서를 보장하려면 각 생산자의 모든 메시지에 고유한 메시지 그룹 ID를 할당합니다.

3. 그룹별 순서
   - FIFO 대기열 로직은 메시지 그룹 ID별로 적용됩니다.
     - 각 메시지 그룹 ID는 고유하고 순서가 지정된 메시지 그룹을 나타냅니다.
     - 메시지 그룹 ID 내에서 모든 메시지는 엄격한 순서로 전송 및 수신됩니다.
     - 메시지 그룹 IDs가 다른 메시지는 서로 비교하여 순서대로 도착하거나 처리될 수 있습니다.
   - 요구 사항 - 메시지 그룹 ID를 각 메시지와 연결해야 합니다. 그룹 ID 없이 메시지가 전송되면 작업이 실패합니다.
   - 단일 그룹 시나리오 - 모든 메시지를 엄격한 순서로 처리해야 하는 경우 모든 메시지에 동일한 메시지 그룹 ID를 사용합니다.

메시지 수신

Amazon SQS FIFO 대기열은 배치 처리, FIFO 주문 보장, 특정 메시지 그룹 IDs 요청 제한 등 메시지 검색을 처리합니다. 이 주제에서는 Amazon SQS가 엄격한 순서 지정 및 표시 규칙을 유지하면서 메시지 그룹 IDs 내부 및 전체에서 메시지를 검색하는 방법을 설명합니다.

1. 배치 가져오기
   - 여러 메시지 그룹 IDs Amazon SQS
     - 단일 호출에서 동일한 메시지 그룹 ID로 가능한 한 많은 메시지를 반환하려고 시도합니다.
     - 다른 소비자가 다른 메시지 그룹 IDs 메시지를 동시에 처리하도록 허용합니다.
   - 중요한 설명
     - 동일한 메시지 그룹 ID에서 하나의 배치로 여러 메시지를 수신할 수 있습니다 (`MaxNumberOfMessages`파라미터를 사용하여 단일 호출에서 최대 10개의 메시지).
     - 그러나 다음 시점까지는 후속 요청에서 동일한 메시지 그룹 ID로부터 추가 메시지를 수신할 수 없습니다.
       - 현재 수신된 메시지가 삭제되거나
       - 다시 표시됩니다(예: 표시 제한 시간이 만료된 후).
2. FIFO 주문 보장
   - 배치로 검색된 메시지는 그룹 내에서 FIFO 순서를 유지합니다.
   - 동일한 메시지 그룹 ID에 사용할 수 있는 메시지가 10개 미만인 경우 Amazon SQS는 다른 메시지 그룹 IDs 메시지를 동일한 배치에 포함할 수 있지만 각 그룹은 FIFO 순서를 유지합니다.

3. 소비자 제한 사항
   - 특정 메시지 그룹 ID에서 메시지를 수신하도록 명시적으로 요청할 수 없습니다.

다수의 재시도 횟수

생산자와 소비자는 메시지 순서를 중단하거나 중복을 도입하지 않고 Amazon SQS FIFO 대기열에서 실패한 작업을 안전하게 재시도할 수 있습니다. 이 주제에서는 중복 제거 IDs 및 제한 시간 초과가 재시도 중에 메시지 무결성을 보장하는 방법을 강조합니다.

1. 생산자 재시도
   - `SendMessage` 작업이 실패하면 생산자는 동일한 메시지 중복 제거 ID로 메시지 전송을 여러 번 다시 시도할 수 있습니다.
   - 중복 제거 간격이 만료되기 전에 생산자가 하나 이상의 승인을 받는 한는 다음을 재시도합니다.
     - 중복 메시지를 도입하지 마십시오.
     - 메시지 순서를 중단하지 마십시오.
2. 소비자 재시도
   - `ReceiveMessage` 작업이 실패하면 소비자는 동일한 수신 요청 시도 ID를 사용하여 필요한 횟수만큼 재시도할 수 있습니다.
   - 가시성 제한 시간이 만료되기 전에 소비자가 하나 이상의 승인을 받는 한는 다음을 재시도합니다.
     - 메시지 순서를 중단하지 마십시오.

FIFO 동작에 대한 추가 참고 사항

가시성 제한 시간 처리, 여러 메시지 그룹 IDs로 병렬 처리 활성화, 단일 그룹 시나리오에서 엄격한 순차 처리 보장에 대해 알아봅니다.

1. 가시성 제한 시간 처리
   - 메시지가 검색되지만 삭제되지 않으면 표시 제한 시간이 만료될 때까지 표시되지 않습니다.
   - 첫 번째 메시지가 삭제되거나 다시 표시될 때까지 동일한 메시지 그룹 ID의 추가 메시지가 반환되지 않습니다.
2. 동시성 및 병렬 처리
   - FIFO 대기열을 사용하면 다양한 메시지 그룹 IDs.
   - 동시성을 극대화하려면 독립적인 워크플로를 위해 여러 메시지 그룹 IDs로 시스템을 설계합니다.
3. 단일 그룹 시나리오

- FIFO 대기열의 모든 메시지를 엄격하게 순차적으로 처리하려면 대기열의 모든 메시지에 단일 메시지 그룹 ID를 사용합니다.

이해를 돕기 위한 예

다음은 Amazon SQS의 FIFO 대기열 동작을 보여주는 실용적인 시나리오입니다.

1. 시나리오 1: 단일 그룹 ID
   - 생산자는 동일한 메시지 그룹 ID 그룹 A로 5개의 메시지를 보냅니다.
   - 소비자는 이러한 메시지를 FIFO 순서로 수신합니다. 소비자가 이러한 메시지를 삭제하거나 제한 시간 초과가 만료될 때까지 그룹 A의 추가 메시지는 수신되지 않습니다.
2. 시나리오 2: 여러 그룹 IDs
   - 생산자는 그룹 A에 5개의 메시지를 보내고 그룹 B에 5개의 메시지를 보냅니다.
   - 소비자 1은 그룹 A의 메시지를 처리하는 반면 소비자 2는 그룹 B의 메시지를 처리합니다. 이렇게 하면 각 그룹 내에서 엄격한 순서를 유지하여 병렬 처리가 가능합니다.
3. 시나리오 3: 배치 검색
   - 생산자가 그룹 A에 7개의 메시지를 보내고 그룹 B에 3개의 메시지를 보냅니다.
   - 단일 소비자는 최대 10개의 메시지를 검색합니다. 대기열에서 허용하는 경우 다음을 반환할 수 있습니다.
     - 그룹 A의 메시지 7개와 그룹 B의 메시지 3개(또는 단일 그룹에서 사용할 수 있는 메시지가 더 적은 경우 더 적음).

Amazon SQS의 정확히 1회 처리

표준 대기열과 달리 FIFO 대기열에는 중복 메시지가 추가되지 않습니다. FIFO 대기열을 사용하면 대기열에 중복 메시지가 전송되는 것을 방지할 수 있습니다. 5분 중복 제거 간격 내에 `SendMessage` 작업을 재시도하면, Amazon SQS가 대기열에 중복 항목을 추가하지 않습니다.

중복 제거를 구성하려면 다음 중 하나를 실시해야 합니다.

- 콘텐츠 기반 중복 제거 활성화. 이 작업을 통해 Amazon SQS는 SHA-256 해시를 사용한 후 메시지 본문을 사용하여 메시지 중복 제거 ID를 생성하지만 메시지 속성은 생성하지 않습니다. 자세한 내용은 Amazon Simple Queue Service API 참조에서 `CreateQueue`, `GetQueueAttributes` 및 `SetQueueAttributes` 작업에 대한 설명서를 참조하세요.

- 메시지에 메시지 중복 제거 ID 값을 명시적으로 지정하거나 시퀀스 번호를 표시합니다. 자세한 내용은 Amazon Simple Queue Service API 참조에서 `SendMessage`, `SendMessageBatch` 및 `ReceiveMessage` 작업에 대한 설명서를 참조하세요.

Amazon SQS의 표준 대기열에서 FIFO 대기열로 이동

기존 애플리케이션에서 표준 대기열을 사용하며 FIFO 대기열의 정렬 또는 정확히 1회 처리 기능을 활용하려는 경우 대기열과 애플리케이션을 모두 올바르게 구성해야 합니다.

주요 고려 사항

- FIFO 대기열 생성: 기존 표준 대기열을 FIFO 대기열로 변환할 수는 없습니다. 애플리케이션의 새 FIFO 대기열을 만들거나 기존의 표준 대기열을 삭제하고 FIFO 대기열로 다시 만들어야 합니다.
- 지연 파라미터: FIFO 대기열은 메시지당 지연을 지원하지 않고 대기열당 지연만 지원합니다. 애플리케이션에서 각 메시지마다 `DelaySeconds` 파라미터를 설정하는 경우, 수정하여 전체 대기열에서 `DelaySeconds`를 설정해야 합니다.
- 메시지 그룹 ID: 전송된 모든 메시지에 대한 메시지 그룹 ID를 제공합니다. 이 ID를 사용하면 메시지를 병렬 처리하면서 해당 순서를 유지할 수 있습니다. FIFO 대기열을 통해 더 효과적으로 규모를 조정하려면 메시지 그룹 ID에 세분화된 비즈니스 차원을 사용하세요. 메시지를 배포하는 메시지 그룹 ID가 많을수록 사용할 수 있는 메시지 수도 많아집니다.
- 높은 처리량 모드: 처리량을 높이려면 FIFO 대기열에 권장되는 높은 처리량 모드를 사용합니다. 메시징 할당량에 대한 자세한 내용은 Amazon SQS 메시지 할당량 섹션을 참조하세요.

FIFO 대기열로 이동하기 위한 체크리스트

FIFO 대기열로 메시지를 전송하기 전에 다음 사항을 확인하세요.

1. 지연 설정 구성
   - 애플리케이션을 수정하여 메시지당 지연을 제거합니다.
   - 전체 대기열에서 `DelaySeconds` 파라미터를 설정합니다.
2. 메시지 그룹 ID 설정
   - 비즈니스 차원을 기반으로 메시지 그룹 ID를 지정하여 메시지를 메시지 그룹으로 구성합니다.
   - 보다 세분화된 비즈니스 차원을 사용하여 확장성을 개선합니다.
3. 메시지 중복 제거 처리

- 애플리케이션이 동일한 메시지 본문으로 메시지를 전송할 수 없는 경우 각 메시지에 고유한 메시지 중복 제거 ID를 제공합니다.
- 애플리케이션에서 고유한 본문 메시지를 전송하는 경우, 콘텐츠 기반 중복 제거를 활성화합니다.

4. 소비자 구성
   - 일반적으로 소비자는 코드를 변경할 필요가 없습니다.
   - 메시지를 처리하는 데 오랜 시간이 걸리고 표시 제한 시간이 높게 설정된 경우, 각 `ReceiveMessage` 작업에 수신 요청 시도 ID를 추가하는 것이 좋습니다. 그러면 네트워킹 오류가 발생한 경우에도 여러 번 수신 시도를 할 수 있고 실패한 수신 시도로 인해 대기열이 일시 중지되지 않습니다.

다음 단계를 따르면 애플리케이션이 FIFO 대기열에서 올바르게 작동하는지 확인하여 주문 및 정확히 한 번 처리 기능을 최대한 활용할 수 있습니다. 자세한 내용은 Amazon Simple Queue Service API 참조를 참조하세요.

Amazon SQS FIFO 대기열 및 Lambda 동시성 동작

Lambda에서 FIFO(선입선출) 대기열을 사용하면 각 메시지 그룹 내에서 메시지의 정렬된 처리를 보장할 수 있습니다. Lambda 함수는 동일한 메시지 그룹에 대해 여러 인스턴스를 동시에 실행하지 않으므로 순서를 유지합니다. 그러나 여러 메시지 그룹을 병렬로 처리하도록 확장할 수 있으므로 대기열 워크로드를 효율적으로 처리할 수 있습니다. 다음 내용에서는 메시지 그룹 ID와 관련하여 Amazon SQS FIFO 대기열에서 메시지를 처리할 때 Lambda 함수의 동작을 설명합니다.

- 메시지 그룹당 단일 인스턴스: 언제든지 Lambda 인스턴스 하나만 특정 메시지 그룹 ID의 메시지를 처리합니다. 이렇게 하면 동일한 그룹 내의 메시지가 순서대로 처리되어 FIFO 시퀀스의 무결성을 유지할 수 있습니다.
- 여러 그룹의 동시 처리: Lambda는 여러 인스턴스를 사용하여 여러 메시지 그룹 ID의 메시지를 동시에 처리할 수 있습니다. 즉, Lambda 함수의 한 인스턴스가 한 메시지 그룹 ID의 메시지를 처리하는 동안 다른 인스턴스는 다른 메시지 그룹 ID의 메시지를 동시에 처리하여 Lambda의 동시성 기능을 활용함으로써 여러 그룹을 병렬로 처리할 수 있습니다.

FIFO 대기열 메시지 그룹화

FIFO 대기열은 메시지가 전송된 정확한 순서로 처리되도록 합니다. 메시지 그룹 ID를 사용하여 순차적으로 처리해야 하는 메시지를 그룹화합니다.

동일한 메시지 그룹 내의 메시지는 순서대로 처리되며, 이 순서를 유지하기 위해 한 번에 각 그룹의 메시지 하나씩만 처리됩니다.

FIFO 대기열을 사용한 Lambda 동시성

대기열을 생성한 후 그 대기열에 메시지를 보낼 수 있습니다.

Amazon SQS FIFO 대기열의 메시지를 처리하도록 Lambda 함수를 설정하면 Lambda는 FIFO 대기열에서 제공하는 순서 보장을 준수합니다. 다음 내용에서는 메시지 그룹 ID를 사용하는 경우 Amazon SQS FIFO 대기열의 메시지를 처리할 때 동시성 및 규모 조정 측면에서 Lambda 함수의 동작을 설명합니다.

- 메시지 그룹 내 동시성: Lambda 인스턴스는 특정 메시지 그룹 ID에 대해 한 번에 하나의 메시지만 처리합니다. 이렇게 하면 그룹 내의 메시지가 순차적으로 처리됩니다.
- 규모 조정 및 여러 메시지 그룹: Lambda는 메시지를 동시에 처리하도록 확장할 수 있지만 이 규모 조정은 다양한 메시지 그룹에서 발생합니다. 메시지 그룹이 여러 개인 경우 Lambda는 여러 그룹을 병렬로 처리할 수 있으며 각 그룹은 별도의 Lambda 인스턴스에서 처리됩니다.

자세한 내용은 AWS Lambda 운영자 가이드의 Lambda의 규모 조정 및 동시성을 참조하세요.

사용 사례 예제

FIFO 대기열이 동일한 메시지 그룹 ID로 메시지를 수신하고 Lambda 함수의 동시성 한도가 높다고 가정해 보겠습니다(최대 1,000개).

그룹 ID 'A'의 메시지가 처리되고 그룹 ID 'A'의 다른 메시지가 도착하면 첫 번째 메시지가 완전히 처리될 때까지 두 번째 메시지가 새 Lambda 인스턴스를 트리거하지 않습니다.

그러나 그룹 ID 'A' 및 'B'의 메시지가 도착하면 별도의 Lambda 인스턴스에서 두 메시지를 동시에 처리할 수 있습니다.

Amazon SQS에서 FIFO 대기열의 높은 처리량

Amazon SQS의 높은 처리량 FIFO 대기열은 메시지 처리량을 효율적으로 관리하는 동시에 엄격한 메시지 순서를 유지하여 수많은 메시지를 처리하는 애플리케이션의 안정성과 확장성을 보장합니다. 이 솔루션은 처리량이 높고 순서가 지정된 메시지 전송이 모두 필요한 시나리오에 적합합니다.

Amazon SQS 높은 처리량 FIFO 대기열은 엄격한 메시지 순서가 중요하지 않고 수신 메시지의 양이 상대적으로 적거나 산발적인 시나리오에서는 필요하지 않습니다. 예를 들어, 빈번하지 않거나 비순차적

인 메시지를 처리하는 소규모 애플리케이션이 있는 경우 처리량이 높은 FIFO 대기열과 관련된 추가 복잡성과 비용이 정당화되지 않을 수 있습니다. 또한 애플리케이션에 처리량이 높은 FIFO 대기열에서 제공하는 향상된 처리량 기능이 필요하지 않은 경우 표준 Amazon SQS 대기열을 선택하는 것이 더욱 비용 효과적이고 쉽게 관리할 수 있습니다.

처리량이 높은 FIFO 대기열에서 요청 용량을 높이려면 메시지 그룹 수를 늘리는 것이 좋습니다. 처리량이 많은 메시지 할당량에 대한 자세한 내용은 Amazon Web Services 일반 참조의 Amazon SQS 서비스 할당량을 참조하세요.

대기열당 할당량 및 데이터 배포 전략에 대한 자세한 내용은 Amazon SQS 메시지 할당량 및 SQS FIFO 대기열의 높은 처리량을 위한 파티션 및 데이터 배포 섹션을 참조하세요.

Amazon SQS FIFO 대기열의 높은 처리량 사용 사례

다음 사용 사례는 여러 산업과 시나리오에서 효율성을 보여주는 높은 처리량 FIFO 대기열의 다양한 애플리케이션을 강조합니다.

1. 실시간 데이터 처리: 이벤트 처리 또는 원격 측정 데이터 수집과 같은 실시간 데이터 스트림을 처리하는 애플리케이션은 정확한 분석을 위해 순서를 유지하면서 메시지의 지속적인 유입을 처리하는 높은 처리량 FIFO 대기열의 이점을 누릴 수 있습니다.
2. 전자 상거래 주문 처리: 고객 트랜잭션의 순서 유지가 중요한 전자 상거래 플랫폼에서 높은 처리량 FIFO 대기열을 사용하면 쇼핑 성수기에도 주문이 지연 없이 순차적으로 처리됩니다.
3. 금융 서비스: 고빈도 거래 또는 트랜잭션 데이터를 처리하는 금융 기관은 처리량이 높은 FIFO 대기열을 사용하여 메시지 순서에 대한 엄격한 규제 요구 사항을 준수하면서 지연 시간을 최소화하고 시장 데이터 및 트랜잭션을 처리합니다.
4. 미디어 스트리밍: 스트리밍 플랫폼 및 미디어 배포 서비스는 처리량이 높은 FIFO 대기열을 사용하여 미디어 파일 및 스트리밍 콘텐츠 전송을 관리하므로 사용자에게 원활한 재생 환경을 보장하는 동시에 콘텐츠 전송 순서를 올바르게 유지합니다.

SQS FIFO 대기열의 높은 처리량을 위한 파티션 및 데이터 배포

Amazon SQS는 FIFO 대기열 데이터를 파티션에 저장합니다. 파티션은 AWS 리전 내의 여러 가용 영역에 자동으로 복제되는 대기열에 대한 스토리지 할당입니다. 사용자는 파티션을 관리하지 않습니다. 대신 Amazon SQS가 파티션 관리를 처리합니다.

FIFO 대기열의 경우 Amazon SQS는 다음과 같은 상황에서 대기열의 파티션 수를 수정합니다.

- 현재 요청 속도가 기존 파티션이 지원할 수 있는 속도에 근접하거나 초과하면 대기열이 리전별 할당량에 도달할 때까지 추가 파티션이 할당됩니다. 할당량에 대한 자세한 내용은 Amazon SQS 메시지 할당량 섹션을 참조하세요.
- 현재 파티션의 사용률이 낮으면 파티션 수가 줄어들 수 있습니다.

파티션 관리는 백그라운드에서 자동으로 이루어지므로 애플리케이션에는 표시되지 않습니다. 대기열과 메시지를 항상 사용할 수 있습니다.

메시지 그룹 ID별로 데이터 배포

FIFO 대기열에 메시지를 추가하기 위해 Amazon SQS는 각 메시지의 메시지 그룹 ID 값을 내부 해시 함수의 입력으로 사용합니다. 해시 함수 출력 값은 메시지를 저장할 파티션을 결정합니다.

다음 다이어그램은 여러 파티션에 걸쳐 있는 대기열을 보여줍니다. 대기열의 메시지 그룹 ID는 항목 번호를 기반으로 합니다. Amazon SQS는 해시 함수를 사용하여 새 항목을 저장할 위치를 결정합니다. 이 경우 문자열 `item0`의 해시 값이 기준으로 사용됩니다. 항목은 대기열에 추가되는 것과 동일한 순서로 저장된다는 점에 유의하세요. 각 항목의 위치는 메시지 그룹 ID의 해시 값으로 결정됩니다.

> **Note**
>
> Amazon SQS는 고유 값이 많을 수 있는 메시지 그룹 IDs를 사용하는 partitions. AWS recommds 수에 관계없이 FIFO 대기열의 파티션에 항목을 균일하게 배포하도록 최적화되어 있습니다.

파티션 사용률 최적화

각 파티션은 배치 처리 시 초당 최대 3,000개의 메시지를 지원하거나 지원되는 리전에서 전송, 수신 및 삭제 작업의 경우 초당 최대 300개의 메시지를 지원합니다. 처리량이 많은 메시지 할당량에 대한 자세한 내용은 Amazon Web Services 일반 참조의 Amazon SQS 서비스 할당량을 참조하세요.

배치 API를 사용하는 경우 각 메시지는 메시지 그룹 ID별로 데이터 배포에 설명된 프로세스에 따라 라우팅됩니다. 동일한 파티션으로 라우팅되는 메시지는 그룹화되어 단일 트랜잭션으로 처리됩니다.

`SendMessageBatch` API의 파티션 사용률을 최적화하기 위해는 가능하면 동일한 메시지 그룹 IDs로 메시지 일괄 처리를 AWS 권장합니다.

`DeleteMessageBatch` 및 `ChangeMessageVisibilityBatch` APIs의 파티션 사용률을 최적화하려면 `MaxNumberOfMessages` 파라미터가 10으로 설정된 `ReceiveMessage` 요청을 사용하고 단일 `ReceiveMessage` 요청으로 반환된 수신 핸들을 일괄 처리하는 것이 AWS 좋습니다.

다음 예에서는 다양한 메시지 그룹 ID를 가진 메시지 배치 처리가 전송됩니다. 배치 처리는 세 그룹으로 나뉘며, 각 그룹은 파티션의 할당량에 포함됩니다.

> **Note**
>
> Amazon SQS는 동일한 메시지 그룹 ID의 내부 해시 함수를 가진 메시지가 배치 요청 내에서만 그룹화되도록 보장합니다. 내부 해시 함수의 출력과 파티션 수에 따라 메시지 그룹 ID가 다른 메시지가 그룹화될 수 있습니다. 해시 함수 또는 파티션 수는 언제든지 변경될 수 있으므로 한 지점에서 그룹화된 메시지가 나중에 그룹화되지 않을 수 있습니다.

Amazon SQS에서 FIFO 대기열의 높은 처리량 활성화

신규 또는 기존 FIFO 대기열에 높은 처리량을 활성화할 수 있습니다. 이 기능에는 FIFO 대기열을 만들고 편집할 때 사용할 수 있는 새로운 옵션 세 가지가 포함되어 있습니다.

- 높은 처리량 FIFO 활성화 - 현재 FIFO 대기열의 메시지에 높은 처리량을 사용합니다.
- 중복 제거 범위 - 중복 제거를 대기열 또는 메시지 그룹 수준에서 수행할지를 지정합니다.
- FIFO 처리량 한도 - FIFO 대기열에 있는 메시지의 처리량 할당량을 대기열 또는 메시지 그룹 수준에서 설정할지를 지정합니다.

FIFO 대기열의 높은 처리량을 활성화하려면(콘솔)

1. FIFO 대기열 생성 또는 편집을 시작합니다.

2. 대기열 옵션을 지정할 때는 높은 처리량 FIFO 활성화를 선택합니다.

   FIFO 대기열에 높은 처리량을 활성화하면 관련 옵션이 다음과 같이 설정됩니다.

   - 중복 제거 범위가 메시지 그룹으로 설정되어 있으며, 이는 FIFO 대기열에 높은 처리량을 사용하는 데 필요한 설정입니다.
   - FIFO 처리량 한도가 메시지 그룹 ID당으로 설정되어 있으며, 이는 FIFO 대기열에 높은 처리량을 사용하는 데 필요한 설정입니다.

   FIFO 대기열에 높은 처리량을 사용하는 데 필요한 설정을 변경하면 대기열에 일반 처리량이 적용되고 지정된 대로 중복 제거가 수행됩니다.
3. 대기열의 모든 옵션을 계속 지정합니다. 지정을 완료하면 대기열 생성 또는 저장을 선택합니다.

FIFO 대기열을 만들거나 편집한 후에는 이 대기열에 메시지를 보내고 메시지를 수신 및 삭제할 수 있으며, 이 모든 작업을 더 높은 TPS로 수행할 수 있습니다. 높은 처리량 할당량에 대해서는 Amazon SQS 메시지 할당량의 메시지 처리량을 참조하세요.

Amazon SQS의 FIFO 대기열 및 메시지 식별자

이 섹션에서는 FIFO 대기열의 식별자에 대해 설명합니다. 이러한 식별자는 특정 대기열과 메시지를 찾고 조작하는 데 도움이 될 수 있습니다.

Amazon SQS의 FIFO 대기열 식별자

다음 식별자에 대한 자세한 내용은 Amazon Simple Queue Service API 참조를 참조하세요.

대기열 이름 및 URL

새 대기열을 생성할 때는 AWS 계정 및 리전에 고유한 대기열 이름을 지정해야 합니다. Amazon SQS는 사용자가 생성한 각 대기열에 대기열 이름과 기타 Amazon SQS 구성 요소가 포함된 대기열 URL이라는 식별자를 할당합니다. 대기열에서 작업을 실시할 때마다 대기열 URL을 지정합니다.

FIFO 대기열의 이름은 `.fifo` 접미사로 끝나야 합니다. 접미사는 문자 80개의 대기열 이름 할당량에 포함됩니다. 해당 대기열이 접미사로 끝나는지 확인하여 FIFO 대기열인지 여부를 알 수 있습니다.

다음은 AWS 계정 번호 `123456789012`를 갖는 사용자가 소유하는 `MyQueue`라는 이름의 FIFO 대기열의 대기열 URL입니다.

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue.fifo
```

대기열을 나열하고 계정 번호 뒤에 오는 문자열을 구문 분석하여 프로그래밍 방식으로 대기열의 URL을 검색할 수 있습니다. 자세한 내용은 `ListQueues` 단원을 참조하십시오.

메시지 ID

각 메시지는 Amazon SQS가 `SendMessage` 응답으로 반환하는 시스템 할당 메시지 ID를 수신합니다. 이 식별자는 메시지를 식별하는 데 유용합니다. 메시지 ID의 최대 길이는 100자입니다.

수신 핸들

대기열에서 메시지를 수신할 때마다 해당 메시지의 수신 핸들을 수신합니다. 이 핸들은 메시지 자체가 아닌 메시지 수신 작업과 관련되어 있습니다. 메시지를 삭제하거나 메시지 표시를 변경하려면, (메시지 ID가 아닌) 수신 핸들을 지정해야 합니다. 또한, 메시지를 수신한 후 언제든지 삭제할 수 있습니다(메시지를 대기열에 배치한 후 회수할 수 없음). 수신 핸들의 최대 길이는 1024자입니다.

> **Important**
>
> 메시지를 두 번 이상 받으면, 받을 때마다 다른 수신 핸들을 받습니다. 메시지 삭제를 요청할 때에는 가장 최근에 받은 수신 핸들을 제공해야 합니다(그렇지 않으면 메시지가 삭제될 수 있음).

다음은 수신 핸들 예제입니다(세 줄로 나뉨).


```
MbZj6wDWli+JvwwJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```


Amazon SQS FIFO 대기열의 추가 식별자

다음 식별자에 대한 자세한 내용은 Amazon SQS의 정확히 1회 처리 및 Amazon Simple Queue Service API 참조를 참조하세요.

메시지 중복 제거 ID

Amazon SQS FIFO 대기열에 사용되는 토큰은 메시지를 고유하게 식별하고 중복을 방지합니다. 중복 제거 ID가 동일한 여러 메시지가 5분 중복 제거 간격 내에 전송되는 경우 중복으로 취급되고 복사본 하나만 전송됩니다. 중복 제거 ID를 지정하지 않고 콘텐츠 기반 중복 제거가 활성화된 경우 Amazon SQS

는 메시지 본문을 해싱하여 중복 제거 ID를 생성합니다. 이 메커니즘은 지정된 기간 내에 중복 메시지를 제거하여 정확히 한 번 전송되도록 합니다.

메시지 그룹 ID

는 Amazon SQS FIFO(First-In-First-Out) 대기열에서만 메시지를 고유한 그룹으로 구성하는 데 사용되는 속성`MessageGroupId`입니다. 동일한 메시지 그룹 내의 메시지는 항상 한 번에 하나씩 엄격한 순서로 처리되므로 동일한 그룹의 두 메시지가 동시에 처리되지 않습니다. 표준 대기열은를 사용하지 `MessageGroupId` 않으며 주문 보증을 제공하지 않습니다. 엄격한 순서 지정이 필요한 경우 대신 FIFO 대기열을 사용합니다.

시퀀스 번호

Amazon SQS가 각 메시지에 할당하는 연속되지 않은 큰 숫자입니다.

Amazon SQS 할당량

이 주제에서는 Amazon SQS FIFO 및 표준 대기열의 할당량 및 제한 사항을 설명하고 대기열 생성, 구성 및 메시지 처리에 미치는 영향을 자세히 설명합니다. 메시지 보존 한도, 진행 중인 메시지 한도 및 처리량 임계값과 같은 제약 조건과 배치 처리, API 호출 최적화 및 긴 폴링을 통해 효율성을 극대화하는 전략에 대해 알아봅니다. 또한이 주제에서는 고수요 워크로드를 충족하기 위해 할당량 증가를 요청하여 효과적인 대기열 관리와 최적의 성능을 보장하기 위한 이름 지정 규칙, 태그 지정 규칙 및 방법도 다룹니다.

Amazon SQS FIFO 대기열 할당량

Amazon SQS 할당량

다음 표에는 FIFO 대기열과 관련된 할당량이 나열되어 있습니다.

할당량	설명
지연 대기열	대기열의 기본(최소) 지연 시간은 0초입니다. 최대값은 15분입니다.
대기열 나열	`ListQueues` 요청당 대기열 1,000개
긴 폴링 대기 시간	긴 폴링 대기 시간의 최대값은 20초입니다.
메시지 그룹	FIFO 대기열 내 메시지 그룹의 수에는 할당량이 없습니다.
대기열당 메시지(백로그)	Amazon SQS 대기열이 저장할 수 있는 메시지 수에는 제한이 없습니다.
대기열당 메시지(이동 중)	FIFO 대기열은 최대 120,000개의 진행 중인 메시지(소비자가 수신했지만 아직 삭제되지 않은 메시지)를 지원합니다. 이 제한에 도달하면 Amazon SQS는 오류를 반환하지 않지만 처리에 영향을 미칠 수 있습니다. AWS Support에 문의하여이 한도를 초과하는 증가를 요청할 수 있습니다.
대기열 이름	FIFO 대기열의 이름은 `.fifo` 접미사로 끝나야 합니다. 접미사는 문자 80개의 대기열 이름 할당량에 포함됩니다. 대

할당량	설명
	기열 이름이 접미사로 끝나는지 확인하여 FIFO 대기열인지 여부를 알 수 있습니다.
대기열 태그	대기열에 50개 이상의 태그를 추가하지 않는 것이 좋습니다. 태그 지정 시 UTF-8 형식의 유니코드 문자를 사용할 수 있습니다.
	`Key` 태그는 필수지만 `Value` 태그는 선택 사항입니다.
	`Key` 태그 및 `Value` 태그는 대소문자를 구분합니다.
	`Key` 태그 및 `Value` 태그에는 UTF-8 형식의 유니코드 알파벳 문자와 공백이 포함될 수 있습니다. 다음 특수 문자가 허용됩니다. `_ . : / = + - @`
	`Key` 태그 또는 `Value` 태그에 예약된 접두사 `aws:`가 포함되며 안 됩니다(이 접두사가 있는 태그 키 또는 값은 삭제할 수 없음).
	`Key` 태그의 최대 길이는 UTF-8 형식의 유니코드 문자 128자입니다. `Key` 태그는 비어 있거나 null이면 안 됩니다.
	`Value` 태그의 최대 길이는 UTF-8 형식의 유니코드 문자 256자입니다. `Value` 태그는 비어 있거나 null일 수 있습니다.
	태그 지정 작업은 당 30TPS로 제한됩니다 AWS 계정. 애플리케이션에 더 많은 처리량이 필요한 경우 요청을 제출하십시오.

Amazon SQS 표준 대기열 할당량

다음 표에는 표준 대기열과 관련된 할당량이 나열되어 있습니다.

할당량	설명
지연 대기열	대기열의 기본(최소) 지연 시간은 0초입니다. 최대값은 15분입니다.
대기열 나열	`ListQueues` 요청당 대기열 1,000개
긴 폴링 대기 시간	긴 폴링 대기 시간의 최대값은 20초입니다.
대기열당 메시지(백로그)	Amazon SQS 대기열이 저장할 수 있는 메시지 수에는 제한이 없습니다.
대기열당 메시지(이동 중)	대부분의 표준 대기열(대기열 트래픽 및 메시지 백로그에 따라 다름)의 경우 최대 약 120,000개의 이동 중 메시지(소비자가 대기열에서 수신했지만 대기열에서 아직 삭제되지 않은 메시지)가 있을 수 있습니다. 짧은 폴링을 사용하는 동안 이 할당량에 도달하면 Amazon SQS에서 `OverLimit` 오류 메시지를 반환합니다. 긴 폴링을 사용하는 경우에는 Amazon SQS에서 오류 메시지를 반환하지 않습니다. 이 할당량에 도달하지 않도록 하려면 메시지를 처리한 후 대기열에서 삭제해야 합니다. 또한 메시지를 처리하는 데 사용하는 대기열의 수를 늘릴 수 있습니다. 할당량 증가를 요청하려면 지원 요청을 제출하십시오.
대기열 이름	대기열 이름은 최대 80자까지 가능합니다. 허용되는 문자는 영숫자, 하이픈(-), 밑줄(_)입니다. **Note** 대기열 이름은 대/소문자를 구분합니다(예: `Test-queue` 와 `test-queue` 는 다른 대기열임).
대기열 태그	대기열에 50개 이상의 태그를 추가하지 않는 것이 좋습니다. 태그 지정 시 UTF-8 형식의 유니코드 문자를 사용할 수 있습니다.
	`Key` 태그는 필수지만 `Value` 태그는 선택 사항입니다.

할당량	설명
	Key 태그 및 Value 태그는 대소문자를 구분합니다.
	Key 태그 및 Value 태그에는 UTF-8 형식의 유니코드 알파벳 문자와 공백이 포함될 수 있습니다. 다음 특수 문자가 허용됩니다. _ . : / = + - @
	Key 태그 또는 Value 태그에 예약된 접두사 aws:가 포함되며 안 됩니다(이 접두사가 있는 태그 키 또는 값은 삭제할 수 없음).
	Key 태그의 최대 길이는 UTF-8 형식의 유니코드 문자 128자입니다. Key 태그는 비어 있거나 null이면 안 됩니다.
	Value 태그의 최대 길이는 UTF-8 형식의 유니코드 문자 256자입니다. Value 태그는 비어 있거나 null일 수 있습니다.
	태그 지정 작업은 당 30TPS로 제한됩니다 AWS 계정. 애플리케이션에 더 많은 처리량이 필요한 경우 요청을 제출하십시오.

Amazon SQS 메시지 할당량

다음 표에 메시지와 관련된 할당량이 나열되어 있습니다.

할당량	설명
일괄 메시지 ID	일괄 메시지 ID는 최대 80자까지 가능합니다. 허용되는 문자는 영숫자, 하이픈(-), 밑줄(_) 입니다.
메시지 속성	메시지 하나에는 최대 10개의 메타데이터 속성이 있을 수 있습니다.
메시지 배치	단일 메시지 배치 요청에는 최대 10개 메시지가 포함될 수 있습니다. 자세한 내용은 Amazon SQS 배치 작업 섹션의

할당량	설명
	AmazonSQSBufferedAsyncClient 구성 섹션을 참조하세요.
메시지 콘텐츠	메시지에는 XML, JSON 및 포맷되지 않은 텍스트만이 포함될 수 있습니다. 다음 유니코드 문자가 허용됩니다. `#x9` | `#xA` | `#xD` | `#x20` ~ `#xD7FF` | `#xE000` to `#xFFFD` | `#x10000` ~ `#x10FFFF` 이 목록에 포함되지 않은 문자는 거부됩니다. 자세한 내용은 문자에 대한 W3C 사양을 참조하세요.
메시지 그룹 ID	`MessageGroupId` 는 FIFO 대기열에 필요합니다. FIFO 대기열로 메시지를 보낼 `MessageGroupId` 때를 제공하지 않으면 작업이 실패합니다. 표준 대기열에서를 사용하면 공정 대기열이 `MessageGroupId` 활성화됩니다. 공정 대기열을 사용할 때는 모든 메시지`MessageGroupId` 에를 포함하는 것이 좋습니다. `MessageGroupId` 의 최대 길이는 128자입니다. 유효한 값은 영숫자 및 문장 부호`(!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)` 입니다.
메시지 보존	기본적으로 메시지는 4일간 보존됩니다. 최소 기간은 60초(1분)입니다. 최소 시간은 1,209,600초(14일)입니다.
메시지 처리량	표준 대기열 표준 대기열은 작업(`SendMessage` , `ReceiveMessage` 또는 `DeleteMessage`)별 초당 거의 무제한의 매우 높은 API 직접 호출 수를 지원합니다. 이렇게 처리량이 높기 때문에 실시간 데이터 스트리밍 또는 대규모 애플리케이션과 같이 대량의 메시지를 빠르게 처리해야 하는 사용 사례에 적합합니다. 표준 대기열은 수요에 따라 자동으로 확장되지만 특히 워크로드가 많은 리전에서는 최적의 성능을 보장하기 위해 사용 패턴을 모니터링하는 것이 중요합니다.

할당량	설명
	FIFO 대기열 • FIFO 대기열의 각 파티션은 API 작업(`SendMessage` , `ReceiveMessage` 및 `DeleteMessage`)별 초당 300개의 트랜잭션으로 제한됩니다. 이 제한은 특히 처리량이 높지 않은 모드에 적용됩니다. 높은 처리량 모드로 전환하면 이 기본 한도를 초과할 수 있습니다. 높은 처리량 모드를 활성화하려면 Amazon SQS에서 FIFO 대기열의 높은 처리량 활성화 섹션을 참조하세요. • 배치 처리를 사용하면 처리량이 높지 않은 FIFO 대기열은 API 작업(`SendMessage` , `ReceiveMessage` 또는 `DeleteMessage`)별 초당 최대 3,000개의 메시지를 지원합니다. 초당 3,000개의 메시지는 300개의 API 호출을 나타내며, 각각 10개의 메시지를 배치 처리합니다. FIFO 대기열의 높은 처리량 Amazon SQS FIFO 제한은 메시지 제한이 아닌 API 요청 수를 기준으로 합니다. 높은 처리량 모드의 경우 이러한 API 요청 제한은 다음과 같습니다. 트랜잭션 처리량 제한(배치가 아닌 API 직접 호출) 이러한 제한은 각 API 작업(예: SendMessage, ReceiveMessage 또는 DeleteMessage)을 독립적으로 수행할 수 있는 빈도를 정의하여 허용되는 초당 트랜잭션(TPS) 내에서 효율적인 시스템 성능을 보장합니다. 다음 제한은 배치가 아닌 API 직접 호출을 기반으로 합니다. • 미국 동부(버지니아 북부), 미국 서부(오리건) 및 유럽(아일랜드): 최대 70,000TPS(초당 트랜잭션 수). • 미국 동부(오하이오) 및 유럽(프랑크푸르트): 최대 19,000TPS.

할당량	설명
	• (아시아 태평양(뭄바이), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄): 최대 9,000TPS. • 유럽(런던) 및 남아메리카(상파울루): 최대 4,500TPS. • 기타: AWS 리전기본 처리량은 2,400TPS입니다. 배치 처리를 통한 처리량 극대화 단일 API 직접 호출로 여러 메시지를 처리하여 효율성을 크게 높입니다. 각 메시지를 개별적으로 처리하는 대신 배치 처리를 사용하면 단일 API 요청으로 최대 10개의 메시지를 전송, 수신 또는 삭제할 수 있습니다. 이렇게 하면 총 API 직접 호출 수가 줄어들어 리전의 트랜잭션 한도(TPS) 내에 있는 동안 초당 더 많은 메시지를 처리할 수 있으므로 처리량과 시스템 성능이 극대화됩니다. 자세한 내용은 Amazon SQS로 수평적 조정과 작업 배치 처리를 사용하여 처리량 증대 단원을 참조하십시오. 다음 제한은 배치 API 직접 호출을 기반으로 합니다. • 미국 동부(버지니아 북부), 미국 서부(오리건) 및 유럽(아일랜드): 초당 최대 700,000개의 메시지(배치가 아닌 제한 70,000TPS의 10배). • 미국 동부(오하이오) 및 유럽(프랑크푸르트): 초당 최대 190,000개의 메시지. • (아시아 태평양(뭄바이), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄): 초당 최대 90,000개의 메시지. • 유럽(런던) 및 남아메리카(상파울루): 초당 최대 45,000개의 메시지. • 기타 모두 AWS 리전: 초당 최대 24,000개의 메시지. 배치 처리 이상으로 처리량 최적화

할당량	설명
	배치 처리는 처리량을 크게 증가시킬 수 있지만 FIFO 성능을 최적화하기 위해서는 다른 전략을 고려하는 것이 중요합니다. • 여러 메시지 그룹 ID에 메시지 배포 - 단일 그룹 내의 메시지는 순차적으로 처리되므로, 여러 메시지 그룹에 워크로드를 분산하면 병렬 처리가 향상되고 전체 처리량이 높아집니다. 자세한 내용은 SQS FIFO 대기열의 높은 처리량을 위한 파티션 및 데이터 배포 단원을 참조하십시오. • API 직접 호출의 효율적인 사용 - 빈번한 표시 여부 변경 또는 반복되는 메시지 삭제와 같은 불필요한 API 직접 호출을 최소화하여 사용 가능한 TPS 사용을 최적화하고 효율성을 개선합니다. • 긴 폴링 수신 사용 - 사용할 수 있는 메시지가 없을 때 빈 응답을 줄이고 불필요한 API 직접 호출을 줄이며 TPS 할당량을 더 잘 사용할 수 있도록 수신 요청에 `WaitTimeSeconds` 를 설정하여 긴 폴링을 활용합니다. • 처리량 증가 요청 - 애플리케이션에 기본 제한보다 높은 처리량이 필요한 경우 Service Quotas 콘솔을 사용하여 증가를 요청합니다. 이는 수요가 많은 워크로드 또는 기본 한도가 더 낮은 리전에 필요할 수 있습니다. 높은 처리량 모드를 활성화하려면 Amazon SQS에서 FIFO 대기열의 높은 처리량 활성화 섹션을 참조하세요.
메시지 타이머	메시지의 기본(최소) 지연 시간은 0초입니다. 최대값은 15분입니다.

할당량	설명
메시지 크기	최소 메시지 크기는 1바이트(1자리)입니다. 최대값은 1,048,576바이트(1MiB)입니다. 1MiB보다 큰 메시지를 보내려면 Java용 Amazon SQS 확장 클라이언트 라이브러리와 Python용 Amazon SQS 확장 클라이언트 라이브러리를 사용할 수 있습니다. 이 라이브러리를 통해 Amazon S3에 메시지 페이로드에 대한 참조를 포함하는 Amazon SQS 메시지를 전송할 수 있습니다. 최대 페이로드 크기는 2GB입니다. **Note** 이 확장 라이브러리는 동기식 클라이언트에서만 작동합니다.
메시지 제한 시간 초과	메시지의 제한 시간 초과는 30초입니다. 최소 시간은 0초입니다. 최대 시간은 12시간입니다.
정책 정보	최대 할당량은 8,192바이트, 20개 문, 주체 50개 또는 조건 10개입니다. 자세한 내용은 Amazon SQS 정책 할당량 단원을 참조하십시오.

Amazon SQS 정책 할당량

다음 표에 정책과 관련된 할당량이 나열되어 있습니다.

명칭	Maximum
바이트	8,192
조건	10
Principal	50
Statement	20

명칭	Maximum
명령문당 작업	7

Amazon SQS의 특징 및 기능

이 주제에서는 메시지 대기열 관리, 성능 최적화, 신뢰할 수 있는 메시지 전송 보장 및 메시지 처리의 효율적인 처리를 위해 Amazon SQS에서 일반적으로 사용되는 기능을 제공합니다.

Amazon SQS에서 배달 못한 편지 대기열 사용

Amazon SQS는 성공적으로 처리되지 않은 메시지를 소스 대기열에서 대상으로 지정할 수 있는 배달 못한 편지 대기열을 지원합니다. 소비되지 않은 메시지를 구분하여 처리가 실패한 이유를 확인할 수 있으므로 DLQ는 애플리케이션을 디버깅하는 데 유용합니다. 최적의 성능을 위해 소스 대기열과 DLQ를 동일한 AWS 계정 및 리전 내에 유지하는 것이 가장 좋습니다. 메시지가 배달 못한 편지 대기열에 있으면 다음을 수행할 수 있습니다.

- 메시지가 DLQ(Dead Letter Queue)로 이동되게 만든 예외적인 상황의 로그를 검사합니다.
- 배달 못한 편지 대기열로 이동한 메시지의 내용을 분석하여 애플리케이션 문제를 진단합니다.
- 소비자에게 메시지를 처리하기에 충분한 시간을 주었는지 확인합니다.
- 배달 못한 편지 대기열 리드라이브를 사용하여 배달 못한 편지 대기열에서 메시지를 이동합니다.

대기열을 배달 못한 편지 대기열로 구성하기 전에 먼저 새 대기열을 생성해야 합니다. Amazon SQS 콘솔을 사용하여 DLQ(Dead Letter Queue)를 구성하는 방법에 대한 자세한 내용은 Amazon SQS 콘솔을 사용하여 배달 못한 편지 대기열 구성 섹션을 참조하세요. 배달 못한 편지 대기열로 이동한 메시지에 대한 경보를 구성하는 방법 등 배달 못한 편지 대기열에 대한 도움말은 Amazon CloudWatch를 사용하여 배달 못한 편지 대기열에 대한 경보 생성 섹션을 참조하세요.

> **Note**
>
> 메시지 또는 작업의 정확한 순서를 지키고 싶다면 FIFO 대기열과 함께 DLQ(Dead Letter Queue)를 사용하지 마세요. 예를 들어, 비디오 편집 모음에 대한 Edit Decision List(EDL)의 지침과 함께 DLQ(Dead Letter Queue)를 사용하지 마세요. 편집 순서를 변경하면 후속 편집의 컨텍스트가 변경됩니다.

배달 못한 편지 대기열에 대한 정책 사용

리드라이브 정책을 사용하여 `maxReceiveCount`를 지정합니다. `maxReceiveCount`는 소비자가 배달 못한 편지 대기열로 이동하기 전에 대기열에서 메시지를 삭제하지 않고 수신하려고 시도한 횟수입

니다. 예를 들어 `maxReceiveCount`를 1과 같이 낮은 값으로 설정하면 메시지 수신에 1회 실패할 경우 메시지가 배달 못한 편지 대기열로 이동하게 됩니다. 시스템이 오류에 대해 복원력을 갖도록 하려면 충분한 재시도를 허용할 수 있을 정도로 `maxReceiveCount`를 높게 설정하세요.

리드라이브 허용 정책은 DLQ(Dead Letter Queue)에 액세스할 수 있는 소스 대기열을 지정합니다. 모든 소스 대기열을 허용할지, 특정 소스 대기열을 허용할지, 모든 소스 대기열의 배달 못한 편지 대기열 사용을 거부할지 선택할 수 있습니다. 기본값은 모든 소스 대기열이 배달 못한 편지 대기열을 사용하도록 허용합니다. `byQueue` 옵션 사용하여 특정 대기열을 허용하도록 선택하는 경우, 소스 대기열 Amazon 리소스 이름(ARN)을 사용하여 최대 10개의 소스 대기열을 지정할 수 있습니다. `denyAll`을 지정하는 경우, 대기열을 DLQ(Dead Letter Queue)로 사용할 수 없습니다.

배달 못한 편지 대기열의 메시지 보존 기간 이해

표준 대기열의 경우, 메시지 만료는 항상 메시지가 원래 대기열에 추가된 타임스탬프를 기준으로 합니다. 메시지가 DLQ(Dead Letter Queue)로 이동하면 대기열에 추가 타임스탬프는 변경되지 않습니다. `ApproximateAgeOfOldestMessage` 지표는 메시지가 원래 전송된 시간이 아니라 메시지가 DLQ(Dead Letter Queue)로 이동한 시간을 나타냅니다. 예를 들어, 메시지가 DLQ(Dead Letter Queue)로 이동하기 전에 원래 대기열에서 1일을 보낸다고 가정합니다. DLQ(Dead Letter Queue)의 보존 기간이 4일인 경우 메시지는 3일 후에 DLQ(Dead Letter Queue)에서 삭제되며 `ApproximateAgeOfOldestMessage`는 3일입니다. 따라서 배달 못한 편지 대기열의 보존 기간을 항상 원래 대기열의 보존 기간보다 더 길게 설정하는 것이 좋습니다.

FIFO 대기열의 경우 메시지가 DLQ(Dead Letter Queue)로 이동하면 대기열에 추가 타임스탬프가 재설정됩니다. `ApproximateAgeOfOldestMessage` 지표는 메시지가 DLQ(Dead Letter Queue)로 이동한 시간을 나타냅니다. 위의 동일한 예제에서 메시지는 4일 후에 배달 못한 편지 대기열에서 삭제되고 `ApproximateAgeOfOldestMessage`는 4일입니다.

Amazon SQS 콘솔을 사용하여 배달 못한 편지 대기열 구성

DLQ(Dead Letter Queue)는 소스 대기열이라고 하는 다른 대기열에서 성공적으로 처리되지 않은 메시지를 수신하는 대기열입니다. Amazon SQS는 DLQ(Dead Letter Queue)를 자동으로 생성하지 않습니다. 대기열을 배달 못한 편지 대기열로 사용하기 전에 먼저 대기열을 생성해야 합니다. DLQ를 구성할 때 대기열 유형은 소스 대기열 유형과 일치해야 합니다. FIFO 대기열은 FIFO DLQ만 사용할 수 있고 표준 대기열은 표준 DLQ만 사용할 수 있습니다. 대기열을 생성하거나 편집할 때 배달 못한 편지 대기열을 구성할 수 있습니다. 자세한 내용은 Amazon SQS에서 배달 못한 편지 대기열 사용 을 참조하세요.

기존 대기열에 대한 배달 못한 편지 대기열을 구성하려면(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 소스 대기열(실패한 메시지를 배달 못한 편지 대기열로 전송할 대기열)을 선택한 다음 편집을 선택합니다.
4. 배달 못한 편지 대기열 섹션으로 스크롤하고 활성화됨을 전환합니다.
5. 배달 못한 편지 대기열 설정에서 배달 못한 편지 대기열로 사용할 기존 대기열의 Amazon 리소스 이름(ARN)을 선택합니다.
6. 배달 못한 편지 대기열로 전송되기 전에 메시지를 수신할 수 있는 횟수를 정의하는 최대 수신 수 값을 설정합니다(유효 범위: 1~1,000).
7. 저장을 선택합니다.

Amazon SQS에서 배달 못한 편지 대기열 리드라이브를 구성하는 방법 알아보기

배달 못한 편지 대기열 리드라이브를 사용하여 소비되지 않은 메시지를 배달 못한 편지 대기열에서 처리를 위해 다른 대상으로 이동합니다. 기본적으로 DLQ(Dead Letter Queue) 리드라이브는 메시지를 DLQ(Dead Letter Queue)에서 소스 대기열로 이동합니다. 그러나 두 대기열이 동일한 유형인 경우 다른 대기열을 리드라이브 대상으로 구성할 수도 있습니다. 예를 들어 DLQ(Dead Letter Queue)가 FIFO 대기열인 경우 리드라이브 대상 대기열도 FIFO 대기열이어야 합니다. 또한 Amazon SQS가 메시지를 이동하는 속도를 설정하도록 리드라이브 속도를 구성할 수 있습니다.

> **Note**
>
> 메시지가 FIFO 대기열에서 FIFO DLQ로 이동되면 원본 메시지의 중복 제거 ID가 원본 메시지의 ID로 대체됩니다. 이는 중복 제거 ID를 공유하는 두 개의 독립적인 메시지가 DLQ 중복 제거로 인해 저장되지 않도록 하기 위한 것입니다.

배달 못한 편지 대기열은 가장 오래된 메시지부터 시작하여 수신된 순서대로 메시지를 리드라이브합니다. 그러나 대상 대기열은 수신 순서에 따라 리드라이브된 메시지뿐만 아니라 다른 생산자의 새 메시지도 수집합니다. 예를 들어 생산자가 소스 FIFO 대기열로 메시지를 보내는 동시에 배달 못한 편지 대기열에서 리드라이브된 메시지를 수신하는 경우, 리드라이브된 메시지는 생산자가 보낸 새 메시지와 섞이게 됩니다.

> **Note**
> 리드라이브 작업은 보존 기간을 재설정합니다. 모든 리드라이브된 메시지는 새 `messageID` 및 `enqueueTime`이 있는 새 메시지로 간주되며 리드라이브된 메시지에 할당됩니다.

Amazon SQS API를 사용하여 기존 표준 대기열에 대한 배달 못한 편지 대기열 리드라이브 구성

`StartMessageMoveTask`, `ListMessageMoveTasks` 및 `CancelMessageMoveTask` API 작업을 사용하여 배달 못한 편지 대기열 리드라이브를 구성할 수 있습니다.

API 작업	설명
`StartMessageMoveTask`	지정된 소스 대기열에서 지정된 대상 대기열로 메시지를 이동하는 비동기 작업을 시작합니다.
`ListMessageMoveTasks`	특정 소스 대기열에서 가장 최근의 메시지 이동 작업(최대 10개)을 가져옵니다.
`CancelMessageMoveTask`	지정된 메시지 이동 작업을 취소합니다. 메시지 이동은 현재 상태가 실행 중일 때만 취소할 수 있습니다.

Amazon SQS 콘솔을 사용하여 기존 표준 대기열에 대한 배달 못한 편지 대기열 리드라이브 구성

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 배달 못한 편지 대기열로 구성한 대기열의 이름을 선택합니다.
4. DLQ 리드라이브 시작을 선택합니다.
5. 리드라이브 구성에서 메시지 대상에 대해 다음 중 하나를 수행합니다.
   - 메시지를 소스 대기열로 리드라이브하려면 소스 대기열로 리드라이브를 선택합니다.
   - 메시지를 다른 대기열로 리드라이브하려면 사용자 지정 대상으로 리드라이브를 선택합니다. 그런 다음 기존 대상 대기열의 Amazon 리소스 이름(ARN)을 입력합니다.

6. 속도 제어 설정에서 다음 중 하나를 선택합니다.

   - 시스템 최적화 - 배달 못한 편지 대기열 메시지를 초당 최대 메시지 수로 리드라이브합니다.

   - 사용자 지정 최고 속도 - 배달 못한 편지 대기열 메시지를 초당 최대 사용자 지정 메시지 속도로 리드라이브합니다. 최대 허용 속도는 초당 500개의 메시지입니다.
     - 사용자 지정 최고 속도를 작은 값으로 시작하고 소스 대기열에 메시지가 넘치지 않는지 확인하는 것이 좋습니다. 그런 다음 사용자 지정 최대 속도 값을 점진적으로 늘려 소스 대기열의 상태를 계속 모니터링합니다.

7. 배달 못한 편지 대기열 리드라이브 구성을 마쳤으면 메시지 리드라이브를 선택합니다.

   > **Important**
   >
   > Amazon SQS는 배달 못한 편지 대기열에서 메시지를 리드라이브하는 동안 메시지를 필터링하고 수정하는 기능을 지원하지 않습니다.
   > DLQ(Dead Letter Queue) 리드라이브 작업은 최대 36시간까지 실행될 수 있습니다.
   > Amazon SQS는 계정당 최대 100개의 활성 리드라이브 작업을 지원합니다.

8. 메시지 리드라이브 작업을 취소하려면 대기열의 세부 정보 페이지에서 DLQ 리드라이브 취소를 선택합니다. 진행 중인 메시지 리드라이브를 취소하면 이미 이동 대상 대기열로 이동된 모든 메시지는 대상 대기열에 남아 있게 됩니다.

배달 못한 편지 대기열 리드라이브에 대한 대기열 권한 구성

정책에 권한을 추가하여 사용자에게 특정 배달 못한 편지 대기열 작업에 대한 액세스 권한을 부여할 수 있습니다. 배달 못한 편지 대기열 리드라이브에 필요한 최소 권한은 다음과 같습니다.

최소 권한	필수 API 메서드
메시지 리드라이브 시작	• 배달 못한 편지 대기열의 `sqs:StartMessageMoveTask` , `sqs:ReceiveMessage` , `sqs:DeleteMessage` 및 `sqs:GetQueueAttributes` 를 추가합니다. 배달 못한 편지 대기열 또는 원본 소스 대기열이 암호화된 경우(SSE 대기열이라고도 함) 메시지를 암호화하는 데 사용된 모든 KMS 키에 대한 `kms:Decrypt` 도 필요합니다. •

최소 권한	필수 API 메서드
	대상 대기열의 `sqs:SendMessage` 를 추가합니다. 대상 대기열이 암호화된 경우 `kms:GenerateDataKey` 및 `kms:Decrypt` 도 필요합니다.
진행 중인 메시지 리드라이브 취소	• 배달 못한 편지 대기열의 `sqs:CancelMessageMoveTask` , `sqs:ReceiveMessage` , `sqs:DeleteMessage` 및 `sqs:GetQueueAttributes` 를 추가합니다. 배달 못한 편지 대기열이 암호화된 경우(SSE 대기열이라고도 함) `kms:Decrypt` 도 필요합니다.
메시지 이동 상태 표시	• 배달 못한 편지 대기열의 `sqs:ListMessageMoveTasks` 및 `sqs:GetQueueAttributes` 를 추가합니다.

암호화된 대기열 페어(배달 못한 편지 대기열이 있는 소스 대기열)에 대한 권한을 구성하려면

다음 단계에 따라 배달 못한 편지 대기열(DLQ) 리드라이브에 대한 최소 권한을 구성합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 새 정책을 생성하고 다음 권한을 추가합니다. 리드라이브 작업을 수행할 IAM 사용자 또는 역할에 정책을 연결합니다.
   - DLQ(소스 대기열)에 대한 권한:
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:GetQueueAttributes`
     - `sqs:ListDeadLetterSourceQueues`
     - DLQ(소스 대기열)의 리소스 ARN을 지정합니다(예: "arn:aws:sqs:*<DLQ_region>*:*<DLQ_accountId>*:*<DLQ_name>*").
   - 대상 대기열에 대한 권한:

    - `sqs:SendMessage`
    - 대상 대기열`Resource ARN`의를 지정합니다(예: "arn:aws:sqs:*<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*").
- KMS 키에 대한 권한:
    - `kms:Decrypt` (DLQ에서 메시지를 복호화하는 데 필요합니다.)
    - `kms:GenerateDataKey` (대상 대기열의 메시지를 암호화하는 데 필요합니다.)
        - `Resource` ARNs
            - DLQ(소스 대기열)에서 메시지를 암호화하는 데 사용되는 KMS 키의 ARN입니다(예: "arn:aws:kms:*<region>*:*<accountId>*:key/*<SourceQueueKeyId>*").
            - 대상 대기열의 메시지를 암호화하는 데 사용되는 KMS 키의 ARN입니다(예: "arn:aws:kms:*<region>*:*<accountId>*:key/*<DestinationQueueKeyId>*").

액세스 정책은 다음과 유사합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:StartMessageMoveTask",
                "sqs:CancelMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:ListDeadLetterSourceQueues"
            ],
            "Resource":
 "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "source"
                }
            }
        },
```

```
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "destination"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": [
                "arn:aws:kms:<region>:<accountId>:key/<SourceQueueKeyId>",
                "arn:aws:kms:<region>:<accountId>:key/<DestQueueKeyId>"
            ]
        }
    ]
}
```

암호화되지 않은 대기열 페어(배달 못한 편지 대기열이 있는 소스 대기열)을 사용하여 권한을 구성하려면

다음 단계에 따라 암호화되지 않은 표준 배달 못한 편지 대기열(DLQ)을 처리하는 데 필요한 최소 권한을 구성합니다. 필요한 최소 권한은 배달 못한 편지 대기열에서 속성을 수신, 삭제 및 가져오고 소스 대기열로 속성을 전송하는 것입니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 새 정책을 생성하고 다음 권한을 추가합니다. 리드라이브 작업을 수행할 IAM 사용자 또는 역할에 정책을 연결합니다.
   - DLQ(소스 대기열)에 대한 권한:
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`

  - `sqs:ListMessageMoveTasks`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:ListDeadLetterSourceQueues`
  - DLQ(소스 대기열)의 리소스 ARN을 지정합니다(예: "arn:aws:sqs:*<DLQ_region>*:*<DLQ_accountId>*:*<DLQ_name>*").
- 대상 대기열에 대한 권한:
  - `sqs:SendMessage`
  - 대상 대기열`Resource ARN`의를 지정합니다(예: "arn:aws:sqs:*<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*").

액세스 정책은 다음과 유사합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:StartMessageMoveTask",
                "sqs:CancelMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:ListDeadLetterSourceQueues"
            ],
            "Resource":
 "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "source"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "destination"
                }
            }
        }
    ]
}
```

VPC 엔드포인트 액세스 제어와 함께 배달 못한 편지 대기열 리드라이브 사용

`aws:sourceVpc` 조건을 사용하여 특정 VPCs에 대한 대기열 액세스를 제한하는 경우 DLQ(Dead Letter Queue) 리드라이브 기능을 활성화하려면 AWS 서비스에 예외를 적용해야 합니다. 이는 메시지를 이동할 때 Amazon SQS 서비스가 VPC 외부에서 작동하기 때문입니다.

DLQ 리드라이브 작업을 허용하려면 대기열 정책에 `aws:CalledViaLast` 조건을 추가합니다. 이를 통해 Amazon SQS는 직접 액세스에 대한 VPC 제한을 유지하면서 사용자를 대신하여 API를 호출할 수 있습니다.

VPC 제한 액세스와 DLQ 리드라이브를 모두 허용하려면:

1. 대기열 정책에서 `aws:CalledViaLast` 조건을 사용합니다.
2. 소스 대기열과 DLQ 모두에 정책 적용
3. 다른 소스에서 직접 액세스하기 위한 VPC 제한 유지

다음은 이러한 요구 사항을 구현하는 정책의 예입니다.

```
{
  "Version": "2012-10-17",
  "Id": "SQSRedriveWithVpcRestriction",
  "Statement": [
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DestQueue",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
```

```
      "Resource": "arn:aws:sqs:*:111122223333:DestQueue",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "StringNotEqualsIfExists": {
          "aws:CalledViaLast": "sqs.amazonaws.com"
        }
      }
    },
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DLQ",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:Dlq",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "StringNotEqualsIfExists": {
          "aws:CalledViaLast": "sqs.amazonaws.com"
        }
      }
    }
  ]
}
```

- 자리 표시자 값을 실제 값으로 바꿉니다.
- 이 정책은 "허용" 문을 사용하는 것보다 더 안전한 조건과 함께 "거부" 문을 사용합니다.
- `StringNotEqualsIfExists` 연산자는 요청 컨텍스트에 조건 키가 없을 수 있는 사례를 처리합니다.

또는 `aws:ViaAWSService` 조건 키를 사용하여 VPC 제한을 유지하면서 서비스 기반 액세스를 허용할 수 있습니다. 이 조건 키는 요청이 AWS 서비스에서 왔는지 여부를 나타냅니다. 다음은 `aws:ViaAWSService` 대신를 사용하는 정책의 예입니다`aws:CalledViaLast`.

```
{
  "Version": "2012-10-17",
  "Id": "SQSRedriveWithVpcRestriction",
  "Statement": [
```

```
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DestQueue",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:DestQueue",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    },
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DLQ",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:Dlq",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

`aws:ViaAWSService` 조건이 있는 BoolIfExists 연산자는 직접 액세스에 대한 VPC 제한을 유지하면서 요청이 서비스에서 오는 경우 요청이 허용되도록 합니다. 이는 마지막 호출을 수행한 AWS 서비스를 확인하는 대신 서비스에 의해 요청이 이루어졌는지 직접 확인하므로 이해하고 유지 관리하는 것이 더 간단할 수 있습니다.

IAM 및 리소스 정책에 사용되는 조건 키에 대한 자세한 내용은 IAM JSON 정책 요소: 조건을 참조하세요.

Amazon SQS 배달 못한 편지 대기열 리드라이브를 위한 CloudTrail 업데이트 및 권한 요구 사항

2023년 6월 8일에 Amazon SQS는 SDK 및 (CLI)용 AWS DLQ AWS Command Line Interface (Dead Letter Queue) 리드라이브를 도입했습니다. 이 기능은 AWS 콘솔에서 이미 지원되는 DLQ 리드라이브에 추가됩니다. 이전에 AWS 콘솔을 사용하여 배달 못한 편지 대기열 메시지를 리드라이브한 경우 다음 변경 사항의 영향을 받을 수 있습니다.

CloudTrail 이벤트 이름 변경

2023년 10월 15일에 Amazon SQS 콘솔에서 DLQ(Dead Letter Queue) 리드라이브에 대한 CloudTrail 이벤트 이름이 변경됩니다. 이러한 CloudTrail 이벤트에 대한 경보를 설정한 경우 지금 업데이트해야 합니다. DLQ 리드라이브를 위한 새로운 CloudTrail 이벤트 이름은 다음과 같습니다.

이전 이벤트 이름	새로운 이벤트 이름
`CreateMoveTask`	`StartMessageMoveTask`
`CancelMoveTask`	`CancelMessageMoveTask`

업데이트된 권한

SDK 및 CLI 릴리스에 포함된 Amazon SQS는 보안 모범 사례를 준수하기 위해 DLQ 리드라이브에 대한 대기열 권한도 업데이트했습니다. 다음 대기열 권한 유형을 사용하여 DLQ에서 메시지를 리드라이브하세요.

1. 작업 기반 권한(DLQ API 작업에 대한 업데이트)
2. 관리형 Amazon SQS 정책 권한
3. sqs:* 와일드카드를 사용하는 권한 정책

> **Important**
>
> SDK 또는 CLI용 DLQ 리드라이브를 사용하려면 위의 옵션 중 하나와 일치하는 DLQ 리드라이브 권한 정책이 있어야 합니다.

DLQ 리드라이브에 대한 대기열 권한이 위의 옵션 중 하나와 일치하지 않는 경우 2023년 8월 31일까지 권한을 업데이트해야 합니다. 지금부터 2023년 8월 31일까지는 이전에 DLQ 리드라이브를 사용한 리전에서만 AWS 콘솔을 통해 구성한 권한을 사용하여 메시지를 리드라이브할 수 있습니다. 예를 들어, us-east-1과 eu-west-1에 모두 '계정 A'가 있다고 가정해 보겠습니다. "계정 A"는 2023년 6월 8일 이전에 us-east-1의 AWS 콘솔에서 메시지를 리드라이브하는 데 사용되었지만 eu-west-1에서는 사용되지 않았습니다. 2023년 6월 8일부터 2023년 8월 31일까지 '계정 A'의 정책 권한이 위의 옵션 중 하나와 일치하지 않는 경우 eu-west-1이 아닌 us-east-1의 AWS 콘솔에서 메시지를 리드라이브하는 데만 사용할 수 있습니다.

> **Important**
>
> 2023년 8월 31일 이후에 DLQ 리드라이브 권한이 이러한 옵션 중 하나와 일치하지 않는 경우 계정은 더 이상 AWS 콘솔을 사용하여 DLQ 메시지를 리드라이브할 수 없게 됩니다.
> 그러나 2023년 8월 중에 AWS 콘솔에서 DLQ 리드라이브 기능을 사용한 경우 이러한 옵션 중 하나에 따라 새 권한을 채택할 수 있도록 2023년 10월 15일까지 확장됩니다.
> 자세한 내용은 the section called “영향을 받는 정책 식별” 단원을 참조하십시오.

다음은 각 DLQ 리드라이브 옵션에 대한 대기열 권한 예제입니다. 서버 측 암호화된(SSE) 대기열을 사용하는 경우 해당 AWS KMS 키 권한이 필요합니다.

작업 기반

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:StartMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:CancelMessageMoveTask"
            ],
            "Resource": "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>"
        },
```

```
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>"
        }
    ]
}
```

관리형 정책

다음 관리형 정책에는 업데이트된 필수 권한이 포함됩니다.

- AmazonSQSFullAccess - 시작, 취소, 나열과 같은 DLQ(Dead Letter Queue) 리드라이브 작업을 포함합니다.
- AmazonSQSReadOnlyAccess - 읽기 전용 액세스를 제공하며, DLQ(Dead Letter Queue) 리드라이브 작업 나열을 포함합니다.

sqs* 와일드카드를 사용하는 권한 정책

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "*"
    }
  ]
}
```

영향을 받는 정책 식별

고객 관리형 정책(CMPs)을 사용하는 경우 AWS CloudTrail 및 IAM을 사용하여 대기열 권한 업데이트의 영향을 받는 정책을 식별할 수 있습니다.

> **Note**
>
> `AmazonSQSFullAccess` 및 `AmazonSQSReadOnlyAccess`를 사용하는 경우 추가 조치가 필요하지 않습니다.

1. AWS CloudTrail 콘솔에 로그인합니다.
2. 이벤트 기록 페이지의 속성 조회에서 드롭다운 메뉴를 사용하여 이벤트 이름을 선택합니다. 그런 다음 `CreateMoveTask`를 검색합니다.
3. 세부 정보 페이지를 열려면 이벤트를 선택합니다. 이벤트 레코드 섹션에서 `userIdentity` ARN의 `UserName` 또는 `RoleName`을 검색합니다.
4. IAM 콘솔에 로그인합니다.
   - 사용자의 경우 사용자를 선택합니다. 이전 단계에서 식별된 `UserName` 사용자를 선택합니다.
   - 역할에서 역할을 선택합니다. 이전 단계에서 식별된 `RoleName` 사용자를 검색합니다.
5. 세부 정보 페이지의 권한 섹션에서 `Action`의 접두사가 `sqs:`인 정책을 검토하거나 `Resource`에서 정의된 Amazon SQS 대기열이 있는 정책을 검토합니다.

Amazon CloudWatch를 사용하여 배달 못한 편지 대기열에 대한 경보 생성

`ApproximateNumberOfMessagesVisible` 지표를 사용하여 배달 못한 편지 대기열의 메시지를 모니터링하도록 CloudWatch 경보를 설정합니다. 자세한 지침은 Amazon SQS 지표를 모니터링하여 CloudWatch 경보 생성 섹션을 참조하세요. 메시지가 배달 못한 편지 대기열로 이동되었음을 나타내는 경보가 트리거되면 대기열을 폴링하여 검토하고 검색할 수 있습니다.

Amazon SQS의 메시지 메타데이터

메시지 속성을 사용하여 애플리케이션의 Amazon SQS 메시지에 사용자 지정 메타데이터를 추가합니다. 메시지 시스템 속성을 사용하여 AWS 서비스와 같은 다른와 통합하기 위한 메타데이터를 저장합니다 AWS X-Ray.

Amazon SQS 메시지 속성

Amazon SQS를 사용하면 메시지 속성을 사용하여 메시지에 구조화된 메타데이터(예: 타임스탬프, 지리 공간 데이터, 서명 및 식별자)를 포함할 수 있습니다. 각 메시지에는 최대 10개의 속성이 있을 수 있습니다. 메시지 속성은 선택 사항이며 메시지 본문과 분리됩니다(단, 메시지 본문과 함께 전송됨). 소비자는 메시지 본문을 먼저 처리하지 않고도 메시지 속성을 사용하여 특정 방식으로 메시지를 처리할 수 있습니다. Amazon SQS 콘솔을 사용하여 속성이 있는 메시지를 전송하는 방법에 대한 자세한 내용은 Amazon SQS를 사용하여 속성을 포함하는 메시지 전송 섹션을 참조하세요.

> **Note**
>
> 메시지 속성을 메시지 시스템 속성과 혼동하지 마세요. 메시지 속성을 사용하여 애플리케이션의 Amazon SQS 메시지에 사용자 지정 메타데이터를 연결할 수 있지만 메시지 시스템 속성을 사용하여와 같은 다른 AWS 서비스에 대한 메타데이터를 저장할 수 있습니다 AWS X-Ray.

주제

- 메시지 속성 구성 요소
- 메시지 속성 데이터 형식
- 메시지 속성의 MD5 메시지 다이제스트 계산

메시지 속성 구성 요소

> **Important**
>
> 메시지 속성의 모든 구성 요소는 1MiB 메시지 크기 제한에 포함됩니다.
> `Name`, `Type`, `Value`, 메시지 본문은 비어있거나 null이면 안됩니다.

각 메시지 속성은 다음 구성 요소로 구성됩니다.

- 이름 - 메시지 속성 이름에 사용할 수 있는 문자는 A-Z, a-z, 0-9, 밑줄(_), 하이픈(-), 마침표(.)입니다. 다음과 같은 제한 사항이 있습니다.
  - 최대 256자일 수 있음
  - `AWS.` 또는 `Amazon.`(또는 모든 대소문자 변형)으로 시작할 수 없음
  - 대소문자 구분
  - 메시지의 모든 속성 이름 중에서 고유해야 함
  - 마침표로 시작하거나 끝나지 않아야 함
  - 시퀀스에 마침표가 없어야 함
- 형식 - 메시지 속성 데이터 형식입니다. 지원되는 형식은 `String`, `Number`, `Binary`입니다. 또한 데이터 형식에 대한 사용자 지정 정보를 추가할 수 있습니다. 데이터 유형에는 메시지 본문과 동일한 제한 사항이 있습니다(자세한 내용은 Amazon Simple Queue Service API 참조의 `SendMessage` 참조). 또한 다음과 같은 제한 사항이 적용됩니다.
  - 최대 256자일 수 있음
  - 대소문자 구분
- 값 - 메시지 속성 값입니다. `String` 데이터 형식의 경우 속성 값은 메시지 본문과 동일한 제한 사항이 있습니다.

메시지 속성 데이터 형식

메시지 속성 데이터 형식은 해당 메시지 속성 값의 처리 방법을 Amazon SQS에 지시합니다. 예를 들어 형식이 `Number`인 경우 Amazon SQS에서 숫자 값을 확인합니다.

Amazon SQS는 `String`, `Number`, `Binary` 논리 데이터 형식과 *`.custom-data-type`* 형식의 선택적 사용자 지정 데이터 형식 레이블을 지원합니다.

- 문자열 - `String` 속성은 모든 유효한 XML 문자를 사용하여 유니코드 텍스트를 저장할 수 있습니다.

- 숫자 - `Number` 속성은 양수 또는 음수 값을 저장할 수 있습니다. 숫자는 최대 38자릿수의 정밀도를 갖출 수 있으며 10^-128에서 10^+126 사이일 수 있습니다.

> **Note**
>
> Amazon SQS는 앞과 끝의 0을 제거합니다.

- 이진 - 이진 속성은 모든 이진 데이터를 저장할 수 있습니다(예: 압축된 데이터, 암호화된 데이터 또는 이미지).
- 사용자 지정 - 사용자 지정 데이터 형식을 생성하려면 사용자 지정 형식 레이블을 데이터 형식에 추가합니다. 예:
  - `Number.byte`, `Number.short`, `Number.int`, `Number.float`는 숫자 형식을 구별하는 데 도움이 될 수 있습니다.
  - `Binary.gif`와 `Binary.png`는 파일 형식을 구별하는 데 도움이 될 수 있습니다.

> **Note**
>
> Amazon SQS는 추가된 데이터를 해석, 확인 또는 사용하지 않습니다.
> 사용자 지정 형식 레이블은 메시지 본문과 동일한 제한 사항이 있습니다.

메시지 속성의 MD5 메시지 다이제스트 계산

를 사용하는 경우이 섹션을 건너뛸 AWS SDK for Java수 있습니다. Java용 SDK의 `MessageMD5ChecksumHandler` 클래스는 Amazon SQS 메시지 속성에서 MD5 메시지 다이제스트를 지원합니다.

쿼리 API 또는 Amazon SQS 메시지 속성에 대해 MD5 메시지 다이제스트를 지원하지 않는 AWS SDKs 중 하나를 사용하는 경우 다음 지침을 사용하여 MD5 메시지 다이제스트 계산을 수행해야 합니다.

> **Note**
>
> MD5 메시지 다이제스트 계산에는 항상 사용자 지정 데이터 유형 접미사를 포함하세요.

개요

다음은 MD5 메시지 다이제스트 계산 알로리즘의 개요입니다.

1. 모든 메시지 속성을 이름에 따라 오름차순으로 정렬합니다.
2. 각 속성의 개별 부분(`Name`, `Type`, `Value`)을 버퍼로 인코딩합니다.
3. 전체 버퍼의 메시지 다이제스트를 계산합니다.

다음 다이어그램은 단일 메시지 속성에서 MD5 메시지의 인코딩을 나타냅니다.

단일 Amazon SQS 메시지 속성을 인코딩하려면

1. 이름(이름의 길이(4바이트) 및 UTF-8바이트)을 인코딩합니다.
2. 데이터 형식(데이터 형식의 길이(4바이트) 및 UTF-8바이트)을 인코딩합니다.
3. 값(1바이트)의 전송 형식(`String` 또는 `Binary`)을 인코딩합니다.

> **Note**
>
> `String` 및 `Number` 논리 데이터 형식은 `String` 전송 형식을 사용합니다.
> `Binary` 논리 데이터 형식은 `Binary` 전송 형식을 사용합니다.

   a. `String` 전송 형식은 1을 인코딩합니다.

   b. `Binary` 전송 형식은 2를 인코딩합니다.

4. 속성 값을 인코딩합니다.

   a. `String` 전송 형식은 속성 값(값의 길이(4바이트) 및 UTF-8바이트)을 인코딩합니다.

   b. `Binary` 전송 형식은 속성 값(값의 길이(4바이트) 및 원시 바이트)을 인코딩합니다.

Amazon SQS 메시지 시스템 속성

메시지 속성을 사용하여 사용자 지정 메타데이터를 응용 프로그램에 대한 Amazon SQS 메시지에 연결할 수 있는 반면, 메시지 시스템 속성을 사용하여 AWS X-Ray같은 다른 AWS 서비스에 대한 메타데이터를 지정할 수 있습니다. 자세한 내용은 Amazon Simple Queue Service API 참조의 `SendMessage` 및 `SendMessageBatch` API 작업의 `MessageSystemAttribute` 요청 파라미터, `ReceiveMessage` API 작업의 `AWSTraceHeader` 속성, `MessageSystemAttributeValue` 데이터 유형을 참조하세요.

메시지 시스템 속성은 다음을 제외하고 메시지 시스템과 똑같이 구조화됩니다.

- 현재 유일하게 지원되는 메시지 시스템 속성은 `AWSTraceHeader`입니다. 유형은 이어야 `String` 하며 값은 올바른 형식의 AWS X-Ray 트레이스 헤더 문자열이어야 합니다.
- 메시지 시스템 속서으이 크기는 총 메시지 크기에 합산되지 않습니다.

Amazon SQS 메시지를 처리하는 데 필요한 리소스

Amazon SQS는 대기열에서 지연, 표시 및 표시되지 않는 메시지의 대략적인 수에 대한 추정치를 제공하여 처리에 필요한 리소스를 평가하는 데 도움이 됩니다. 표시에 대한 자세한 정보는 Amazon SQS 제한 시간 초과 단원을 참조하십시오.

> **Note**
>
> 일부 지표의 경우, Amazon SQS의 분산 아키텍처로 인해 그 결과는 거의 비슷합니다. 대부분의 경우 그 수는 대기열에 있는 메시지의 실제 개수에 근접해야 합니다.

다음 표에는 `GetQueueAttributes` 작업에 사용할 속성 이름이 나열되어 있습니다:

Task	속성 이름
대기열로부터 검색이 가능한 메시지의 대략적인 수를 확인합니다.	`ApproximateNumberOfMessagesVisible`
대기열 중 지연되고 즉시 읽기가 불가능한 메시지의 대략적인 수를 확인합니다. 이러한 경우는 대기열이 지연 대기열로 구성되거나 메시지가 지연 파라미터와 함께 전송되었을 때 발생할 수 있습니다.	`ApproximateNumberOfMessagesDelayed`
이동 중인 메시지의 대략적인 수를 확인합니다. 클라이언트에게 전송되었으나 아직 삭제되지 않았거나 가시성 창 말단에 이르지 않은 경우 이동 중인 것으로 간주됩니다.	`ApproximateNumberOfMessagesNotVisible`

Amazon SQS 목록 대기열 페이지 매김

`listQueues` 및 `listDeadLetterQueues` API 메서드는 선택적 페이지 매김 제어를 지원합니다. 기본적으로 이러한 API 메서드는 응답 메시지에 최대 1000개의 대기열을 반환합니다. 각 응답에서 반환되는 결과 수가 적도록 `MaxResults` 파라미터를 설정할 수 있습니다.

`listQueues` 또는 `listDeadLetterQueues` 요청에서 `MaxResults` 파라미터를 설정하여 응답에서 반환할 최대 결과 수를 지정합니다. `MaxResults`를 설정하지 않으면 응답에는 최대 1,000개의 결과가 포함되며 응답의 `NextToken` 값은 null입니다.

`MaxResults`를 설정한 경우, 표시할 추가 결과가 있으면 응답에 `NextToken`에 대한 값이 포함됩니다. `listQueues`에 대한 다음 요청에서 `NextToken`을 파라미터로 사용하여 다음 결과 페이지를 수신합니다. 표시할 추가 결과가 없으면 응답의 `NextToken` 값은 null입니다.

Amazon SQS 비용 할당 태그

비용 할당에 대한 Amazon SQS 대기열을 구성하고 식별하려면 대기열의 목적, 소유자 또는 환경을 식별하는 메타데이터 태그를 추가할 수 있습니다. 이 기능은 대기열이 많을 때 특히 유용합니다. Amazon SQS 콘솔을 사용하여 태그를 구성하려면 the section called “대기열에 대해 태그 구성” 섹션을 참조하세요.

비용 할당 태그를 사용하여 자체 비용 구조를 반영하도록 AWS 청구서를 구성할 수 있습니다. 이렇게 하려면 가입하여 태그 키와 값을 포함하는 AWS 계정 청구서를 받습니다. 자세한 내용은 AWS Billing 사용 설명서에서 월간 비용 할당 보고서 설정을 참조하세요.

각 태그는 사용자가 정의하는 키-값 페어로 구성됩니다. 예를 들어, 다음과 같이 대기열에 태그를 지정하면 프로덕션 및 테스트 대기열을 쉽게 식별할 수 있습니다.

대기열	키	값
`MyQueueA`	`QueueType`	`Production`
`MyQueueB`	`QueueType`	`Testing`

> **Note**
>
> 대기열 태그를 사용할 경우 다음 지침을 유념해야 합니다.
>
> - 대기열에 50개 이상의 태그를 추가하지 않는 것이 좋습니다. 태그 지정 시 UTF-8 형식의 유니코드 문자를 사용할 수 있습니다.
> - 태그에는 의미론적 의미가 없습니다. Amazon SQS는 태그를 문자열로 해석합니다.
> - 태그는 대/소문자를 구분합니다.
> - 키가 기존 태그와 동일한 새 태그는 기존 태그를 덮어씁니다.
> - 태그 지정 작업은 당 30TPS로 제한됩니다 AWS 계정. 애플리케이션에 더 많은 처리량이 필요한 경우 요청을 제출하십시오.
>
> 태그 제한 사항 전체 목록은 Amazon SQS 표준 대기열 할당량 단원을 참조하십시오.

Amazon SQS 짧은 폴링 및 긴 폴링

Amazon SQS는 대기열에서 메시지를 수신하기 위한 짧은 폴링 및 긴 폴링 옵션을 제공합니다. 이러한 두 가지 폴링 옵션 중에서 선택할 때는 응답성과 비용 효율성에 대한 애플리케이션의 요구 사항을 고려하세요.

- 짧은 폴링(기본값) - `ReceiveMessage` 요청은 서버 하위 집합(가중 무작위 배포 기반)을 쿼리하여 사용 가능한 메시지를 찾고 메시지가 발견되지 않더라도 즉각적인 응답을 보냅니다.

- 긴 폴링 - `ReceiveMessage`는 모든 서버에서 메시지를 쿼리하여 메시지를 하나 이상 사용할 수 있게 되면 지정된 최대값까지 응답을 보냅니다. 빈 응답은 폴링 대기 시간이 만료된 경우에만 전송됩니다. 이 옵션은 빈 응답 수를 줄이고 잠재적으로 비용을 낮출 수 있습니다.

다음 섹션에서는 짧은 폴링과 긴 폴링의 세부 사항을 설명합니다.

짧은 폴링을 사용한 메시지 사용

짧은 폴링을 사용하여 대기열(FIFO 또는 표준)에서 메시지를 사용하면, Amazon SQS는 가중치 기반 무작위 배포를 기반으로 이 서버의 하위 세트를 샘플링하고 해당 서버에서만 메시지를 반환합니다. 따라서 특정 `ReceiveMessage` 요청은 모든 메시지를 반환하지 않을 수도 있습니다. 하지만 대기열의 메시지 수가 1,000개 미만인 경우 이후 요청은 메시지를 반환합니다. 대기열에서 계속 사용할 경우 Amazon SQS가 서버를 모두 샘플링하며, 사용자는 메시지를 모두 받게 됩니다.

다음 다이어그램은 시스템 구성 요소 중 하나가 수신 요청을 한 후 표준 대기열에서 반환된 메시지의 짧은 폴링 동작을 보여줍니다. Amazon SQS는 여러 서버(회색)를 샘플링하고 이러한 서버에서 메시지 A, C, D, B를 반환합니다. 메시지 E는 이 요청에서 반환되지 않지만 이후 요청에서 반환됩니다.

긴 폴링을 사용한 메시지 사용

`ReceiveMessage` API 작업에 대한 대기 시간이 0보다 큰 경우 긴 폴링이 유효합니다. 긴 폴링 대기 시간의 최대값은 20초입니다. 긴 폴링은 빈 응답 수를 줄이고(`ReceiveMessage` 요청에 사용 가능한 메시지가 없는 경우) 거짓의 빈 응답을 제거하여(메시지를 사용할 수 있지만 응답에 포함되지 않은 경우) Amazon SQS 사용 비용을 줄여 줍니다. Amazon SQS 콘솔을 사용하여 새 대기열 또는 기존 대기열에

긴 폴링을 활성화하는 방법에 대한 자세한 내용은 Amazon SQS 콘솔을 사용하여 대기열 파라미터 구성 섹션을 참조하세요. 모범 사례는 Amazon SQS에서 긴 폴링 설정 섹션을 참조하세요.

긴 폴링은 다음과 같은 이점이 있습니다.

- 응답을 전송하기 전에 대기열에서 메시지를 사용할 수 있을 때까지 Amazon SQS를 대기시켜 빈 응답을 줄입니다. 연결 시간이 초과되지 않은 경우, `ReceiveMessage` 요청에 대한 응답은 하나 이상의 사용 가능한 메시지부터 `ReceiveMessage` 작업에 지정된 최대 메시지 개수까지 포함합니다. 드문 경우지만, 대기열에 여전히 메시지가 있는 경우에도 빈 응답을 받을 수 있습니다. 특히 `ReceiveMessageWaitTimeSeconds` 파라미터의 값을 낮게 지정한 경우 더욱 그렇습니다.
- Amazon SQS 서버의 하위 집합이 아닌 모든 서버를 쿼리하여 잘못된 빈 응답을 줄이세요.
- 사용 가능한 즉시 메시지를 반환합니다.

대기열이 비어 있는지 확인하는 방법에 대한 자세한 내용은 Amazon SQS 대기열이 비어 있는지 확인 섹션을 참조하세요.

긴 폴링과 짧은 폴링의 차이점

짧은 폴링은 `WaitTimeSeconds` 요청의 `ReceiveMessage` 파라미터가 다음 두 가지 방식 중 하나에서 `0`으로 설정될 때 나타납니다.

- `ReceiveMessage` 호출이 `WaitTimeSeconds`를 `0`으로 설정합니다.
- `ReceiveMessage` 호출이 `WaitTimeSeconds`를 설정하지 않지만 대기열 속성 `ReceiveMessageWaitTimeSeconds`이 `0`으로 설정되어 있습니다.

Amazon SQS 제한 시간 초과

Amazon SQS 대기열에서 메시지를 받으면 대기열에 남아 있지만 다른 소비자에게 일시적으로 표시되지 않습니다. 이러한 가시성이 제한 시간 초과에 의해 제어되므로 작업 중에 다른 소비자가 동일한 메시지를 처리할 수 없습니다. Amazon SQS는 처리 후 메시지를 삭제하는 두 가지 옵션을 제공합니다.

- 수동 삭제 - `DeleteMessage` 작업을 사용하여 메시지를 명시적으로 삭제합니다.
- 자동 삭제 - 특정 AWS SDKs에서 지원되며 처리가 성공하면 메시지가 자동으로 삭제되어 워크플로가 간소화됩니다.

가시성 제한 시간 사용 사례

장기 실행 작업 관리 - 가시성 제한 시간을 사용하여 처리 시간 연장이 필요한 작업을 처리합니다. 처리 시간을 늘려야 하는 메시지에 대해 적절한 제한 시간 초과를 설정합니다. 이렇게 하면 다른 소비자가 처리 중에 동일한 메시지를 선택하지 않아 중복 작업을 방지하고 시스템 효율성을 유지할 수 있습니다.

재시도 메커니즘 구현 - 초기 제한 시간 내에 완료되지 않는 작업에 대해 프로그래밍 방식으로 가시성 제한 시간을 연장합니다. 초기 가시성 제한 시간 내에 작업이 완료되지 않으면 프로그래밍 방식으로 제한 시간을 연장할 수 있습니다. 이렇게 하면 시스템에서 다른 소비자에게 표시되지 않고 메시지 처리를 다시 시도할 수 있으므로 내결함성과 신뢰성이 향상됩니다. DLQs(Dead-Letter Queue)와 결합하여 지속적인 장애를 관리합니다.

분산 시스템 조정 - SQS 가시성 제한 시간을 사용하여 분산 시스템 전반에서 작업을 조정합니다. 다양한 구성 요소에 대한 예상 처리 시간에 맞게 가시성 제한 시간을 설정합니다. 이를 통해 일관성을 유지하고 복잡한 분산 아키텍처에서 레이스 조건을 방지할 수 있습니다.

리소스 사용률 최적화 - SQS 가시성 제한 시간을 조정하여 애플리케이션의 리소스 사용률을 최적화합니다. 적절한 제한 시간을 설정하면 리소스를 불필요하게 연결하지 않고도 메시지가 효율적으로 처리되도록 할 수 있습니다. 이로 인해 전반적인 시스템 성능과 비용 효율성이 향상됩니다.

표시 제한 시간 설정 및 조정

제한 시간 초과는 메시지가 전달되는 즉시 시작됩니다. 이 기간 동안 메시지를 처리하고 삭제해야 합니다. 제한 시간이 만료되기 전에 삭제하지 않으면 메시지가 대기열에 다시 표시되며 다른 소비자가 검색할 수 있습니다. 대기열의 기본 제한 시간은 30초이지만 애플리케이션에서 메시지를 처리하고 삭제해야 하는 시간에 맞게 조정할 수 있습니다. 대기열의 전체 설정을 변경하지 않고도 개별 메시지에 대해 특정 제한 시간 초과를 설정할 수도 있습니다. `ChangeMessageVisibility` 작업을 사용하여 필요에 따라 제한 시간을 프로그래밍 방식으로 연장하거나 단축합니다.

비행 중 메시지 및 할당량

Amazon SQS에서 진행 중인 메시지는 소비자가 수신했지만 아직 삭제되지 않은 메시지입니다. 표준 대기열의 경우 대기열 트래픽 및 메시지 백로그에 따라 약 120,000개의 진행 중인 메시지로 제한됩니다. 이 제한에 도달하면 Amazon SQS는 `OverLimit` 오류를 반환하여 일부 진행 중인 메시지가 삭제될 때까지 추가 메시지를 수신할 수 없음을 나타냅니다. FIFO 대기열의 경우 제한은 활성 메시지 그룹에 따라 다릅니다.

- 짧은 폴링 사용 시 - 짧은 폴링을 사용하는 동안이 제한에 도달하면 Amazon SQS는 `OverLimit` 오류를 반환하여 일부 진행 중인 메시지가 삭제될 때까지 추가 메시지를 수신할 수 없음을 나타냅니다.
- 긴 폴링 사용 시 - 긴 폴링을 사용하는 경우 진행 중인 메시지 제한에 도달하면 Amazon SQS가 오류를 반환하지 않습니다. 대신 처리 중 메시지 수가 한도 아래로 떨어질 때까지 새 메시지를 반환하지 않습니다.

진행 중인 메시지를 효과적으로 관리하려면:

1. 프롬프트 삭제 - 처리 후 메시지를 삭제(수동 또는 자동)하여 진행 중인 수를 줄입니다.
2. CloudWatch로 모니터링 - 한도에 도달하지 않도록 높은 진행 중 횟수에 대한 경보를 설정합니다.
3. 로드 분산 - 대량의 메시지를 처리하는 경우 추가 대기열 또는 소비자를 사용하여 로드의 균형을 맞추고 병목 현상을 방지합니다.
4. 할당량 증가 요청 - 더 높은 한도가 필요한 경우 AWS Support에 요청을 제출합니다.

표준 및 FIFO 대기열의 표시 제한 시간 이해

표준 대기열과 FIFO(First-In-First-Out) 대기열 모두에서 제한 시간 초과는 여러 소비자가 동일한 메시지를 동시에 처리하지 못하도록 하는 데 도움이 됩니다. 그러나 Amazon SQS의 at-least-once 전송 모델로 인해 가시성 제한 시간 동안 메시지가 2회 이상 전송되지 않을 것이라는 절대적인 보장은 없습니다.

- 표준 대기열 - 표준 대기열의 제한 시간 초과로 인해 여러 소비자가 동시에 동일한 메시지를 처리할 수 없습니다. 그러나 at-least-once 전송 모델로 인해 Amazon SQS는 가시성 제한 시간 내에 메시지가 2회 이상 전송되지 않을 것이라고 보장하지 않습니다.
- FIFO 대기열 - FIFO 대기열의 경우 메시지 그룹 ID가 동일한 메시지는 엄격한 순서로 처리됩니다. 메시지 그룹 ID가 인 메시지가 진행 중이면 해당 그룹의 후속 메시지는 진행 중 메시지가 삭제되거나 제한 시간 초과가 만료될 때까지 사용할 수 없습니다. 그러나 그룹을 무기한 "잠그지" 않습니다. 각 메시지는 순서대로 처리되며, 각 메시지가 삭제되거나 다시 표시될 때만 해당 그룹의 다음 메시지를

소비자가 사용할 수 있습니다. 이 접근 방식은 그룹의 메시지 전송을 불필요하게 잠그지 않고도 그룹 내에서 순서가 지정된 처리를 보장합니다.

실패 처리

애플리케이션 오류, 충돌 또는 연결 문제로 인해 표시 제한 시간이 만료되기 전에 메시지를 처리하고 삭제하지 않으면 메시지가 대기열에 다시 표시됩니다. 그런 다음 동일한 소비자 또는 다른 소비자가 다른 처리 시도를 위해 검색할 수 있습니다. 이렇게 하면 초기 처리에 실패하더라도 메시지가 손실되지 않습니다. 그러나 표시 제한 시간을 너무 높게 설정하면 처리되지 않은 메시지의 재출현이 지연되어 재시도 속도가 느려질 수 있습니다. 적시에 메시지를 처리할 수 있도록 예상 처리 시간을 기준으로 적절한 가시성 제한 시간을 설정하는 것이 중요합니다.

표시 제한 시간 변경 및 종료

`ChangeMessageVisibility` 작업을 사용하여 표시 제한 시간을 변경하거나 종료할 수 있습니다.

- 제한 시간 변경 -를 사용하여 가시성 제한 시간을 동적으로 조정합니다`ChangeMessageVisibility`. 이렇게 하면 처리 요구 사항에 맞게 제한 시간을 연장하거나 줄일 수 있습니다.
- 제한 시간 종료 - 수신된 메시지를 처리하지 않기로 결정한 경우 `ChangeMessageVisibility` 작업을 통해를 `0`초`VisibilityTimeout`로 설정하여 표시 제한 시간을 종료합니다. 이렇게 하면 즉시 다른 소비자가 메시지를 처리할 수 있게 됩니다.

모범 사례

다음 모범 사례를 사용하여 제한 시간 설정, 조정 및 연장, Dead Letter Queue(DLQ)를 사용하여 처리되지 않은 메시지를 처리하는 등 표시 제한 시간을 관리합니다.

- 제한 시간 설정 및 조정. 먼저 애플리케이션에서 일반적으로 메시지를 처리하고 삭제하는 데 필요한 최대 시간과 일치하도록 표시 제한 시간을 설정합니다. 정확한 처리 시간이 확실하지 않은 경우 더 짧은 제한 시간(예: 2분)으로 시작하고 필요에 따라 연장합니다. 하트비트 메커니즘을 구현하여 가시성 제한 시간을 주기적으로 연장하여 처리가 완료될 때까지 메시지가 보이지 않도록 합니다. 이렇게 하면 처리되지 않은 메시지의 재처리 지연이 최소화되고 조기 가시성이 방지됩니다.
- 제한 시간 연장 및 12시간 제한 처리. 처리 시간이 다르거나 처음에 설정된 제한 시간을 초과할 수 있는 경우 `ChangeMessageVisibility` 작업을 사용하여 메시지를 처리하는 동안 표시 제한 시간을 연장합니다. 표시 제한 시간은 메시지가 처음 수신된 시점으로부터 최대 12시간입니다. 제한 시간을

연장해도 이 12시간 제한은 재설정되지 않습니다. 처리에이 제한보다 많은 시간이 필요한 경우 작업을 AWS Step Functions 사용하거나 더 작은 단계로 나누는 것이 좋습니다.

- 처리되지 않은 메시지 처리. 여러 처리 시도에 실패한 메시지를 관리하려면 DLQ(Dead-Letter Queue)를 구성합니다. 이렇게 하면 추가 분석 또는 처리를 위해 여러 번의 재시도 후 처리할 수 없는 메시지를 별도로 캡처하여 기본 대기열에서 반복적으로 순환하지 않도록 할 수 있습니다.

Amazon SQS 공정 대기열

Amazon SQS 공정 대기열은 고객, 클라이언트 애플리케이션 또는 메시지 유형과 같은 여러 논리적 엔터티의 메시지가 포함된 다중 테넌트 대기열에서 노이즈가 많은 이웃에 미치는 영향을 자동으로 완화합니다. 이러한 공유 대기열 환경에서 중요한 성능 지표 중 하나는 유지 시간입니다.이 지표는 메시지의 도착부터 처리까지 대기열에서 소요된 총 시간을 측정합니다. 한 테넌트가 시스템에서 처리할 수 있는 것보다 더 많은 메시지를 게시하여 대기열에 백로그를 생성하면 공정 대기열은 다른 테넌트의 체류 시간에 미치는 영향을 최소화합니다.

정상 상태

다음 다이어그램은 네 개의 개별 테넌트(레이블 A, B, C, D)의 메시지가 포함된 다중 테넌트 대기열을 보여줍니다. 대기열은 안정적인 상태로 작동하며, 소비자가 대기열에 메시지가 나타나는 즉시 메시지를 수신할 때 메시지 백로그가 없습니다. 모든 테넌트의 체류 시간이 짧습니다. 이 안정 상태에서 모든 소비자 용량이 완전히 활용되는 것은 아닙니다.

시끄러운 이웃 영향

노이즈가 많은 이웃 영향은 다중 테넌트 대기열의 한 테넌트가 백로그를 생성하여 다른 모든 테넌트의 메시지 유지 시간을 늘릴 때 발생합니다. 테넌트는 다른 테넌트보다 더 많은 양의 메시지를 보내거나 소비자가 특정 테넌트의 메시지를 처리하는 데 더 오래 걸리는 경우 시끄러운 이웃이 될 수 있습니다.

이 다이어그램은 테넌트 A에서 증가된 트래픽이 대기열에 백로그를 생성하는 방법을 보여줍니다. 소비자는 테넌트 A의 메시지만 처리하는 데 바쁜 반면, 다른 테넌트의 메시지는 백로그에서 대기하여 모든 테넌트의 유지 시간이 늘어납니다.

공정 대기열을 사용한 완화

Amazon SQS는 처리 중에 테넌트 간의 메시지 분포를 모니터링하여 시끄러운 이웃을 감지합니다("사용 중" 상태). 테넌트가 다른 테넌트에 비해 전송 중 메시지가 불균형적으로 많은 경우 Amazon SQS는 해당 테넌트를 노이즈가 많은 이웃으로 식별하고 다른 테넌트에 대한 메시지 전송의 우선순위를 지정합니다. 이 접근 방식은 다른 테넌트에 대한 유지 시간 영향을 줄입니다.

이 다이어그램은 Amazon SQS 공정 대기열이 시끄러운 이웃 문제를 해결하는 방법을 보여줍니다. 한 테넌트(테넌트 A)에 노이즈가 발생하면 Amazon SQS는 다른 테넌트(B, C, D)의 메시지 반환을 우선시합니다. 이 우선 순위는 조용한 테넌트 테넌트 B, C 및 D의 유지 시간을 낮게 유지하는 데 도움이 되며, 테넌트 A 메시지의 유지 시간은 다른 테넌트에 영향을 주지 않고 대기열 백로그가 사용될 때까지 상승됩니다.

> **Note**
>
> Amazon SQS는 테넌트당 소비율을 제한하지 않습니다. 소비자 용량이 있고 대기열에 반환할 다른 메시지가 없는 경우 소비자가 시끄러운 이웃 테넌트로부터 메시지를 수신할 수 있습니다.

Amazon SQS 표준 대기열과 마찬가지로 공정 대기열은 사실상 무제한의 처리량을 허용하며 대기열에 있을 수 있는 테넌트 수에는 제한이 없습니다.

FIFO 대기열과의 차이점

FIFO 대기열은 각 테넌트의 진행 중인 메시지 수를 제한하여 엄격한 순서를 유지합니다. 이렇게 하면 노이즈가 많은 이웃이 방지되지만 각 테넌트의 처리량이 제한됩니다. 공정 대기열은 높은 처리량, 낮은 체류 시간, 공정한 리소스 할당이 우선순위인 다중 테넌트 시나리오를 위해 설계되었습니다. 공정 대기열을 사용하면 여러 소비자가 동일한 테넌트의 메시지를 동시에 처리하는 동시에 모든 테넌트가 일관된 유지 시간을 유지할 수 있습니다.

공정 대기열 사용

메시지 생산자는 발신 메시지`MessageGroupId`에를 설정하여 테넌트 식별자를 추가할 수 있습니다.

```
// Send message with tenant identifier
SendMessageRequest request = new SendMessageRequest()
    .withQueueUrl(queueUrl)
    .withMessageBody(messageBody)
    .withMessageGroupId("tenant-123"); // Tenant identifier
sqs.sendMessage(request);
```

공정성 기능은 MessageGroupId 속성이 있는 메시지의 모든 Amazon SQS 표준 대기열에 자동으로 적용됩니다. 소비자 코드를 변경할 필요가 없으며 API 지연 시간에 영향을 주지 않으며 처리량 제한이 없습니다.

공정 대기열 CloudWatch 지표

Amazon SQS는 노이즈가 많은 이웃 영향의 완화를 모니터링하는 데 도움이 되는 추가 CloudWatch 지표를 제공합니다. 예를 들어 `Approximate..InQuietGroups` 지표를 표준 대기열 수준 지표와 비교할 수 있습니다. 특정 테넌트에 대한 트래픽이 급증하는 동안 일반 대기열 수준 지표에서 백로그 증가 또는 메시지 수명 증가를 확인할 수 있습니다. 그러나 저소음 그룹을 개별적으로 살펴보면 대부분의 비저소음 메시지 그룹 또는 테넌트가 영향을 받지 않음을 식별할 수 있습니다.

아래에서 노이즈가 없는 테넌트(ApproximateNumberOfMessagesVisibleInQuietGroups)에 대한 백로그가 낮게 유지되는 동안 노이즈가 있는 테넌트로 인해 표준 대기열 백로그 지표(ApproximateNumberOfMessagesVisible)가 증가하는 예를 찾을 수 있습니다.

Amazon SQS CloudWatch 지표 및 해당 설명의 전체 목록은 Amazon SQS에 대한 CloudWatch 지표를 참조하세요.

Amazon SQS 지연 대기열

지연 대기열을 사용하면 새 메시지를 소비자에게 전송하는 것을 특정 시간(초) 동안 연기할 수 있습니다(예: 소비자 애플리케이션에서 메시지를 처리하기 위해 추가 시간이 필요한 경우). 지연 대기열을 생성할 경우, 이 대기열로 전송하는 모든 메시지는 지연 기간 동안 소비자에게 표시되지 않습니다. 대기열의 기본(최소) 지연 시간은 0초입니다. 최대값은 15분입니다. 콘솔을 사용해 지연 대기열을 구성하는 방법에 대한 자세한 내용은 Amazon SQS 콘솔을 사용하여 대기열 파라미터 구성 단원을 참조하세요.

> **Note**
>
> 표준 대기열에서 대기열당 지연 설정은 소급 적용되지 않습니다. 설정을 변경할 경우, 이것은 대기열에 이미 있는 메시지의 지연에 영향을 주지 않습니다.
> FIFO 대기열에서 대기열당 지연 설정은 소급 적용됩니다. 설정을 변경할 경우, 이것은 대기열에 이미 있는 메시지의 지연에 영향을 줍니다.

지연 대기열은 두 기능 모두가 소비자가 특정 기간 동안 사용하지 못하도록 하기 때문에 제한 시간 초과와 비슷합니다. 두 속성의 차이는 지연 대기열은 처음 대기열에 추가될 때 메시지가 숨겨지지만, 제한 시간 초과는 대기열에서 사용된 후에만 메시지가 숨겨진다는 것입니다. 다음 그림은 지연 대기열과 제한 시간 초과의 관계를 잘 보여줍니다.

확장 예약 옵션

Amazon SQS 지연 대기열 및 메시지 타이머를 사용하면 향후 최대 15분 동안 메시지 전송을 예약할 수 있지만 보다 유연한 예약 기능이 필요할 수 있습니다. 이러한 경우 시간 제한 없이 수십억 개의 일회성 또는 반복 API 작업을 예약할 수 있는 EventBridge 스케줄러를 사용하는 것이 좋습니다. EventBridge 스케줄러는 고급 메시지 예약 사용 사례에 권장되는 솔루션입니다.

전체 대기열이 아닌 개별 메시지에 지연 초를 설정하려면 메시지 타이머를 사용하여 Amazon SQS가 지연 대기열 `DelaySeconds` 값 대신 메시지 타이머의 `DelaySeconds` 값을 사용하도록 허용합니다. EventBridge 스케줄러는 개별 메시지 예약도 지원합니다.

Amazon SQS 임시 대기열

임시 대기열을 사용하면 request-response와 같은 공통 메시지 패턴을 사용할 때 개발 시간 및 배포 비용을 줄일 수 있습니다. 임시 대기열 클라이언트를 사용하여 대량고속처리가 가능하고 비용 효율적인 애플리케이션 관리 임시 대기열을 생성할 수 있습니다.

클라이언트는 여러 개의 임시 대기열(특정 프로세스에 대한 요구 시 생성되는 애플리케이션 관리형 대기열)을 하나의 Amazon SQS 대기열로 자동으로 매핑합니다. 따라서 애플리케이션에서 API 호출 수를 줄이고 각 임시 대기열에 대한 트래픽이 낮을 때도 높은 처리량을 얻을 수 있습니다. 임시 대기열을 더 이상 사용하지 않을 때는 클라이언트를 사용하는 프로세스가 명확하게 종료된 경우라 하더라도 클라이언트가 임시 대기열을 자동으로 정리합니다.

임시 대기열은 다음과 같은 이점이 있습니다.

- 임시 대기열은 특정 스레드 또는 프로세스에 대해 경량 통신 채널 역할을 합니다.
- 또한 추가 비용 없이도 생성 및 삭제가 가능합니다.

- 이는 정적(정상) Amazon SQS 대기열과 API 호환이 가능합니다. 따라서 메시지를 전송 및 수신하는 기존 코드가 가상 대기열과 메시지를 주고 받을 수 있습니다.

가상 대기열

가상 대기열은 임시 대기열 클라이언트가 생성하는 로컬 데이터 구조입니다. 가상 대기열을 사용하면 Amazon SQS 대기열에 트래픽이 낮은 대상을 여러 개 결합할 수 있습니다. 모범 사례는 가상 대기열에서 동일한 메시지 그룹 ID 재사용 방지를 참조하세요.

> **Note**
>
> - 가상 대기열을 생성하면 소비자가 들어오는 메시지를 수신할 수 있는 임시 데이터 구조만 생성됩니다. 가상 대기열은 Amazon SQS에 대해 API 호출을 수행하지 않기 때문에 어떤 비용도 발생하지 않습니다.
> - 한 호스트 대기열 내의 모든 가상 대기열에는 TPS 할당량이 적용됩니다. 자세한 내용은 Amazon SQS 메시지 할당량 단원을 참조하십시오.

`AmazonSQSVirtualQueuesClient` 래퍼 클래스는 가상 대기열과 관련된 속성 지원을 추가합니다. 가상 대기열을 생성하려면 `HostQueueURL` 속성을 사용하여 `CreateQueue` API 작업을 호출해야 합니다. 이 속성은 가상 대기열을 호스팅하는 기존 대기열을 지정합니다.

가상 대기열의 URL은 다음 형식을 따릅니다.

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue#MyVirtualQueueName
```

생산자가 가상 대기열 URL에서 `SendMessage` 또는 `SendMessageBatch` API 작업을 호출하면 임시 대기열 클라이언트가 다음과 같은 작업을 수행합니다.

1. 가상 대기열 이름을 추출합니다.
2. 가상 대기열 이름을 추가 메시지 속성으로 연결합니다.
3. 호스트 대기열에 이 메시지를 전송합니다.

생산자가 메시지를 전송할 때 백그라운드 스레드는 호스트 대기열을 폴링하고, 해당되는 메시지 속성에 따라 가상 대기열에 수신한 메시지를 전송합니다.

소비자가 가상 대기열 URL에서 `ReceiveMessage` API 작업을 호출하는 동안 임시 대기열 클라이언트는 백그라운드 스레드가 가상 대기열에 메시지를 전송할 때까지 로컬 호출을 차단합니다. (이 프로세스는 버퍼링된 비동기식 클라이언트에서의 메시지 미리 가져오기와 비슷한데, 단일 API 작업이 최대 10개의 가상 대기열에 메시지를 전송하는 것이 가능합니다). 가상 대기열을 삭제하면 Amazon SQS 자체를 호출하지 않고 클라이언트측 리소스가 제거됩니다.

`AmazonSQSTemporaryQueuesClient` 클래스는 생성한 모든 대기열을 임시 대기열로 자동 전환합니다. 또한 요구 시 같은 대기열 속성을 가진 호스트 대기열을 자동 생성합니다. 이러한 대기열의 이름은 임시 대기열로 식별하는 공통된 구성 가능 접두사(기본적으로 `__RequesterClientQueues__`)를 공유합니다. 따라서 클라이언트는 대기열을 생성 및 삭제하는 기존 코드를 최적화하는 즉각적인 대체물 역할을 할 수 있습니다. 또한 클라이언트에는 대기열 간에 양방향 통신을 가능하게 하는 `AmazonSQSRequester` 및 `AmazonSQSResponder` 인터페이스가 포함되어 있습니다.

요청-응답 메시징 패턴(가상 대기열)

임시 대기열의 가장 일반적인 사용 사례는 요청-응답 메시징 패턴인데, 여기에서는 요청이 각 응답 메시지를 수신하기 위해 임시 대기열을 생성합니다. 각 응답 메시지에 대해 Amazon SQS 대기열을 생성하는 것을 방지하기 위해 임시 대기열 클라이언트는 어떤 Amazon SQS API도 호출하지 않고 여러 개의 임시 대기열을 생성 및 삭제할 수 있도록 해줍니다. 자세한 내용은 요청-응답 시스템 구현을 참조하세요.

다음 다이어그램은 이 패턴을 사용하는 공통 구성을 보여줍니다.

예제 시나리오: 로그인 요청 처리

다음 예제 시나리오는 AmazonSQSRequester 및 AmazonSQSResponder 인터페이스를 사용하여 사용자의 로그인 요청을 처리할 수 있는 방법을 보여줍니다.

클라이언트 측

```
public class LoginClient {

    // Specify the Amazon SQS queue to which to send requests.
    private final String requestQueueUrl;

    // Use the AmazonSQSRequester interface to create
    // a temporary queue for each response.
    private final AmazonSQSRequester sqsRequester =
            AmazonSQSRequesterClientBuilder.defaultClient();

    LoginClient(String requestQueueUrl) {
        this.requestQueueUrl = requestQueueUrl;
```

```
    }

    // Send a login request.
    public String login(String body) throws TimeoutException {
        SendMessageRequest request = new SendMessageRequest()
                .withMessageBody(body)
                .withQueueUrl(requestQueueUrl);

        // If no response is received, in 20 seconds,
        // trigger the TimeoutException.
        Message reply = sqsRequester.sendMessageAndGetResponse(request,
                20, TimeUnit.SECONDS);

        return reply.getBody();
    }
}
```

로그인 요청을 전송하는 과정은 다음과 같습니다.

1. 임시 대기열을 생성합니다.
2. 임시 대기열의 URL을 메시지에 속성으로써 연결합니다.
3. 이 메시지를 전송합니다.
4. 임시 대기열에서 응답을 수신합니다.
5. 임시 대기열을 삭제합니다.
6. 응답을 반환합니다.

서버 측

이 예제에서는 구성 시 대기열을 폴링하고 모든 메시지에 대해 `handleLoginRequest()` 메서드를 호출하기 위해 스레드가 생성된다고 가정합니다. 게다가 `doLogin()`는 위임된 메서드입니다.

```
public class LoginServer {

    // Specify the Amazon SQS queue to poll for login requests.
    private final String requestQueueUrl;

    // Use the AmazonSQSResponder interface to take care
    // of sending responses to the correct response destination.
    private final AmazonSQSResponder sqsResponder =
```

```
            AmazonSQSResponderClientBuilder.defaultClient();

    LoginServer(String requestQueueUrl) {
        this.requestQueueUrl = requestQueueUrl;
    }

    // Process login requests from the client.
    public void handleLoginRequest(Message message) {

        // Process the login and return a serialized result.
        String response = doLogin(message.getBody());

        // Extract the URL of the temporary queue from the message attribute
        // and send the response to the temporary queue.
        sqsResponder.sendResponseMessage(MessageContent.fromMessage(message),
                new MessageContent(response));
    }
}
```

대기열 정리

Amazon SQS가 이 가상 대기열에서 사용되는 인 메모리 리소스를 회수하는지 확인하려면 애플리케이션에서 임시 대기열 클라이언트가 더 이상 필요하지 않을 때 `shutdown()` 메서드를 호출해야 합니다. 또한 `AmazonSQSRequester` 인터페이스의 `shutdown()` 메서드를 사용할 수도 있습니다.

임시 대기열 클라이언트는 사용하지 않는 호스트 대기열을 제거할 수 있는 방법도 제공합니다. 일정 기간(기본적으로 5분) 동안 API 호출을 수신하는 각 대기열에서 클라이언트는 `TagQueue` API 작업을 사용하여 사용 중인 상태로 남아 있는 대기열에 태그 지정을 합니다.

> **Note**
>
> `ReceiveMessage` 작업을 포함하여 대기열에서 이루어진 모든 API 작업은 대기열을 비유휴 상태로 표시합니다.

백그라운드 스레드는 `ListQueues` 및 `ListTags` API 작업을 사용하여 접두사가 구성된 모든 대기열을 확인하고 최소 5분 동안 태그 지정이 되지 않은 대기열을 모두 삭제합니다. 이 과정에서 한 클라이언트가 명확하게 종료되지 않으면 다른 활성 클라이언트가 종료 이후에 정리합니다. 작업 중복을 줄이기 위해 같은 접두사를 가진 모든 클라이언트가 접두사 뒤에 명명된 내부의 공유 대기열을 통해 통신합니다.

Amazon SQS 메시지 타이머

메시지 타이머를 사용하면 메시지가 대기열에 추가될 때 메시지의 초기 보이지 않는 기간을 설정할 수 있습니다. 예를 들어 45초 타이머로 메시지를 보내면 처음 45초 동안 소비자에게 숨겨진 상태로 유지됩니다. 메시지의 기본(최소) 지연 시간은 0초입니다. 최대값은 15분입니다. 콘솔을 사용하여 타이머가 있는 메시지를 전송하는 방법에 대한 자세한 내용은 표준 대기열을 사용하여 메시지 전송 섹션을 참조하세요.

> **Note**
>
> FIFO 대기열은 개별 메시지의 타이머를 지원하지 않습니다.

개별 메시지가 아닌 전체 대기열에서 지연 기간을 설정하려면 지연 대기열을 사용합니다. 개별 메시지에 대한 타이머 설정은 Amazon SQS 지연 대기열의 `DelaySeconds` 값을 재정의합니다.

확장 예약 옵션

Amazon SQS 지연 대기열 및 메시지 타이머를 사용하면 향후 최대 15분 동안 메시지 전송을 예약할 수 있지만 보다 유연한 예약 기능이 필요할 수 있습니다. 이러한 경우 시간 제한 없이 수십억 개의 일회성 또는 반복 API 작업을 예약할 수 있는 EventBridge 스케줄러를 사용하는 것이 좋습니다. EventBridge 스케줄러는 고급 메시지 예약 사용 사례에 권장되는 솔루션입니다.

Amazon SQS 콘솔을 통해 Amazon EventBridge 파이프에 액세스

Amazon EventBridge 파이프는 소스를 대상에 연결합니다. 파이프는 지원되는 소스와 대상 간의 지점 간 통합을 위한 것으로 고급 변환 및 보강을 지원합니다. EventBridge 파이프는 Step Functions, Amazon SQS, API Gateway와 같은 AWS 서비스뿐만 아니라 Salesforce와 같은 타사 서비스형 소프트웨어(SaaS) 애플리케이션에 Amazon SQS 대기열을 연결할 수 있는 확장성이 뛰어난 방법을 제공합니다.

파이프를 설정하려면 소스를 선택하고, 선택적 필터링을 추가하고, 선택적 보강을 정의하고, 이벤트 데이터의 대상을 선택합니다.

Amazon SQS 대기열의 세부 정보 페이지에서 해당 대기열을 소스로 사용하는 파이프를 볼 수 있습니다. 다음을 수행할 수도 있습니다.

- EventBridge 콘솔을 실행하여 파이프 세부 정보를 확인합니다.
- EventBridge 콘솔을 실행하여 대기열이 소스인 새 파이프를 생성합니다.

Amazon SQS 대기열을 파이프 소스로 구성하는 방법에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge 사용 설명서를 참조하세요. 일반적인 EventBridge 파이프에 대한 자세한 내용은 EventBridge 파이프를 참조하세요.

지정된 Amazon SQS 대기열에 대한 EventBridge 파이프에 액세스하려면 다음을 수행합니다.

1. Amazon SQS 콘솔의 대기열 페이지를 엽니다.
2. 대기열을 선택합니다.
3. 대기열 세부 정보 페이지에서 EventBridge 파이프 탭을 선택합니다.

   EventBridge 파이프 탭에는 선택한 대기열을 소스로 사용하도록 현재 구성된 모든 파이프 목록이 포함되어 있습니다.

   - 파이프 이름
   - 현재 상태
   - 파이프 대상
   - 파이프를 마지막으로 수정한 시점
4. 원하는 경우 더 많은 파이프 세부 정보를 보거나 새 파이프를 만듭니다.

   - 파이프에 대한 추가 세부 정보에 액세스:

     파이프 이름을 선택합니다.

     그러면 EventBridge 콘솔의 파이프 세부 정보 페이지가 시작됩니다.
   - 새 파이프를 생성하려면:

     파이프에 Amazon SQS 대기열 연결을 선택합니다.

     그러면 Amazon SQS 대기열이 파이프 소스로 지정된 EventBridge 콘솔의 파이프 생성 페이지가 시작됩니다. 자세한 내용은 Amazon EventBridge 사용 설명서에서 EventBridge 파이프 생성을 참조하세요.

     > **Important**
     >
     > Amazon SQS 대기열의 메시지는 메시지가 해당 파이프에 대해 구성할 수 있는 필터와 일치하는지 여부에 관계없이 단일 파이프로 읽힌 다음 처리된 후 대기열에서 삭제됩니다. 동일한 대기열을 소스로 사용하도록 여러 파이프를 구성할 때는 주의하세요.

확장 클라이언트 라이브러리 및 Amazon Simple Storage Service를 사용하여 대규모 Amazon SQS 메시지 관리

Java용 Amazon SQS 확장 클라이언트 라이브러리와 Python용 Amazon SQS 확장 클라이언트 라이브러리를 사용하여 특히 256KB~2GB의 페이로드에 대해 대용량 메시지를 보냅니다. 이러한 라이브러리는 메시지 페이로드를 Amazon S3 버킷에 저장하고 Amazon SQS 대기열에 저장된 객체에 대한 참조를 전송합니다.

> **Note**
>
> Amazon SQS 확장 클라이언트 라이브러리는 표준 및 FIFO 대기열과 호환됩니다.

Java 및 Amazon S3를 사용하여 대규모 Amazon SQS 메시지 관리

Amazon S3와 함께 Java용 Amazon SQS 확장 클라이언트 라이브러리를 사용하여 특히 256KB~2GB 범위의 페이로드에 대해 대규모 Amazon SQS 메시지를 관리할 수 있습니다. 라이브러리는 메시지 페이로드를 Amazon S3 버킷에 저장하고 Amazon SQS 대기열에 저장된 객체에 대한 참조가 포함된 메시지를 보냅니다.

Java용 Amazon SQS 확장 클라이언트 라이브러리를 사용하면 다음을 수행할 수 있습니다.

- 메시지를 항상 Amazon S3에만 저장할지를 지정하거나, 메시지의 크기가 256KB를 초과할 때만 저장할지를 지정합니다.
- S3 버킷에 저장된 단일 메시지 객체를 참조하는 메시지를 전송합니다.
- Amazon S3 버킷에서 메시지 객체 검색
- Amazon S3 버킷에서 메시지 객체 삭제

사전 조건

다음 예제에서는 AWS Java SDK를 사용합니다. SDK를 설치하고 설정하려면 AWS SDK for Java 개발자 안내서의 Java용 AWS SDK 설정을 참조하세요.

예제 코드를 실행하기 전에 AWS 자격 증명을 구성합니다. 자세한 내용은 AWS SDK for Java 개발자 안내서의 개발을 위한 AWS 자격 증명 및 리전 설정을 참조하세요.

Java용 SDK 및 Java용 Amazon SQS 확장 클라이언트 라이브러리를 사용하려면 J2SE 개발 키트 8.0 이상이 필요합니다.

> **Note**
>
> Java용 Amazon SQS 확장 클라이언트 라이브러리를 사용하면 Amazon S3에 AWS SDK for Java만 사용하여 Amazon SQS 메시지를 관리할 수 있습니다. AWS CLI, Amazon SQS 콘솔, Amazon SQS HTTP API 또는 기타 AWS SDKs로는이 작업을 수행할 수 없습니다.

AWS SDK for Java 2.x 예: Amazon S3를 사용하여 대규모 Amazon SQS 메시지 관리

다음 SDK for SDK for Java 2.x 예제에서는 Java용 확장 클라이언트 라이브러리를 사용하여 대용량 메시지로 작업합니다. 생성자에서 다음 코드:

- 임의의 이름으로 Amazon S3 버킷을 생성합니다.
- 로 시작하는 SQS 대기열을 생성합니다. `MyQueue`
- 의 인스턴스에서 표준 Java SDK Amazon S3 클라이언트 래핑 `AmazonSQSExtendedClient`

`sendAnReceiveMessage` 메서드에서 예제는 Amazon S3 버킷에 저장된 임의의 메시지를 전송합니다. 이는 256KB(표준 최대 메시지 크기)를 초과하기 때문입니다. 마지막으로 메서드는 메시지를 검색하고 이에 대한 정보를 콘솔에 표시합니다.

에서 전체 예제를 볼 수 있습니다https://github.com/awsdocs/aws-doc-sdk-examples/blob/94d1b24df12deda0f4fd91433b8231fed6d18b85/javav2/example_code/sqs/src/main/java/com/example/sqs/SqsExtendedClientExample.java#L1.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */
```

```
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.BucketLifecycleConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ExpirationStatus;
import software.amazon.awssdk.services.s3.model.LifecycleExpiration;
import software.amazon.awssdk.services.s3.model.LifecycleRule;
import software.amazon.awssdk.services.s3.model.LifecycleRuleFilter;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.PutBucketLifecycleConfigurationRequest;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;


/**
 * Examples of using Amazon SQS Extended Client Library for Java 2.x
 *
 */
public class SqsExtendedClientExamples {
    // Create an Amazon S3 bucket with a random name.
    private final static String amzn-s3-demo-bucket = UUID.randomUUID() + "-"
            + DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());

    public static void main(String[] args) {
```

```
        /*
         * Create a new instance of the builder with all defaults (credentials
         * and region) set automatically. For more information, see
         * Creating Service Clients in the AWS SDK for Java Developer Guide.
         */
        final S3Client s3 = S3Client.create();

        /*
         * Set the Amazon S3 bucket name, and then set a lifecycle rule on the
         * bucket to permanently delete objects 14 days after each object's
         * creation date.
         */
        final LifecycleRule lifeCycleRule = LifecycleRule.builder()
                .expiration(LifecycleExpiration.builder().days(14).build())
                .filter(LifecycleRuleFilter.builder().prefix("").build())
                .status(ExpirationStatus.ENABLED)
                .build();
        final BucketLifecycleConfiguration lifecycleConfig =
 BucketLifecycleConfiguration.builder()
                .rules(lifeCycleRule)
                .build();

        // Create the bucket and configure it
        s3.createBucket(CreateBucketRequest.builder().bucket(amzn-s3-demo-
bucket).build());

 s3.putBucketLifecycleConfiguration(PutBucketLifecycleConfigurationRequest.builder()
                .bucket(amzn-s3-demo-bucket)
                .lifecycleConfiguration(lifecycleConfig)
                .build());
        System.out.println("Bucket created and configured.");

        // Set the Amazon SQS extended client configuration with large payload support
 enabled
        final ExtendedClientConfiguration extendedClientConfig = new
 ExtendedClientConfiguration().withPayloadSupportEnabled(s3, amzn-s3-demo-bucket);

        final SqsClient sqsExtended = new
 AmazonSQSExtendedClient(SqsClient.builder().build(), extendedClientConfig);

        // Create a long string of characters for the message object
        int stringLength = 300000;
        char[] chars = new char[stringLength];
        Arrays.fill(chars, 'x');
```

```
        final String myLongString = new String(chars);

        // Create a message queue for this example
        final String queueName = "MyQueue-" + UUID.randomUUID();
        final CreateQueueResponse createQueueResponse =
sqsExtended.createQueue(CreateQueueRequest.builder().queueName(queueName).build());
        final String myQueueUrl = createQueueResponse.queueUrl();
        System.out.println("Queue created.");

        // Send the message
        final SendMessageRequest sendMessageRequest = SendMessageRequest.builder()
                .queueUrl(myQueueUrl)
                .messageBody(myLongString)
                .build();
        sqsExtended.sendMessage(sendMessageRequest);
        System.out.println("Sent the message.");

        // Receive the message
        final ReceiveMessageResponse receiveMessageResponse =
sqsExtended.receiveMessage(ReceiveMessageRequest.builder().queueUrl(myQueueUrl).build());
        List<Message> messages = receiveMessageResponse.messages();

        // Print information about the message
        for (Message message : messages) {
            System.out.println("\nMessage received.");
            System.out.println("  ID: " + message.messageId());
            System.out.println("  Receipt handle: " + message.receiptHandle());
            System.out.println("  Message body (first 5 characters): " +
message.body().substring(0, 5));
        }

        // Delete the message, the queue, and the bucket
        final String messageReceiptHandle = messages.get(0).receiptHandle();

sqsExtended.deleteMessage(DeleteMessageRequest.builder().queueUrl(myQueueUrl).receiptHandle(me
        System.out.println("Deleted the message.");


sqsExtended.deleteQueue(DeleteQueueRequest.builder().queueUrl(myQueueUrl).build());
        System.out.println("Deleted the queue.");

        deleteBucketAndAllContents(s3);
        System.out.println("Deleted the bucket.");
```

```
    }

    private static void deleteBucketAndAllContents(S3Client client) {
        ListObjectsV2Response listObjectsResponse =
 client.listObjectsV2(ListObjectsV2Request.builder().bucket(amzn-s3-demo-
bucket).build());

        listObjectsResponse.contents().forEach(object -> {
            client.deleteObject(DeleteObjectRequest.builder().bucket(amzn-s3-demo-
bucket).key(object.key()).build());
        });

        ListObjectVersionsResponse listVersionsResponse =
 client.listObjectVersions(ListObjectVersionsRequest.builder().bucket(amzn-s3-demo-
bucket).build());

        listVersionsResponse.versions().forEach(version -> {
            client.deleteObject(DeleteObjectRequest.builder().bucket(amzn-s3-demo-
bucket).key(version.key()).versionId(version.versionId()).build());
        });

        client.deleteBucket(DeleteBucketRequest.builder().bucket(amzn-s3-demo-
bucket).build());
    }
}
```

Apache Maven을 사용하여 Java 프로젝트용 Amazon SQS 확장 클라이언트를 구성 및 빌드하거나 SDK 자체를 빌드할 수 있습니다. 애플리케이션에서 사용하는 SDK의 개별 모듈을 지정합니다.

```
<properties>
    <aws-java-sdk.version>2.20.153</aws-java-sdk.version>
</properties>

<dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>sqs</artifactId>
      <version>${aws-java-sdk.version}</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
```

```
      <artifactId>s3</artifactId>
      <version>${aws-java-sdk.version}</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>amazon-sqs-java-extended-client-lib</artifactId>
      <version>2.0.4</version>
    </dependency>

    <dependency>
      <groupId>joda-time</groupId>
      <artifactId>joda-time</artifactId>
      <version>2.12.6</version>
    </dependency>
</dependencies>
```

Python 및 Amazon S3를 사용하여 대규모 Amazon SQS 메시지 관리

Amazon S3에서 Python용 Amazon SQS Amazon SQS 확장 클라이언트 라이브러리를 사용하여 특히 256KB~2GB의 페이로드에 대해 대규모 Amazon SQS 메시지를 관리할 수 있습니다. Amazon S3 라이브러리는 메시지 페이로드를 Amazon S3 버킷에 저장하고 Amazon SQS 대기열에 저장된 객체에 대한 참조가 포함된 메시지를 보냅니다.

Python용 Amazon SQS 확장 클라이언트 라이브러리를 사용하면 다음을 수행할 수 있습니다.

- 페이로드 크기가 256KB를 초과할 때 페이로드가 항상 Amazon S3에 저장되는지 아니면 Amazon S3에만 저장되는지 지정합니다.
- Amazon S3 버킷에 저장된 단일 메시지 객체를 참조하는 메시지를 전송합니다.
- Amazon S3 버킷에서 해당 페이로드 객체 검색
- Amazon S3 버킷에서 해당 페이로드 객체 삭제

사전 조건

다음은 Python용 Amazon SQS 확장 클라이언트 라이브러리 사용을 위한 사전 조건입니다.

- 필요한 자격 증명이 있는 AWS 계정입니다. AWS 계정을 생성하려면 AWS 홈 페이지로 이동한 다음 AWS 계정 생성을 선택합니다. 지침을 따릅니다. 자격 증명에 대한 자세한 내용은 자격 증명을 참조하세요.

- AWS SDK:이 페이지의 예제에서는 AWS Python SDK Boto3를 사용합니다. SDK를 설치하고 설정하려면 AWS SDK for Python 개발자 안내서의 AWS SDK for Python 설명서를 참조하세요.
- Python 3.x(또는 이상) 및 `pip`.
- Python용 Amazon SNS 확장 클라이언트 라이브러리, PyPI에서 사용 가능

> **Note**
>
> Python용 Amazon SQS 확장 클라이언트 라이브러리를 사용하여 Python용 AWS SDK에서만 Amazon S3를 사용하여 Amazon Amazon SQS 메시지를 관리할 수 있습니다. AWS CLI, Amazon SQS 콘솔, Amazon SQS HTTP API 또는 기타 AWS SDKs로는이 작업을 수행할 수 없습니다.

메시지 스토리지 구성

Amazon SQS 확장 클라이언트는 다음 메시지 속성을 사용하여 Amazon S3 메시지 스토리지 옵션을 구성합니다.

- `large_payload_support`: 대용량 메시지를 저장하는 Amazon S3 버킷 이름입니다.
- `always_through_s3`: `True`인 경우 모든 메시지가 Amazon S3에 저장됩니다. `False`인 경우 256KB보다 작은 메시지는 s3 버킷에 직렬화되지 않습니다. 기본값은 `False`입니다.
- `use_legacy_attribute`: `True`인 경우 게시된 모든 메시지는 현재 예약된 메시지 속성(`ExtendedPayloadSize`) 대신 이전 예약 메시지 속성(`SQSLargePayloadSize`)을 사용합니다.

Python용 확장 클라이언트 라이브러리를 사용하여 대규모 Amazon SQS 메시지 관리

다음 예에서는 임의의 이름을 사용하여 Amazon S3 버킷을 생성합니다. 그런 다음 이름이 `MyQueue`인 Amazon SQS 대기열을 생성하고 S3 버킷에 저장된 256KB 이상의 메시지를 대기열로 보냅니다. 마지막으로 이 코드는 메시지를 검색하고, 메시지에 대한 정보를 반환한 후 메시지, 대기열 및 버킷을 삭제합니다.

```
import boto3
import sqs_extended_client

#Set the Amazon SQS extended client configuration with large payload.
sqs_extended_client = boto3.client("sqs", region_name="us-east-1")
```

```
sqs_extended_client.large_payload_support = "amzn-s3-demo-bucket"
sqs_extended_client.use_legacy_attribute = False


# Create an SQS message queue for this example. Then, extract the queue URL.
queue = sqs_extended_client.create_queue(
    QueueName = "MyQueue"
)
queue_url = sqs_extended_client.get_queue_url(
    QueueName = "MyQueue"
)['QueueUrl']


# Create the S3 bucket and allow message objects to be stored in the bucket.
sqs_extended_client.s3_client.create_bucket(Bucket=sqs_extended_client.large_payload_support)

# Sending a large message
small_message = "s"
large_message = small_message * 300000 # Shall cross the limit of 256 KB

send_message_response = sqs_extended_client.send_message(
    QueueUrl=queue_url,
    MessageBody=large_message
)
assert send_message_response['ResponseMetadata']['HTTPStatusCode'] == 200

# Receiving the large message
receive_message_response = sqs_extended_client.receive_message(
    QueueUrl=queue_url,
    MessageAttributeNames=['All']
)
assert receive_message_response['Messages'][0]['Body'] == large_message
receipt_handle = receive_message_response['Messages'][0]['ReceiptHandle']

# Deleting the large message
# Set to True for deleting the payload from S3
sqs_extended_client.delete_payload_from_s3 = True
delete_message_response = sqs_extended_client.delete_message(
    QueueUrl=queue_url,
    ReceiptHandle=receipt_handle
)

assert delete_message_response['ResponseMetadata']['HTTPStatusCode'] == 200
```

```
# Deleting the queue
delete_queue_response = sqs_extended_client.delete_queue(
    QueueUrl=queue_url
)

assert delete_queue_response['ResponseMetadata']['HTTPStatusCode'] == 200
```

Amazon SQS 콘솔을 사용하여 Amazon SQS 대기열 구성

Amazon SQS 콘솔을 사용하여 Amazon SQS 대기열 및 기능을 구성하고 관리합니다. 다른 방법:

- 보안 강화를 위해 서버 측 암호화를 활성화합니다.
- 배달 못한 편지 대기열을 연결하여 처리되지 않은 메시지를 처리합니다.
- 이벤트 기반 처리를 위해 Lambda 함수를 호출하도록 트리거를 설정합니다.

Amazon SQS에 대한 속성 기반 액세스 제어

ABAC란 무엇입니까?

속성 기반 액세스 제어(ABAC)는 사용자 및 AWS 리소스에 연결된 태그를 기반으로 권한을 정의하는 권한 부여 프로세스입니다. ABAC는 속성 및 값을 기반으로 세분화되고 유연한 액세스 제어를 제공하고 재구성된 역할 기반 정책과 관련된 보안 위험을 줄이며 감사 및 액세스 정책 관리를 중앙 집중화합니다. ABAC에 대한 자세한 내용은 IAM 사용 설명서의 AWS용 ABAC란 무엇입니까? 섹션을 참조하세요.

Amazon SQS는 Amazon SQS 대기열과 연결된 태그 및 별칭을 기반으로 Amazon SQS 대기열에 대한 액세스를 제어할 수 있도록 하여 ABAC를 지원합니다. Amazon SQS에서 ABAC를 활성화하는 태그 및 별칭 조건 키는 정책을 편집하거나 권한 부여를 관리하지 않고도 IAM 보안 주체가 Amazon SQS 대기열을 사용할 수 있는 권한을 부여합니다. ABAC 조건 키에 대해 자세히 알아보려면 서비스 승인 참조의 Amazon SQS에 대한 조건 키를 참조하세요.

ABAC를 사용하면 태그를 사용하여 Amazon SQS 대기열에 대한 IAM 액세스 권한 및 정책을 구성할 수 있으며, 이는 권한 관리의 규모를 조정하는 데 도움이 됩니다. 새 리소스를 추가할 때마다 정책을 업데이트할 필요 없이 각 비즈니스 역할에 추가하는 태그를 사용하여 IAM에서 단일 권한 정책을 생성할 수 있습니다. 또한 IAM 보안 주체에 태그를 추가하여 ABAC 정책을 생성할 수 있습니다. 호출하는 IAM 사용자 역할의 태그가 Amazon SQS 대기열 태그와 일치할 때 Amazon SQS 작업을 허용하도록 ABAC 정책을 설계할 수 있습니다. 태그 지정에 대한 자세한 내용은 AWS 태그 지정 전략 및 섹션을 AWS참조하세요Amazon SQS 비용 할당 태그.

> **Note**
>
> Amazon SQS용 ABAC는 현재 Amazon SQS를 사용할 수 있는 모든 AWS 상용 리전에서 사용할 수 있습니다. 단, 다음은 예외입니다.

- 아시아 태평양(하이데라바드)
- 아시아 태평양(멜버른)
- 유럽(스페인)
- 유럽(취리히)

Amazon SQS에 ABAC를 사용해야 하는 이유는 무엇입니까?

Amazon SQS에서 ABAC를 사용할 때 얻을 수 있는 몇 가지 이점은 다음과 같습니다.

- Amazon SQS의 ABAC에는 필요한 권한 정책이 적습니다. 각 직무에 서로 다른 정책을 생성할 필요가 없습니다. 둘 이상의 대기열에 적용되는 리소스 및 요청 태그를 사용할 수 있으므로 운영 오버헤드가 줄어듭니다.
- ABAC를 사용하여 팀을 빠르게 확장할 수 있습니다. 리소스 생성 중에 리소스에 태그가 적절하게 지정되면 새 리소스에 대한 권한이 태그에 따라 자동으로 부여됩니다.
- IAM 보안 주체에 대한 권한을 사용하여 리소스 액세스를 제한할 수 있습니다. IAM 보안 주체용 태그를 생성하고 이를 사용하여 IAM 보안 주체의 태그와 일치하는 특정 작업에 대한 액세스를 제한할 수 있습니다. 이렇게 하면 요청 권한 부여 프로세스를 자동화할 수 있습니다.
- 누가 리소스에 액세스하는지 추적할 수 있습니다. AWS CloudTrail에서 사용자 속성을 살펴봄으로써 세션의 ID를 확인할 수 있습니다.

주제

- Amazon SQS에서 액세스 제어용 태그 지정
- IAM 사용자 및 Amazon SQS 대기열 생성
- Amazon SQS에서 속성 기반 액세스 제어 테스트

Amazon SQS에서 액세스 제어용 태그 지정

다음은 Amazon SQS에서 액세스 제어에 태그를 사용하는 예입니다. IAM 정책은 IAM 사용자를 키 ‘environment’ 및 값 ‘production’인 리소스 태그를 포함하는 모든 대기열에 대한 모든 Amazon SQS 작업으로 제한합니다. 자세한 내용은 태그 및 AWS Organizations를 사용한 속성 기반 액세스 제어를 참조하세요.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessForProd",
      "Effect": "Deny",
      "Action": "sqs:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "prod"
        }
      }
    }
  ]
}
```

IAM 사용자 및 Amazon SQS 대기열 생성

다음 예제에서는 AWS Management Console 및를 사용하여 Amazon SQS에 대한 액세스를 제어하는 ABAC 정책을 생성하는 방법을 설명합니다 AWS CloudFormation.

사용 AWS Management Console

IAM 사용자 생성

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 사용자를 선택합니다.
3. 사용자 추가를 선택하고 사용자 이름 텍스트 상자에 이름을 입력합니다.
4. 액세스 키 - 프로그래밍 방식 액세스를 선택하고 다음: 권한을 선택합니다.
5. 다음: 태그를 선택합니다.
6. 태그 키를 `environment`로 추가하고 태그 값을 `beta`로 추가합니다.
7. 다음: 검토를 선택한 후 사용자 생성을 선택합니다.
8. 액세스 키 ID와 비밀 액세스 키를 복사하여 안전한 위치에 저장합니다.

IAM 사용자 권한 추가

1. 생성한 IAM 사용자를 선택합니다.
2. 인라인 정책 추가(Add inline policy)를 선택합니다.
3. JSON 탭에 다음 정책을 붙여넣습니다.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessForSameResTag",
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage",
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "${aws:PrincipalTag/environment}"
        }
      }
    },
    {
      "Sid": "AllowAccessForSameReqTag",
      "Effect": "Allow",
      "Action": [
        "sqs:CreateQueue",
        "sqs:DeleteQueue",
        "sqs:SetQueueAttributes",
        "sqs:tagqueue"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/environment": "${aws:PrincipalTag/environment}"
        }
      }
    },
```

```
  ]
}
```

4. 정책 검토를 선택합니다.
5. 정책 생성을 선택합니다.

사용 AWS CloudFormation

다음 샘플 AWS CloudFormation 템플릿을 사용하여 인라인 정책과 Amazon SQS 대기열이 연결된 IAM 사용자를 생성합니다.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "CloudFormation template to create IAM user with custom inline policy"
Resources:
    IAMPolicy:
        Type: "AWS::IAM::Policy"
        Properties:
            PolicyDocument: |
                {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "AllowAccessForSameResTag",
                            "Effect": "Allow",
                            "Action": [
                                "sqs:SendMessage",
                                "sqs:ReceiveMessage",
                                "sqs:DeleteMessage"
                            ],
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
                                    "aws:ResourceTag/environment": "${aws:PrincipalTag/
environment}"
                                }
                            }
                        },
                        {
                            "Sid": "AllowAccessForSameReqTag",
                            "Effect": "Allow",
                            "Action": [
                                "sqs:CreateQueue",
```

```
                                "sqs:DeleteQueue",
                                "sqs:SetQueueAttributes",
                                "sqs:tagqueue"
                            ],
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
                                    "aws:RequestTag/environment": "${aws:PrincipalTag/
environment}"
                                }
                            }
                        },
                        {
                            "Sid": "DenyAccessForProd",
                            "Effect": "Deny",
                            "Action": "sqs:*",
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
                                    "aws:ResourceTag/stage": "prod"
                                }
                            }
                        }
                    ]
                }

            Users:
              - "testUser"
            PolicyName: tagQueuePolicy

    IAMUser:
        Type: "AWS::IAM::User"
        Properties:
            Path: "/"
            UserName: "testUser"
            Tags:
              -
                Key: "environment"
                Value: "beta"
```

Amazon SQS에서 속성 기반 액세스 제어 테스트

다음 예제는 Amazon SQS에 속성 기반 액세스 제어를 테스트하는 방법을 보여줍니다.

태그 키를 environment로 설정하고 태그 값을 prod로 설정하여 대기열을 생성합니다.

이 AWS CLI 명령을 실행하여 태그 키가 환경으로 설정되고 태그 값이 prod로 설정된 대기열 생성을 테스트합니다. AWS CLI가 없는 경우 시스템에 맞게 CLI를 다운로드하고 구성할 수 있습니다.

```
aws sqs create-queue --queue-name prodQueue –region us-east-1 –tags "environment=prod"
```

Amazon SQS 엔드포인트에서 다음과 같은 `AccessDenied` 오류가 발생합니다.

```
An error occurred (AccessDenied) when calling the CreateQueue operation: Access to the
 resource <queueUrl> is denied.
```

이는 IAM 사용자의 태그 값이 `CreateQueue` API 호출에서 전달된 태그와 일치하지 않기 때문입니다. 키가 `environment`로 설정되고 값이 `beta`로 설정된 태그를 IAM 사용자에게 적용했다는 점을 기억하세요.

태그 키를 environment로 설정하고 태그 값을 beta로 설정하여 대기열을 생성합니다.

이 CLI 명령을 실행하여 태그 키가 `environment`로 설정되고 태그 값이 `beta`로 설정된 대기열 생성을 테스트합니다.

```
aws sqs create-queue --queue-name betaQueue –region us-east-1 –tags "environment=beta"
```

아래와 비슷한 방법으로 대기열이 성공적으로 생성되었음을 확인하는 메시지가 나타납니다.

```
{
"QueueUrl": "<queueUrl>“
}
```

대기열로 메시지 전송

이 CLI 명령을 실행하여 대기열에 메시지 전송을 테스트합니다.

```
aws sqs send-message --queue-url <queueUrl> --message-body testMessage
```

응답은 Amazon SQS 대기열로 메시지가 성공적으로 전송되었음을 보여줍니다. IAM 사용자 권한을 사용하면 `beta` 태그가 있는 대기열에 메시지를 보낼 수 있습니다. 응답에는 메시지가 포함된 `MD5OfMessageBody` 및 `MessageId`가 포함됩니다.

```
{
"MD5OfMessageBody": "<MD5OfMessageBody>",
"MessageId": "<MessageId>"
}
```

Amazon SQS 콘솔을 사용하여 대기열 파라미터 구성

대기열을 생성하거나 편집할 때 다음 파라미터를 구성할 수 있습니다.

- 표시 제한 시간 - 한 소비자가 대기열에서 수신한 메시지가 다른 메시지 소비자에게 표시되지 않는 시간입니다. 자세한 내용은 표시 제한 시간을 참조하세요.

> **Note**
>
> 콘솔을 사용하여 표시 제한 시간을 구성하면 대기열에 있는 모든 메시지에 대한 제한 시간 값이 구성됩니다. 단일 또는 여러 메시지에 대한 제한 시간을 구성하려면 AWS SDKs 중 하나를 사용해야 합니다.

- 메시지 보존 기간 - Amazon SQS가 대기열에 남아 있는 메시지를 유지하는 기간입니다. 기본적으로 대기열의 메시지는 4일 동안 유지됩니다. 메시지를 최대 14일 동안 유지하도록 대기열을 구성할 수 있습니다. 자세한 내용은 메시지 보존 기간을 참조하세요.
- 전송 지연 - Amazon SQS가 대기열에 추가된 메시지를 전송하기 전에 지연하는 시간입니다. 자세한 내용은 전송 지연을 참조하세요.
- 최대 메시지 크기 - 이 대기열의 최대 메시지 크기입니다. 자세한 내용은 최대 메시지 크기를 참조하세요.
- 메시지 수신 대기 시간 - 대기열이 수신 요청을 받은 후 Amazon SQS가 메시지를 사용할 수 있을 때까지 기다리는 최대 시간입니다. 자세한 내용은 Amazon SQS 짧은 폴링 및 긴 폴링 단원을 참조하십시오.
- 콘텐츠 기반 중복 제거 활성화 - Amazon SQS는 메시지 본문을 기반으로 중복 제거 ID를 자동으로 생성할 수 있습니다. 자세한 내용은 Amazon SQS FIFO 대기열 단원을 참조하십시오.
- 높은 처리량 FIFO 활성화 - 대기열의 메시지에 대해 높은 처리량을 활성화하는 데 사용합니다. 이 옵션을 선택하면 관련 옵션(중복 제거 범위 및 FIFO 처리량 한도)이 FIFO 대기열의 높은 처리량을 활성화하는 데 필요한 설정으로 변경됩니다. 자세한 내용은 Amazon SQS에서 FIFO 대기열의 높은 처리량 및 Amazon SQS 메시지 할당량 섹션을 참조하세요.
- 리드라이브 허용 정책: 이 대기열을 배달 못한 편지 대기열로 사용할 수 있는 소스 대기열을 정의합니다. 자세한 내용은 Amazon SQS에서 배달 못한 편지 대기열 사용 단원을 참조하십시오.

기존 대기열에 대한 대기열 파라미터를 구성하려면(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다. 대기열을 선택하고 편집을 선택합니다.
3. 구성 섹션으로 스크롤합니다.
4. 표시 제한 시간에 기간과 단위를 입력합니다. 범위는 0초~12시간입니다. 기본값은 30초입니다.
5. 메시지 보존 기간에 기간과 단위를 입력합니다. 범위는 1분~14일입니다. 기본값은 4일입니다.
6. 표준 대기열의 경우 메시지 수신 대기 시간 값을 입력합니다. 범위는 0~20초입니다. 기본값은 0초이며 짧은 폴링을 설정합니다. 0이 아닌 값은 모두 긴 폴링을 설정합니다.
7. 전송 지연에 기간과 단위를 입력합니다. 범위는 0초~15분입니다. 기본값은 0초입니다.
8. 최대 메시지 크기에는 값을 입력합니다. 범위는 1KiB~1024KiB입니다. 기본값은 1024KiB입니다.
9. FIFO 대기열의 경우 콘텐츠 기반 중복 제거 활성화를 선택하여 콘텐츠 기반 중복 제거를 활성화합니다. 기본 설정은 비활성화입니다.
10. (선택 사항) FIFO 대기열에서 메시지를 보내고 받을 때 처리량을 높이려면 높은 처리량 FIFO 활성화를 선택합니다.

    이 옵션을 선택하면 관련 옵션(중복 제거 범위 및 FIFO 처리량 한도)이 FIFO 대기열의 높은 처리량을 활성화하는 데 필요한 설정으로 변경됩니다. 높은 처리량 FIFO를 사용하는 데 필요한 설정을 변경하면 대기열에 일반 처리량이 적용되고 지정된 대로 중복 제거가 수행됩니다. 자세한 내용은 Amazon SQS에서 FIFO 대기열의 높은 처리량 및 Amazon SQS 메시지 할당량 섹션을 참조하세요.
11. 리드라이브 허용 정책의 경우 활성화됨을 선택합니다. 모두 허용(기본값), 대기열 기준 또는 모두 거부 중에서 선택합니다. 대기열 기준 선택 시 Amazon 리소스 이름(ARN)별로 최대 10개의 소스 대기열 목록을 지정합니다.
12. 대기열 파라미터 구성을 마쳤으면 저장을 선택합니다.

Amazon SQS에서 액세스 정책 구성

대기열을 편집할 때 대기열과 상호 작용할 수 있는 사용자를 제어하도록 액세스 정책을 구성할 수 있습니다.

- 액세스 정책은 대기열에 액세스할 수 있는 권한이 있는 계정, 사용자 및 역할을 정의합니다.
- , `SendMessage` `ReceiveMessage`또는와 같이 허용되는 작업을 지정합니다`DeleteMessage`.
- 기본적으로 대기열 소유자만 메시지를 보내고 받을 수 있는 권한이 있습니다.

기존 대기열에 대한 액세스 정책을 구성하려면(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열을 선택하고 편집을 선택합니다.
4. 액세스 정책 섹션으로 스크롤합니다.
5. 입력 상자에서 액세스 정책 설명을 편집합니다. 액세스 정책 설명에 대한 자세한 내용은 Amazon SQS의 Identity and Access Management 섹션을 참조하세요.
6. 액세스 정책 구성을 마쳤으면 저장을 선택합니다.

SQS 관리형 암호화 키를 사용하여 대기열에 대해 서버 측 암호화 구성

기본 Amazon SQS 관리형 서버 측 암호화(SSE) 옵션 외에도 Amazon SQS 관리형 SSE(SSE-SQS)를 사용하면 SQS 관리형 암호화 키 기반의 사용자 지정 관리형 서버 측 암호화를 생성하여 메시지 대기열을 통해 전송되는 민감한 데이터를 보호할 수 있습니다. SSE-SQS를 사용하면 암호화 키를 생성 및 관리하거나 데이터 암호화를 위해 코드를 수정할 필요가 없습니다. SSE-SQS를 사용하면 데이터를 안전하게 전송하는 한편, 추가 비용 없이 엄격한 암호화 규정 준수 및 규제 요구 사항을 충족할 수 있습니다.

SSE-SQS는 256비트 Advanced Encryption Standard(AES-256) 암호화를 사용하여 저장 데이터를 보호합니다. Amazon SQS가 메시지를 수신하면 SSE가 메시지를 즉시 암호화합니다. Amazon SQS는 메시지를 암호화된 형태로 저장하고 권한 있는 소비자에게 메시지를 보낼 때만 암호를 해독합니다.

> **Note**
>
> - 기본 SSE 옵션은 암호화 속성을 지정하지 않고 대기열을 생성할 때만 유효합니다.
> - Amazon SQS는 모든 대기열 암호화를 끌 수 있습니다. 따라서 KMS-SSE를 끄면 SQS-SSE가 자동으로 활성화되지 않습니다. KMS-SSE를 끈 후 SQS-SSE를 활성화하려면 요청에 속성 변경을 추가해야 합니다.

대기열에 대해 SSE-SQS 암호화를 구성하려면(콘솔)

> **Note**
>
> HTTP(비 TLS) 엔드포인트를 사용하여 만든 새 대기열에서는 기본적으로 SSE-SQS 암호화가 활성화되지 않습니다. HTTPS 또는 서명 버전 4 엔드포인트를 사용하여 Amazon SQS 대기열을 생성하는 것이 보안 모범 사례입니다.

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열을 선택한 다음 편집을 선택합니다.
4. 암호화를 확장합니다.
5. 서버 측 암호화에서 활성화(기본)를 선택합니다.

   > **Note**
   >
   > SSE를 활성화하면 암호화된 대기열에 대한 익명 `SendMessage` 및 `ReceiveMessage` 요청이 거부됩니다. Amazon SQS 보안 모범 사례에서는 익명 요청을 사용하지 말 것을 권장합니다. Amazon SQS 대기열로 익명 요청을 보내려면 SSE를 비활성화해야 합니다.

6. Amazon SQS 키(SSE-SQS)를 선택합니다. 이 옵션 사용에 대한 추가 요금은 부과되지 않습니다.
7. 저장을 선택합니다.

Amazon SQS 콘솔을 사용하여 대기열에 대한 서버 측 암호화 구성

대기열 메시지의 데이터를 보호하기 위해 Amazon SQS는 새로 생성된 모든 대기열에 대해 기본적으로 서버 측 암호화(SSE)를 사용하도록 설정되어 있습니다. Amazon SQS는 Amazon Web Services Key Management Service(Amazon Web Services KMS)와 통합하여 서버 측 암호화(SSE)용 KMS 키를 관리합니다. SSE 사용에 대한 자세한 내용은 Amazon SQS에서 저장 시 암호화 섹션을 참조하세요.

대기열에 할당하는 KMS 키에는 대기열을 사용할 수 있는 권한이 있는 모든 주체에 대한 권한이 포함된 키 정책이 있어야 합니다. 자세한 내용은 키 관리를 참조하세요.

해당 KMS 키의 소유자가 아니거나 `kms:ListAliases` 및 `kms:DescribeKey` 권한이 없는 계정으로 로그인하는 경우 Amazon SQS 콘솔에서 해당 KMS 키에 대한 정보를 볼 수 없습니다. KMS 키의 소유자에게 이 권한을 부여해 달라고 요청해야 합니다. 자세한 내용은 키 관리를 참조하세요.

대기열을 만들 때 또는 편집할 때 SSE-KMS를 구성할 수 있습니다.

기존 대기열에 대해 SSE-KMS 구성(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열을 선택한 다음 편집을 선택합니다.
4. 암호화를 확장합니다.
5. 서버 측 암호화에서 활성화(기본)를 선택합니다.

> **Note**
>
> SSE를 활성화하면 암호화된 대기열에 대한 익명 `SendMessage` 및 `ReceiveMessage` 요청이 거부됩니다. Amazon SQS 보안 모범 사례에서는 익명 요청을 사용하지 말 것을 권장합니다. Amazon SQS 대기열로 익명 요청을 보내려면 SSE를 비활성화해야 합니다.

6. AWS 키 관리 서비스 키(SSE-KMS)를 선택합니다.

   콘솔에 설명, 계정 및 KMS 키의 KMS 키 ARN이 표시됩니다.
7. 대기열의 KMS 키 ID를 지정합니다. 자세한 내용은 주요 용어 단원을 참조하십시오.

   a. KMS 키 별칭 선택 옵션을 선택합니다.

   b. 기본 키는 Amazon SQS용 Amazon Web Services 관리형 KMS 키입니다. 이 키를 사용하려면 KMS 키 목록에서 키를 선택합니다.

   c. Amazon Web Services 계정에서 사용자 지정 KMS 키를 사용하려면 KMS 키 목록에서 해당 키를 선택합니다. 사용자 지정 KMS 키를 만드는 방법에 대한 지침은 Amazon Web Services Key Management Service 개발자 가이드에서 키 만들기를 참조하세요.

   d. 목록에 없는 사용자 지정 KMS 키 또는 다른 Amazon Web Services 계정의 사용자 지정 KMS 키를 사용하려면 KMS 키 별칭 입력을 선택하고 KMS 키 Amazon 리소스 이름(ARN)을 입력합니다.
8. (선택 사항) 데이터 키 재사용 기간에서 1분에서 24시간 사이의 값을 지정합니다. 기본값은 5분입니다. 자세한 내용은 데이터 키 재사용 기간 이해 단원을 참조하십시오.
9. SSE-KMS 구성을 마치면 저장을 선택합니다.

Amazon SQS 콘솔을 사용하여 대기열에 대한 비용 할당 태그 구성

Amazon SQS 대기열을 구성하고 식별하기 위해 비용 할당 태그를 추가할 수 있습니다. 자세한 내용은 Amazon SQS 비용 할당 태그 단원을 참조하십시오.

- 세부 정보 페이지의 태그 지정 탭에는 대기열의 태그가 표시됩니다.
- 대기열을 생성하거나 편집할 때 태그를 추가하거나 수정할 수 있습니다.

기존 대기열에 대해 태그 구성(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열을 선택하고 편집을 선택합니다.
4. 스크롤하여 태그 섹션으로 이동합니다.
5. 다음과 같이 대기열 태그를 추가, 수정 또는 제거합니다.
   a. 태그를 추가하려면 새 태그 추가를 선택하고 키 및 값을 입력한 다음 새 태그 추가를 선택합니다.
   b. 태그를 업데이트하려면 해당 태그의 키와 값을 변경합니다.
   c. 태그를 제거하려면 키-값 페어 옆에 있는 제거를 선택합니다.
6. 태그 구성을 마치면 저장을 선택합니다.

Amazon SQS 콘솔을 사용하여 대기열에서 Amazon SNS 주제 구독

하나 이상의 Amazon SQS 대기열에서 Amazon SNS 주제를 구독할 수 있습니다. 주제에 메시지를 게시하면 Amazon SNS는 구독한 각 대기열에 메시지를 보냅니다. Amazon SQS는 구독을 관리하고 필요한 권한을 처리합니다. Amazon SNS에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 설명서의 Amazon SNS란 무엇입니까?를 참조하세요.

Amazon SQS 대기열에서 Amazon SNS 주제를 구독하면 Amazon SNS는 HTTPS를 사용하여 Amazon SQS에 메시지를 전달합니다. 암호화된 Amazon SQS 대기열과 함께 Amazon SNS를 사용하는 방법에 대한 자세한 내용은 AWS 서비스에 대한 KMS 권한 구성 섹션을 참조하세요.

> **Important**
>
> Amazon SQS는 각 액세스 정책에 대해 최대 20개의 문을 지원합니다. Amazon SNS 주제를 구독하면 이러한 명령문이 하나 추가됩니다. 이 개수를 초과하면 주제 구독 전송이 실패합니다.

Amazon SNS 주제에 대한 대기열을 구독하려면(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열 목록에서 Amazon SNS 주제를 구독할 대기열을 선택합니다.
4. 작업(Actions)에서 Amazon SNS 주제 구독(Subscribe to Amazon SNS topic)을 선택합니다.
5. 이 대기열에 사용할 수 있는 Amazon SNS 주제 지정 메뉴에서 대기열의 Amazon SNS 주제를 선택합니다.

   SNS 주제가 나열되지 않은 경우 Amazon SNS 주제 ARN 입력을 선택한 다음 주제의 Amazon 리소스 이름(ARN)을 입력합니다.
6. 저장을 선택합니다.
7. 구독을 확인하려면 주제에 메시지를 게시하고 대기열에서 메시지를 확인합니다. 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 메시지 게시를 참조하세요.

교차 계정 구독

Amazon SQS 대기열과 Amazon SNS 주제가 다른 경우 AWS 계정추가 권한이 필요합니다.

주제 소유자(계정 A)

Amazon SNS 주제의 액세스 정책을 수정하여 Amazon SQS 대기열의 구독 AWS 계정 을 허용합니다. 정책 설명의 예:

```
{
  "Effect": "Allow",
  "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
  "Action": "sns:Subscribe",
  "Resource": "arn:aws:sns:us-east-1:123456789012:MyTopic"
}
```

이 정책은 계정이 `111122223333`를 구독하도록 허용합니다`MyTopic`.

대기열 소유자(계정 B)

Amazon SNS 주제가 메시지를 전송할 수 있도록 Amazon SQS Amazon SNS 대기열의 액세스 정책을 수정합니다. 정책 설명의 예:

```
{
  "Effect": "Allow",
  "Principal": { "Service": "sns.amazonaws.com" },
  "Action": "sqs:SendMessage",
  "Resource": "arn:aws:sqs:us-east-1:111122223333:MyQueue",
  "Condition": {
    "ArnEquals": { "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:MyTopic" }
  }
}
```

이 정책은 `MyTopic`가에 메시지를 보내도록 허용합니다`MyQueue`.

리전 간 구독

다른의 Amazon SNS 주제를 구독하려면 다음을 AWS 리전확인하세요.

- Amazon SNS 주제의 액세스 정책은 리전 간 구독을 허용합니다.
- Amazon SQS 대기열의 액세스 정책은 Amazon SNS 주제가 리전 간에 메시지를 전송하도록 허용합니다.

자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 다른 리전에 있는 Amazon SQS 대기열 또는 함수로 Amazon SNS Amazon SQS 메시지 전송 AWS Lambda을 참조하세요.

AWS Lambda 함수를 트리거하도록 Amazon SQS 대기열 구성

Lambda 함수를 사용하여 Amazon SQS 대기열의 메시지를 처리할 수 있습니다. Lambda는 대기열을 폴링하고 함수를 동기적으로 호출하여 메시지 배치를 이벤트로 전달합니다.

가시성 제한 시간 구성

대기열의 표시 제한 시간을 함수 제한 시간의 6배 이상으로 설정합니다. 이렇게 하면 이전 배치를 처리하는 동안 함수가 제한되는 경우 Lambda가 재시도할 시간을 충분히 확보할 수 있습니다.

DLQ(Dead Letter Queue) 사용

배달 못한 편지 대기열을 지정하여 Lambda 함수가 처리하지 못하는 메시지를 캡처합니다.

여러 대기열 및 함수 처리

Lambda 함수는 각 대기열에 대해 별도의 이벤트 소스를 생성하여 여러 대기열을 처리할 수 있습니다. 여러 Lambda 함수를 동일한 대기열에 연결할 수도 있습니다.

암호화된 대기열에 대한 권한

암호화된 대기열을 Lambda 함수와 연결하지만 Lambda가 메시지를 폴링하지 않는 경우 Lambda 실행 역할에 `kms:Decrypt` 권한을 추가합니다.

제한 사항

대기열과 Lambda 함수는 동일한에 있어야 합니다 AWS 리전.

기본 키(Amazon SQS용AWS 관리형 KMS 키)를 사용하는 암호화된 대기열은 다른에서 Lambda 함수를 호출할 수 없습니다 AWS 계정.

구현 세부 정보는 AWS Lambda 개발자 안내서의 Amazon SQS에서 사용을 AWS Lambda 참조하세요.

사전 조건

Lambda 함수 트리거를 구성하려면 다음 요구 사항을 충족해야 합니다.

- 사용자를 사용하는 경우 Amazon SQS 역할에 다음 권한이 포함되어야 합니다.
  - `lambda:CreateEventSourceMapping`
  - `lambda:ListEventSourceMappings`
  - `lambda:ListFunctions`
- Lambda 실행 역할에는 다음 권한이 포함되어야 합니다.
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `sqs:ReceiveMessage`
- 암호화된 대기열을 Lambda 함수와 연결하는 경우 Lambda 실행 역할에 `kms:Decrypt` 권한을 추가합니다.

자세한 내용은 Amazon SQS의 액세스 관리 개요 단원을 참조하십시오.

Lambda 함수를 트리거하는 대기열을 구성하려면(콘솔)

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열 페이지에서 구성할 대기열을 선택합니다.
4. 대기열 페이지에서 Lambda 트리거 탭을 선택합니다.
5. Lambda 트리거 페이지에서 Lambda 트리거를 선택합니다.

   목록에 필요한 Lambda 트리거가 포함되어 있지 않은 경우 Lambda 함수 트리거 구성을 선택합니다. Lambda 함수의 Amazon 리소스 이름(ARN)을 입력하거나 기존 리소스를 선택합니다. 그런 다음 저장을 선택합니다.
6. 저장을 선택합니다. 콘솔에서 구성을 저장하고 대기열의 세부 정보 페이지를 표시합니다.

   세부 정보 페이지의 Lambda 트리거 탭에는 Lambda 함수와 해당 상태가 표시됩니다. Lambda 함수가 대기열에 연결되기까지 약 1분 정도 소요됩니다.
7. 구성 결과를 확인하려면 대기열에 메시지를 전송한 후 Lambda 콘솔에서 트리거된 Lambda 함수를 확인합니다.

Amazon EventBridge를 사용하여 AWS 서비스에서 Amazon SQS로 알림 자동화 Amazon SQS

Amazon EventBridge를 사용하면 애플리케이션 문제 또는 리소스 변경과 같은 이벤트를 거의 실시간으로 자동화 AWS 서비스 하고 대응할 수 있습니다.

- 규칙을 생성하여 특정 이벤트를 필터링하고 규칙이 이벤트와 일치할 때 자동화된 작업을 정의할 수 있습니다.
- EventBridge는 JSON 형식으로 이벤트를 수신하는 Amazon SQS 표준 및 FIFO 대기열을 포함한 여러 대상을 지원합니다.

자세한 정보는 Amazon EventBridge 사용 설명서의 Amazon EventBridge 대상을 참조하세요.

Amazon SQS를 사용하여 속성을 포함하는 메시지 전송

표준 및 FIFO 대기열의 경우 타임스탬프, 지리 공간 데이터, 서명 및 식별자를 포함하여 메시지에 구조화된 메타데이터를 포함할 수 있습니다. 자세한 내용은 Amazon SQS 메시지 속성 단원을 참조하십시오.

Amazon SQS 콘솔을 사용하여 속성이 있는 메시지를 대기열로 전송하려면

1. https://console.aws.amazon.com/sqs/에서 Amazon SQS 콘솔을 엽니다.
2. 탐색 창에서 대기열을 선택합니다.
3. 대기열 페이지에서 대기열을 선택합니다.
4. [메시지 전송 및 수신(Send and receive messages)]을 선택합니다.
5. 메시지 속성 파라미터를 입력합니다.
   a. 이름 텍스트 상자에 최대 256자의 고유한 이름을 입력합니다.
   b. 속성 유형으로는 문자열, 숫자 또는 2진수를 선택합니다.
   c. (선택 사항) 사용자 지정 데이터 유형을 입력합니다. 예를 들어, 번호에 **`byte`**, **`int`** 또는 **`float`**를 사용자 지정 데이터 유형으로 추가할 수 있습니다.
   d. 값 텍스트 상자에 메시지 속성 값을 입력합니다.

6. 다른 메시지 속성을 추가하려면 새 속성 추가를 선택합니다.

7. 메시지를 전송하기 전에 언제든지 속성 값을 수정할 수 있습니다.
8. 속성을 삭제하려면 제거를 선택합니다. 첫 번째 속성을 삭제하려면 메시지 속성을 닫습니다.

9. 속성을 메시지에 추가했으면 메시지 전송을 선택합니다. 메시지가 전송되고 콘솔에 성공 메시지가 표시됩니다. 보낸 메시지의 메시지 속성에 대한 정보를 보려면 세부 정보 보기를 선택합니다. 완료를 선택하여 메시지 세부 정보 대화 상자를 닫습니다.

Amazon SQS 모범 사례

Amazon SQS는 메시지 대기열을 관리하고 처리하여 애플리케이션의 여러 부분이 메시지를 안정적이고 대규모로 교환할 수 있도록 합니다. 이 주제에서는 긴 폴링을 사용하여 빈 응답을 줄이고, 배달 못한 편지 대기열을 구현하여 처리 오류를 처리하고, 보안을 위해 대기열 권한을 최적화하는 등 주요 운영 모범 사례를 다룹니다.

주제

- Amazon SQS 오류 처리 및 문제가 있는 메시지
- Amazon SQS 메시지 중복 제거 및 그룹화
- Amazon SQS 메시지 처리 및 타이밍

Amazon SQS 오류 처리 및 문제가 있는 메시지

이 주제에서는 요청 오류 처리, 문제가 있는 메시지 캡처, 메시지 신뢰성을 보장하기 위한 Dead Letter Queue(DLQ) 보존 구성 등의 기술을 포함하여 Amazon SQS에서 오류를 관리하고 완화하는 방법에 대한 자세한 지침을 제공합니다.

주제

- Amazon SQS에서 요청 오류 처리
- Amazon SQS에서 문제가 있는 메시지 캡처
- Amazon SQS에서 Dead Letter Queue(DLQ) 보존 설정

Amazon SQS에서 요청 오류 처리

요청 오류를 처리하려면 다음 전략 중 하나를 사용하십시오.

- AWS SDK를 사용하는 경우 이미 자동 재시도 및 백오프 로직을 사용할 수 있습니다. 자세한 내용은 Amazon Web Services 일반 참조의 AWS의 오류 재시도 및 지수 백오프 단원을 참조하세요.
- 재시도 및 백오프에 AWS SDK 기능을 사용하지 않는 경우 Amazon SQS에서 메시지, 제한 시간 또는 오류 메시지를 수신하지 않은 후 ReceiveMessage 작업을 재시도하기 전에 일시 중지(예: 200ms)를 허용하세요. 그 다음에 동일한 결과를 반환하는 `ReceiveMessage`를 사용하려면, 더 긴 일시 중지 시간(예: 400ms)을 둡니다.

Amazon SQS에서 문제가 있는 메시지 캡처

처리할 수 없는 모든 메시지를 캡처하고 정확한 CloudWatch 지표를 수집하려면 배달 못한 편지 대기열을 구성합니다.

- 소스 대기열이 지정된 횟수까지 메시지를 처리하지 못하고 나면 리드라이브 정책에 따라 메시지를 배달 못한 편지 대기열로 리디렉션합니다.
- 배달 못한 편지 대기열을 사용하면 메시지 개수가 감소하고 사용자가 독약 메시지(수신하지만 처리 불가능한 메시지)를 받을 가능성이 낮아집니다.
- 대기열에 독약 메시지가 포함되면 독약 메시지의 잘못된 수명 지정으로 인해 `ApproximateAgeOfOldestMessage` CloudWatch 지표가 왜곡될 수 있습니다. 배달 못한 편지 대기열을 구성하면 이 측정치를 사용할 때 거짓 경보를 방지할 수 있습니다.

Amazon SQS에서 Dead Letter Queue(DLQ) 보존 설정

표준 대기열의 경우, 메시지 만료는 항상 메시지가 원래 대기열에 추가된 타임스탬프를 기준으로 합니다. 메시지가 DLQ(Dead Letter Queue)로 이동하면 대기열에 추가 타임스탬프는 변경되지 않습니다. `ApproximateAgeOfOldestMessage` 지표는 메시지가 원래 전송된 시간이 아니라 메시지가 DLQ(Dead Letter Queue)로 이동한 시간을 나타냅니다. 예를 들어, 메시지가 DLQ(Dead Letter Queue)로 이동하기 전에 원래 대기열에서 1일을 보낸다고 가정합니다. DLQ(Dead Letter Queue)의 보존 기간이 4일인 경우 메시지는 3일 후에 DLQ(Dead Letter Queue)에서 삭제되며 `ApproximateAgeOfOldestMessage`는 3일입니다. 따라서 배달 못한 편지 대기열의 보존 기간을 항상 원래 대기열의 보존 기간보다 더 길게 설정하는 것이 좋습니다.

FIFO 대기열의 경우 메시지가 DLQ(Dead Letter Queue)로 이동하면 대기열에 추가 타임스탬프가 재설정됩니다. `ApproximateAgeOfOldestMessage` 지표는 메시지가 DLQ(Dead Letter Queue)로 이동한 시간을 나타냅니다. 위의 동일한 예에서 메시지는 4일 후에 배달 못한 편지 대기열에서 삭제되며 `ApproximateAgeOfOldestMessage`는 4일입니다.

Amazon SQS 메시지 중복 제거 및 그룹화

이 주제에서는 Amazon SQS에서 일관된 메시지 처리를 보장하는 모범 사례를 제공합니다. 사용 방법을 설명합니다.

- `MessageDeduplicationId` FIFO 대기열에서 중복 메시지를 방지합니다.
- `MessageGroupId` - 고유한 메시지 그룹 내에서 메시지 순서를 관리합니다.

주제

- Amazon SQS에서 일관되지 않은 메시지 처리 방지
- Amazon SQS에서 메시지 중복 제거 ID 사용
- Amazon SQS FIFO 대기열에서 메시지 그룹 ID 사용
- Amazon SQS 수신 요청 시도 ID 사용

Amazon SQS에서 일관되지 않은 메시지 처리 방지

Amazon SQS는 분산 시스템이므로 메시지가 `ReceiveMessage` API 메서드 호출에서 성공적으로 반환되고 Amazon SQS에서 배달된 것으로 표시되더라도 소비자가 해당 메시지를 받지 못할 수 있습니다. 이 경우 Amazon SQS는 소비자가 수신한 적이 없는 메시지를 한 번 이상 배달된 것으로 기록합니다. 이 상태에서는 메시지 배달이 추가로 시도되지 않으므로 배달 못한 편지 대기열의 최대 수신 수를 1로 설정하지 않는 것이 좋습니다.

Amazon SQS에서 메시지 중복 제거 ID 사용

`MessageDeduplicationId`는 중복 메시지 전송을 방지하기 위해 Amazon SQS FIFO 대기열에서만 사용되는 토큰입니다. 이를 통해 5분 중복 제거 기간 내에 중복 제거 ID가 동일한 메시지 인스턴스 하나만 처리되고 전달됩니다.

Amazon SQS가 특정 중복 제거 ID가 있는 메시지를 이미 수락한 경우 ID가 동일한 후속 메시지는 확인되지만 소비자에게 전달되지 않습니다.

> **Note**
>
> Amazon SQS는 메시지를 수신하고 삭제한 후에도 중복 제거 ID를 계속 추적합니다.

주제

- Amazon SQS에서 메시지 중복 제거 ID를 제공해야 하는 경우
- Amazon SQS에서 단일 생산자/소비자 시스템에 대한 중복 제거 활성화
- Amazon SQS의 중단 복구 시나리오
- Amazon SQS에서 가시성 제한 시간 구성

Amazon SQS에서 메시지 중복 제거 ID를 제공해야 하는 경우

생산자는 다음 시나리오에서 메시지 중복 제거 ID를 지정해야 합니다.

- 동일한 메시지 본문을 전송할 때 이를 고유으로 처리해야 합니다.
- 콘텐츠는 동일하지만 메시지 속성은 다른 메시지를 전송할 때 각 메시지가 별도로 처리되도록 합니다.
- 콘텐츠가 다르지만(예: 메시지 본문의 재시도 카운터) Amazon SQS가 중복으로 인식하도록 요구하는 메시지를 전송하는 경우.

Amazon SQS에서 단일 생산자/소비자 시스템에 대한 중복 제거 활성화

단일 생산자와 단일 소비자가 있고 본문에 애플리케이션별 메시지 ID가 포함되어 있기 때문에 메시지가 고유한 경우 다음 모범 사례를 따르세요.

- 대기열에서 콘텐츠 기반 중복 제거 기능을 활성화합니다(각 메시지에는 고유한 본문이 있음). 생성자는 메시지 중복 제거 ID를 생략할 수 있습니다.
- Amazon SQS FIFO 대기열에 대해 콘텐츠 기반 중복 제거가 사용 설정되고 메시지가 중복 제거 ID와 함께 전송되면 `SendMessage` 중복 제거 ID가 생성된 콘텐츠 기반 중복 제거 ID를 재정의합니다.
- 소비자가 각 요청별 수신 요청 시도 ID를 제공하지 않아도 되지만, 이렇게 하면 실패-재시도 시퀀스가 빠르게 실행되기 때문에 이는 권장되는 모범 사례입니다.
- 전송 또는 수신 요청은 FIFO 대기열에서 메시지 정렬을 방해하지 않기 때문에 이 요청을 다시 시도할 수 있습니다.

Amazon SQS의 중단 복구 시나리오

FIFO 대기열의 중복 제거 프로세스는 시간에 민감합니다. 애플리케이션을 설계할 때 생산자와 소비자 모두 중복이나 처리 실패 없이 클라이언트 또는 네트워크 중단으로부터 복구할 수 있는지 확인합니다.

생산자 고려 사항

- Amazon SQS는 5분의 중복 제거 기간을 적용합니다.
- 생산자가 5분 후에 `SendMessage` 요청을 재시도하면 Amazon SQS는 요청을 새 메시지로 취급하여 잠재적으로 중복을 생성합니다.

소비자 고려 사항

- 소비자가 표시 제한 시간이 만료되기 전에 메시지를 처리하지 못하면 다른 소비자가 메시지를 수신하고 처리하여 중복 처리로 이어질 수 있습니다.
- 애플리케이션의 처리 시간에 따라 표시 제한 시간을 조정합니다.
- `ChangeMessageVisibility` API를 사용하여 메시지가 처리 중인 동안 제한 시간을 연장합니다.
- 메시지가 반복적으로 처리되지 않는 경우 무기한 재처리되는 대신 DLQ(Dead Letter Queue)로 라우팅합니다.

- 생산자는 대기열의 중복 제거 간격을 알고 있어야 합니다. Amazon SQS의 중복 제거 간격은 5분입니다. 중복 제거 간격이 만료된 이후에 `SendMessage` 요청을 재시도하면 대기열에 중복 메시지가 생성될 수 있습니다. 예를 들어, 자동차 안의 모바일 디바이스에서 순서가 중요한 메시지를 전송할 수 있습니다. 승인을 받기 전에 일정 기간 동안 자동차의 셀룰러 연결이 끊길 경우, 셀룰러가 다시 연결되고 나서 요청을 다시 시도하면 중복이 생성될 수 있습니다.
- 제한 시간 초과가 만료되기 전에 메시지를 처리하지 못할 위험을 최소화해 주는 제한 시간 초과를 소비자에게 적용해야 합니다. 메시지를 처리하는 동안 제한 시간 초과를 연장하려면 `ChangeMessageVisibility` 작업을 호출합니다. 그렇지만, 제한 시간 초과가 만료된 경우 다른 소비자가 메시지를 바로 처리할 수 있고, 이렇게 되면 한 메시지를 여러 번 처리하게 됩니다. 이 시나리오를 방지하려면 배달 못한 편지 대기열을 구성합니다.

Amazon SQS에서 가시성 제한 시간 구성

안정적인 메시지 처리를 보장하려면 표시 제한 시간을 AWS SDK 읽기 제한 시간보다 길게 설정합니다. 이는 짧은 폴링과 긴 폴링 모두에서 `ReceiveMessage` API를 사용할 때 적용됩니다. 제한 시간 초과가 길어지면 원래 요청이 완료되기 전에 다른 소비자가 메시지를 사용할 수 없어 중복 처리 위험이 줄어듭니다.

Amazon SQS FIFO 대기열에서 메시지 그룹 ID 사용

FIFO(First-In-First-Out) 대기열에서 `MessageGroupId`는 메시지를 개별 그룹으로 구성하는 속성입니다. 동일한 메시지 그룹 내의 메시지는 항상 한 번에 하나씩 엄격한 순서로 처리되므로 동일한 그룹의 두 메시지가 동시에 처리되지 않습니다. 표준 대기열에서를 사용하면 공정 대기열이 `MessageGroupId` 활성화됩니다. 엄격한 주문이 필요한 경우 FIFO 대기열을 사용합니다.

주제

- Amazon SQS에서 여러 개의 정렬된 메시지 그룹 인터리브
- Amazon SQS의 다중 생산자/소비자 시스템에서 중복 처리 방지

- Amazon SQS에서 동일한 메시지 그룹 ID를 가진 큰 메시지 백로그 방지
- Amazon SQS의 가상 대기열에서 동일한 메시지 그룹 ID 재사용 방지

Amazon SQS에서 여러 개의 정렬된 메시지 그룹 인터리브

단일 FIFO 대기열 내에서 정렬된 여러 메시지 그룹을 인터리브하려면 각 그룹에를 할당`MessageGroupId`합니다(예: 다른 사용자의 세션 데이터). 이렇게 하면 여러 소비자가 대기열에서 동시에 읽는 동시에 동일한 그룹 내의 메시지가 순서대로 처리되도록 할 수 있습니다.

특정가 포함된 메시지가 처리 `MessageGroupId` 중이고 보이지 않는 경우 표시 제한 시간이 만료되거나 메시지가 삭제될 때까지 다른 소비자는 동일한 그룹의 메시지를 처리할 수 없습니다.

Amazon SQS의 다중 생산자/소비자 시스템에서 중복 처리 방지

메시지 순서 지정이 우선순위가 아닌 처리량이 높고 지연 시간이 짧은 시스템에서 생산자는 각 메시지`MessageGroupId`에 고유한를 할당할 수 있습니다. 이렇게 하면 다중 생산자/다중 소비자 설정에서도 Amazon SQS FIFO 대기열이 중복을 제거할 수 있습니다. 이 접근 방식은 중복 메시지를 방지하지만 각 메시지는 자체 독립 그룹으로 취급되므로 메시지 순서를 보장하지 않습니다.

생산자와 소비자가 여러 개인 시스템에서는 항상 중복 전달 위험이 있습니다. 소비자가 표시 제한 시간이 만료되기 전에 메시지를 처리하지 못하면 Amazon SQS는 메시지를 다시 사용할 수 있도록 하여 잠재적으로 다른 소비자가 메시지를 픽업할 수 있도록 합니다. 이를 완화하려면 처리 시간에 따라 메시지 확인 및 표시 제한 시간 설정이 적절한지 확인합니다.

Amazon SQS에서 동일한 메시지 그룹 ID를 가진 큰 메시지 백로그 방지

FIFO 대기열은 최대 120,000개의 진행 중인 메시지(소비자가 수신했지만 아직 삭제되지 않은 메시지)를 지원합니다. 이 제한에 도달하면 Amazon SQS는 오류를 반환하지 않지만 처리에 영향을 미칠 수 있습니다. AWS Support에 문의하여이 한도를 초과하는 증가를 요청할 수 있습니다.

FIFO 대기열은 처음 120,000개의 메시지를 스캔하여 사용 가능한 메시지 그룹을 결정합니다. 대규모 백로그가 단일 메시지 그룹에 빌드되면 나중에 전송된 다른 그룹의 메시지는 백로그가 처리될 때까지 차단된 상태로 유지됩니다.

> **Note**
>
> 소비자가 반복적으로 메시지를 처리하지 못하면 메시지 백로그가 발생할 수 있습니다. 이는 메시지 콘텐츠 문제 또는 소비자 측 장애 때문일 수 있습니다. 메시지 처리 지연을 방지하려면 여러 번 실패한 후 처리되지 않은 메시지를 이동하도록 배달 못한 편지 대기열을 구성합니다. 이

렇게 하면 동일한 메시지 그룹의 다른 메시지를 처리할 수 있으므로 시스템 병목 현상을 방지할 수 있습니다.

Amazon SQS의 가상 대기열에서 동일한 메시지 그룹 ID 재사용 방지

공유 호스트 대기열과 함께 가상 대기열을 사용하는 경우 서로 다른 가상 대기열`MessageGroupId`에서 동일한를 재사용하지 마세요. 여러 가상 대기열이 동일한 호스트 대기열을 공유하고 동일한의 메시지를 포함하는 경우 `MessageGroupId`해당 메시지가 서로 차단되어 효율적인 처리를 방해할 수 있습니다. 원활한 메시지 처리를 위해 서로 다른 가상 대기열의 메시지에 고유한 `MessageGroupId` 값을 할당합니다.

Amazon SQS 수신 요청 시도 ID 사용

수신 요청 시도 ID는 Amazon SQS에서 `ReceiveMessage` 호출을 중복 제거하는 데 사용되는 고유한 토큰입니다. 애플리케이션과 Amazon SQS 간의 네트워크 중단 또는 연결 문제가 발생하는 경우 다음을 수행하는 것이 가장 좋습니다.

- `ReceiveMessage` 호출할 때 수신 요청 시도 ID를 제공합니다.
- 작업이 실패할 경우 동일한 수신 요청 시도 ID를 사용하여 재시도합니다.

Amazon SQS 메시지 처리 및 타이밍

이 주제에서는 메시지를 적시에 처리하기 위한 전략, 최상의 폴링 모드 선택, 성능 향상을 위한 긴 폴링 구성 등 Amazon SQS에서 메시지 처리의 속도와 효율성을 최적화하는 포괄적인 지침을 제공합니다.

주제

- Amazon SQS에서 적시에 메시지 처리
- Amazon SQS에서 긴 폴링 설정
- Amazon SQS에서 적절한 폴링 모드 사용

Amazon SQS에서 적시에 메시지 처리

제한 시간 초과 설정은 애플리케이션에서 메시지를 처리하고 삭제하는 데 얼마나 걸리느냐에 따라 다릅니다. 예를 들어 애플리케이션에서 메시지를 처리하려면 10초가 필요한데 제한 시간 초과 값을 15분으로 설정한다면, 이전의 메시지 처리 시도가 실패할 경우 다시 메시지 처리를 시도하기 위해 비교적

오랜 시간을 기다려야 합니다. 또한 애플리케이션에서 메시지를 처리하려면 10초가 필요한데 제한 시간 초과 값을 불과 2초로 설정한다면, 원래 소비자가 아직 메시지를 처리하는 동안 다른 소비자로부터 중복 메시지를 받을 수 있습니다.

메시지 처리 시간을 충분히 확보하려면 다음 전략 중 하나를 사용하세요.

- 메시지 처리 소요 시간을 알고 있거나 합리적으로 추정할 수 있는 경우, 메시지의 제한 시간 초과를 메시지 처리와 삭제에 소요되는 최대 시간으로 연장합니다. 자세한 내용은 표시 제한 시간 구성을 참조하세요.
- 메시지를 처리하는 데 걸리는 시간을 모르는 경우 소비자 프로세스에 대한 하트비트를 생성합니다. 초기 표시 제한 시간(예: 2분)을 지정한 다음 소비자가 메시지에서 계속 작업하는 경우 표시 제한 시간을 1분마다 2분씩 연장합니다.

> **Important**
>
> 최대 표시 제한 시간은 Amazon SQS가 `ReceiveMessage` 요청을 수신한 시간으로부터 12시간입니다. 표시 제한 시간을 연장해도 최대 12시간이 재설정되지는 않습니다.
> 또한 `ReceiveMessage` 요청으로 타이머가 시작되기 때문에 개별 메시지의 제한 시간을 전체 12시간(예: 43,200초)으로 설정하지 못할 수도 있습니다. 예를 들어 메시지를 수신한 후 `VisibilityTimeout`이 43,200초인 `ChangeMessageVisibility` 호출을 전송하여 즉시 최대 12시간을 설정하면 실패할 가능성이 높습니다. 하지만 43,195초 값을 사용하면 `ReceiveMessage`를 통해 메시지를 요청하는 것과 표시 제한 시간을 업데이트하는 것 사이에 상당한 지연이 있는 경우가 아니면 작동합니다. 소비자에게 12시간 이상 필요한 경우 Step Functions 사용을 고려하세요.

Amazon SQS에서 긴 폴링 설정

`ReceiveMessage` API 작업에 대한 대기 시간이 0보다 큰 경우 긴 폴링이 유효합니다. 긴 폴링 대기 시간의 최대값은 20초입니다. 긴 폴링은 빈 응답 수를 줄이고(`ReceiveMessage` 요청에 사용 가능한 메시지가 없는 경우) 거짓의 빈 응답을 제거하여(메시지를 사용할 수 있지만 응답에 포함되지 않은 경우) Amazon SQS 사용 비용을 줄여 줍니다. 자세한 내용은 Amazon SQS 짧은 폴링 및 긴 폴링 단원을 참조하십시오.

메시지 처리를 최적화하려면 다음 전략을 사용하세요.

- 대부분의 경우 `ReceiveMessage` 대기 시간을 20초로 설정할 수 있습니다. 애플리케이션에 20초가 너무 긴 경우 `ReceiveMessage` 대기 시간을 더 짧게 설정할 수 있습니다(최소값은 1초). AWS SDK

를 사용하여 Amazon SQS에 액세스하지 않거나 대기 시간이 더 짧도록 AWS SDK를 구성하는 경우 더 긴 요청을 허용하거나 긴 폴링에 더 짧은 대기 시간을 사용하도록 Amazon SQS 클라이언트를 수정해야 할 수 있습니다.

- 여러 대기열에 대해 긴 폴링을 구현할 경우에는 모든 대기열에 단일 스레드를 사용하지 말고 각 대기열마다 하나의 스레드를 사용하는 것이 좋습니다. 각 대기열마다 하나의 스레드를 사용하면 애플리케이션이 각 대기열에서 메시지를 사용 가능한 즉시 처리할 수 있지만, 하나의 스레드를 사용하여 여러 대기열을 폴링하면 애플리케이션이 사용 가능한 메시지가 없는 대기열을 대기하느라(최대 20초) 다른 대기열에서 사용 가능한 메시지를 처리하지 못하게 될 수 있습니다.

> **Important**
>
> HTTP 오류를 방지하려면 `ReceiveMessage` 요청에 대한 HTTP 응답 제한 시간이 `WaitTimeSeconds` 파라미터보다 긴지 확인합니다. 자세한 내용은 ReceiveMessage를 참조하십시오.

Amazon SQS에서 적절한 폴링 모드 사용

- 긴 폴링은 Amazon SQS 대기열의 메시지가 사용 가능 상태가 되자마자 사용할 수 있게 해줍니다.
  - Amazon SQS 사용 비용을 절감하고 빈 대기열로의 빈 수신(반환되는 메시지가 없는 `ReceiveMessage` 작업의 응답) 횟수를 줄이려면 긴 폴링을 활성화합니다. 자세한 내용은 Amazon SQS 긴 폴링을 참조하세요.
  - 여러 번의 수신으로 여러 스레드를 폴링할 때 효율성을 높이려면, 스레드 수를 줄입니다.
  - 대다수의 경우에 긴 폴링이 짧은 폴링보다 선호됩니다.
- 폴링된 Amazon SQS 대기열이 비어 있더라도 짧은 폴링은 즉시 응답을 반환합니다.
  - `ReceiveMessage` 요청에 대한 즉각적인 응답이 예상되는 애플리케이션의 요구 사항을 충족하려면, 짧은 폴링을 사용합니다.
  - 긴 폴링과 마찬가지로 짧은 폴링도 요금이 청구됩니다.

Amazon SQS Java SDK 예제

를 AWS SDK for Java 사용하면 Amazon SQS 및 기타와 상호 작용하는 Java 애플리케이션을 구축할 수 있습니다 AWS 서비스.

- SDK를 설치하고 설정하려면 AWS SDK for Java 2.x 개발자 안내서의 시작하기를 참조하세요.
- 대기열 생성 또는 메시지 전송과 같은 기본 대기열 작업은 AWS SDK for Java 2.x 개발자 안내서의 Amazon SQS 메시지 대기열 작업을 참조하세요.
- 이 가이드에는 다음과 같은 추가 Amazon SQS 기능의 예도 포함되어 있습니다.
  - Amazon SQS 대기열에서 서버 측 암호화 사용
  - Amazon SQS 대기열에 대한 태그 구성
  - Amazon SQS 대기열로 메시지 속성 보내기

Amazon SQS 대기열에서 서버 측 암호화 사용

AWS SDK for Java 를 사용하여 Amazon SQS 대기열에 서버 측 암호화(SSE)를 추가합니다. 각 대기열은 AWS Key Management Service (AWS KMS) KMS 키를 사용하여 데이터 암호화 키를 생성합니다. 이 예시에서는 Amazon SQS의 AWS 관리형 KMS 키를 사용합니다.

SSE 사용 및 KMS 키의 역할에 대한 자세한 정보는 Amazon SQS에서 저장 시 암호화 섹션을 참조하세요.

기존 대기열에 SSE 추가

기존 대기열에 대해 서버 측 암호화를 활성화하려면 `SetQueueAttributes` 메서드를 사용하여 `KmsMasterKeyId` 속성을 설정합니다.

다음 코드 예제에서는를 Amazon SQS의 AWS 관리형 KMS 키 AWS KMS key 로 설정합니다. 또한 이 예제에서는 AWS KMS key 재사용 기간을 140초로 설정합니다.

예제 코드를 실행하기 전에 자격 AWS 증명을 설정했는지 확인합니다. 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 개발을 위한 AWS 자격 증명 및 리전 설정을 참조하세요.

```
    public static void addEncryption(String queueName, String kmsMasterKeyAlias) {
        SqsClient sqsClient = SqsClient.create();
```

```
        GetQueueUrlRequest urlRequest = GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build();

        GetQueueUrlResponse getQueueUrlResponse;
        try {
            getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest);
        } catch (QueueDoesNotExistException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        }
        String queueUrl = getQueueUrlResponse.queueUrl();


        Map<QueueAttributeName, String> attributes = Map.of(
                QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias,
                QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140" // Set the
data key reuse period to 140 seconds.
        );                                                                  // This is
how long SQS can reuse the data key before requesting a new one from KMS.

        SetQueueAttributesRequest attRequest = SetQueueAttributesRequest.builder()
                .queueUrl(queueUrl)
                .attributes(attributes)
                .build();
        try {
            sqsClient.setQueueAttributes(attRequest);
            LOGGER.info("The attributes have been applied to {}", queueName);
        } catch (InvalidAttributeNameException | InvalidAttributeValueException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        } finally {
            sqsClient.close();
        }
    }
```

대기열에 SSE 비활성화

기존 대기열의 서버 측 암호화를 비활성화하려면 `SetQueueAttributes` 메서드를 사용하여 `KmsMasterKeyId` 속성을 빈 문자열로 설정하세요.

> **Important**
>
> `null`은 유효한 `KmsMasterKeyId` 값이 아닙니다.

SSE를 사용하여 대기열 생성

대기열을 생성할 때 SSE를 활성화하려면 `CreateQueue` API 메서드에 `KmsMasterKeyId` 속성을 추가합니다.

다음 예제에서는 SSE가 활성화된 새 대기열을 만듭니다. 대기열은 Amazon SQS에 AWS 관리형 KMS 키를 사용합니다. 또한 이 예제에서는 AWS KMS key 재사용 기간을 160초로 설정합니다.

예제 코드를 실행하기 전에 자격 AWS 증명을 설정했는지 확인합니다. 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 개발을 위한 AWS 자격 증명 및 리전 설정을 참조하세요.

```
// Create an SqsClient for the specified Region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Create a hashmap for the attributes. Add the key alias and reuse period to the
 hashmap.
HashMap<QueueAttributeName, String> attributes = new HashMap<QueueAttributeName,
 String>();
final String kmsMasterKeyAlias = "alias/aws/sqs";  // the alias of the AWS managed KMS
 key for Amazon SQS.
attributes.put(QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias);
attributes.put(QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140");

// Add the attributes to the CreateQueueRequest.
CreateQueueRequest createQueueRequest =
                CreateQueueRequest.builder()
                        .queueName(queueName)
                        .attributes(attributes)
                        .build();
sqsClient.createQueue(createQueueRequest);
```

SSE 속성 검색

대기열 속성 검색에 대한 자세한 내용은 Amazon Simple Queue Service API 참조의 예를 참조합니다.

특정 대기열의 KMS 키 ID 또는 데이터 키 재사용 기간을 검색하려면 `GetQueueAttributes` 메서드를 실행하고 `KmsMasterKeyId` 및 `KmsDataKeyReusePeriodSeconds` 값을 검색합니다.

Amazon SQS 대기열에 대한 태그 구성

비용 할당 태그를 사용하여 Amazon SQS 대기열을 구성하고 식별할 수 있습니다. 다음 예제에서는 AWS SDK for Java를 사용하여 태그를 구성하는 방법을 보여줍니다. 자세한 내용은 Amazon SQS 비용 할당 태그 단원을 참조하십시오.

예제 코드를 실행하기 전에 자격 AWS 증명을 설정했는지 확인합니다. 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 개발을 위한 AWS 자격 증명 및 리전 설정을 참조하세요.

태그 나열

대기열의 태그를 나열하려면 `ListQueueTags` 메서드를 사용합니다.

```
// Create an SqsClient for the specified region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Get the queue URL.
String queueName = "MyStandardQ1";
GetQueueUrlResponse getQueueUrlResponse =

 sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
String queueUrl = getQueueUrlResponse.queueUrl();

// Create the ListQueueTagsRequest.
final ListQueueTagsRequest listQueueTagsRequest =

 ListQueueTagsRequest.builder().queueUrl(queueUrl).build();

// Retrieve the list of queue tags and print them.
final ListQueueTagsResponse listQueueTagsResponse =
                                  sqsClient.listQueueTags(listQueueTagsRequest);
System.out.println(String.format("ListQueueTags: \tTags for queue %s are %s.\n",
                queueName, listQueueTagsResponse.tags() ));
```

태그 추가 또는 업데이트

대기열에 태그 값을 추가하거나 업데이트하려면 `TagQueue` 메서드를 사용합니다.

```
 // Create an SqsClient for the specified Region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Get the queue URL.
String queueName = "MyStandardQ1";
GetQueueUrlResponse getQueueUrlResponse =

 sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
String queueUrl = getQueueUrlResponse.queueUrl();

// Build a hashmap of the tags.
final HashMap<String, String> addedTags = new HashMap<>();
        addedTags.put("Team", "Development");
        addedTags.put("Priority", "Beta");
        addedTags.put("Accounting ID", "456def");

//Create the TagQueueRequest and add them to the queue.
final TagQueueRequest tagQueueRequest = TagQueueRequest.builder()
        .queueUrl(queueUrl)
        .tags(addedTags)
        .build();
sqsClient.tagQueue(tagQueueRequest);
```

태그 제거

대기열에서 하나 이상의 태그를 삭제하려면 `UntagQueue` 메서드를 사용합니다. 다음 예제에서는 `Accounting ID` 태그를 제거합니다.

```
// Create the UntagQueueRequest.
final UntagQueueRequest untagQueueRequest = UntagQueueRequest.builder()
        .queueUrl(queueUrl)
        .tagKeys("Accounting ID")
        .build();

// Remove the tag from this queue.
sqsClient.untagQueue(untagQueueRequest);
```

Amazon SQS 대기열로 메시지 속성 보내기

메시지 속성을 사용하여 메시지에 구조화된 메타데이터(예: 타임스탬프, 지리 공간 데이터, 서명 및 식별자)를 포함할 수 있습니다. 자세한 내용은 Amazon SQS 메시지 속성 단원을 참조하십시오.

예제 코드를 실행하기 전에 자격 AWS 증명을 설정했는지 확인합니다. 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서의 개발을 위한 AWS 자격 증명 및 리전 설정을 참조하세요.

속성 정의

메시지에 대한 속성을 정의하려면 `MessageAttributeValue` 데이터 유형을 사용하는 다음 코드를 추가합니다. 자세한 내용은 메시지 속성 구성 요소 및 메시지 속성 데이터 형식 섹션을 참조하세요.

는 메시지 본문 및 메시지 속성 체크섬을 AWS SDK for Java 자동으로 계산하고 Amazon SQS가 반환하는 데이터와 비교합니다. 자세한 내용은 AWS SDK for Java 2.x 개발자 안내서 및 다른 프로그래밍 언어의 메시지 속성의 MD5 메시지 다이제스트 계산을 참조하세요.

String

이 예제에서는 값 `Jane`을 사용하여 `Name`이라는 `String` 속성을 정의합니다.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("Name", new MessageAttributeValue()
.withDataType("String")
.withStringValue("Jane"));
```

Number

이 예제에서는 값 `230.00000000000000001`을 사용하여 `AccurateWeight`이라는 `Number` 속성을 정의합니다.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccurateWeight", new MessageAttributeValue()
.withDataType("Number")
.withStringValue("230.00000000000000001"));
```

Binary

이 예제에서는 초기화되지 않은 10바이트 어레이의 값을 사용하여 `ByteArray`라는 `Binary` 속성을 정의합니다.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("ByteArray", new MessageAttributeValue()
.withDataType("Binary")
.withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

String (custom)

이 예제에서는 값 `ABC123456`을 사용하여 `EmployeeId`라는 사용자 지정 속성 `String.EmployeeId`를 정의합니다.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("EmployeeId", new MessageAttributeValue()
.withDataType("String.EmployeeId")
.withStringValue("ABC123456"));
```

Number (custom)

이 예제에서는 값 `000123456`을 사용하여 `AccountId`라는 사용자 지정 속성 `Number.AccountId`를 정의합니다.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccountId", new MessageAttributeValue()
.withDataType("Number.AccountId")
.withStringValue("000123456"));
```

> **Note**
>
> 기본 데이터 입력이 `Number`이므로 `ReceiveMessage` 메서드가 `123456`을 반환합니다.

Binary (custom)

이 예제에서는 초기화되지 않은 10바이트 어레이의 값을 사용하여 `ApplicationIcon`이라는 사용자 지정 속성 `Binary.JPEG`를 정의합니다.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("ApplicationIcon", new MessageAttributeValue()
.withDataType("Binary.JPEG")
.withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

속성을 포함하는 메시지 전송

이 예에서는 메시지를 보내기 전에 SendMessageRequest에 속성을 추가합니다.

```
// Send a message with an attribute.
final SendMessageRequest sendMessageRequest = new SendMessageRequest();
sendMessageRequest.withMessageBody("This is my message text.");
sendMessageRequest.withQueueUrl(myQueueUrl);
sendMessageRequest.withMessageAttributes(messageAttributes);
sqs.sendMessage(sendMessageRequest);
```

> **Important**
>
> 선입선출(FIFO) 대기열로 메시지를 보내는 경우 메시지 그룹 ID를 제공한 후 sendMessage 메서드가 실행되는지 확인하세요.
>
> SendMessage 대신 SendMessageBatch 메서드를 사용하는 경우 배치의 개별 메시지에 대해 메시지 속성을 지정해야 합니다.

Amazon SQS에서 API 사용

이 주제에서는 Amazon SQS 엔드포인트 구성, GET 및 POST 메서드를 사용한 쿼리 API 요청, 배치 API 작업 사용에 대한 정보를 제공합니다. 파라미터, 오류, 예제, 데이터 유형 등 Amazon SQS 작업에 대한 자세한 내용은 Amazon Simple Queue Service API 참조를 참조하세요.

다양한 프로그래밍 언어를 사용하여 Amazon SQS에 액세스하려는 경우 다음과 같은 자동 기능을 제공하는 AWS SDK를 사용할 수도 있습니다.

- 서비스 요청에 대한 암호화 서명
- 요청 재시도
- 오류 응답 처리

자세한 내용은 the section called “ AWS SDKs 작업” 단원을 참조하십시오.

명령줄 도구 정보는 AWS CLI 명령 참조 및 AWS Tools for PowerShell Cmdlet 참조의 Amazon SQS 섹션을 참조하세요.

AWS JSON 프로토콜을 사용하는 Amazon SQS APIs

Amazon SQS는 지정된 AWS SDK 버전의 모든 Amazon SQS APIs에 대한 전송 메커니즘으로 AWS JSON 프로토콜을 사용합니다. AWS JSON 프로토콜은 더 높은 처리량, 더 짧은 지연 시간 및 더 빠른 application-to-application 통신을 제공합니다. AWS JSON 프로토콜은 AWS 쿼리 프로토콜과 비교할 때 요청 및 응답의 직렬화/역직렬화에 더 효율적입니다. SQS APIs에서 AWS 쿼리 프로토콜을 계속 사용하려는 경우 Amazon SQS AWS 쿼리 프로토콜을 지원하는 AWS SDK 버전Amazon SQS API에서 사용되는 AWS JSON 프로토콜에는 어떤 언어가 지원되나요?은 섹션을 참조하세요.

Amazon SQS는 AWS JSON 프로토콜을 사용하여 AWS SDK 클라이언트(예: Java, Python, Golang, JavaScript)와 Amazon SQS 서버 간에 통신합니다. Amazon SQS API 작업의 HTTP 요청은 JSON 형식의 입력을 수락합니다. Amazon SQS 작업이 실행되고 실행 응답이 JSON 형식으로 SDK 클라이언트에 다시 전송됩니다. AWS 쿼리에 비해 AWS JSON은 클라이언트와 서버 간에 데이터를 전송하는 데 더 간단하고 빠르며 효율적입니다.

- AWS JSON 프로토콜은 Amazon SQS 클라이언트와 서버 간의 매개자 역할을 합니다.
- 서버는 Amazon SQS 작업이 생성되는 프로그래밍 언어를 이해하지 못하지만 AWS JSON 프로토콜을 이해합니다.
- AWS JSON 프로토콜은 Amazon SQS 클라이언트와 서버 간의 직렬화(객체를 JSON 형식으로 변환) 및 역직렬화(JSON 형식을 객체로 변환)를 사용합니다.

Amazon SQS를 사용하는 AWS JSON 프로토콜에 대한 자세한 내용은 섹션을 참조하세요Amazon SQS AWS JSON 프로토콜 FAQs.

AWS JSON 프로토콜은 지정된 AWS SDK 버전에서 사용할 수 있습니다. 언어 변형에 따른 SDK 버전 및 릴리스 날짜를 검토하려면AWS SDK 및 도구 참조 안내서의 AWS SDK 및 도구 버전 지원 매트릭스를 참조하세요.

Amazon SQS에서 AWS JSON 프로토콜을 사용하여 쿼리 API 요청 수행

이 주제에서는 Amazon SQS 엔드포인트를 구성하고, POST 요청을 하고, 응답을 해석하는 방법을 설명합니다.

> **Note**
>
> AWS JSON 프로토콜은 대부분의 언어 변형에서 지원됩니다. 지원되는 언어 변형의 전체 목록은 Amazon SQS API에서 사용되는 AWS JSON 프로토콜에는 어떤 언어가 지원되나요? 섹션을 참조하세요.

엔드포인트 구성

Amazon SQS 대기열로 작업하려면 엔드포인트를 생성해야 합니다. Amazon SQS 엔드포인트에 대한 자세한 내용은 Amazon Web Services 일반 참조의 다음 페이지를 참조하세요.

- 리전 엔드포인트
- Amazon Simple Queue Service 엔드포인트 및 할당량

모든 Amazon SQS 엔드포인트는 독립적입니다. 예를 들어 두 대기열의 이름이 MyQueue인데 한 대기열에는 `sqs.us-east-2.amazonaws.com` 엔드포인트가 있고 다른 대기열에는 `sqs.eu-west-2.amazonaws.com` 엔드포인트가 있는 경우 두 대기열은 서로 어떤 데이터도 공유하지 않습니다.

다음은 대기열을 생성하는 요청을 만드는 엔드포인트의 예제입니다.

```
POST / HTTP/1.1
Host: sqs.us-west-2.amazonaws.com
```

```
X-Amz-Target: AmazonSQS.CreateQueue
X-Amz-Date: <Date>
Content-Type: application/x-amz-json-1.0
Authorization: <AuthParams>
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
{
    "QueueName":"MyQueue",
    "Attributes": {
        "VisibilityTimeout": "40"
    },
    "tags": {
        "QueueType": "Production"
    }
}
```

> **Note**
>
> 대기열 이름과 대기열 URL은 대소문자를 구분합니다.
>
> *AUTHPARAMS*의 구조는 API 요청의 서명에 따라 달라집니다. 자세한 내용은 Amazon Web Services 일반 참조의 AWS API 요청 서명을 참조하세요.

POST 요청 만들기

Amazon SQS POST 요청은 쿼리 파라미터를 HTTP 요청 본문에 포함시켜 보냅니다.

다음은 `X-Amz-Target`이 `AmazonSQS.<operationName>`으로 설정된 HTTP 헤더와 `Content-Type`이 `application/x-amz-json-1.0`으로 설정된 HTTP 헤더의 예입니다.

```
POST / HTTP/1.1
Host: sqs.<region>.<domain>
X-Amz-Target: AmazonSQS.SendMessage
X-Amz-Date: <Date>
Content-Type: application/x-amz-json-1.0
Authorization: <AuthParams>
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
{
    "QueueUrl": "https://sqs.<region>.<domain>/<awsAccountId>/<queueName>/",
    "MessageBody": "This is a test message"
```

```
}
```

이 HTTP POST 요청은 Amazon SQS 대기열로 메시지를 전송합니다.

> **Note**
>
> HTTP 헤더 `X-Amz-Target` 및 `Content-Type` 모두 필수입니다.
> HTTP 클라이언트는 클라이언트의 HTTP 버전에 따라 HTTP 요청에 다른 항목을 추가할 수 있습니다.

아마존 SQS JSON API 응답 해석

Amazon SQS에 요청을 보내면 결과와 함께 JSON 응답을 반환합니다. 응답 구조는 사용한 API 작업에 따라 달라집니다.

이러한 응답의 세부 정보를 이해하려면 다음을 참조하세요.

- Amazon Simple Queue Service API 참조의 API 작업에서 특정 API 작업
- Amazon SQS AWS JSON 프로토콜 FAQs은

성공적인 JSON 응답 구조

요청이 성공하면 기본 응답 요소는 요청의 범용 고유 식별자(UUID)와 기타 추가된 응답 필드를 포함하는 `x-amzn-RequestId`입니다. 예를 들어 다음 `CreateQueue` 응답에는 생성된 대기열의 URL이 포함된 `QueueUrl` 필드가 포함되어 있습니다.

```
HTTP/1.1 200 OK
x-amzn-RequestId: <requestId>
Content-Length: <PayloadSizeBytes>
Date: <Date>
Content-Type: application/x-amz-json-1.0
{
    "QueueUrl":"https://sqs.us-east-1.amazonaws.com/111122223333/MyQueue"
}
```

JSON 오류 응답 구조

요청이 실패하면 Amazon SQS는 HTTP 헤더와 본문을 포함한 기본 응답을 반환합니다.

HTTP 헤더에서 `x-amzn-RequestId`에는 요청의 UUID가 포함됩니다. `x-amzn-query-error`에는 오류 유형, 오류가 생산자 오류인지 소비자 오류인지 여부라는 두 가지 정보가 포함되어 있습니다.

응답 본문에서 `"__type"`은 기타 오류 세부 정보를 나타내고, `Message`는 오류 조건을 읽을 수 있는 형식으로 나타냅니다.

다음은 JSON 형식의 오류 응답 예제입니다.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: 66916324-67ca-54bb-a410-3f567a7a0571
x-amzn-query-error: AWS.SimpleQueueService.NonExistentQueue;Sender
Content-Length: <PayloadSizeBytes>
Date: <Date>
Content-Type: application/x-amz-json-1.0
{
    "__type": "com.amazonaws.sqs#QueueDoesNotExist",
    "message": "The specified queue does not exist."
}
```

Amazon SQS AWS JSON 프로토콜 FAQs

이 주제에서는 Amazon SQS에서 AWS JSON 프로토콜을 사용하는 방법에 대해 자주 묻는 질문을 다룹니다.

AWS JSON 프로토콜이란 무엇이며 기존 Amazon SQS API 요청 및 응답과 어떻게 다릅니까?

JSON은 이기종 시스템 간 통신에 가장 널리 사용되고 수용되는 연결 방법 중 하나입니다. Amazon SQS는 JSON을 매체로 사용하여 AWS SDK 클라이언트(예: Java, Python, Golang, JavaScript)와 Amazon SQS 서버 간에 통신합니다. Amazon SQS API 작업의 HTTP 요청은 JSON 형식의 입력을 수락합니다. Amazon SQS 작업이 실행되고 실행 응답이 JSON 형식으로 SDK 클라이언트에 다시 공유됩니다. JSON은 AWS 쿼리에 비해 클라이언트와 서버 간에 데이터를 전송하는 데 효율적입니다.

- Amazon SQS AWS JSON 프로토콜은 Amazon SQS 클라이언트와 서버 간의 매개자 역할을 합니다.
- 서버는 Amazon SQS 작업이 생성되는 프로그래밍 언어를 이해하지 못하지만 AWS JSON 프로토콜을 이해합니다.
- Amazon SQS AWS JSON 프로토콜은 Amazon SQS 클라이언트와 서버 간의 직렬화(객체를 JSON 형식으로 변환) 및 역직렬화(JSON 형식을 객체로 변환)를 사용합니다.

Amazon SQS용 AWS JSON 프로토콜을 시작하려면 어떻게 해야 하나요? Amazon SQS

Amazon SQS에 대한 더 빠른 메시징을 위해 최신 AWS SDK 버전을 시작하려면 AWS SDK를 지정된 버전 또는 후속 버전으로 업그레이드하세요. SDK 클라이언트에 대해 자세히 알아보려면 아래 표의 안내서 열을 참조하세요.

다음은 Amazon SQS API와 함께 사용할 수 있는 AWS JSON 프로토콜의 언어 변형 간 SDK 버전 목록입니다. APIs

Language	SDK 클라이언트 리포지토리	필수 SDK 클라이언트 버전	안내서
C++	aws/aws-sdk-cpp	1.11.98	AWS SDK for C++
Golang 1.x	aws/aws-sdk-go	v1.47.7	AWS SDK for Go
Golang 2.x	aws/aws-sdk-go-v2	v1.28.0	AWS SDK for Go V2
Java 1.x	aws/aws-sdk-java	1.12.585	AWS SDK for Java
Java 2.x	aws/aws-sdk-java-v2	2.21.19	AWS SDK for Java
JavaScript v2.x	aws/aws-sdk-js	의 JavaScript AWS	
JavaScript v3.x	aws/aws-sdk-js-v3	v3.447.0	의 JavaScript AWS
.NET	aws/aws-sdk-net	3.7.681.0	AWS SDK for .NET
PHP	aws/aws-sdk-php	3.285.2	AWS SDK for PHP
Python-boto3	boto/boto3	1.28.82	AWS SDK for Python (Boto3)

Language	SDK 클라이언트 리포지토리	필수 SDK 클라이언트 버전	안내서
Python-botocore	boto/botocore	1.31.82	AWS SDK for Python (Boto3)
awscli	AWS CLI	1.29.82	AWS Command Line Interface
Ruby	aws/aws-sdk-ruby	1.67.0	AWS SDK for Ruby

Amazon SQS 워크로드에 JSON 프로토콜을 활성화하면 어떤 위험이 있나요?

AWS SDK의 사용자 지정 구현 또는 사용자 지정 클라이언트와 AWS SDK의 조합을 사용하여 AWS 쿼리 기반(XML 기반) 응답을 생성하는 Amazon SQS와 상호 작용하는 경우 AWS JSON 프로토콜과 호환되지 않을 수 있습니다. 문제가 발생하면 AWS Support에 문의하십시오.

이미 최신 AWS SDK 버전을 사용하고 있지만 오픈 소스 솔루션이 JSON을 지원하지 않는 경우 어떻게 해야 합니까?

SDK 버전을 사용 중인 것보다 이전 버전으로 변경해야 합니다. Amazon SQS용 AWS JSON 프로토콜을 시작하려면 어떻게 해야 하나요? Amazon SQS 자세한 내용은 섹션을 참조하세요.에 나열된 AWS SDK 버전Amazon SQS용 AWS JSON 프로토콜을 시작하려면 어떻게 해야 하나요? Amazon SQS은 Amazon SQS APIs용 JSON 유선 프로토콜을 사용합니다. AWS SDK를 이전 버전으로 변경하면 Amazon SQS APIs AWS 쿼리를 사용합니다.

Amazon SQS API에서 사용되는 AWS JSON 프로토콜에는 어떤 언어가 지원되나요?

Amazon SQS는 AWS SDKs을 지원합니다. 현재 Kotlin, Rust 또는 Swift는 지원하지 않습니다. 다른 언어 변형에 대해 자세히 알아보려면 AWS에서의 빌드를 위한 도구를 참조하세요.

Amazon SQS API에서 사용되는 AWS JSON 프로토콜이 지원되는 리전

Amazon SQS는 Amazon SQS를 사용할 수 있는 모든 AWS 리전에서 AWS JSON 프로토콜을 지원합니다.

AWS JSON 프로토콜을 사용하여 Amazon SQS용 지정된 AWS SDK 버전으로 업그레이드할 때 어떤 지연 시간 개선을 기대할 수 있습니까?

AWS JSON 프로토콜은 AWS 쿼리 프로토콜에 비해 요청 및 응답의 직렬화 및 역직렬화에 더 효율적입니다. 5KB 메시지 페이로드에 대한 AWS 성능 테스트를 기반으로 Amazon SQS용 JSON 프로토콜은 end-to-end 메시지 처리 지연 시간을 최대 23% 줄이고 애플리케이션 클라이언트 측 CPU 및 메모리 사용량을 줄입니다.

AWS 쿼리 프로토콜이 더 이상 사용되지 않나요?

AWS 쿼리 프로토콜은 계속 지원됩니다. AWS SDK 버전이 Amazon SQS용 JSON 프로토콜을 시작하려면 어떻게 해야 합니까?에 나열된 버전 이외의 이전 버전으로 설정되어 있는 한 AWS 쿼리 프로토콜을 계속 사용할 수 있습니다. AWS Amazon SQS

AWS JSON 프로토콜에 대한 자세한 정보는 어디에서 찾을 수 있나요?

Smithy 설명서의 AWS JSON 1.0 프로토콜에서 JSON 프로토콜에 대한 자세한 내용을 확인할 수 있습니다. AWS JSON 프로토콜을 사용하는 Amazon SQS API 요청에 대한 자세한 내용은 Amazon SQS에서 AWS JSON 프로토콜을 사용하여 쿼리 API 요청 수행 섹션을 참조하세요.

Amazon SQS에서 쿼리 프로토콜을 사용하여 AWS 쿼리 API 요청 수행

이 주제에서는 Amazon SQS 엔드포인트를 구성하고, GET 및 POST 요청을 하고, 응답을 해석하는 방법을 설명합니다.

엔드포인트 구성

Amazon SQS 대기열을 사용하려면 엔드포인트를 생성해야 합니다. Amazon SQS 엔드포인트에 대한 자세한 내용은 Amazon Web Services 일반 참조의 다음 페이지를 참조하세요.

- 리전 엔드포인트
- Amazon Simple Queue Service 엔드포인트 및 할당량

모든 Amazon SQS 엔드포인트는 독립적입니다. 예를 들어 두 대기열의 이름이 MyQueue인데 한 대기열에는 `sqs.us-east-2.amazonaws.com` 엔드포인트가 있고 다른 대기열에는 `sqs.eu-west-2.amazonaws.com` 엔드포인트가 있는 경우 두 대기열은 서로 어떤 데이터도 공유하지 않습니다.

다음은 대기열을 생성하는 요청을 만드는 엔드포인트의 예제입니다.

```
https://sqs.eu-west-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> 대기열 이름과 대기열 URL은 대소문자를 구분합니다.
> *AUTHPARAMS*의 구조는 API 요청의 서명에 따라 달라집니다. 자세한 내용은 Amazon Web Services 일반 참조의 AWS API 요청 서명을 참조하세요.

GET 요청 만들기

Amazon SQS GET 요청은 다음으로 구성된 URL로 생성됩니다.

- 엔드포인트 - 요청이 작용하는 리소스(대기열 이름 및 URL)입니다. 예: `https://sqs.us-east-2.amazonaws.com/`*`123456789012`*`/MyQueue`
- 작업 - 엔드포인트에서 수행하려는 작업입니다. 물음표(?)는 엔드포인트와 작업을 구분합니다. 예: `?Action=SendMessage&MessageBody=Your%20Message%20Text`
- 파라미터 - 요청 파라미터입니다. 각 파라미터는 앰퍼샌드(&)로 구분됩니다. 예: `&Version=2012-11-05&`*`AUTHPARAMS`*

다음은 Amazon SQS 대기열로 메시지를 보내는 GET 요청의 예제입니다.

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
?Action=SendMessage&MessageBody=Your%20message%20text
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> 대기열 이름과 대기열 URL은 대소문자를 구분합니다.

GET 요청은 URL 형식이므로 모든 파라미터 값을 URL 인코딩해야 합니다. URL에 공백이 허용되지 않기 때문에 각 공백은 `%20`으로 URL 인코딩됩니다. 나머지 예제는 URL 인코딩되지 않아 쉽게 판독할 수 있습니다.

POST 요청 만들기

Amazon SQS POST 요청은 쿼리 파라미터를 HTTP 요청 본문에 포함시켜 보냅니다.

다음은 `Content-Type`이 `application/x-www-form-urlencoded`로 설정된 HTTP 헤더의 예제입니다.

```
POST /123456789012/MyQueue HTTP/1.1
Host: sqs.us-east-2.amazonaws.com
Content-Type: application/x-www-form-urlencoded
```

헤더 뒤에 Amazon SQS 대기열로 메시지를 보내는 `form-urlencoded` GET 요청이 옵니다. 각 파라미터는 앰퍼샌드(&)로 구분됩니다.

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Expires=2020-10-15T12%3A00%3A00Z
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> `Content-Type` HTTP 헤더만 필수 항목입니다. *AUTHPARAMS*는 GET 요청과 동일합니다. HTTP 클라이언트는 클라이언트의 HTTP 버전에 따라 HTTP 요청에 다른 항목을 추가할 수 있습니다.

Amazon SQS XML API 응답 해석

Amazon SQS에 요청을 보내면 요청 결과가 포함된 XML 응답이 반환됩니다. 이러한 응답의 구조와 세부 정보를 이해하려면 Amazon Simple Queue Service API 참조의 특정 API 작업을 참조하세요.

성공적인 XML 응답 구조

요청이 성공하면 주 응답 요소는 작업의 이름을 따서 이름이 지정되고 `Response`가 추가됩니다(예: *`ActionName`*`Response`).

이 요소에는 다음과 같은 하위 요소가 포함되어 있습니다.

- ***`ActionName`*`Result`** - 작업별 요소를 포함합니다. 예를 들어 `CreateQueueResult` 요소는 `QueueUrl` 요소를 포함하며, 후자는 생성된 대기열의 URL을 포함합니다.
- **`ResponseMetadata`** - `RequestId`를 포함하며, 결과적으로 요청의 Universal Unique Identifier(UUID)를 포함합니다.

다음은 XML 형식의 성공적인 응답의 예입니다.

```
<CreateQueueResponse
   xmlns=https://sqs.us-east-2.amazonaws.com/doc/2012-11-05/
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:type=CreateQueueResponse>
   <CreateQueueResult>
      <QueueUrl>https://sqs.us-east-2.amazonaws.com/770098461991/queue2</QueueUrl>
   </CreateQueueResult>
   <ResponseMetadata>
      <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
   </ResponseMetadata>
</CreateQueueResponse>
```

XML 오류 응답 구조

요청이 실패한 경우 Amazon SQS는 항상 주 응답 요소 `ErrorResponse`를 반환합니다. 이 요소에는 `Error` 요소와 `RequestId` 요소가 포함되어 있습니다.

`Error` 요소에는 다음과 같은 하위 요소가 포함되어 있습니다.

- **`Type`** - 오류가 생성자 오류였는지 소비자 오류였는지 지정합니다.
- **`Code`** - 오류 유형을 지정합니다.
- **`Message`** - 오류 조건을 읽을 수 있는 형식으로 지정합니다.
- **`Detail`** - (선택 사항) 오류에 대한 추가 세부 정보를 지정합니다.

`RequestId` 요소에는 요청의 UUID가 포함되어 있습니다.

다음은 XML 형식의 오류 응답의 예입니다.

```
<ErrorResponse>
   <Error>
      <Type>Sender</Type>
      <Code>InvalidParameterValue</Code>
      <Message>
         Value (quename_nonalpha) for parameter QueueName is invalid.
         Must be an alphanumeric String of 1 to 80 in length.
      </Message>
   </Error>
   <RequestId>42d59b56-7407-4c4a-be0f-4c88daeea257</RequestId>
</ErrorResponse>
```

Amazon SQS에 대한 요청 인증

인증은 요청을 보내는 당사자를 식별하고 확인하는 프로세스입니다. 인증의 첫 단계에서 AWS 는 생성자의 자격 증명과 생성자가 AWS를 사용하도록 등록되었는지 여부를 확인합니다(자세한 내용은 1단계: AWS 계정 및 IAM 사용자 생성 참조). 다음으로, 다음 절차를 AWS 따릅니다.

1. 생성자(발신자)는 필요한 자격 증명을 획득합니다.
2. 생성자는 요청과 자격 증명은 소비자(수신자)에게 전송합니다.
3. 소비자는 자격 증명을 사용하여 생성자가 요청을 전송했는지 여부를 확인합니다.
4. 다음 중 하나가 발생합니다.
   - 인증에 성공할 경우, 소비자는 요청을 처리합니다.
   - 인증에 실패할 경우 소비자는 요청을 거부하고 오류를 반환합니다.

HMAC-SHA를 사용한 기본 인증 프로세스

쿼리 API를 사용하여 Amazon SQS에 액세스할 때는 요청을 인증할 수 있도록 다음 항목을 제공해야 합니다.

- 가 보안 AWS 액세스 키를 조회하는 데 사용하는를 식별하는 액세스 키 ID입니다. AWS 계정 AWS
- 비밀 액세스 키(사용자와 AAA에게만 알려진 공유 비밀 - 자세한 내용은 RFC2104 참조)를 사용하여 계산된 HMAC-SHA 요청 서명입니다. AWS SDK는 서명 프로세스를 처리합니다. 그러나 HTTP 또는 HTTPS를 통해 쿼리 요청을 제출하는 경우 모든 쿼리 요청에 서명을 포함시켜야 합니다.

1. 서명 버전 4 서명 키를 생성합니다. 자세한 내용은 Java를 사용하여 서명 키 생성을 참조하십시오.

> **Note**
>
> Amazon SQS는 이전 버전보다 SHA256 기반 보안과 성능이 강화된 서명 버전 4를 지원합니다. Amazon SQS를 사용하는 새 애플리케이션을 만들 때 서명 버전 4를 사용합니다.

2. 요청 서명을 base64로 인코딩합니다. 다음 샘플 Java 코드는 이를 수행합니다.

```
package amazon.webservices.common;

// Define common routines for encoding data in AWS requests.
public class Encoding {

    /* Perform base64 encoding of input bytes.
     * rawData is the array of bytes to be encoded.
     * return is the base64-encoded string representation of rawData.
     */
    public static String EncodeBase64(byte[] rawData) {
        return Base64.encodeBytes(rawData);
    }
}
```

- 요청의 타임스탬프(또는 만료)입니다. 요청에 사용하는 타임스탬프는 시간, 분, 초를 포함한 전체 날짜가 포함된 `dateTime` 객체여야 합니다. 예를 들면 `2007-01-31T23:59:59Z`입니다. 필수는 아니지만 협정 세계시(그리니치 표준시) 시간대를 사용하여 객체를 제공하는 것이 좋습니다.

> **Note**
>
> 서버 시간을 올바르게 설정했는지 확인하십시오. 타임스탬프(만료되지 않음)를 지정하면 지정된 시간으로부터 15분 후에 요청이 자동으로 만료됩니다(타임스탬프가 현재 서버 시간 AWS 보다 15분 이상 이전인 요청은 처리AWS 하지 않음).
> .NET을 사용하는 경우, 시간 정밀도의 유효 자릿수에 대한 해석이 다양하기 때문에 너무 자세한 타임스탬프를 전송하지 않아야 합니다. 이 경우 정밀도가 1밀리초 이하인 `dateTime` 객체를 수동으로 생성해야 합니다.

1부: 사용자의 요청

다음은 HMAC-SHA AWS 요청 서명을 사용하여 요청을 인증하기 위해 따라야 하는 프로세스입니다.

1. 요청을 구성합니다 AWS.
2. 보안 액세스 키를 사용하여 키가 추가된 해시 메시지 인증 코드(HMAC-SHA) 서명을 계산합니다.
3. 요청에 서명과 액세스 키 ID를 포함시킨 다음 요청을 보냅니다 AWS.

2부:의 응답 AWS

AWS 는 응답으로 다음 프로세스를 시작합니다.

1. AWS 는 액세스 키 ID를 사용하여 보안 액세스 키를 조회합니다.
2. AWS 는 요청에서 전송한 서명을 계산하는 데 사용한 것과 동일한 알고리즘을 사용하여 요청 데이터와 보안 액세스 키에서 서명을 생성합니다.
3. 다음 중 하나가 발생합니다.
   - 가 AWS 생성하는 서명이 요청에서 보내는 서명과 일치하는 경우는 요청을 신뢰할 수 있는 것으로 AWS 간주합니다.
   - 비교에 실패하면 요청이 삭제되고가 오류를 AWS 반환합니다.

Amazon SQS 배치 작업

Amazon SQS는 비용을 절감하고 단일 작업으로 최대 10개의 메시지를 조작할 수 있는 배치 작업을 제공합니다. 이러한 배치 작업에는 다음이 포함됩니다.

- `SendMessageBatch`
- `DeleteMessageBatch`

- ChangeMessageVisibilityBatch

배치 작업을 사용하면 단일 API 직접 호출로 여러 작업을 수행할 수 있으므로 성능을 최적화하고 비용을 절감할 수 있습니다. Amazon SQS 배치 작업을 지원하는 쿼리 API 또는 AWS SDK를 사용하여 배치 기능을 활용할 수 있습니다.

중요한 세부 정보

- 메시지 크기 제한: 단일 `SendMessageBatch` 호출로 전송된 모든 메시지의 총 크기는 1,048,576바이트(1MiB)를 초과할 수 없습니다.
- 권한: `SendMessageBatch`, `DeleteMessageBatch` 또는 `ChangeMessageVisibilityBatch`에 대한 권한을 명시적으로 설정할 수 없습니다. `SendMessage`, `DeleteMessage` 또는 `ChangeMessageVisibility`에 대한 권한을 설정하면 작업의 해당 배치 버전에 대한 권한도 설정됩니다.
- 콘솔 지원: Amazon SQS 콘솔은 배치 작업을 지원하지 않습니다. 쿼리 API 또는 AWS SDK를 사용하여 배치 작업을 수행해야 합니다.

메시지 작업 일괄 처리

비용 및 효율성을 더욱 최적화하려면 메시지 작업 배치 처리를 위한 다음 모범 사례를 고려하세요.

- 배치 API 작업: Amazon SQS 배치 API 작업을 사용하여 메시지를 전송, 수신 및 삭제하고 한 번의 작업으로 여러 메시지에 대한 메시지 표시 시간 제한을 변경합니다. 이렇게 하면 API 직접 호출 수와 관련 비용이 줄어듭니다.
- 클라이언트 측 버퍼링 긴 롱 폴링: AWS SDK for Java에 포함된 버퍼링된 비동기식 클라이언트와 함께 긴 폴링을 사용하여 클라이언트 측 버퍼링과 요청 배치 처리를 결합합니다. 이 접근 방식은 요청 수를 최소화하고 대량의 메시지 처리를 최적화하는 데 도움이 됩니다.

> **Note**
>
> Amazon SQS의 버퍼링된 비동기식 클라이언트는 현재 FIFO 대기열을 지원하지 않습니다.

Amazon SQS로 클라이언트 측 버퍼링 및 요청 배치 처리 활성화

AWS SDK for Java는 Amazon SQS에 액세스하는 `AmazonSQSBufferedAsyncClient`를 포함합니다. 이 클라이언트는 클라이언트 측 버퍼링을 사용하여 간단한 요청 일괄 처리를 허용합니다. 클라이언트에서 이루어진 호출은 먼저 버퍼링된 다음 Amazon SQS에 배치 요청으로 전송됩니다.

클라이언트 측 버퍼링을 통해 최대 10개의 요청을 버퍼링하여 배치 요청으로 전송할 수 있으므로 Amazon SQS 사용 비용이 절감되고 전송되는 요청 수가 줄어듭니다. `AmazonSQSBufferedAsyncClient` 버퍼는 동기식 및 비동기식 호출을 모두 버퍼링합니다. 배치 처리된 요청과 긴 폴링 지원을 통해 처리량도 높일 수 있습니다. 자세한 내용은 Amazon SQS로 수평적 조정과 작업 배치 처리를 사용하여 처리량 증대 단원을 참조하십시오.

`AmazonSQSBufferedAsyncClient`는 `AmazonSQSAsyncClient`와 동일한 인터페이스를 구현하므로 `AmazonSQSAsyncClient`에서 `AmazonSQSBufferedAsyncClient`로 마이그레이션하려면 일반적으로 기존 코드를 최소한만 변경하면 됩니다.

> **Note**
>
> Amazon SQS의 버퍼링된 비동기식 클라이언트는 현재 FIFO 대기열을 지원하지 않습니다.

AmazonSQSBufferedAsyncClient 사용

시작하기 전에 Amazon SQS 설정의 단계를 완료해야 합니다.

AWS Java 1.x용 SDK

AWS SDK for Java 1.x의 경우 다음 예제를 `AmazonSQSBufferedAsyncClient` 기반으로 새를 생성할 수 있습니다.

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

새 `AmazonSQSBufferedAsyncClient`를 생성한 후 이것을 이용해서 Amazon SQS에 다중 요청을 전송할 수 있습니다(`AmazonSQSAsyncClient`와 마찬가지로). 예를 들면 다음과 같습니다.

```
final CreateQueueRequest createRequest = new
 CreateQueueRequest().withQueueName("MyQueue");
```

```
final CreateQueueResult res = bufferedSqs.createQueue(createRequest);

final SendMessageRequest request = new SendMessageRequest();
final String body = "Your message text" + System.currentTimeMillis();
request.setMessageBody( body );
request.setQueueUrl(res.getQueueUrl());

final Future<SendMessageResult> sendResult = bufferedSqs.sendMessageAsync(request);

final ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
final ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

AmazonSQSBufferedAsyncClient 구성

`AmazonSQSBufferedAsyncClient`는 대부분의 사용 사례에 적용되는 설정으로 사전 구성됩니다. `AmazonSQSBufferedAsyncClient`를 추가로 구성할 수 있습니다. 예를 들면 다음과 같습니다.

1. 필요한 구성 파라미터로 `QueueBufferConfig` 클래스의 인스턴스를 만듭니다.
2. `AmazonSQSBufferedAsyncClient` 생성자에 인스턴스를 제공합니다.

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

final QueueBufferConfig config = new QueueBufferConfig()
    .withMaxInflightReceiveBatches(5)
    .withMaxDoneReceiveBatches(15);

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync, config);
```

QueueBufferConfig 구성 파라미터

파라미터	기본값	설명
`longPoll`	`true`	`longPoll`이 `true`로 설정된 경우 `AmazonSQSBufferedAsyncClient` 는 메시지를

파라미터	기본값	설명
		사용할 때 긴 폴링을 사용하려고 시도합니다.
`longPollWaitTimeoutSeconds`	20s	`ReceiveMessage` 호출이 서버에서 차단되고 빈 수신 결과를 반환하기 전에 메시지가 대기열에 나타나기까지 대기하는 최대 시간(초)입니다. **Note** 긴 폴링을 비활성화하면 이 설정은 영향을 주지 않습니다.
`maxBatchOpenMs`	200ms	발신 호출이 동일한 유형의 메시지를 배치 처리할 다른 호출을 대기하는 최대 시간(밀리초)입니다. 설정이 높을수록 동일한 작업량을 수행하는 데 필요한 배치 수가 감소합니다(그러나 배치의 첫 번째 호출을 대기하는 시간이 길어짐). 이 파라미터를 `0`으로 설정하면, 제출된 요청은 다른 요청을 기다리지 않으므로 사실상 일괄 처리하지 않게 됩니다.

파라미터	기본값	설명
`maxBatchSize`	배치당 요청 10개	단일 요청에서 배치 방식으로 함께 처리되는 최대 메시지 수입니다. 설정이 높을수록 동일한 요청 수를 수행하는 데 필요한 배치 수가 감소합니다. **Note** 배치당 요청 10개는 Amazon SQS에 허용되는 최대값입니다.
`maxBatchSizeBytes`	1MiB	클라이언트가 Amazon SQS에 전송하려고 시도하는 메시지 배치의 최대 크기(바이트 단위)입니다. **Note** 1MiB는 Amazon SQS에 허용되는 최대 값입니다.

파라미터	기본값	설명
`maxDoneReceiveBatches`	배치 10개	`AmazonSQSBufferedAsyncClient` 가 클라이언트 측에서 미리 가져와서 저장하는 수신 배치의 최대 개수입니다. 설정이 높을수록 Amazon SQS에 호출을 전송하지 않고도 만족할 수 있는 수신 요청 개수가 늘어납니다(그러나 미리 가져오는 메시지가 많을수록 메시지가 버퍼에 있는 시간이 길어지므로 표시 제한 시간이 만료될 수 있음). **Note** `0`은 모든 메시지 미리 가져오기 기능이 비활성화되고 메시지가 요청 시에만 사용됨을 나타냅니다.

파라미터	기본값	설명
`maxInflightOutboundBatches`	배치 5개	동시에 처리될 수 있는 활성 아웃바운드 배치의 최대 개수입니다. 설정이 높을수록 아웃바운드 배치 전송 속도가 빨라지고 (CPU 또는 대역폭과 같은 할당량도 영향을 줌), `AmazonSQSBufferedAsyncClient` 에서 사용하는 스레드 수가 많아집니다.
`maxInflightReceiveBatches`	배치 10개	동시에 처리될 수 있는 활성 수신 배치의 최대 개수입니다. 설정이 높을수록 수신 가능한 메시지가 많아지고(CPU 또는 대역폭과 같은 할당량도 영향을 줌), `AmazonSQSBufferedAsyncClient` 에서 사용하는 스레드 수가 많아집니다. **Note** `0`은 모든 메시지 미리 가져오기 기능이 비활성화되고 메시지가 요청 시에만 사용됨을 나타냅니다.

파라미터	기본값	설명
`visibilityTimeoutSeconds`	-1	이 파라미터를 0이 아닌 양수 값으로 설정하면, 여기에 설정된 제한 시간 초과는 메시지가 사용되는 대기열에서 설정된 제한 시간 초과를 재정의합니다. **Note** -1은 대기열에서 기본 설정이 선택되어 있음을 나타냅니다. 제한 시간 초과를 0으로 설정할 수 없습니다.

AWS Java 2.x용 SDK

AWS SDK for Java 2.x의 경우 다음 예제를 `SqsAsyncBatchManager` 기반으로 새를 생성할 수 있습니다.

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

새 `SqsAsyncBatchManager`를 생성한 후 이것을 이용해서 Amazon SQS에 다중 요청을 전송할 수 있습니다(`SqsAsyncClient`와 마찬가지로). 예를 들면 다음과 같습니다.

```
final String queueName = "MyAsyncBufferedQueue" + UUID.randomUUID();
final CreateQueueRequest request =
 CreateQueueRequest.builder().queueName(queueName).build();
```

```
final String queueUrl = sqs.createQueue(request).join().queueUrl();
System.out.println("Queue created: " + queueUrl);


// Send messages
CompletableFuture<SendMessageResponse> sendMessageFuture;
for (int i = 0; i < 10; i++) {
    final int index = i;
    sendMessageFuture = sqsAsyncBatchManager.sendMessage(
            r -> r.messageBody("Message " + index).queueUrl(queueUrl));
    SendMessageResponse response= sendMessageFuture.join();
    System.out.println("Message " + response.messageId() + " sent!");
}

// Receive messages with customized configurations
CompletableFuture<ReceiveMessageResponse> receiveResponseFuture =
 customizedBatchManager.receiveMessage(
        r -> r.queueUrl(queueUrl)
                .waitTimeSeconds(10)
                .visibilityTimeout(20)
                .maxNumberOfMessages(10)
);
System.out.println("You have received " +
 receiveResponseFuture.join().messages().size() + " messages in total.");

// Delete messages
DeleteQueueRequest deleteQueueRequest =
 DeleteQueueRequest.builder().queueUrl(queueUrl).build();
int code = sqs.deleteQueue(deleteQueueRequest).join().sdkHttpResponse().statusCode();
System.out.println("Queue is deleted, with statusCode " + code);
```

SqsAsyncBatchManager 구성

`SqsAsyncBatchManager`는 대부분의 사용 사례에 적용되는 설정으로 사전 구성됩니다. `SqsAsyncBatchManager`를 추가로 구성할 수 있습니다. 예를 들면 다음과 같습니다.

를 통해 사용자 지정 구성 생성`SqsAsyncBatchManager.Builder`:

```
SqsAsyncBatchManager customizedBatchManager = SqsAsyncBatchManager.builder()
    .client(sqs)
    .scheduledExecutor(Executors.newScheduledThreadPool(5))
    .overrideConfiguration(b -> b
        .maxBatchSize(10)
```

```
        .sendRequestFrequency(Duration.ofMillis(200))
        .receiveMessageMinWaitDuration(Duration.ofSeconds(10))
        .receiveMessageVisibilityTimeout(Duration.ofSeconds(20))
        .receiveMessageAttributeNames(Collections.singletonList("*"))

 .receiveMessageSystemAttributeNames(Collections.singletonList(MessageSystemAttributeName.ALL))
    .build();
```

`BatchOverrideConfiguration` 파라미터

파라미터	기본값	설명
`maxBatchSize`	배치당 요청 10개	단일 요청에서 배치 방식으로 함께 처리되는 최대 메시지 수입니다. 설정이 높을수록 동일한 요청 수를 수행하는 데 필요한 배치 수가 감소합니다. **Note** Amazon SQS의 최대 허용 값은 배치당 요청 10개입니다.
`sendRequestFrequency`	200ms	발신 호출이 동일한 유형의 메시지를 배치 처리할 다른 호출을 대기하는 최대 시간(밀리초)입니다. 설정이 높을수록 동일한 작업량을 수행하는 데 필요한 배치 수가 감소합니다(그러나 배치의 첫 번째 호출을 대기하는 시간이 길어짐). 이 파라미터를 `0`으로 설정하면, 제출된 요청은 다른 요청을 기다리지 않으므로 사실상 일괄 처리하지 않게 됩니다.

파라미터	기본값	설명
`receiveMessageVisibilityTimeout`	-1	이 파라미터를 0이 아닌 양수 값으로 설정하면, 여기에 설정된 제한 시간 초과는 메시지가 사용되는 대기열에서 설정된 제한 시간 초과를 재정의합니다. **Note** 1은 대기열에서 기본 설정이 선택되어 있음을 나타냅니다. 제한 시간 초과를 0으로 설정할 수 없습니다.
`receiveMessageMinWaitDuration`	50ms	`receiveMessage` 통화가 사용 가능한 메시지를 가져올 때까지 대기하는 최소 시간(밀리초)입니다. 설정이 높을수록 동일한 수의 요청을 수행하는 데 필요한 배치 수가 줄어듭니다.

Amazon SQS로 수평적 조정과 작업 배치 처리를 사용하여 처리량 증대

Amazon SQS는 처리량이 많은 메시징을 지원합니다. 처리량 제한에 대한 자세한 내용은 섹션을 참조하세요Amazon SQS 메시지 할당량.

처리량을 최대화하려면:

- 각 인스턴스를 더 추가하여 생산자와 소비자를 수평적으로 확장합니다.
- 작업 일괄 처리를 사용하여 단일 요청으로 여러 메시지를 보내거나 수신하여 API 호출 오버헤드를 줄입니다.

수평 크기 조정

HTTP 요청-응답 프로토콜을 통해 Amazon SQS에 액세스하기 때문에 요청 지연 시간(요청을 시작해서 응답을 받기까지의 시간 간격)은 단일 연결을 사용하여 단일 스레드에서 확보할 수 있는 처리량을 제한합니다. 예를 들어, 동일한 리전에서 Amazon EC2 기반 클라이언트부터 Amazon SQS까지 지연 시간이 평균 20ms인 경우 단일 연결을 통한 단일 스레드의 최대 처리량은 평균 50TPS입니다.

수평적 조정은 메시지 생성자(`SendMessage` 요청 전송)와 소비자(`ReceiveMessage` 및 `DeleteMessage` 요청 전송) 수를 늘리는 것으로 전체 대기열 처리량을 높여줍니다. 다음 세 가지 방법으로 수평 확장할 수 있습니다.

- 클라이언트당 스레드 수 늘리기
- 클라이언트 추가
- 클라이언트당 스레드 수를 늘리고 클라이언트 추가

클라이언트를 더 추가하면 대기열 처리량이 선형적으로 증가합니다. 예를 들어 클라이언트 개수를 2배 늘리면 처리량도 2배가 됩니다.

작업 일괄 처리

일괄 처리는 서비스까지 왕복할 때마다 더 많은 작업을 수행합니다(예: `SendMessageBatch` 요청 하나로 여러 메시지를 전송하는 경우). Amazon SQS 배치 작업은 `SendMessageBatch`, `DeleteMessageBatch` 및 `ChangeMessageVisibilityBatch`입니다. 생산자 또는 소비자를 변경하지 않고 배치 처리를 활용하려면 Amazon SQS의 버퍼링된 비동기식 클라이언트를 사용하면 됩니다.

> **Note**
>
> `ReceiveMessage`는 한 번에 메시지를 10개 처리할 수 있으므로 `ReceiveMessageBatch` 작업이 없습니다.

일괄 처리는 메시지 하나의 전체 지연 시간을 수락하는 것이 아니라, 배치 작업의 지연 시간을 배치 요청의 여러 메시지로 분산시키는 것입니다(예: `SendMessage` 요청). 각 왕복마다 더 많은 작업을 수행하기 때문에 배치 요청은 스레드와 연결을 보다 효율적으로 사용하므로 처리량이 개선됩니다.

일괄 처리와 수평적 조정을 함께 사용하면 개별 메시지 요청보다 스레드, 연결 및 요청 수가 적은 처리량을 얻습니다. 배치 처리된 Amazon SQS 작업을 사용하여 한 번에 최대 10개 메시지를 전송, 수신 또

는 삭제할 수 있습니다. Amazon SQS는 요청별로 요금을 부과하기 때문에 배치 처리 시 비용을 대폭 절감할 수 있습니다.

일괄 처리를 사용하면 애플리케이션에 어느 정도 복잡성이 수반됩니다(예를 들어 애플리케이션이 메시지를 누적한 후 메시지를 전송하고 때로는 응답을 받기까지 대기 시간이 길어짐). 그러나 다음과 같은 상황에서는 일괄 처리가 효과적일 수 있습니다.

- 애플리케이션이 단시간에 많은 메시지를 생성하지만 지연 시간이 매우 길지 않은 경우.
- 제어하지 못하는 이벤트에 대응하여 메시지를 전송해야 하는 일반적인 메시지 생성자와 달리 메시지 소비자가 단독 재량으로 대기열에서 메시지를 가져오는 경우.

> **Important**
>
> 배치의 개별 메시지에 오류가 발생하더라도 배치 요청이 성공할 수 있습니다. 배치 요청 후에는 항상 개별 메시지의 오류 여부를 확인하고 필요에 따라 작업을 다시 시도해야 합니다.

단일 작업 및 배치 요청에 대한 Java 사용 예제

사전 조건

`aws-java-sdk-sqs.jar`, `aws-java-sdk-ec2.jar`, `commons-logging.jar` 패키지를 Java 빌드 클래스 경로에 추가합니다. 다음 예제는 Maven 프로젝트의 `pom.xml` 파일 내에 존재하는 이러한 종속성을 보여줍니다.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-ec2</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>LATEST</version>
```

```
    </dependency>
</dependencies>
```

SimpleProducerConsumer.java

다음 Java 코드 예제는 단순한 생성자-소비자 패턴을 구현한 것입니다. 기본 스레드는 지정된 시간 동안 1KB 메시지를 처리하는 생성자와 소비자 스레드를 다수 생성합니다. 이 예제에는 단일 작업을 요청하는 생성자와 소비자뿐 아니라 배치 요청을 하는 생성자와 소비자가 포함되어 있습니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

import com.amazonaws.AmazonClientException;
import com.amazonaws.ClientConfiguration;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
```

```
 * Start a specified number of producer and consumer threads, and produce-consume
 * for the least of the specified duration and 1 hour. Some messages can be left
 * in the queue because producers and consumers might not be in exact balance.
 */
public class SimpleProducerConsumer {

    // The maximum runtime of the program.
    private final static int MAX_RUNTIME_MINUTES = 60;
    private final static Log log = LogFactory.getLog(SimpleProducerConsumer.class);

    public static void main(String[] args) throws InterruptedException {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the number of producers: ");
        final int producerCount = input.nextInt();

        System.out.print("Enter the number of consumers: ");
        final int consumerCount = input.nextInt();

        System.out.print("Enter the number of messages per batch: ");
        final int batchSize = input.nextInt();

        System.out.print("Enter the message size in bytes: ");
        final int messageSizeByte = input.nextInt();

        System.out.print("Enter the run time in minutes: ");
        final int runTimeMinutes = input.nextInt();

        /*
         * Create a new instance of the builder with all defaults (credentials
         * and region) set automatically. For more information, see Creating
         * Service Clients in the AWS SDK for Java Developer Guide.
         */
        final ClientConfiguration clientConfiguration = new ClientConfiguration()
                .withMaxConnections(producerCount + consumerCount);

        final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
                .withClientConfiguration(clientConfiguration)
                .build();
```

```
        final String queueUrl = sqsClient
                .getQueueUrl(new GetQueueUrlRequest(queueName)).getQueueUrl();

        // The flag used to stop producer, consumer, and monitor threads.
        final AtomicBoolean stop = new AtomicBoolean(false);

        // Start the producers.
        final AtomicInteger producedCount = new AtomicInteger();
        final Thread[] producers = new Thread[producerCount];
        for (int i = 0; i < producerCount; i++) {
            if (batchSize == 1) {
                producers[i] = new Producer(sqsClient, queueUrl, messageSizeByte,
                        producedCount, stop);
            } else {
                producers[i] = new BatchProducer(sqsClient, queueUrl, batchSize,
                        messageSizeByte, producedCount,
                        stop);
            }
            producers[i].start();
        }

        // Start the consumers.
        final AtomicInteger consumedCount = new AtomicInteger();
        final Thread[] consumers = new Thread[consumerCount];
        for (int i = 0; i < consumerCount; i++) {
            if (batchSize == 1) {
                consumers[i] = new Consumer(sqsClient, queueUrl, consumedCount,
                        stop);
            } else {
                consumers[i] = new BatchConsumer(sqsClient, queueUrl, batchSize,
                        consumedCount, stop);
            }
            consumers[i].start();
        }

        // Start the monitor thread.
        final Thread monitor = new Monitor(producedCount, consumedCount, stop);
        monitor.start();

        // Wait for the specified amount of time then stop.
        Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runTimeMinutes,
                MAX_RUNTIME_MINUTES)));
        stop.set(true);
```

```
        // Join all threads.
        for (int i = 0; i < producerCount; i++) {
            producers[i].join();
        }

        for (int i = 0; i < consumerCount; i++) {
            consumers[i].join();
        }

        monitor.interrupt();
        monitor.join();
    }

    private static String makeRandomString(int sizeByte) {
        final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
        new Random().nextBytes(bs);
        bs[0] = (byte) ((bs[0] | 64) & 127);
        return new BigInteger(bs).toString(32);
    }

    /**
     * The producer thread uses {@code SendMessage}
     * to send messages until it is stopped.
     */
    private static class Producer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        Producer(AmazonSQS sqsQueueBuffer, String queueUrl, int messageSizeByte,
                 AtomicInteger producedCount, AtomicBoolean stop) {
            this.sqsClient = sqsQueueBuffer;
            this.queueUrl = queueUrl;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /*
         * The producedCount object tracks the number of messages produced by
         * all producer threads. If there is an error, the program exits the
         * run() method.
```

```
         */
        public void run() {
            try {
                while (!stop.get()) {
                    sqsClient.sendMessage(new SendMessageRequest(queueUrl,
                            theMessage));
                    producedCount.incrementAndGet();
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("Producer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * The producer thread uses {@code SendMessageBatch}
     * to send messages until it is stopped.
     */
    private static class BatchProducer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        BatchProducer(AmazonSQS sqsQueueBuffer, String queueUrl, int batchSize,
                      int messageSizeByte, AtomicInteger producedCount,
                      AtomicBoolean stop) {
            this.sqsClient = sqsQueueBuffer;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        public void run() {
            try {
```

```
                while (!stop.get()) {
                    final SendMessageBatchRequest batchRequest =
                            new SendMessageBatchRequest().withQueueUrl(queueUrl);

                    final List<SendMessageBatchRequestEntry> entries =
                            new ArrayList<SendMessageBatchRequestEntry>();
                    for (int i = 0; i < batchSize; i++)
                        entries.add(new SendMessageBatchRequestEntry()
                                .withId(Integer.toString(i))
                                .withMessageBody(theMessage));
                    batchRequest.setEntries(entries);

                    final SendMessageBatchResult batchResult =
                            sqsClient.sendMessageBatch(batchRequest);
                    producedCount.addAndGet(batchResult.getSuccessful().size());

                    /*
                     * Because SendMessageBatch can return successfully, but
                     * individual batch items fail, retry the failed batch items.
                     */
                    if (!batchResult.getFailed().isEmpty()) {
                        log.warn("Producer: retrying sending "
                                + batchResult.getFailed().size() + " messages");
                        for (int i = 0, n = batchResult.getFailed().size();
                             i < n; i++) {
                            sqsClient.sendMessage(new
                                    SendMessageRequest(queueUrl, theMessage));
                            producedCount.incrementAndGet();
                        }
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("BatchProducer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * The consumer thread uses {@code ReceiveMessage} and {@code DeleteMessage}
```

```
     * to consume messages until it is stopped.
     */
    private static class Consumer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        Consumer(AmazonSQS sqsClient, String queueUrl, AtomicInteger consumedCount,
                 AtomicBoolean stop) {
            this.sqsClient = sqsClient;
            this.queueUrl = queueUrl;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /*
         * Each consumer thread receives and deletes messages until the main
         * thread stops the consumer thread. The consumedCount object tracks the
         * number of messages that are consumed by all consumer threads, and the
         * count is logged periodically.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    try {
                        final ReceiveMessageResult result = sqsClient
                                .receiveMessage(new
                                        ReceiveMessageRequest(queueUrl));

                        if (!result.getMessages().isEmpty()) {
                            final Message m = result.getMessages().get(0);
                            sqsClient.deleteMessage(new
                                    DeleteMessageRequest(queueUrl,
                                    m.getReceiptHandle()));
                            consumedCount.incrementAndGet();
                        }
                    } catch (AmazonClientException e) {
                        log.error(e.getMessage());
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
```

```
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("Consumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * The consumer thread uses {@code ReceiveMessage} and {@code
     * DeleteMessageBatch} to consume messages until it is stopped.
     */
    private static class BatchConsumer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        BatchConsumer(AmazonSQS sqsClient, String queueUrl, int batchSize,
                      AtomicInteger consumedCount, AtomicBoolean stop) {
            this.sqsClient = sqsClient;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        public void run() {
            try {
                while (!stop.get()) {
                    final ReceiveMessageResult result = sqsClient
                            .receiveMessage(new ReceiveMessageRequest(queueUrl)
                                    .withMaxNumberOfMessages(batchSize));

                    if (!result.getMessages().isEmpty()) {
                        final List<Message> messages = result.getMessages();
                        final DeleteMessageBatchRequest batchRequest =
                                new DeleteMessageBatchRequest()
                                        .withQueueUrl(queueUrl);

                        final List<DeleteMessageBatchRequestEntry> entries =
                                new ArrayList<DeleteMessageBatchRequestEntry>();
                        for (int i = 0, n = messages.size(); i < n; i++)
```

```
                            entries.add(new DeleteMessageBatchRequestEntry()
                                    .withId(Integer.toString(i))
                                    .withReceiptHandle(messages.get(i)
                                            .getReceiptHandle()));
                        batchRequest.setEntries(entries);

                        final DeleteMessageBatchResult batchResult = sqsClient
                                .deleteMessageBatch(batchRequest);
                        consumedCount.addAndGet(batchResult.getSuccessful().size());

                        /*
                         * Because DeleteMessageBatch can return successfully,
                         * but individual batch items fail, retry the failed
                         * batch items.
                         */
                        if (!batchResult.getFailed().isEmpty()) {
                            final int n = batchResult.getFailed().size();
                            log.warn("Producer: retrying deleting " + n
                                    + " messages");
                            for (BatchResultErrorEntry e : batchResult
                                    .getFailed()) {

                                sqsClient.deleteMessage(
                                        new DeleteMessageRequest(queueUrl,
                                                messages.get(Integer
                                                        .parseInt(e.getId()))
                                                        .getReceiptHandle()));

                                consumedCount.incrementAndGet();
                            }
                        }
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("BatchConsumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }
```

```
    /**
     * This thread prints every second the number of messages produced and
     * consumed so far.
     */
    private static class Monitor extends Thread {
        private final AtomicInteger producedCount;
        private final AtomicInteger consumedCount;
        private final AtomicBoolean stop;

        Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
                AtomicBoolean stop) {
            this.producedCount = producedCount;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        public void run() {
            try {
                while (!stop.get()) {
                    Thread.sleep(1000);
                    log.info("produced messages = " + producedCount.get()
                            + ", consumed messages = " + consumedCount.get());
                }
            } catch (InterruptedException e) {
                // Allow the thread to exit.
            }
        }
    }
}
```

예제 실행에서 볼륨 측정치 모니터링

Amazon SQS는 전송, 수신 및 삭제된 메시지에 대한 볼륨 지표를 자동으로 생성합니다. 대기열의 모니터링 탭이나 CloudWatch 콘솔을 통해 이러한 지표 및 기타 지표에 액세스할 수 있습니다.

> **Note**
>
> 이 측정치의 경우 대기열을 사용할 수 있을 때까지 최대 15분이 걸릴 수 있습니다.

AWS SDK에서 Amazon SQS 사용

AWS 소프트웨어 개발 키트(SDKs)는 널리 사용되는 많은 프로그래밍 언어에 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예제 및 설명서를 제공합니다.

SDK 설명서	코드 예제
AWS SDK for C++	AWS SDK for C++ 코드 예제
AWS CLI	AWS CLI 코드 예제
AWS SDK for Go	AWS SDK for Go 코드 예제
AWS SDK for Java	AWS SDK for Java 코드 예제
AWS SDK for JavaScript	AWS SDK for JavaScript 코드 예제
AWS SDK for Kotlin	AWS SDK for Kotlin 코드 예제
AWS SDK for .NET	AWS SDK for .NET 코드 예제
AWS SDK for PHP	AWS SDK for PHP 코드 예제
AWS Tools for PowerShell	AWS Tools for PowerShell 코드 예제
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 코드 예제
AWS SDK for Ruby	AWS SDK for Ruby 코드 예제
AWS SDK for Rust	AWS SDK for Rust 코드 예제
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP 코드 예제
AWS SDK for Swift	AWS SDK for Swift 코드 예제

가용성 예제

필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예시를 요청하세요.

Amazon SQS에서 JMS 사용

Amazon SQS Java 메시징 라이브러리는 이미 JMS를 사용하고 있는 애플리케이션에서 Amazon SQS를 활용하도록 해주는 Amazon SQS용 Java 메시지 서비스(JMS) 인터페이스입니다. 이 인터페이스를 통해 코드 변경을 최소화하면서 Amazon SQS를 JMS 공급자로서 사용할 수 있습니다. AWS SDK for Java와 함께 Amazon SQS Java 메시징 라이브러리를 통해 JMS 연결과 세션뿐 아니라 Amazon SQS 대기열과 메시지를 주고 받는 생성자 및 소비자를 생성할 수 있습니다.

이 라이브러리는 JMS 1.1 사양에 따라 대기열(JMS 지점 간 모델)과의 메시지 송수신을 지원합니다. 이 라이브러리는 Amazon SQS 대기열로의 문자, 바이트 또는 객체 메시지 동기식 전송을 지원합니다. 또한, 동기 또는 비동기식 객체 수신도 지원합니다.

JMS 1.1 사양을 지원하는 Amazon SQS Java 메시징 라이브러리의 기능에 대한 자세한 내용은 Amazon SQS 지원 JMS 1.1 구현 및 Amazon SQS FAQ를 참조하세요.

JMS 및 Amazon SQS 작업을 위한 사전 조건

시작하기 전에 다음 사전 요구 사항을 완비해야 합니다.

- Java용 SDK

  프로젝트에 Java용 SDK를 포함시키는 두 가지 방법이 있습니다.

  - Java용 SDK 다운로드 및 설치
  - Maven을 사용하여 Amazon SQS Java 메시징 라이브러리를 다운로드합니다.

  > **Note**
  >
  > Java용 SDK는 종속적으로 포함되어 있습니다.
  > Java용 SDK 및 Java용 Amazon SQS 확장 클라이언트 라이브러리를 사용하려면 J2SE 개발 키트 8.0 이상이 필요합니다.

  Java용 SDK 다운로드에 대한 자세한 내용은 Java용 SDK를 참조하세요.

- Amazon SQS Java 메시징 라이브러리

  Maven을 사용하지 않을 경우, `amazon-sqs-java-messaging-lib.jar` 패키지를 Java 빌드 클래스 경로에 추가해야 합니다. 라이브러리 다운로드에 대한 자세한 내용은 Amazon SQS Java 메시징 라이브러리를 참조하세요.

> **Note**
>
> Amazon SQS Java 메시징 라이브러리에는 Maven과 Spring 프레임워크에 대한 지원이 포함되어 있습니다.
> Maven, Spring Framework, Amazon SQS Java 메시징 라이브러리를 사용하는 코드 샘플은 Amazon SQS 표준 대기열에 JMS를 사용하는 Java 작업 예제 섹션을 참조하세요.
>
> ```
> <dependency>
>  <groupId>com.amazonaws</groupId>
>  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
>  <version>1.0.4</version>
>  <type>jar</type>
> </dependency>
> ```

- Amazon SQS 대기열

  Amazon SQS AWS Management Console 용 , `CreateQueue` API 또는 Amazon SQS Java 메시징 라이브러리에 포함된 래핑된 Amazon SQS 클라이언트를 사용하여 대기열을 생성합니다.

  - AWS Management Console 또는 `CreateQueue` API를 사용하여 Amazon SQS에서 대기열을 만드는 방법에 대한 정보는 대기열 만들기를 참조하세요.
  - Amazon SQS Java 메시징 라이브러리 사용에 대한 자세한 내용은 Amazon SQS Java 메시징 라이브러리 사용 섹션을 참조하세요.

Amazon SQS Java 메시징 라이브러리 사용

Amazon SQS에서 Java 메시지 서비스(JMS) 사용을 시작하려면 이 섹션의 코드 예제를 사용합니다. 다음 섹션에는 JMS 연결 및 세션 생성 방법과 메시지 전송 및 수신 방법이 나와 있습니다.

Amazon SQS Java 메시징 라이브러리에 포함된 래핑된 Amazon SQS 클라이언트 객체는 Amazon SQS 대기열이 존재하는지 확인합니다. 대기열이 존재하지 않는 경우, 클라이언트가 이 대기열을 생성합니다.

JMS 연결 생성

시작하기 전에 JMS 및 Amazon SQS 작업을 위한 사전 조건에서 사전 조건을 참조하세요.

1. 연결 팩토리를 생성하고 이 팩토리에 대한 `createConnection` 메서드를 호출합니다.

```
// Create a new connection factory with all defaults (credentials and region) set
 automatically
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.defaultClient()
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

SQSConnection 클래스는 javax.jms.Connection을 확장합니다. JMS 표준 연결 메서드와 더불어 SQSConnection은 getAmazonSQSClient 및 getWrappedAmazonSQSClient와 같은 추가 메서드를 제공합니다. 이 두 메서드를 통해 새로운 대기열 생성과 같이 JMS 사양에 포함되지 않은 관리 작업을 수행할 수 있습니다. 그렇지만, getWrappedAmazonSQSClient 메서드는 최신 연결을 통해 사용되는 래핑된 버전의 Amazon SQS 클라이언트도 제공합니다. 래퍼는 클라이언트에서 JMSException으로 모든 예외를 변환하므로, JMSException 발생이 예상되는 기존 코드에서 보다 쉽게 사용할 수 있습니다.

2. getAmazonSQSClient 및 getWrappedAmazonSQSClient에서 반환되는 클라이언트 객체를 사용하여 JMS 사양에 포함되지 않은 관리 작업(예: Amazon SQS 대기열 생성 가능)을 수행할 수 있습니다.

   JMS 예외가 예상되는 기존 코드가 있는 경우, getWrappedAmazonSQSClient를 사용해야 합니다.

   - getWrappedAmazonSQSClient를 사용한 경우, 반환된 클라이언트 객체는 모든 예외를 JMS 예외로 변환합니다.
   - getAmazonSQSClient를 사용하는 경우, 예외는 모두 Amazon SQS 예외입니다.

Amazon SQS 대기열 생성

래핑된 클라이언트 객체는 Amazon SQS 대기열이 있는지를 확인합니다.

대기열이 존재하지 않는 경우, 클라이언트가 이 대기열을 생성합니다. 대기열이 존재하지 않는 경우, 이 함수는 어느 항목도 반환하지 않습니다. 자세한 내용은 TextMessageSender.java 예제에서 "필요한 경우 대기열 생성"을 참조하십시오.

표준 대기열을 생성하려면

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an SQS queue named MyQueue, if it doesn't already exist
if (!client.queueExists("MyQueue")) {
    client.createQueue("MyQueue");
}
```

FIFO 대기열을 생성하려면

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an Amazon SQS FIFO queue named MyQueue.fifo, if it doesn't already exist
if (!client.queueExists("MyQueue.fifo")) {
    Map<String, String> attributes = new HashMap<String, String>();
    attributes.put("FifoQueue", "true");
    attributes.put("ContentBasedDeduplication", "true");
    client.createQueue(new
 CreateQueueRequest().withQueueName("MyQueue.fifo").withAttributes(attributes));
}
```

> **Note**
>
> FIFO 대기열의 이름은 `.fifo` 접미사로 끝나야 합니다. `ContentBasedDeduplication` 속성에 대한 자세한 내용은 Amazon SQS의 정확히 1회 처리를 참조하십시오.

메시지 동기식 전송

1. 연결 및 기본 Amazon SQS 대기열이 준비가 되면, `AUTO_ACKNOWLEDGE` 모드로 트랜잭션 처리되지 않는 JMS 세션을 생성합니다.

   ```
   // Create the nontransacted session with AUTO_ACKNOWLEDGE mode
   Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
   ```

2. 대기열에 문자 메시지를 전송하려면, JMS 대기열 자격 증명과 메시지 생성자를 생성합니다.

```
// Create a queue identity and specify the queue name to the session
Queue queue = session.createQueue("MyQueue");

// Create a producer for the 'MyQueue'
MessageProducer producer = session.createProducer(queue);
```

3. 문자 메시지를 작성하고 대기열로 전송합니다.

   - 메시지를 표준 대기열로 전송하는 경우 추가 파라미터를 설정할 필요가 없습니다.

     ```
     // Create the text message
     TextMessage message = session.createTextMessage("Hello World!");

     // Send the message
     producer.send(message);
     System.out.println("JMS Message " + message.getJMSMessageID());
     ```

   - 메시지를 FIFO 대기열로 전송하는 경우 메시지 그룹 ID를 설정해야 합니다. 메시지 중복 제거 ID를 설정할 수도 있습니다. 자세한 내용은 Amazon SQS FIFO 대기열 주요 용어 단원을 참조하십시오.

     ```
     // Create the text message
     TextMessage message = session.createTextMessage("Hello World!");

     // Set the message group ID
     message.setStringProperty("JMSXGroupID", "Default");

     // You can also set a custom message deduplication ID
     // message.setStringProperty("JMS_SQS_DeduplicationId", "hello");
     // Here, it's not needed because content-based deduplication is enabled for the
      queue

     // Send the message
     producer.send(message);
     System.out.println("JMS Message " + message.getJMSMessageID());
     System.out.println("JMS Message Sequence Number " +
      message.getStringProperty("JMS_SQS_SequenceNumber"));
     ```

메시지 동기식 수신

1. 메시지를 수신하려면 동일한 대기열에 대한 소비자를 생성하고 `start` 메서드를 호출합니다.

   언제든지 연결에서 `start` 메서드를 호출할 수 있습니다. 그렇지만, 소비자는 사용자가 메시지를 호출할 때까지 메시지 수신을 시작하지 않습니다.

```
// Create a consumer for the 'MyQueue'
MessageConsumer consumer = session.createConsumer(queue);
// Start receiving incoming messages
connection.start();
```

2. 제한 시간을 1초로 설정한 소비자에서 `receive` 메서드를 호출한 다음 수신된 메시지의 내용을 인쇄합니다.

   - 표준 대기열에서 메시지를 수신한 후 해당 메시지의 내용에 액세스할 수 있습니다.

```
// Receive a message from 'MyQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and display the text
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
}
```

   - FIFO 대기열에서 메시지를 수신한 후 메시지 그룹 ID, 메시지 중복 제거 ID 및 시퀀스 번호와 같은 메시지 내용 및 기타 FIFO 관련 메시지 속성에 액세스할 수 있습니다. 자세한 내용은 Amazon SQS FIFO 대기열 주요 용어 단원을 참조하십시오.

```
// Receive a message from 'MyQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and display the text
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    System.out.println("Group id: " +
 receivedMessage.getStringProperty("JMSXGroupID"));
    System.out.println("Message deduplication id: " +
 receivedMessage.getStringProperty("JMS_SQS_DeduplicationId"));
    System.out.println("Message sequence number: " +
 receivedMessage.getStringProperty("JMS_SQS_SequenceNumber"));
```

```
}
```

3. 연결과 세션을 종료합니다.

```
// Close the connection (and the session).
connection.close();
```

출력 결과는 다음과 비슷합니다:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

> **Note**
>
> Spring Framework를 사용하여 이 객체를 초기화할 수 있습니다.
> 추가 정보는 `SpringExampleConfiguration.xml`, `SpringExample.java` 및 `ExampleConfiguration.java` 단원의 `ExampleCommon.java`와 Amazon SQS 표준 대기열에 JMS를 사용하는 Java 작업 예제에 있는 기타 헬퍼 클래스를 참조하십시오.

객체 전송 및 수신 예제 전체를 보려면, TextMessageSender.java 및 SyncMessageReceiver.java를 참조하십시오.

메시지 비동기식 수신

Amazon SQS Java 메시징 라이브러리 사용의 예제에서 메시지는 `MyQueue`로 전송되고 동기식으로 수신됩니다.

다음 예제는 리스너를 통해 비동기식으로 메시지를 수신하는 방법을 나타냅니다.

1. `MessageListener` 인터페이스를 구현합니다.

```
class MyListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to
 screen.
```

```
            System.out.println("Received: " + ((TextMessage) message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

메시지를 수신하면 `MessageListener` 인터페이스의 `onMessage` 메서드가 호출됩니다. 이 리스너 구현에서 메시지에 저장된 텍스트가 인쇄됩니다.

2. 소비자에서 `receive` 메서드를 명시적으로 호출하지 않고, 소비자의 메시지 리스너를 `MyListener` 구현의 인스턴스로 설정합니다. 기본 스레드는 1초간 대기합니다.

```
// Create a consumer for the 'MyQueue'.
MessageConsumer consumer = session.createConsumer(queue);

// Instantiate and set the message listener for the consumer.
consumer.setMessageListener(new MyListener());

// Start receiving incoming messages.
connection.start();

// Wait for 1 second. The listener onMessage() method is invoked when a message is
 received.
Thread.sleep(1000);
```

나머지 단계는 Amazon SQS Java 메시징 라이브러리 사용 예제의 단계와 동일합니다. 비동기식 소비자의 예제 전체는 `AsyncMessageReceiver.java`의 Amazon SQS 표준 대기열에 JMS를 사용하는 Java 작업 예제를 참조하십시오.

이 예제의 출력은 다음과 비슷합니다.

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

클라이언트 승인 모드 사용

Amazon SQS Java 메시징 라이브러리 사용의 예제는 모든 수신 메시지가 자동으로 승인되고 나서 기본 Amazon SQS 대기열에서 삭제되는 `AUTO_ACKNOWLEDGE` 모드를 사용합니다.

1. 메시지를 처리한 후 명시적으로 승인하려면, CLIENT_ACKNOWLEDGE 모드로 세션을 생성해야 합니다.

```
// Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

2. 메시지를 수신하면 메시지를 표시한 후 명시적으로 승인합니다.

```
// Cast the received message as TextMessage and print the text to screen. Also
 acknowledge the message.
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    receivedMessage.acknowledge();
    System.out.println("Acknowledged: " + message.getJMSMessageID());
}
```

> **Note**
>
> 이 모드에서 메시지를 승인하면, 이 메시지를 이전에 받은 모든 메시지도 명시적으로 승인됩니다. 예를 들어, 메시지 10개를 수신하고 10번째(메시지 수신 순서) 메시지만을 승인하면, 이전의 9개 메시지 전체도 승인됩니다.

나머지 단계는 Amazon SQS Java 메시징 라이브러리 사용 예제의 단계와 동일합니다. 클라이언트 승인 모드를 사용하는 비동기식 소비자의 예제 전체는 SyncMessageReceiverClientAcknowledge.java의 Amazon SQS 표준 대기열에 JMS를 사용하는 Java 작업 예제를 참조하십시오.

이 예제의 출력은 다음과 비슷합니다.

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

정렬되어 있지 않은 승인 모드 사용

CLIENT_ACKNOWLEDGE 모드를 사용하면, 명시적으로 승인한 메시지 이전에 받은 모든 메시지가 자동으로 승인됩니다. 자세한 내용은 클라이언트 승인 모드 사용 단원을 참조하십시오.

Amazon SQS Java 메시징 라이브러리는 또 다른 승인 모드를 제공합니다. UNORDERED_ACKNOWLEDGE 모드를 사용하면, 수신한 모든 메시지가 수신 순서에 상관 없이 개별적 및 명시적으로 승인됩니다. 이렇게 하려면 UNORDERED_ACKNOWLEDGE 모드를 사용하여 세션을 생성합니다.

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

나머지 단계는 클라이언트 승인 모드 사용 예제의 단계와 동일합니다. UNORDERED_ACKNOWLEDGE 모드에서 동기식 소비자의 예제 전체는 SyncMessageReceiverUnorderedAcknowledge.java를 참조하십시오.

이 예제의 출력은 다음과 비슷합니다.

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Java 메시지 서비스를 다른 Amazon SQS 클라이언트와 함께 사용

AWS SDK와 함께 Amazon SQS Java Message Service(JMS) 클라이언트를 사용하면 Amazon SQS 메시지 크기가 256KB로 제한됩니다. 그러나 Amazon SQS 클라이언트를 사용하여 JMS 공급자를 생성할 수 있습니다. 예를 들어, JMS 클라이언트와 함께 Java용 Amazon SQS 확장 클라이언트 라이브러리를 사용하여 Amazon S3의 메시지 페이로드(최대 2GB)에 대한 참조가 들어 있는 Amazon SQS 메시지를 전송할 수 있습니다. 자세한 내용은 Java 및 Amazon S3를 사용하여 대규모 Amazon SQS 메시지 관리 단원을 참조하십시오.

다음의 Java 코드 예제에서는 확장 클라이언트 라이브러리에 대한 JMS 공급자를 생성합니다.

이 예제를 테스트하기 전에 JMS 및 Amazon SQS 작업을 위한 사전 조건에서 사전 조건을 참조하세요.

```
AmazonS3 s3 = new AmazonS3Client(credentials);
Region s3Region = Region.getRegion(Regions.US_WEST_2);
s3.setRegion(s3Region);

// Set the Amazon S3 bucket name, and set a lifecycle rule on the bucket to
// permanently delete objects a certain number of days after each object's creation
 date.
```

```
// Next, create the bucket, and enable message objects to be stored in the bucket.
BucketLifecycleConfiguration.Rule expirationRule = new
 BucketLifecycleConfiguration.Rule();
expirationRule.withExpirationInDays(14).withStatus("Enabled");
BucketLifecycleConfiguration lifecycleConfig = new
 BucketLifecycleConfiguration().withRules(expirationRule);

s3.createBucket(s3BucketName);
s3.setBucketLifecycleConfiguration(s3BucketName, lifecycleConfig);
System.out.println("Bucket created and configured.");

// Set the SQS extended client configuration with large payload support enabled.
ExtendedClientConfiguration extendedClientConfig = new ExtendedClientConfiguration()
    .withLargePayloadSupportEnabled(s3, s3BucketName);

AmazonSQS sqsExtended = new AmazonSQSExtendedClient(new AmazonSQSClient(credentials),
 extendedClientConfig);
Region sqsRegion = Region.getRegion(Regions.US_WEST_2);
sqsExtended.setRegion(sqsRegion);
```

다음의 Java 코드 예제에서는 연결 팩토리를 생성합니다.

```
// Create the connection factory using the environment variable credential provider.
// Pass the configured Amazon SQS Extended Client to the JMS connection factory.
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        sqsExtended
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

Amazon SQS 표준 대기열에 JMS를 사용하는 Java 작업 예제

다음 코드 예시에서는 Amazon SQS 표준 대기열과 함께 Java 메시지 서비스(JMS)를 사용하는 방법을 보여줍니다. FIFO 대기열 작업에 대한 자세한 내용은 FIFO 대기열을 생성하려면, 메시지 동기식 전송 및 메시지 동기식 수신 섹션을 참조하세요. (메시지를 동기적으로 수신하는 것은 표준 대기열과 FIFO 대기열에서 동일합니다. 그러나 FIFO 대기열의 메시지에는 더 많은 속성이 포함되어 있습니다.)

다음 예제를 테스트하기 전에 JMS 및 Amazon SQS 작업을 위한 사전 조건에서 사전 조건을 참조하세요.

ExampleConfiguration.java

다음 Java SDK v 1.x 코드 예제에서는 다른 Java 예제에서 사용되는 기본 대기열 이름, 리전 및 자격 증명을 설정한 것입니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleConfiguration {
    public static final String DEFAULT_QUEUE_NAME = "SQSJMSClientExampleQueue";

    public static final Region DEFAULT_REGION = Region.getRegion(Regions.US_EAST_2);

    private static String getParameter( String args[], int i ) {
        if( i + 1 >= args.length ) {
            throw new IllegalArgumentException( "Missing parameter for " + args[i] );
        }
        return args[i+1];
    }

    /**
     * Parse the command line and return the resulting config. If the config parsing
 fails
     * print the error and the usage message and then call System.exit
     *
     * @param app the app to use when printing the usage string
     * @param args the command line arguments
     * @return the parsed config
     */
    public static ExampleConfiguration parseConfig(String app, String args[]) {
```

```
        try {
            return new ExampleConfiguration(args);
        } catch (IllegalArgumentException e) {
            System.err.println( "ERROR: " + e.getMessage() );
            System.err.println();
            System.err.println( "Usage: " + app + " [--queue <queue>] [--region
<region>] [--credentials <credentials>] ");
            System.err.println( "  or" );
            System.err.println( "       " + app + " <spring.xml>" );
            System.exit(-1);
            return null;
        }
    }

    private ExampleConfiguration(String args[]) {
        for( int i = 0; i < args.length; ++i ) {
            String arg = args[i];
            if( arg.equals( "--queue" ) ) {
                setQueueName(getParameter(args, i));
                i++;
            } else if( arg.equals( "--region" ) ) {
                String regionName = getParameter(args, i);
                try {
                    setRegion(Region.getRegion(Regions.fromName(regionName)));
                } catch( IllegalArgumentException e ) {
                    throw new IllegalArgumentException( "Unrecognized region " +
regionName );
                }
                i++;
            } else if( arg.equals( "--credentials" ) ) {
                String credsFile = getParameter(args, i);
                try {
                    setCredentialsProvider( new
PropertiesFileCredentialsProvider(credsFile) );
                } catch (AmazonClientException e) {
                    throw new IllegalArgumentException("Error reading credentials from
" + credsFile, e );
                }
                i++;
            } else {
                throw new IllegalArgumentException("Unrecognized option " + arg);
            }
        }
    }
```

```
    private String queueName = DEFAULT_QUEUE_NAME;
    private Region region = DEFAULT_REGION;
    private AWSCredentialsProvider credentialsProvider = new
 DefaultAWSCredentialsProviderChain();

    public String getQueueName() {
        return queueName;
    }

    public void setQueueName(String queueName) {
        this.queueName = queueName;
    }

    public Region getRegion() {
        return region;
    }

    public void setRegion(Region region) {
        this.region = region;
    }

    public AWSCredentialsProvider getCredentialsProvider() {
        return credentialsProvider;
    }

    public void setCredentialsProvider(AWSCredentialsProvider credentialsProvider) {
        // Make sure they're usable first
        credentialsProvider.getCredentials();
        this.credentialsProvider = credentialsProvider;
    }
}
```

TextMessageSender.java

다음 Java 코드 예제는 문자 메시지 생성자를 생성합니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
```

```
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class TextMessageSender {
    public static void main(String args[]) throws JMSException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("TextMessageSender", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer =
 session.createProducer( session.createQueue( config.getQueueName() ) );

        sendMessages(session, producer);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static void sendMessages( Session session, MessageProducer producer ) {
```

```
        BufferedReader inputReader = new BufferedReader(
            new InputStreamReader( System.in, Charset.defaultCharset() ) );

        try {
            String input;
            while( true ) {
                System.out.print( "Enter message to send (leave empty to exit): " );
                input = inputReader.readLine();
                if( input == null || input.equals("" ) ) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
                System.out.println( "Send message " + message.getJMSMessageID() );
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
            System.err.println( "Failed reading input: " + e.getMessage() );
        } catch (JMSException e) {
            System.err.println( "Failed sending message: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

SyncMessageReceiver.java

다음 Java 코드 예제는 동기식 메시지 소비자를 생성합니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
```

```
 */

public class SyncMessageReceiver {
public static void main(String args[]) throws JMSException {
    ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

    ExampleCommon.setupLogging();

    // Create the connection factory based on the config
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                    .withRegion(config.getRegion().getName())
                    .withCredentials(config.getCredentialsProvider())
            );

    // Create the connection
    SQSConnection connection = connectionFactory.createConnection();

    // Create the queue if needed
    ExampleCommon.ensureQueueExists(connection, config.getQueueName());

    // Create the session
    Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
    MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

    connection.start();

    receiveMessages(session, consumer);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println( "Connection closed" );
}

private static void receiveMessages( Session session, MessageConsumer consumer ) {
    try {
        while( true ) {
            System.out.println( "Waiting for messages");
            // Wait 1 minute for a message
            Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
            if( message == null ) {
```

```
                System.out.println( "Shutting down after 1 minute of silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        }
    } catch (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
```

AsyncMessageReceiver.java

다음 Java 코드 예제는 비동기식 메시지 소비자를 생성합니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class AsyncMessageReceiver {
    public static void main(String args[]) throws JMSException, InterruptedException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
```

```
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

        // No messages are processed until this is called
        connection.start();

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }


    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();

        public void waitForOneMinuteOfSilence() throws InterruptedException {
            for(;;) {
                long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
                long remainingTillOneMinuteOfSilence =
                    TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
                if( remainingTillOneMinuteOfSilence < 0 ) {
                    break;
                }
```

```
                TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
            }
        }


        @Override
        public void onMessage(Message message) {
            try {
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message " +
 message.getJMSMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSException e) {
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}
```

SyncMessageReceiverClientAcknowledge.java

다음 Java 코드 예제는 클라이언트 승인 모드를 사용하는 비동기식 소비자를 생성합니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

/**
```

```
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also
 acknowledged.
 *
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
```

```
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with client acknowledge mode
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
 session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
 session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
 in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
 receiving no messages since
        // we have acknowledged the second message. Although we didn't explicitly
 acknowledge the first message,
        // in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the
 explicitly acknowledged message
        // are also acknowledged. Therefore, we have implicitly acknowledged the first
 message.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }
```

```
    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
 messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
 "Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
```

```
}
```

SyncMessageReceiverUnorderedAcknowledge.java

다음 Java 코드 예제는 정렬되어 있지 않은 승인 모드를 사용하는 비동기식 소비자를 생성합니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for
 received messages. This example
 * complements the example given in {@link SyncMessageReceiverClientAcknowledge} for
 CLIENT_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that the first message received in the
 prior attempt is returned again
 * for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
 explicitly acknowledged no matter what
 * the order they're received.
 *
 * This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverClientAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {
```

```
   // Visibility time-out for the queue. It must match to the one set for the queue
for this example to work.
   private static final long TIME_OUT_SECONDS = 1;

   public static void main(String args[]) throws JMSException, InterruptedException {
       // Create the configuration for the example
       ExampleConfiguration config =
ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);

       // Setup logging for the example
       ExampleCommon.setupLogging();

       // Create the connection factory based on the config
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               AmazonSQSClientBuilder.standard()
                       .withRegion(config.getRegion().getName())
                       .withCredentials(config.getCredentialsProvider())
               );

       // Create the connection
       SQSConnection connection = connectionFactory.createConnection();

       // Create the queue if needed
       ExampleCommon.ensureQueueExists(connection, config.getQueueName());

       // Create the session  with unordered acknowledge mode
       Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

       // Create the producer and consume
       MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
       MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

       // Open the connection
       connection.start();

       // Send two text messages
       sendMessage(producer, session, "Message 1");
       sendMessage(producer, session, "Message 2");
```

```
        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving the first message since
        // we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE
mode, all the messages must
        // be explicitly acknowledged.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
(TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
"Queue is empty!" is
     * printed.
     *
```

```
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

SpringExampleConfiguration.xml

다음 XML 코드 예제는 SpringExample.java의 빈(bean) 구성 파일입니다.

```
<!--
    Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.

    Licensed under the Apache License, Version 2.0 (the "License").
    You may not use this file except in compliance with the License.
    A copy of the License is located at

    https://aws.amazon.com/apache2.0

    or in the "license" file accompanying this file. This file is distributed
    on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
    express or implied. See the License for the specific language governing
    permissions and limitations under the License.
-->
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
schema/util/spring-util-3.0.xsd
    ">

    <bean id="CredentialsProviderBean"
 class="com.amazonaws.auth.DefaultAWSCredentialsProviderChain"/>

    <bean id="ClientBuilder" class="com.amazonaws.services.sqs.AmazonSQSClientBuilder"
 factory-method="standard">
        <property name="region" value="us-east-2"/>
        <property name="credentials" ref="CredentialsProviderBean"/>
    </bean>

    <bean id="ProviderConfiguration"
 class="com.amazon.sqs.javamessaging.ProviderConfiguration">
        <property name="numberOfMessagesToPrefetch" value="5"/>
    </bean>

    <bean id="ConnectionFactory"
 class="com.amazon.sqs.javamessaging.SQSConnectionFactory">
        <constructor-arg ref="ProviderConfiguration" />
        <constructor-arg ref="ClientBuilder" />
    </bean>

    <bean id="Connection" class="javax.jms.Connection"
        factory-bean="ConnectionFactory"
        factory-method="createConnection"
        init-method="start"
        destroy-method="close" />

    <bean id="QueueName" class="java.lang.String">
        <constructor-arg value="SQSJMSClientExampleQueue"/>
    </bean>
</beans>
```

SpringExample.java

다음 Java 코드 예제는 객체를 초기화하는 빈(bean) 구성 파일을 사용합니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SpringExample {
    public static void main(String args[]) throws JMSException {
        if( args.length != 1 || !args[0].endsWith(".xml")) {
            System.err.println( "Usage: " + SpringExample.class.getName() + " <spring
 config.xml>" );
            System.exit(1);
        }

        File springFile = new File( args[0] );
        if( !springFile.exists() || !springFile.canRead() ) {
            System.err.println( "File " + args[0] + " doesn't exist or isn't
 readable.");
            System.exit(2);
        }

        ExampleCommon.setupLogging();

        FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext( "file://" +
 springFile.getAbsolutePath() );

        Connection connection;
        try {
```

```
            connection = context.getBean(Connection.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the JMS connection to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        String queueName;
        try {
            queueName = context.getBean("QueueName", String.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the name of the queue to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        if( connection instanceof SQSConnection ) {
            ExampleCommon.ensureQueueExists( (SQSConnection) connection, queueName );
        }

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
session.createConsumer( session.createQueue( queueName) );

        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                    break;
                }
```

```
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message" );
            }
        } catch (JMSException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

ExampleCommon.java

다음 Java 코드 예제는 Amazon SQS 대기열이 존재하는지 확인하고 나서 존재하지 않으면 대기열 하나를 생성합니다. 또한 예제 로깅 코드를 포함합니다.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is
 run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName)
 throws JMSException {
        AmazonSQSMessagingClientWrapper client =
 connection.getWrappedAmazonSQSClient();
```

```
        /**
         * In most cases, you can do this with just a createQueue call, but
 GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
 queue
         * already exists. Also many users and roles have permission to call
 GetQueueUrl
         * but don't have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: ");
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ){
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up to 256k so
 that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
            System.out.println( "\t" +  Base64.encodeAsString( bytes ) );
        } else if( message instanceof ObjectMessage ) {
            ObjectMessage objMessage = (ObjectMessage) message;
            System.out.println( "\t" + objMessage.getObject() );
        }
    }
}
```

Amazon SQS 지원 JMS 1.1 구현

Amazon SQS 자바 메시징 라이브러리는 다음과 같은 JMS 1.1 구현을 지원합니다. Amazon SQS Java 메시징 라이브러리의 지원 기능 및 역량에 대한 자세한 내용은 Amazon SQS FAQ를 참조하세요.

지원되는 공통 인터페이스

- `Connection`
- `ConnectionFactory`
- `Destination`
- `Session`
- `MessageConsumer`
- `MessageProducer`

지원되는 메시지 유형

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

지원되는 메시지 승인 모드

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

> **Note**
>
> `UNORDERED_ACKNOWLEDGE` 모드는 JMS 1.1 사양의 일부가 아닙니다. 이 모드를 통해 Amazon SQS에서 JMS 클라이언트가 메시지를 명시적으로 승인하도록 허용합니다.

JMS 정의된 헤더 및 예약된 속성

메시지 전송의 경우

메시지를 전송하는 경우 사용자는 각 메시지에 대해 다음과 같은 헤더 및 속성을 설정할 수 있습니다.

- `JMSXGroupID`(FIFO 대기열에서 필수, 표준 대기열에서는 허용되지 않음)
- `JMS_SQS_DeduplicationId`(FIFO 대기열에서 선택 사항, 표준 대기열에서는 허용되지 않음)

사용자가 메시지를 전송한 후 Amazon SQS는 각 메시지에 대해 다음과 같은 헤더 및 속성을 설정합니다.

- `JMSMessageID`
- `JMS_SQS_SequenceNumber`(FIFO 대기열에만 해당)

메시지 수신의 경우

사용자가 메시지를 수신한 후 Amazon SQS는 각 메시지에 대해 다음과 같은 헤더 및 속성을 설정합니다.

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID`(FIFO 대기열에만 해당)
- `JMS_SQS_DeduplicationId`(FIFO 대기열에만 해당)
- `JMS_SQS_SequenceNumber`(FIFO 대기열에만 해당)

Amazon SQS 자습서

이 주제에서는 Amazon SQS 기능을 탐색하는 데 도움이 되는 자습서를 제공합니다.

자습서

- 를 사용하여 Amazon SQS 대기열 생성 AWS CloudFormation
- 자습서: Amazon Virtual Private Cloud에서 Amazon SQS 대기열로 메시지 보내기

를 사용하여 Amazon SQS 대기열 생성 AWS CloudFormation

AWS CloudFormation 콘솔을 JSON 또는 YAML 템플릿과 함께 사용하여 Amazon SQS 대기열을 생성합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 템플릿 및 리소스 작업을 AWS CloudFormation 참조하세요. `AWS::SQS::Queue`

AWS CloudFormation 를 사용하여 Amazon SQS 대기열을 생성합니다.

1. 다음 JSON 코드를 `MyQueue.json`이라는 파일에 복사합니다. 표준 대기열을 생성하려면 `FifoQueue` 및 `ContentBasedDeduplication` 속성을 생략합니다. 콘텐츠 기반 중복 제거에 대한 자세한 내용은 Amazon SQS의 정확히 1회 처리 단원을 참조하십시오.

> **Note**
>
> FIFO 대기열의 이름은 `.fifo` 접미사로 끝나야 합니다.

```
{
   "AWSTemplateFormatVersion": "2010-09-09",
   "Resources": {
      "MyQueue": {
         "Properties": {
            "QueueName": "MyQueue.fifo",
            "FifoQueue": true,
            "ContentBasedDeduplication": true
             },
         "Type": "AWS::SQS::Queue"
         }
      },
   "Outputs": {
```

```
        "QueueName": {
           "Description": "The name of the queue",
           "Value": {
              "Fn::GetAtt": [
                 "MyQueue",
                 "QueueName"
              ]
           }
        },
        "QueueURL": {
           "Description": "The URL of the queue",
           "Value": {
              "Ref": "MyQueue"
           }
        },
        "QueueARN": {
           "Description": "The ARN of the queue",
           "Value": {
              "Fn::GetAtt": [
                 "MyQueue",
                 "Arn"
              ]
           }
        }
    }
}
```

2. AWS CloudFormation 콘솔에 로그인한 후 스택 생성을 선택합니다.
3. 템플릿 지정 패널에서 템플릿 파일 업로드를 선택하고, `MyQueue.json` 파일을 선택한 후 다음을 선택합니다.
4. 세부 정보 지정 페이지에서 스택 이름에 `MyQueue`를 입력한 후 다음을 선택합니다.
5. 옵션 페이지에서 다음을 선택합니다.
6. 검토 페이지에서 생성을 선택합니다.

   AWS CloudFormation 가 `MyQueue` 스택 생성을 시작하고 CREATE_IN_PROGRESS 상태를 표시합니다. 이 과정이 완료되면 AWS CloudFormation 에는 CREATE_COMPLETE 상태가 표시됩니다.

7. (선택 사항) 대기열의 이름, URL 및 ARN을 표시하려면, 스택의 이름을 선택하고 나서 다음 페이지에서 [Outputs] 섹션을 확장합니다.

자습서: Amazon Virtual Private Cloud에서 Amazon SQS 대기열로 메시지 보내기

이 자습서에서는 안전한 프라이빗 네트워크를 통해 Amazon SQS 대기열로 메시지를 보내는 방법을 보여줍니다. 네트워크에는 다음이 포함됩니다.

- Amazon EC2 인스턴스가 포함된 VPC입니다.
- Amazon EC2 인스턴스가 퍼블릭 인터넷을 사용하지 않고 Amazon SQS에 연결할 수 있는 인터페이스 VPC 엔드포인트입니다.

완전 프라이빗 네트워크에서도 Amazon EC2 인스턴스에 연결하고 Amazon SQS 대기열로 메시지를 보낼 수 있습니다. 자세한 내용은 Amazon SQS용 Amazon Virtual Private Cloud 엔드포인트 단원을 참조하십시오.

Important

- Amazon Virtual Private Cloud는 HTTPS Amazon SQS 엔드포인트에서만 사용할 수 있습니다.
- Amazon VPC에서 메시지를 보내도록 Amazon SQS를 구성할 때는 프라이빗 DNS를 활성화하고 듀얼 스택 엔드포인트에 `sqs.us-east-2.api.aws` 대해 `sqs.us-east-2.amazonaws.com` 또는 형식으로 엔드포인트를 지정해야 합니다.
- Amazon SQS는 엔드포인트 서비스를 사용하여 PrivateLink를 통해 FIPS `com.amazonaws.region.sqs-fips` 엔드포인트도 지원합니다. 형식으로 FIPS 엔드포인트에 연결할 수 있습니다`sqs-fips.region.amazonaws.com`.
- Amazon Virtual Private Cloud에서 듀얼 스택 엔드포인트를 사용하는 경우 IPv4 및 IPv6를 사용하여 요청이 전송됩니다.
- 프라이빗 DNS는 `queue.amazonaws.com` 또는 `us-east-2.queue.amazonaws.com` 같은 레거시 엔드포인트를 지원하지 않습니다.

1단계: Amazon EC2 키 페어 생성

키 페어는 Amazon EC2 인스턴스에 연결할 수 있게 해줍니다. 키 페어는 로그인 정보를 암호화하는 퍼블릭 키와 로그인 정보 암호를 해독하는 프라이빗 키로 구성됩니다.

1. Amazon EC2 콘솔에 로그인합니다.
2. 탐색 창의 네트워크 및 보안에서 키 페어를 선택합니다.
3. 키 페어 생성(Create Key Pair)을 선택합니다.
4. 키 페어 생성 대화 상자에서 키 페어 이름에`SQS-VPCE-Tutorial-Key-Pair`를 입력한 후 생성을 선택합니다.
5. 브라우저가 프라이빗 키 파일 `SQS-VPCE-Tutorial-Key-Pair.pem`을 자동으로 다운로드합니다.

   > **Important**
   > 이 파일을 안전한 위치에 저장합니다. EC2는 동일한 키 페어에 대해 `.pem` 파일을 두 번 생성하지 않습니다.

6. SSH 클라이언트가 EC2 인스턴스에 연결하도록 허용하려면 최종 사용자만 읽기 권한을 가질 수 있도록 프라이빗 키 파일에 대한 권한을 설정합니다. 예:

```
chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
```

2단계: AWS 리소스 생성

필요한 인프라를 설정하려면 Amazon EC2 인스턴스 및 Amazon SQS 대기열과 같은 AWS 리소스로 구성된 스택을 생성하기 위한 청사진인 an AWS CloudFormation template를 사용해야 합니다.

이 자습서에서 사용할 스택으로 다음과 같은 리소스가 있습니다.

- VPC 및 이에 연결된 네트워킹 리소스(예: 서브넷, 보안 그룹, 인터넷 게이트웨이, 라우팅 테이블)
- VPC 서브넷으로 시작된 Amazon EC2 인스턴스
- Amazon SQS 대기열

1. GitHubSQS-VPCE-Tutorial-CloudFormation.yaml에서 라는 AWS CloudFormation 템플릿을 다운로드합니다.
2. AWS CloudFormation 콘솔에 로그인합니다.
3. 스택 생성을 선택합니다.
4. 템플릿 선택 페이지에서 Amazon S3에 템플릿 업로드를 선택하고 `SQS-VPCE-SQS-Tutorial-CloudFormation.yaml` 파일을 선택한 후 다음을 선택합니다.
5. 세부 정보 지정 페이지에서 다음 작업을 수행합니다.
   a. 스택 이름에 `SQS-VPCE-Tutorial-Stack`을 입력합니다.
   b. KeyName에서 SQS-VPCE-Tutorial-Key-Pair를 선택합니다.
   c. 다음을 선택합니다.
6. 옵션 페이지에서 다음을 선택합니다.
7. 검토 페이지의 기능 섹션에서 AWS CloudFormation이 사용자 지정 이름으로 IAM 리소스를 생성할 수 있음을 승인합니다를 선택한 다음 생성을 선택합니다.

AWS CloudFormation 가 스택 생성을 시작하고 CREATE_IN_PROGRESS 상태를 표시합니다. 이 과정이 완료되면 AWS CloudFormation 에는 CREATE_COMPLETE 상태가 표시됩니다.

3단계: EC2 인스턴스에 공개적으로 액세스할 수 없는지 확인

AWS CloudFormation 템플릿은 라는 EC2 인스턴스를 VPC`SQS-VPCE-Tutorial-EC2-Instance`로 시작합니다. 이 EC2 인스턴스는 아웃바운드 트래픽을 허용하지 않으며 Amazon SQS로 메시지를 전송할 수 없습니다. 이를 확인하려면, 인스턴스에 연결하고 퍼블릭 엔드포인트에 연결을 시도한 후 Amazon SQS에 메시지를 게시해 보아야 합니다.

1. Amazon EC2 콘솔에 로그인합니다.
2. 탐색 메뉴의 인스턴스에서 인스턴스를 선택합니다.
3. SQS-VPCE-Tutorial-EC2Instance를 선택합니다.
4. 퍼블릭 DNS 아래에 호스트 이름을 복사합니다. 예: ec2-203-0-113-0.us-west-2.compute.amazonaws.com.
5. 앞서 생성한 키 페어를 포함하는 디렉터리에서, 다음과 같은 명령을 사용하여 인스턴스에 연결합니다.

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. 임의의 퍼블릭 엔드포인트에 연결해 봅니다. 예:

```
ping amazon.com
```

예상대로 연결 시도가 실패합니다.

7. Amazon SQS 콘솔에 로그인합니다.
8. 대기열 목록에서 VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK와 같이 AWS CloudFormation 템플릿으로 생성된 대기열을 선택합니다.
9. 세부 정보 테이블에서 URL을 복사합니다(예: https://sqs.us-east-2.amazonaws.com/123456789012/).
10. EC2 인스턴스에서 다음과 같은 명령을 사용하여 대기열에 메시지를 게시해 봅니다.

```
aws sqs send-message --region us-east-2 --endpoint-url https://sqs.us-
east-2.amazonaws.com/ --queue-url https://sqs.us-east-2.amazonaws.com/123456789012/
 --message-body "Hello from Amazon SQS."
```

예상대로 전송 시도가 실패합니다.

> **Important**
> 나중에 Amazon SQS용 VPC 엔드포인트를 생성하면 전송 시도가 성공할 것입니다.

4단계: Amazon SQS용 Amazon VPC 엔드포인트 생성

VPC를 Amazon SQS에 연결하려면 인터페이스 VPC 엔드포인트를 정의해야 합니다. 엔드포인트를 추가한 후에는 VPC의 EC2 인스턴스에서 Amazon SQS API를 사용할 수 있습니다. 이렇게 하면 퍼블릭 인터넷을 통과하지 않고도 AWS 네트워크 내의 대기열로 메시지를 보낼 수 있습니다.

> **Note**
> EC2 인스턴스는 여전히 인터넷의 다른 AWS 서비스 및 엔드포인트에 액세스할 수 없습니다.

1. Amazon VPC 콘솔에 로그인합니다.
2. 탐색 메뉴에서 엔드포인트를 선택합니다.
3. 엔드포인트 생성을 선택합니다.

4. 엔드포인트 생성 페이지의 서비스 이름에서 Amazon SQS의 서비스 이름을 선택합니다.

> **Note**
> 서비스 이름은 현재 AWS 리전에 따라 다릅니다. 예를 들어, 미국 동부(오하이오)에서는 서비스 이름이 com.amazonaws.*us-east-2*.sqs입니다.

5. VPC에서 SQS-VPCE-Tutorial-VPC를 선택합니다.
6. 서브넷에서 서브넷 ID에 SQS-VPCE-Tutorial-Subnet이 포함된 서브넷을 선택합니다.
7. 보안 그룹에서 Select security groups(보안 그룹 선택)를 선택한 다음 그룹 이름에 SQS VPCE Tutorial Security Group이 포함된 보안 그룹을 선택합니다.
8. Create endpoint(엔드포인트 생성)을 선택합니다.

   인터페이스 VPC 엔드포인트가 생성되고 ID가 표시됩니다. 예: vpce-0ab1cdef2ghi3j456k.
9. 닫기를 선택하세요.

   Amazon VPC 콘솔에서 엔드포인트 페이지가 열립니다.

Amazon VPC가 엔드포인트를 생성하기 시작하고 대기 중 상태를 표시합니다. 이 과정이 완료되면 Amazon VPC에는 사용 가능 상태가 표시됩니다.

5단계: Amazon SQS 대기열에 메시지 전송

이제 VPC에 Amazon SQS용 엔드포인트가 포함되므로 EC2 인스턴스에 연결하여 대기열로 메시지를 보낼 수 있습니다.

1. EC2 인스턴스에 다시 연결합니다. 예:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

2. 다음과 같은 명령을 사용하여 다시 대기열에 메시지를 게시해 봅니다.

```
aws sqs send-message --region us-east-2 --endpoint-url https://sqs.us-
east-2.amazonaws.com/ --queue-url https://sqs.us-east-2.amazonaws.com/123456789012/
 --message-body "Hello from Amazon SQS."
```

   전송 시도가 성공하고 메시지 본문의 MD5 다이제스트와 메시지 ID가 표시됩니다. 예:

```
{
 "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
 "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

AWS CloudFormation 템플릿에서 생성한 대기열(예: VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK)에서 메시지를 수신하고 삭제하는 방법에 대한 자세한 내용은 섹션을 참조하세요Amazon SQS에서 메시지 수신 및 삭제 .

리소스 삭제에 대한 자세한 내용은 다음을 참조하십시오.

- Amazon VPC 사용 설명서의 VPC 엔드포인트 삭제
- Amazon SQS 대기열 삭제
- Amazon EC2 사용 설명서의 인스턴스 종료
- Amazon VPC 사용 설명서의 VPC 삭제
- AWS CloudFormation 사용 설명서의 AWS CloudFormation 콘솔에서 스택 삭제
- Amazon EC2 사용 설명서의 키 페어 삭제

AWS SDKs를 사용하는 Amazon SQS의 코드 예제

다음 코드 예제에서는 Amazon SQS를 AWS 소프트웨어 개발 키트(SDK)와 함께 사용하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 관련 시나리오의 컨텍스트에 따라 표시되며, 개별 서비스 함수를 직접적으로 호출하는 방법을 보여줍니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

시작

Hello Amazon SNS

다음 코드 예는 Amazon SQS를 사용하여 시작하는 방법을 보여줍니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
```

```
        var sqsClient = new AmazonSQSClient();

        Console.WriteLine($"Hello Amazon SQS! Following are some of your
 queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });

        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListQueues를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS sqs)
```

```
# Set this project's name.
project("hello_sqs")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if(WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif()

add_executable(${PROJECT_NAME}
        hello_sqs.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_sqs.cpp 소스 파일의 코드입니다.

```
#include <aws/core/Aws.h>
```

```
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ListQueuesRequest.h>
#include <iostream>

/*
 *  A "Hello SQS" starter application that initializes an Amazon Simple Queue
 Service
 *  (Amazon SQS) client and lists the SQS queues in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_sqs'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::SQS::SQSClient sqsClient(clientConfig);

        Aws::Vector<Aws::String> allQueueUrls;
        Aws::String nextToken; // Next token is used to handle a paginated
 response.
        do {
            Aws::SQS::Model::ListQueuesRequest request;

            Aws::SQS::Model::ListQueuesOutcome outcome =
 sqsClient.ListQueues(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::String> &pageOfQueueUrls =
 outcome.GetResult().GetQueueUrls();
                if (!pageOfQueueUrls.empty()) {
                    allQueueUrls.insert(allQueueUrls.cend(),
 pageOfQueueUrls.cbegin(),
                                        pageOfQueueUrls.cend());
                }
```

```
            }
            else {
                std::cerr << "Error with SQS::ListQueues. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
            nextToken = outcome.GetResult().GetNextToken();
        } while (!nextToken.empty());


        std::cout << "Hello Amazon SQS! You have " << allQueueUrls.size() << "
 queue"
                  << (allQueueUrls.size() == 1 ? "" : "s") << " in your account."
                  << std::endl;

        if (!allQueueUrls.empty()) {
            std::cout << "Here are your queue URLs." << std::endl;
            for (const Aws::String &queueUrl: allQueueUrls) {
                std::cout << "  * " << queueUrl << std::endl;
            }
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ListQueues를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
```

```
}
```

- API에 대한 세부 정보는AWS SDK for Go API 참조의 ListQueues를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.SqsException;
import software.amazon.awssdk.services.sqs.paginators.ListQueuesIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloSQS {
    public static void main(String[] args) {
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueues(sqsClient);
        sqsClient.close();
    }

    public static void listQueues(SqsClient sqsClient) {
        try {
```

```
            ListQueuesIterable listQueues = sqsClient.listQueuesPaginator();
            listQueues.stream()
                    .flatMap(r -> r.queueUrls().stream())
                    .forEach(content -> System.out.println(" Queue URL: " +
 content.toLowerCase()));

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListQueues를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 클라이언트를 초기화하고 대기열을 나열합니다.

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];
```

```
  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your
 account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListQueues를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package com.kotlin.sqs

import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.paginators.listQueuesPaginated
import kotlinx.coroutines.flow.transform

suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
```

```
            .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
            .collect { queue ->
                println("The Queue URL is $queue")
            }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ListQueues를 참조하십시오.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

`Package.swift` 파일.

```
import PackageDescription

let package = Package(
    name: "sqs-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
```

```
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sqs-basics",
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

Swift 소스 코드 `entry.swift`.

```
import ArgumentParser
import AWSClientRuntime
import AWSSQS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "sqs-basics",
        abstract: """
        This example shows how to list all of your available Amazon SQS queues.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
```

```
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }

            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
        }

        print("You have \(queues.count) queues:")
        for queue in queues {
            print("   \(queue)")
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API 세부 정보는 SDK for Swift API 참조의 ListQueues를 참조하세요. AWS

코드 예제

- AWS SDKs 사용한 Amazon SQS의 기본 예제
  - Hello Amazon SNS
  - AWS SDKs를 사용한 Amazon SQS 작업
    - CLI로 AddPermission 사용
    - AWS SDK 또는 CLI와 ChangeMessageVisibility 함께 사용
    - CLI로 ChangeMessageVisibilityBatch 사용
    - AWS SDK 또는 CLI와 CreateQueue 함께 사용
    - AWS SDK 또는 CLI와 DeleteMessage 함께 사용
    - AWS SDK 또는 CLI와 DeleteMessageBatch 함께 사용
    - AWS SDK 또는 CLI와 DeleteQueue 함께 사용
    - AWS SDK 또는 CLI와 GetQueueAttributes 함께 사용
    - AWS SDK 또는 CLI와 GetQueueUrl 함께 사용
    - CLI로 ListDeadLetterSourceQueues 사용
    - AWS SDK 또는 CLI와 ListQueues 함께 사용
    - CLI로 PurgeQueue 사용
    - AWS SDK 또는 CLI와 ReceiveMessage 함께 사용
    - CLI로 RemovePermission 사용
    - AWS SDK 또는 CLI와 SendMessage 함께 사용
    - AWS SDK 또는 CLI와 SendMessageBatch 함께 사용
    - AWS SDK 또는 CLI와 SetQueueAttributes 함께 사용
- AWS SDKs를 사용한 Amazon SQS 시나리오
  - Amazon SQS를 사용하여 메시지를 보내고 검색하는 웹 애플리케이션 생성
  - Step Functions를 사용하여 메신저 애플리케이션 생성
  - Amazon Textract 탐색기 애플리케이션 생성
  - AWS SDK를 사용하여 FIFO Amazon SNS 주제 생성 및 게시
  - AWS SDK를 사용하여 Amazon Rekognition을 사용하여 비디오에서 사람 및 객체 감지
  - AWS SDK와 함께 Amazon S3를 사용하여 대규모 Amazon SQS 메시지 관리 Amazon S3
  - AWS SDK를 사용하여 Amazon S3 이벤트 알림 수신 및 처리

  - AWS SDK를 사용하여 Amazon SNS 메시지를 Amazon SQS 대기열에 게시

- AWS SDK를 사용하여 Amazon SQS로 메시지 배치 전송 및 수신
- .NET용 AWS 메시지 처리 프레임워크를 사용하여 Amazon SQS 메시지 게시 및 수신
- Amazon SQS Java 메시징 라이브러리를 사용하여 Amazon SQS용 Java Message Service(JMS) 인터페이스로 작업
- AWS SDK를 사용하여 대기열 태그 및 Amazon SQS 작업
- Amazon SQS용 서버리스 예제
  - Amazon SQS 트리거에서 간접적으로 Lambda 함수 간접 호출
  - Amazon SQS 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

AWS SDKs 사용한 Amazon SQS의 기본 예제

다음 코드 예제에서는 AWS SDK와 함께 Amazon Simple Queue Service의 기본 사항을 사용하는 방법을 보여줍니다.

예시

- Hello Amazon SNS
- AWS SDKs를 사용한 Amazon SQS 작업
  - CLI로 AddPermission 사용
  - AWS SDK 또는 CLI와 ChangeMessageVisibility 함께 사용
  - CLI로 ChangeMessageVisibilityBatch 사용
  - AWS SDK 또는 CLI와 CreateQueue 함께 사용
  - AWS SDK 또는 CLI와 DeleteMessage 함께 사용
  - AWS SDK 또는 CLI와 DeleteMessageBatch 함께 사용
  - AWS SDK 또는 CLI와 DeleteQueue 함께 사용
  - AWS SDK 또는 CLI와 GetQueueAttributes 함께 사용
  - AWS SDK 또는 CLI와 GetQueueUrl 함께 사용
  - CLI로 ListDeadLetterSourceQueues 사용
  - AWS SDK 또는 CLI와 ListQueues 함께 사용
  - CLI로 PurgeQueue 사용
  - AWS SDK 또는 CLI와 ReceiveMessage 함께 사용
  - CLI로 RemovePermission 사용
  - AWS SDK 또는 CLI와 SendMessage 함께 사용

- AWS SDK 또는 CLI와 SendMessageBatch 함께 사용
- AWS SDK 또는 CLI와 SetQueueAttributes 함께 사용

Hello Amazon SNS

다음 코드 예는 Amazon SQS를 시작하는 방법을 보여줍니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
        var sqsClient = new AmazonSQSClient();

        Console.WriteLine($"Hello Amazon SQS! Following are some of your
 queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });
```

```
        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListQueues를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS sqs)

# Set this project's name.
project("hello_sqs")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
```

```
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if(WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif()

add_executable(${PROJECT_NAME}
        hello_sqs.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_sqs.cpp 소스 파일의 코드입니다.

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ListQueuesRequest.h>
#include <iostream>

/*
 *  A "Hello SQS" starter application that initializes an Amazon Simple Queue
 Service
 *  (Amazon SQS) client and lists the SQS queues in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_sqs'
 *
```

```
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::SQS::SQSClient sqsClient(clientConfig);

        Aws::Vector<Aws::String> allQueueUrls;
        Aws::String nextToken; // Next token is used to handle a paginated
 response.
        do {
            Aws::SQS::Model::ListQueuesRequest request;

            Aws::SQS::Model::ListQueuesOutcome outcome =
 sqsClient.ListQueues(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::String> &pageOfQueueUrls =
 outcome.GetResult().GetQueueUrls();
                if (!pageOfQueueUrls.empty()) {
                    allQueueUrls.insert(allQueueUrls.cend(),
 pageOfQueueUrls.cbegin(),
                                        pageOfQueueUrls.cend());
                }
            }
            else {
                std::cerr << "Error with SQS::ListQueues. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
            nextToken = outcome.GetResult().GetNextToken();
        } while (!nextToken.empty());


        std::cout << "Hello Amazon SQS! You have " << allQueueUrls.size() << "
 queue"
```

```
                    << (allQueueUrls.size() == 1 ? "" : "s") << " in your account."
                    << std::endl;

        if (!allQueueUrls.empty()) {
            std::cout << "Here are your queue URLs." << std::endl;
            for (const Aws::String &queueUrl: allQueueUrls) {
                std::cout << "  * " << queueUrl << std::endl;
            }
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ListQueues를 참조하십시오.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
```

```
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- API에 대한 세부 정보는AWS SDK for Go API 참조의 ListQueues를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.SqsException;
import software.amazon.awssdk.services.sqs.paginators.ListQueuesIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloSQS {
    public static void main(String[] args) {
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueues(sqsClient);
        sqsClient.close();
    }

    public static void listQueues(SqsClient sqsClient) {
        try {
            ListQueuesIterable listQueues = sqsClient.listQueuesPaginator();
            listQueues.stream()
                    .flatMap(r -> r.queueUrls().stream())
                    .forEach(content -> System.out.println(" Queue URL: " +
 content.toLowerCase()));
```

```
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListQueues를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 클라이언트를 초기화하고 대기열을 나열합니다.

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }
```

```
  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your
 account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListQueues를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package com.kotlin.sqs

import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.paginators.listQueuesPaginated
import kotlinx.coroutines.flow.transform

suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
            .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
            .collect { queue ->
                println("The Queue URL is $queue")
            }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ListQueues를 참조하십시오.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

`Package.swift` 파일.

```
import PackageDescription

let package = Package(
    name: "sqs-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sqs-basics",
```

```
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

Swift 소스 코드 `entry.swift`.

```
import ArgumentParser
import AWSClientRuntime
import AWSSQS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "sqs-basics",
        abstract: """
        This example shows how to list all of your available Amazon SQS queues.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
        )

        // Each time a page of results arrives, process its contents.
```

```
        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }

            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
        }

        print("You have \(queues.count) queues:")
        for queue in queues {
            print("   \(queue)")
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API 세부 정보는 SDK for Swift API 참조의 ListQueues를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDKs를 사용한 Amazon SQS 작업

다음 코드 예제에서는 AWS SDKs를 사용하여 개별 Amazon SQS 작업을 수행하는 방법을 보여줍니다. 각 예제에는 GitHub에 대한 링크가 포함되어 있습니다. 여기에서 코드 설정 및 실행에 대한 지침을 찾을 수 있습니다.

이들 발췌문은 Amazon SQS API를 직접적으로 호출하며, 컨텍스트에서 실행되어야 하는 더 큰 프로그램에서 발췌한 코드입니다. AWS SDKs를 사용한 Amazon SQS 시나리오 에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 Amazon Simple Queue Service API 참조를 참조하세요.

예시

- CLI로 AddPermission 사용
- AWS SDK 또는 CLI와 ChangeMessageVisibility 함께 사용
- CLI로 ChangeMessageVisibilityBatch 사용
- AWS SDK 또는 CLI와 CreateQueue 함께 사용
- AWS SDK 또는 CLI와 DeleteMessage 함께 사용
- AWS SDK 또는 CLI와 DeleteMessageBatch 함께 사용
- AWS SDK 또는 CLI와 DeleteQueue 함께 사용
- AWS SDK 또는 CLI와 GetQueueAttributes 함께 사용
- AWS SDK 또는 CLI와 GetQueueUrl 함께 사용
- CLI로 ListDeadLetterSourceQueues 사용
- AWS SDK 또는 CLI와 ListQueues 함께 사용
- CLI로 PurgeQueue 사용
- AWS SDK 또는 CLI와 ReceiveMessage 함께 사용
- CLI로 RemovePermission 사용
- AWS SDK 또는 CLI와 SendMessage 함께 사용
- AWS SDK 또는 CLI와 SendMessageBatch 함께 사용
- AWS SDK 또는 CLI와 SetQueueAttributes 함께 사용

CLI로 **AddPermission** 사용

다음 코드 예시는 AddPermission의 사용 방법을 보여 줍니다.

CLI

AWS CLI

대기열에 권한을 추가하는 방법

이 예제에서는 지정된 AWS 계정이 지정된 대기열로 메시지를 보낼 수 있도록 합니다.

명령:

```
aws sqs add-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue --aws-
account-ids 12345EXAMPLE --actions SendMessage
```

출력:

```
None.
```

- API 세부 정보는 AWS CLI 명령 참조의 AddPermission을 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1:이 예제에서는 지정된가 지정된 대기열에서 메시지를 보내 AWS 계정 도록 허용합니다.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 AddPermission을 참조하세요.

Tools for PowerShell V5

예제 1:이 예제에서는 지정된가 지정된 대기열에서 메시지를 보내 AWS 계정 도록 허용합니다.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 AddPermission을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **`ChangeMessageVisibility`** 함께 사용

다음 코드 예시는 `ChangeMessageVisibility`의 사용 방법을 보여 줍니다.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Changes the visibility timeout of a message in an Amazon Simple Queue Service
//! (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageReceiptHandle: A message receipt handle.
  \param visibilityTimeoutSeconds: Visibility timeout in seconds.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::changeMessageVisibility(
        const Aws::String &queue_url,
```

```
        const Aws::String &messageReceiptHandle,
        int visibilityTimeoutSeconds,
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ChangeMessageVisibilityRequest request;
    request.SetQueueUrl(queue_url);
    request.SetReceiptHandle(messageReceiptHandle);
    request.SetVisibilityTimeout(visibilityTimeoutSeconds);

    auto outcome = sqsClient.ChangeMessageVisibility(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully changed visibility of message " <<
                  messageReceiptHandle << " from queue " << queue_url <<
 std::endl;
    }
    else {
        std::cout << "Error changing visibility of message from queue "
                  << queue_url << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는AWS SDK for C++ API 참조의 ChangeMessageVisibility를 참조하세요.

CLI

AWS CLI

메시지의 제한 시간 가시성을 변경하는 방법

이 예시에서는 지정된 메시지의 제한 시간 가시성을 10시간(10시간 * 60분 * 60초)으로 변경합니다.

명령:

```
aws sqs change-message-visibility --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --
visibility-timeout 36000
```

출력:

```
None.
```

- API 세부 정보는 AWS CLI 명령 참조의 ChangeMessageVisibility를 참조하세요.

JavaScript

SDK for JavaScript (v3)

> **Note**
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 제한 시간 표시 여부를 변경합니다.

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);
```

```
  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ChangeMessageVisibility를 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 제한 시간 표시 여부를 변경합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
```

```
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 ChangeMessageVisibility를 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 메시지의 표시 제한 시간을 10시간(10시간 * 60분 * 60초 = 36,000초)으로 변경합니다.

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- API 세부 정보는 Cmdlet 참조(V4)의 ChangeMessageVisibility를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 메시지의 표시 제한 시간을 10시간(10시간 * 60분 * 60초 = 36,000초)으로 변경합니다.

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- API 세부 정보는 Cmdlet 참조(V5)의 ChangeMessageVisibility를 참조하세요. AWS Tools for PowerShell

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-sqs' # v2: require 'aws-sdk'
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
sqs = Aws::SQS::Client.new(region: 'us-west-2')

begin
  queue_name = 'my-queue'
  queue_url = sqs.get_queue_url(queue_name: queue_name).queue_url

  # Receive up to 10 messages
  receive_message_result_before = sqs.receive_message({
                                                        queue_url: queue_url,
                                                        max_number_of_messages:
 10
                                                      })

  puts "Before attempting to change message visibility timeout: received
 #{receive_message_result_before.messages.count} message(s)."
```

```
  receive_message_result_before.messages.each do |message|
    sqs.change_message_visibility({
                                    queue_url: queue_url,
                                    receipt_handle: message.receipt_handle,
                                    visibility_timeout: 30 # This message will
 not be visible for 30 seconds after first receipt.
                                  })
  end

  # Try to retrieve the original messages after setting their visibility timeout.
  receive_message_result_after = sqs.receive_message({
                                                       queue_url: queue_url,
                                                       max_number_of_messages: 10
                                                     })

  puts "\nAfter attempting to change message visibility timeout: received
 #{receive_message_result_after.messages.count} message(s)."
rescue Aws::SQS::Errors::NonExistentQueue
  puts "Cannot receive messages for a queue named '#{queue_name}', as it does not
 exist."
end
```

- API 세부 정보는AWS SDK for Ruby API 참조의 ChangeMessageVisibility를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

CLI로 **ChangeMessageVisibilityBatch** 사용

다음 코드 예시는 `ChangeMessageVisibilityBatch`의 사용 방법을 보여 줍니다.

CLI

AWS CLI

여러 메시지의 제한 시간 가시성을 배치로 변경하는 방법

이 예시에서는 지정된 2개 메시지의 제한 시간 가시성을 10시간(10시간 * 60분 * 60초)으로 변경합니다.

명령:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

입력 파일(change-message-visibility-batch.json):

```
[
  {
    "Id": "FirstMessage",
        "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
        "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBkTUH...HifSnw==",
        "VisibilityTimeout": 36000
  }
]
```

출력:

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ChangeMessageVisibilityBatch를 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 2개의 메시지의 표시 제한 시간을 변경합니다. 첫 번째 메시지의 표시 제한 시간이 10시간(10시간 * 60분 * 60초 = 36,000초)으

로 변경됩니다. 두 번째 메시지의 표시 제한 시간이 5시간(5시간 * 60분 * 60초 = 18,000초)으로 변경됩니다.

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

출력:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- API 세부 정보는 Cmdlet 참조(V4)의 ChangeMessageVisibilityBatch를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 2개의 메시지의 표시 제한 시간을 변경합니다. 첫 번째 메시지의 표시 제한 시간이 10시간(10시간 * 60분 * 60초 = 36,000초)으로 변경됩니다. 두 번째 메시지의 표시 제한 시간이 5시간(5시간 * 60분 * 60초 = 18,000초)으로 변경됩니다.

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000
```

```
$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

출력:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- API 세부 정보는 Cmdlet 참조(V5)의 ChangeMessageVisibilityBatch를 참조하세요. AWS Tools for PowerShell

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `CreateQueue` 함께 사용

다음 코드 예시는 `CreateQueue`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 대기열에 메시지 게시
- 메시지 일괄 전송 및 수신
- Amazon SQS Java 메시징 라이브러리를 사용하여 JMS 인터페이스 작업

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

특정 이름으로 대기열을 생성합니다.

```
    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
 useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }
```

```
            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }
```

Amazon SQS 대기열을 생성하고 메시지를 전송합니다.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SQS;
    using Amazon.SQS.Model;

    public class CreateSendExample
    {
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
        private static readonly RegionEndpoint ServiceRegion =
 RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
 Dictionary<string, MessageAttributeValue>
            {
```

```
                { "Title",   new MessageAttributeValue { DataType = "String",
 StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
 StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl,
 messageBody, messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS
 client, string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }
```

```
        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateQueue를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Create an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueName: An Amazon SQS queue name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::createQueue(const Aws::String &queueName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::CreateQueueRequest request;
    request.SetQueueName(queueName);

    const Aws::SQS::Model::CreateQueueOutcome outcome =
 sqsClient.CreateQueue(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created queue " << queueName << " with a queue
 URL "
                  << outcome.GetResult().GetQueueUrl() << "." << std::endl;
    }
    else {
        std::cerr << "Error creating queue " << queueName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는AWS SDK for C++ API 참조의 CreateQueue를 참조하세요.

CLI

AWS CLI

대기열 생성

이 예시에서는 지정된 이름의 대기열을 만들고, 메시지 보존 기간을 3일(3일 * 24시간 * 60분 * 60초)로 설정하고, 대기열의 Dead Letter Queue(DLQ)를 최대 메시지 수신 개수가 1,000개인 지정된 대기열로 설정합니다.

명령:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

입력 파일(create-queue.json):

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}
```

출력:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- API 세부 정보는 AWS CLI 명령 참조의 CreateQueue를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can
 specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
```

```
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 CreateQueue를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.ChangeMessageVisibilityRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.ListQueuesRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesResponse;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;
import java.util.List;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SQSExample {
    public static void main(String[] args) {
        String queueName = "queue" + System.currentTimeMillis();
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        // Perform various tasks on the Amazon SQS queue.
        String queueUrl = createQueue(sqsClient, queueName);
        listQueues(sqsClient);
        listQueuesFilter(sqsClient, queueUrl);
        List<Message> messages = receiveMessages(sqsClient, queueUrl);
        sendBatchMessages(sqsClient, queueUrl);
        changeMessages(sqsClient, queueUrl, messages);
        deleteMessages(sqsClient, queueUrl, messages);
        sqsClient.close();
    }

    public static String createQueue(SqsClient sqsClient, String queueName) {
        try {
            System.out.println("\nCreate Queue");

            CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();

            sqsClient.createQueue(createQueueRequest);

            System.out.println("\nGet queue url");

            GetQueueUrlResponse getQueueUrlResponse = sqsClient

 .getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

```
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void listQueues(SqsClient sqsClient) {

        System.out.println("\nList Queues");
        String prefix = "que";

        try {
            ListQueuesRequest listQueuesRequest =
ListQueuesRequest.builder().queueNamePrefix(prefix).build();
            ListQueuesResponse listQueuesResponse =
sqsClient.listQueues(listQueuesRequest);
            for (String url : listQueuesResponse.queueUrls()) {
                System.out.println(url);
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void listQueuesFilter(SqsClient sqsClient, String queueUrl) {
        // List queues with filters
        String namePrefix = "queue";
        ListQueuesRequest filterListRequest = ListQueuesRequest.builder()
                .queueNamePrefix(namePrefix)
                .build();

        ListQueuesResponse listQueuesFilteredResponse =
sqsClient.listQueues(filterListRequest);
        System.out.println("Queue URLs with prefix: " + namePrefix);
        for (String url : listQueuesFilteredResponse.queueUrls()) {
            System.out.println(url);
        }

        System.out.println("\nSend message");
        try {
            sqsClient.sendMessage(SendMessageRequest.builder()
```

```
                    .queueUrl(queueUrl)
                    .messageBody("Hello world!")
                    .delaySeconds(10)
                    .build());

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void sendBatchMessages(SqsClient sqsClient, String queueUrl) {

        System.out.println("\nSend multiple messages");
        try {
            SendMessageBatchRequest sendMessageBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)

.entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello
from msg 1").build(),

SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
2").delaySeconds(10)
                                    .build())
                    .build();
            sqsClient.sendMessageBatch(sendMessageBatchRequest);

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static List<Message> receiveMessages(SqsClient sqsClient, String
queueUrl) {

        System.out.println("\nReceive messages");
        try {
            ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(5)
                    .build();
```

```
            return sqsClient.receiveMessage(receiveMessageRequest).messages();

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static void changeMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {

        System.out.println("\nChange Message Visibility");
        try {

            for (Message message : messages) {
                ChangeMessageVisibilityRequest req =
ChangeMessageVisibilityRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .visibilityTimeout(100)
                        .build();
                sqsClient.changeMessageVisibility(req);
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {
        System.out.println("\nDelete Messages");

        try {
            for (Message message : messages) {
                DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .build();
                sqsClient.deleteMessage(deleteMessageRequest);
            }
```

```
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateQueue를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 표준 대기열을 생성합니다.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

긴 폴링이 있는 Amazon SQS 대기열을 생성합니다.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
        // When the wait time for the ReceiveMessage API action is greater than
 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
  );
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateQueue를 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 표준 대기열을 생성합니다.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

메시지가 도착하기를 기다리는 Amazon SQS 대기열을 생성합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 CreateQueue를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createQueue(queueNameVal: String): String {
    println("Create Queue")
    val createQueueRequest =
        CreateQueueRequest {
            queueName = queueNameVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.createQueue(createQueueRequest)
        println("Get queue url")

        val getQueueUrlRequest =
            GetQueueUrlRequest {
                queueName = queueNameVal
            }

        val getQueueUrlResponse = sqsClient.getQueueUrl(getQueueUrlRequest)
        return getQueueUrlResponse.queueUrl.toString()
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateQueue를 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제는 지정된 이름으로 대기열을 생성합니다.

```
New-SQSQueue -QueueName MyQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 CreateQueue를 참조하세요.

Tools for PowerShell V5

예제 1: 이 예제는 지정된 이름으로 대기열을 생성합니다.

```
New-SQSQueue -QueueName MyQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 CreateQueue를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def create_queue(name, attributes=None):
```

```
    """
    Creates an Amazon SQS queue.

    :param name: The name of the queue. This is part of the URL assigned to the
 queue.
    :param attributes: The attributes of the queue, such as maximum message size
 or
                       whether it's a FIFO queue.
    :return: A Queue object that contains metadata about the queue and that can
 be used
             to perform queue operations like sending and receiving messages.
    """
    if not attributes:
        attributes = {}

    try:
        queue = sqs.create_queue(QueueName=name, Attributes=attributes)
        logger.info("Created queue '%s' with URL=%s", name, queue.url)
    except ClientError as error:
        logger.exception("Couldn't create queue named '%s'.", name)
        raise error
    else:
        return queue
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CreateQueue를 참조하십시오.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# This code example demonstrates how to create a queue in Amazon Simple Queue
 Service (Amazon SQS).
```

```
require 'aws-sdk-sqs'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_name [String] The name of the queue.
# @return [Boolean] true if the queue was created; otherwise, false.
# @example
#   exit 1 unless queue_created?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'my-queue'
#   )
def queue_created?(sqs_client, queue_name)
  sqs_client.create_queue(queue_name: queue_name)
  true
rescue StandardError => e
  puts "Error creating queue: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Creating the queue named '#{queue_name}'..."

  if queue_created?(sqs_client, queue_name)
    puts 'Queue created.'
  else
    puts 'Queue not created.'
  end
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 CreateQueue를 참조하십시오.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 표준 대기열을 생성합니다.

```
    TRY.
        oo_result = lo_sqs->createqueue( iv_queuename = iv_queue_name ).        "
 oo_result is returned for testing purposes. "
        MESSAGE 'SQS queue created.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedeldrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
 queue with the same name.' TYPE 'E'.
      CATCH /aws1/cx_sqsqueuenameexists.
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
    ENDTRY.
```

메시지가 도착하기를 기다리는 Amazon SQS 대기열을 생성합니다.

```
    TRY.
        DATA lt_attributes TYPE /aws1/cl_sqsqueueattrmap_w=>tt_queueattributemap.
        DATA ls_attribute TYPE /aws1/
cl_sqsqueueattrmap_w=>ts_queueattributemap_maprow.
        ls_attribute-key = 'ReceiveMessageWaitTimeSeconds'.               " Time
 in seconds for long polling, such as how long the call waits for a message to
 arrive in the queue before returning. "
        ls_attribute-value = NEW /aws1/cl_sqsqueueattrmap_w( iv_value =
 iv_wait_time ).
        INSERT ls_attribute INTO TABLE lt_attributes.
        oo_result = lo_sqs->createqueue(                  " oo_result is returned
 for testing purposes. "
                iv_queuename = iv_queue_name
                it_attributes = lt_attributes ).
        MESSAGE 'SQS queue created.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedeldrecently.
```

```
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
 queue with the same name.' TYPE 'E'.
      CATCH /aws1/cx_sqsqueuenameexists.
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
    ENDTRY.
```

- API에 대한 세부 정보는 AWS SDK for SAP ABAP API 참조의 CreateQueue를 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.createQueue(
            input: CreateQueueInput(
                queueName: queueName
            )
        )

        guard let queueUrl = output.queueUrl else {
            print("No queue URL returned.")
            return
        }
```

- API 세부 정보는 SDK for Swift API 참조의 CreateQueue를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DeleteMessage** 함께 사용

다음 코드 예시는 DeleteMessage의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 메시지 일괄 전송 및 수신

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에서 메시지를 수신한 다음 메시지를 삭제합니다.

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
```

```
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
 queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await
 client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</
param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
 ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };
```

```
            var receiveMessageResponse = await
 client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteMessage를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Delete a message from an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageReceiptHandle: A message receipt handle.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::deleteMessage(const Aws::String &queueUrl,
```

```
                                const Aws::String &messageReceiptHandle,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::DeleteMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetReceiptHandle(messageReceiptHandle);

    const Aws::SQS::Model::DeleteMessageOutcome outcome =
 sqsClient.DeleteMessage(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted message from queue " << queueUrl
                  << std::endl;
    }
    else {
        std::cerr << "Error deleting message from queue " << queueUrl << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 DeleteMessage를 참조하세요.

CLI

AWS CLI

메시지를 삭제하는 방법

이 예시에서는 지정된 메시지를 삭제합니다.

명령:

```
aws sqs delete-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

출력:

```
None.
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteMessage를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        try {
            for (Message message : messages) {
                DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .build();
                sqsClient.deleteMessage(deleteMessageRequest);
            }
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteMessage를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 삭제합니다.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
```

```
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteMessage를 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지를 수신하고 삭제합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
```

```
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteMessage를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.purgeQueue(purgeRequest)
        println("Messages are successfully deleted from $queueUrlVal")
    }
```

```
}

suspend fun deleteQueue(queueUrlVal: String) {
    val request =
        DeleteQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteMessage를 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 메시지를 삭제합니다.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 DeleteMessage를 참조하세요.

Tools for PowerShell V5

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 메시지를 삭제합니다.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 DeleteMessage를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def delete_message(message):
    """
    Delete a message from a queue. Clients must delete messages after they
    are received and processed to remove them from the queue.

    :param message: The message to delete. The message's queue URL is contained
 in
                    the message's metadata.
    :return: None
    """
    try:
        message.delete()
        logger.info("Deleted message: %s", message.message_id)
    except ClientError as error:
        logger.exception("Couldn't delete message: %s", message.message_id)
        raise error
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteMessage를 참조하십시오.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `DeleteMessageBatch` 함께 사용

다음 코드 예시는 `DeleteMessageBatch`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- S3 이벤트 알림 처리
- 대기열에 메시지 게시
- 메시지 일괄 전송 및 수신

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl,
List<Message> messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
_amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);
```

```
        return deleteResponse.Failed.Any();
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteMessageBatch를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

            Aws::SQS::Model::DeleteMessageBatchRequest request;
            request.SetQueueUrl(queueURLS[i]);
            int id = 1; // Ids must be unique within a batch delete request.
            for (const Aws::String &receiptHandle: receiptHandles) {
                Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
                entry.SetId(std::to_string(id));
                ++id;
                entry.SetReceiptHandle(receiptHandle);
                request.AddEntries(entry);
            }

            Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
                    sqsClient.DeleteMessageBatch(request);

            if (outcome.IsSuccess()) {
                std::cout << "The batch deletion of messages was successful."
                          << std::endl;
            }
            else {
```

```
                std::cerr << "Error with SQS::DeleteMessageBatch. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteMessageBatch를 참조하세요.

CLI

AWS CLI

여러 메시지를 배치로 삭제하는 방법

이 예시에서는 지정된 메시지를 삭제합니다.

명령:

```
aws sqs delete-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-batch.json
```

입력 파일(delete-message-batch.json):

```
[
  {
        "Id": "FirstMessage",
        "ReceiptHandle": "AQEB1mgl...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

출력:

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteMessageBatch를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
```

```
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of
 messages from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 DeleteMessageBatch를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
```

```
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
```

```
  }
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteMessageBatch를 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 메시지 2개를 삭제합니다.

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

출력:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- API 세부 정보는 Cmdlet 참조(V4)의 DeleteMessageBatch를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제는 지정된 대기열에서 지정된 수신 핸들이 있는 메시지 2개를 삭제합니다.

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
```

```
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

출력:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- API 세부 정보는 Cmdlet 참조(V5)의 DeleteMessageBatch를 참조하세요. AWS Tools for PowerShell

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def delete_messages(queue, messages):
    """
    Delete a batch of messages from a queue in a single request.

    :param queue: The queue from which to delete the messages.
    :param messages: The list of messages to delete.
    :return: The response from SQS that contains the list of successful and
 failed
```

```
            message deletions.
    """
    try:
        entries = [
            {"Id": str(ind), "ReceiptHandle": msg.receipt_handle}
            for ind, msg in enumerate(messages)
        ]
        response = queue.delete_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info("Deleted %s",
messages[int(msg_meta["Id"])].receipt_handle)
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Could not delete %s",
messages[int(msg_meta["Id"])].receipt_handle
                )
    except ClientError:
        logger.exception("Couldn't delete messages from queue %s", queue)
    else:
        return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteMessageBatch를 참조하십시오.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSSQS
```

```
        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        // Create the list of message entries.

        var entries: [SQSClientTypes.DeleteMessageBatchRequestEntry] = []
        var messageNumber = 1

        for handle in handles {
            let entry = SQSClientTypes.DeleteMessageBatchRequestEntry(
                id: "\(messageNumber)",
                receiptHandle: handle
            )
            entries.append(entry)
            messageNumber += 1
        }

        // Delete the messages.

        let output = try await sqsClient.deleteMessageBatch(
            input: DeleteMessageBatchInput(
                entries: entries,
                queueUrl: queue
            )
        )

        // Get the lists of failed and successful deletions from the output.

        guard let failedEntries = output.failed else {
            print("Failed deletion list is missing!")
            return
        }
        guard let successfulEntries = output.successful else {
            print("Successful deletion list is missing!")
            return
        }

        // Display a list of the failed deletions along with their
        // corresponding explanation messages.

        if failedEntries.count != 0 {
            print("Failed deletions:")

            for entry in failedEntries {
```

```
                print("Message #\(entry.id ?? "<unknown>") failed:
\(entry.message ?? "<unknown>")")
            }
        } else {
            print("No failed deletions.")
        }

        // Output a list of the message numbers that were successfully deleted.

        if successfulEntries.count != 0 {
            var successes = ""

            for entry in successfulEntries {
                if successes.count == 0 {
                    successes = entry.id ?? "<unknown>"
                } else {
                    successes = "\(successes), \(entry.id ?? "<unknown>")"
                }
            }
            print("Succeeded: ", successes)
        } else {
            print("No successful deletions.")
        }
```

- API 세부 정보는 SDK for Swift API 참조의 DeleteMessageBatch를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DeleteQueue** 함께 사용

다음 코드 예시는 DeleteQueue의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 대기열에 메시지 게시
- 메시지 일괄 전송 및 수신
- Amazon SQS Java 메시징 라이브러리를 사용하여 JMS 인터페이스 작업

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

URL을 사용하여 대기열을 삭제합니다.

```
/// <summary>
/// Delete a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteQueueByUrl(string queueUrl)
{
    var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
        new DeleteQueueRequest()
        {
            QueueUrl = queueUrl
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteQueue를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Delete an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueURL: An Amazon SQS queue URL.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::deleteQueue(const Aws::String &queueURL,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);
    Aws::SQS::Model::DeleteQueueRequest request;
    request.SetQueueUrl(queueURL);

    const Aws::SQS::Model::DeleteQueueOutcome outcome =
 sqsClient.DeleteQueue(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted queue with url " << queueURL <<
                  std::endl;
    }
    else {
        std::cerr << "Error deleting queue " << queueURL << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는AWS SDK for C++ API 참조의 DeleteQueue를 참조하세요.

CLI

AWS CLI

대기열 삭제

이 예시에서는 지정된 대기열을 삭제합니다.

명령:

```
aws sqs delete-queue --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

출력:

```
None.
```

- API 세부 정보는 AWS CLI 명령 참조의 DeleteQueue를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// DeleteQueue deletes an Amazon SQS queue.
```

```
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

- API에 대한 세부 정보는AWS SDK for Go API 참조의 DeleteQueue를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteQueue {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:    <queueName>

                Where:
                   queueName - The name of the Amazon SQS queue to delete.

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String queueName = args[0];
        SqsClient sqs = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        deleteSQSQueue(sqs, queueName);
        sqs.close();
    }

    public static void deleteSQSQueue(SqsClient sqsClient, String queueName) {
        try {
            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                    .queueName(queueName)
                    .build();

            String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            DeleteQueueRequest deleteQueueRequest = DeleteQueueRequest.builder()
                    .queueUrl(queueUrl)
                    .build();

            sqsClient.deleteQueue(deleteQueueRequest);

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteQueue를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 삭제합니다.

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteQueue를 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 삭제합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 DeleteQueue를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.purgeQueue(purgeRequest)
```

```
        println("Messages are successfully deleted from $queueUrlVal")
    }
}

suspend fun deleteQueue(queueUrlVal: String) {
    val request =
        DeleteQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteQueue를 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 지정된 대기열을 삭제합니다.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- API 세부 정보는 Cmdlet 참조(V4)의 DeleteQueue를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제에서는 지정된 대기열을 삭제합니다.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- API 세부 정보는 Cmdlet 참조(V5)의 DeleteQueue를 참조하세요. AWS Tools for PowerShell

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def remove_queue(queue):
    """
    Removes an SQS queue. When run against an AWS account, it can take up to
    60 seconds before the queue is actually deleted.

    :param queue: The queue to delete.
    :return: None
    """
    try:
        queue.delete()
        logger.info("Deleted queue with URL=%s.", queue.url)
    except ClientError as error:
        logger.exception("Couldn't delete queue with URL=%s!", queue.url)
        raise error
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteQueue를 참조하십시오.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-sqs' # v2: require 'aws-sdk'
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
sqs = Aws::SQS::Client.new(region: 'us-west-2')

sqs.delete_queue(queue_url: URL)
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 DeleteQueue를 참조하십시오.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
        MESSAGE 'SQS queue deleted' TYPE 'I'.
    ENDTRY.
```

- API에 대한 세부 정보는 AWS SDK for SAP ABAP API 참조의 DeleteQueue를 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSSQS
```

```
        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        do {
            _ = try await sqsClient.deleteQueue(
                input: DeleteQueueInput(
                    queueUrl: queueUrl
                )
            )
        } catch _ as AWSSQS.QueueDoesNotExist {
            print("Error: The specified queue doesn't exist.")
            return
        }
```

- API 세부 정보는 SDK for Swift API 참조의 DeleteQueue를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **GetQueueAttributes** 함께 사용

다음 코드 예시는 `GetQueueAttributes`의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- S3 이벤트 알림 처리
- 대기열에 메시지 게시

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await
_amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetQueueAttributes를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

            Aws::SQS::Model::GetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);
```

```
request.AddAttributeNames(Aws::SQS::Model::QueueAttributeName::QueueArn);

            Aws::SQS::Model::GetQueueAttributesOutcome outcome =
                    sqsClient.GetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                const Aws::Map<Aws::SQS::Model::QueueAttributeName, Aws::String>
&attributes =
                        outcome.GetResult().GetAttributes();
                const auto &iter = attributes.find(
                        Aws::SQS::Model::QueueAttributeName::QueueArn);
                if (iter != attributes.end()) {
                    queueARN = iter->second;
                    std::cout << "The queue ARN '" << queueARN
                              << "' has been retrieved."
                              << std::endl;
                }

            }
            else {
                std::cerr << "Error with SQS::GetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;


            }
```

- API 세부 정보는 AWS SDK for C++ API 참조의 GetQueueAttributes를 참조하세요.

CLI

AWS CLI

대기열의 속성을 가져오는 방법

이 예시에서는 지정된 대기열의 모든 속성을 가져옵니다.

명령:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

출력:

```
{
  "Attributes": {
    "ApproximateNumberOfMessagesNotVisible": "0",
    "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":1000}",
    "MessageRetentionPeriod": "345600",
    "ApproximateNumberOfMessagesDelayed": "0",
    "MaximumMessageSize": "262144",
    "CreatedTimestamp": "1442426968",
    "ApproximateNumberOfMessages": "0",
    "ReceiveMessageWaitTimeSeconds": "0",
    "DelaySeconds": "0",
    "VisibilityTimeout": "30",
    "LastModifiedTimestamp": "1442426968",
    "QueueArn": "arn:aws:sqs:us-east-1:80398EXAMPLE:MyNewQueue"
  }
}
```

이 예시에서는 지정된 대기열의 최대 메시지 크기 및 가시성 제한 시간 속성만 가져옵니다.

명령:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-
names MaximumMessageSize VisibilityTimeout
```

출력:

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- API 세부 정보는 AWS CLI 명령 참조의 GetQueueAttributes를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string)
 (string, error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
```

```
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 GetQueueAttributes를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetQueueAttributes를 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 지정된 대기열의 모든 속성을 나열합니다.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":
```

```
 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                        arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                        [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

예제 2: 이 예제에서는 지정된 대기열에 대해 지정된 속성만 별도로 나열합니다.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
VisibilityTimeout                       : 30
DelaySeconds                            : 0
MaximumMessageSize                      : 262144
MessageRetentionPeriod                  : 345600
ApproximateNumberOfMessages             : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed      : 0
CreatedTimestamp                        : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                   : 12/29/2015 2:23:17 PM
QueueARN                                : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                  :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                        398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                        arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
```

```
Attributes                              : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- API 세부 정보는 Cmdlet 참조(V4)의 GetQueueAttributes를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제에서는 지정된 대기열의 모든 속성을 나열합니다.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
VisibilityTimeout                       : 30
DelaySeconds                            : 0
MaximumMessageSize                      : 262144
MessageRetentionPeriod                  : 345600
ApproximateNumberOfMessages             : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed      : 0
CreatedTimestamp                        : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                   : 12/29/2015 2:23:17 PM
QueueARN                                : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                  :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
```

```
                                          [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

예제 2: 이 예제에서는 지정된 대기열에 대해 지정된 속성만 별도로 나열합니다.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- API 세부 정보는 Cmdlet 참조(V5)의 GetQueueAttributes를 참조하세요. AWS Tools for PowerShell

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.getQueueAttributes(
            input: GetQueueAttributesInput(
                attributeNames: [
                    .approximatenumberofmessages,
                    .maximummessagesize
                ],
                queueUrl: url
            )
        )

        guard let attributes = output.attributes else {
            print("No queue attributes returned.")
            return
        }

        for (attr, value) in attributes {
            switch(attr) {
            case "ApproximateNumberOfMessages":
                print("Approximate message count: \(value)")
            case "MaximumMessageSize":
                print("Maximum message size: \(value)kB")
            default:
                continue
            }
        }
```

- API 세부 정보는 SDK for Swift API 참조의 GetQueueAttributes를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **GetQueueUrl** 함께 사용

다음 코드 예시는 `GetQueueUrl`의 사용 방법을 보여 줍니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
using System;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

public class GetQueueUrl
{
    /// <summary>
    /// Initializes the Amazon SQS client object and then calls the
    /// GetQueueUrlAsync method to retrieve the URL of an Amazon SQS
    /// queue.
    /// </summary>
    public static async Task Main()
    {
        // If the Amazon SQS message queue is not in the same AWS Region as
your
        // default user, you need to provide the AWS Region as a parameter to
the
        // client constructor.
        var client = new AmazonSQSClient();
```

```
            string queueName = "New-Example-Queue";

            try
            {
                var response = await client.GetQueueUrlAsync(queueName);

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine($"The URL for {queueName} is:
 {response.QueueUrl}");
                }
            }
            catch (QueueDoesNotExistException ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine($"The queue {queueName} was not found.");
            }
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetQueueUrl을 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Get the URL for an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueName: An Amazon SQS queue name.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::getQueueUrl(const Aws::String &queueName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::GetQueueUrlRequest request;
    request.SetQueueName(queueName);

    const Aws::SQS::Model::GetQueueUrlOutcome outcome =
 sqsClient.GetQueueUrl(request);
    if (outcome.IsSuccess()) {
        std::cout << "Queue " << queueName << " has url " <<
                  outcome.GetResult().GetQueueUrl() << std::endl;
    }
    else {
        std::cerr << "Error getting url for queue " << queueName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 GetQueueUrl을 참조하세요.

CLI

AWS CLI

대기열 URL을 가져오는 방법

이 예시에서는 지정된 대기열의 URL을 가져옵니다.

명령:

```
aws sqs get-queue-url --queue-name MyQueue
```

출력:

```
{
```

```
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- API 세부 정보는 AWS CLI 명령 참조의 GetQueueUrl을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
            GetQueueUrlResponse getQueueUrlResponse = sqsClient

 .getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 GetQueueUrl을 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열의 URL을 가져옵니다.

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";
```

```
export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetQueueUrl을 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열의 URL을 가져옵니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 GetQueueUrl을 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 지정된 이름이 있는 대기열의 URL을 나열합니다.

```
Get-SQSQueueUrl -QueueName MyQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 Cmdlet 참조(V4)의 GetQueueUrl을 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제에서는 지정된 이름이 있는 대기열의 URL을 나열합니다.

```
Get-SQSQueueUrl -QueueName MyQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 Cmdlet 참조(V5)의 GetQueueUrl을 참조하세요. AWS Tools for PowerShell

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def get_queue(name):
    """
    Gets an SQS queue by name.

    :param name: The name that was used to create the queue.
    :return: A Queue object.
    """
    try:
        queue = sqs.get_queue_by_name(QueueName=name)
        logger.info("Got queue '%s' with URL=%s", name, queue.url)
    except ClientError as error:
        logger.exception("Couldn't get queue named %s.", name)
        raise error
    else:
        return queue
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 GetQueueUrl를 참조하십시오.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_sqs->getqueueurl( iv_queuename = iv_queue_name ).        "
oo_result is returned for testing purposes. "
        MESSAGE 'Queue URL retrieved.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedoesnotexist.
        MESSAGE 'The requested queue does not exist.' TYPE 'E'.
    ENDTRY.
```

- API에 대한 세부 정보는 AWS SDK for SAP ABAP API 참조의 GetQueueUrl을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

CLI로 **ListDeadLetterSourceQueues** 사용

다음 코드 예시는 `ListDeadLetterSourceQueues`의 사용 방법을 보여 줍니다.

CLI

AWS CLI

배달 못한 편지 소스 대기열을 나열하는 방법

이 예시에서는 지정된 Dead Letter 소스 대기열에 연결된 대기열을 나열합니다.

명령:

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

출력:

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ListDeadLetterSourceQueues를 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 지정된 대기열을 배달 못한 편지 대기열로 사용하는 대기열의 URL을 나열합니다.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- API 세부 정보는 Cmdlet 참조(V4)의 ListDeadLetterSourceQueues를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제에서는 지정된 대기열을 배달 못한 편지 대기열로 사용하는 대기열의 URL을 나열합니다.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- API 세부 정보는 Cmdlet 참조(V5)의 ListDeadLetterSourceQueues를 참조하세요. AWS Tools for PowerShell

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **`ListQueues`** 함께 사용

다음 코드 예시는 `ListQueues`의 사용 방법을 보여 줍니다.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! List the Amazon Simple Queue Service (Amazon SQS) queues within an AWS
 account.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SQS::listQueues(const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ListQueuesRequest listQueuesRequest;

    Aws::String nextToken; // Used for pagination.
    Aws::Vector<Aws::String> allQueueUrls;

    do {
        if (!nextToken.empty()) {
            listQueuesRequest.SetNextToken(nextToken);
        }
        const Aws::SQS::Model::ListQueuesOutcome outcome = sqsClient.ListQueues(
                listQueuesRequest);
        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String> &queueUrls =
 outcome.GetResult().GetQueueUrls();
            allQueueUrls.insert(allQueueUrls.end(),
                                queueUrls.begin(),
                                queueUrls.end());

            nextToken = outcome.GetResult().GetNextToken();
        }
        else {
            std::cerr << "Error listing queues: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

    } while (!nextToken.empty());
```

```
    std::cout << allQueueUrls.size() << " Amazon SQS queue(s) found." <<
 std::endl;
    for (const auto &iter: allQueueUrls) {
        std::cout << " " << iter << std::endl;
    }

    return true;
}
```

- API에 대한 세부 정보는AWS SDK for C++ API 참조의 ListQueues를 참조하세요.

CLI

AWS CLI

대기열을 나열하는 방법

이 예시에서는 모든 대기열을 나열합니다.

명령:

```
aws sqs list-queues
```

출력:

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/TestQueue1",
        "https://queue.amazonaws.com/80398EXAMPLE/TestQueue2"
  ]
}
```

이 예시에서는 'My'로 시작하는 대기열만 나열합니다.

명령:

```
aws sqs list-queues --queue-name-prefix My
```

출력:

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ListQueues를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
```

```
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- API에 대한 세부 정보는AWS SDK for Go API 참조의 ListQueues를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        String prefix = "que";

        try {
            ListQueuesRequest listQueuesRequest =
 ListQueuesRequest.builder().queueNamePrefix(prefix).build();
            ListQueuesResponse listQueuesResponse =
 sqsClient.listQueues(listQueuesRequest);
            for (String url : listQueuesResponse.queueUrls()) {
                System.out.println(url);
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListQueues를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 나열합니다.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
```

```
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListQueues를 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 나열합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 ListQueues를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun listQueues() {
    println("\nList Queues")

    val prefix = "que"
    val listQueuesRequest =
        ListQueuesRequest {
            queueNamePrefix = prefix
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.listQueues(listQueuesRequest)
        response.queueUrls?.forEach { url ->
            println(url)
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ListQueues를 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 모든 대기열을 나열합니다.

```
Get-SQSQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/AnotherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/DeadLetterQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

예제 2: 이 예제에서는 지정된 이름으로 시작하는 대기열을 나열합니다.

```
Get-SQSQueue -QueueNamePrefix My
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 ListQueues를 참조하세요.

Tools for PowerShell V5

예제 1: 이 예제에서는 모든 대기열을 나열합니다.

```
Get-SQSQueue
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/AnotherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/DeadLetterQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

예제 2: 이 예제에서는 지정된 이름으로 시작하는 대기열을 나열합니다.

```
Get-SQSQueue -QueueNamePrefix My
```

출력:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 ListQueues를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def get_queues(prefix=None):
    """
    Gets a list of SQS queues. When a prefix is specified, only queues with names
    that start with the prefix are returned.

    :param prefix: The prefix used to restrict the list of returned queues.
    :return: A list of Queue objects.
    """
    if prefix:
        queue_iter = sqs.queues.filter(QueueNamePrefix=prefix)
    else:
        queue_iter = sqs.queues.all()
    queues = list(queue_iter)
    if queues:
        logger.info("Got queues: %s", ", ".join([q.url for q in queues]))
    else:
        logger.warning("No queues found.")
    return queues
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListQueues를 참조하십시오.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @example
#   list_queue_urls(Aws::SQS::Client.new(region: 'us-west-2'))
def list_queue_urls(sqs_client)
  queues = sqs_client.list_queues

  queues.queue_urls.each do |url|
    puts url
  end
rescue StandardError => e
  puts "Error listing queue URLs: #{e.message}"
end

# Lists the attributes of a queue in Amazon Simple Queue Service (Amazon SQS).
#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @example
#   list_queue_attributes(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
#   )
def list_queue_attributes(sqs_client, queue_url)
  attributes = sqs_client.get_queue_attributes(
    queue_url: queue_url,
    attribute_names: ['All']
  )
```

```
  attributes.attributes.each do |key, value|
    puts "#{key}: #{value}"
  end
rescue StandardError => e
  puts "Error getting queue attributes: #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'

  sqs_client = Aws::SQS::Client.new(region: region)

  puts 'Listing available queue URLs...'
  list_queue_urls(sqs_client)

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  puts "\nGetting information about queue '#{queue_name}'..."
  list_queue_attributes(sqs_client, queue_url)
end
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 ListQueues를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

지역에 나열된 첫 번째 Amazon SQS 대기열을 검색합니다.

```
async fn find_first_queue(client: &Client) -> Result<String, Error> {
    let queues = client.list_queues().send().await?;
    let queue_urls = queues.queue_urls();
    Ok(queue_urls
        .first()
        .expect("No queues in this account and Region. Create a queue to
 proceed.")
        .to_string())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 ListQueues를 참조하십시오.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_sqs->listqueues( ).        " oo_result is returned for
 testing purposes. "
        MESSAGE 'Retrieved list of queues.' TYPE 'I'.
    ENDTRY.
```

- API에 대한 세부 정보는 AWS SDK for SAP ABAP API 참조의 ListQueues를 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }

            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
        }
```

- API 세부 정보는 AWS SDK for Swift API 참조의 ListQueues를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

CLI로 **PurgeQueue** 사용

다음 코드 예시는 PurgeQueue의 사용 방법을 보여 줍니다.

CLI

AWS CLI

대기열을 제거하는 방법

이 예시에서는 지정된 대기열의 모든 메시지를 삭제합니다.

명령:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue
```

출력:

```
None.
```

- API 세부 정보는 AWS CLI 명령 참조의 PurgeQueue를 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제는 지정된 대기열에서 모든 메시지를 삭제합니다.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 PurgeQueue를 참조하세요.

Tools for PowerShell V5

예제 1: 이 예제는 지정된 대기열에서 모든 메시지를 삭제합니다.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 PurgeQueue를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **ReceiveMessage** 함께 사용

다음 코드 예시는 ReceiveMessage의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- S3를 사용하여 대용량 메시지 관리
- S3 이벤트 알림 처리
- 대기열에 메시지 게시
- 메시지 일괄 전송 및 수신

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

대기열의 URL을 사용하여 대기열에서 메시지를 수신합니다.

```
    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
```

```
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }
```

Amazon SQS 대기열에서 메시지를 수신한 다음 메시지를 삭제합니다.

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
 queueName)
```

```
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await
 client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</
param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
 ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };

            var receiveMessageResponse = await
 client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };
```

```
            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ReceiveMessage를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Receive a message from an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::receiveMessage(const Aws::String &queueUrl,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ReceiveMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetMaxNumberOfMessages(1);

    const Aws::SQS::Model::ReceiveMessageOutcome outcome =
 sqsClient.ReceiveMessage(
            request);
```

```
    if (outcome.IsSuccess()) {

        const Aws::Vector<Aws::SQS::Model::Message> &messages =
                outcome.GetResult().GetMessages();
        if (!messages.empty()) {
            const Aws::SQS::Model::Message &message = messages[0];
            std::cout << "Received message:" << std::endl;
            std::cout << "  MessageId: " << message.GetMessageId() << std::endl;
            std::cout << "  ReceiptHandle: " << message.GetReceiptHandle() <<
 std::endl;
            std::cout << "  Body: " << message.GetBody() << std::endl <<
 std::endl;
        }
        else {
            std::cout << "No messages received from queue " << queueUrl <<
                      std::endl;

        }
    }
    else {
        std::cerr << "Error receiving message from queue " << queueUrl << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 ReceiveMessage를 참조하세요.

CLI

AWS CLI

메시지를 수신하는 방법

이 예시에서는 사용 가능한 메시지를 최대 10개까지 수신하고 사용 가능한 속성을 모두 반환합니다.

명령:

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All --message-
attribute-names All --max-number-of-messages 10
```

출력:

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEBzbVv...fqNzFw==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "9424c491...26bc3ae7",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "ApproximateFirstReceiveTimestamp": "1442428276921",
        "SenderId": "AIDAIAZKMSNQ7TEXAMPLE",
        "ApproximateReceiveCount": "5",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        },
        "City": {
          "DataType": "String",
          "StringValue": "Any City"
        }
      }
    }
  ]
}
```

이 예시에서는 다음으로 사용 가능한 메시지를 수신하여 SenderID 및 SentTimestamp 속성과 PostalCode 메시지 속성만 반환합니다.

명령:

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-
names SenderId SentTimestamp --message-attribute-names PostalCode
```

출력:

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEB6nR4...HzlvZQ==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "b8e89563...e088e74f",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "SenderId": "AIDAIAZKMSNQ7TEXAMPLE",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        }
      }
    }
  ]
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ReceiveMessage를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "encoding/json"
 "fmt"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 } else {
  messages = result.Messages
 }
 return messages, err
}
```

- API에 대한 세부 정보는 AWS SDK for Go API 참조의 ReceiveMessage를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        try {
            ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(5)
                    .build();
            return sqsClient.receiveMessage(receiveMessageRequest).messages();

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ReceiveMessage를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에서 메시지를 수신합니다.

```
import {
```

```
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
```

```
      }),
    );
  }
};
```

긴 폴링 지원을 사용하여 Amazon SQS 대기열에서 메시지를 수신합니다.

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    // The duration (in seconds) for which the call waits for a message
    // to arrive in the queue before returning. If a message is available,
    // the call returns sooner than WaitTimeSeconds. If no messages are
    // available and the wait time expires, the call returns successfully
    // with an empty list of messages.
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 ReceiveMessage를 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

긴 폴링 지원을 사용하여 Amazon SQS 대기열에서 메시지를 수신합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 ReceiveMessage를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun receiveMessages(queueUrlVal: String?) {
    println("Retrieving messages from $queueUrlVal")

    val receiveMessageRequest =
        ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.receiveMessage(receiveMessageRequest)
        response.messages?.forEach { message ->
            println(message.body)
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ReceiveMessage를 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 지정된 대기열에 대해 수신할 다음 메시지 최대 10개에 대한 정보를 나열합니다. 지정된 메시지 속성이 있는 경우 해당 속성에 대한 값이 정보에 포함됩니다.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 ReceiveMessage를 참조하세요.

Tools for PowerShell V5

예제 1: 이 예제에서는 지정된 대기열에 대해 수신할 다음 메시지 최대 10개에 대한 정보를 나열합니다. 지정된 메시지 속성이 있는 경우 해당 속성에 대한 값이 정보에 포함됩니다.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- API 세부 정보는 Cmdlet 참조(V5)의 ReceiveMessage를 참조하세요. AWS Tools for PowerShell

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def receive_messages(queue, max_number, wait_time):
    """
    Receive a batch of messages in a single request from an SQS queue.

    :param queue: The queue from which to receive messages.
    :param max_number: The maximum number of messages to receive. The actual
 number
                        of messages received might be less.
    :param wait_time: The maximum time to wait (in seconds) before returning.
 When
                       this number is greater than zero, long polling is used.
 This
                       can result in reduced costs and fewer false empty
 responses.
    :return: The list of Message objects received. These each contain the body
             of the message and metadata and custom attributes.
    """
    try:
        messages = queue.receive_messages(
            MessageAttributeNames=["All"],
            MaxNumberOfMessages=max_number,
            WaitTimeSeconds=wait_time,
        )
        for msg in messages:
            logger.info("Received message: %s: %s", msg.message_id, msg.body)
    except ClientError as error:
        logger.exception("Couldn't receive messages from queue: %s", queue)
        raise error
    else:
        return messages
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ReceiveMessage를 참조하십시오.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# Receives messages in a queue in Amazon Simple Queue Service (Amazon SQS).
#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param max_number_of_messages [Integer] The maximum number of messages
#   to receive. This number must be 10 or less. The default is 10.
# @example
#   receive_messages(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     10
#   )
def receive_messages(sqs_client, queue_url, max_number_of_messages = 10)
  if max_number_of_messages > 10
    puts 'Maximum number of messages to receive must be 10 or less. ' \
      'Stopping program.'
    return
  end

  response = sqs_client.receive_message(
    queue_url: queue_url,
    max_number_of_messages: max_number_of_messages
```

```
  )

  if response.messages.count.zero?
    puts 'No messages to receive, or all messages have already ' \
      'been previously received.'
    return
  end

  response.messages.each do |message|
    puts '-' * 20
    puts "Message body: #{message.body}"
    puts "Message ID:   #{message.message_id}"
  end
rescue StandardError => e
  puts "Error receiving messages: #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  max_number_of_messages = 10

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Receiving messages from queue '#{queue_name}'..."

  receive_messages(sqs_client, queue_url, max_number_of_messages)
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 ReceiveMessage를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
async fn receive(client: &Client, queue_url: &String) -> Result<(), Error> {
    let rcv_message_output =
 client.receive_message().queue_url(queue_url).send().await?;

    println!("Messages from queue with url: {}", queue_url);

    for message in rcv_message_output.messages.unwrap_or_default() {
        println!("Got the message: {:#?}", message);
    }

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 ReceiveMessage를 참조하십시오.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에서 메시지를 수신합니다.

```
    TRY.
```

```
        oo_result = lo_sqs->receivemessage( iv_queueurl = iv_queue_url ).    "
oo_result is returned for testing purposes. "
        DATA(lt_messages) = oo_result->get_messages( ).
        MESSAGE 'Message received from SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsoverlimit.
        MESSAGE 'Maximum number of in-flight messages reached.' TYPE 'E'.
    ENDTRY.
```

긴 폴링 지원을 사용하여 Amazon SQS 대기열에서 메시지를 수신합니다.

```
    TRY.
        oo_result = lo_sqs->receivemessage(           " oo_result is returned for
testing purposes. "
                iv_queueurl = iv_queue_url
                iv_waittimeseconds = iv_wait_time ).    " Time in seconds for
long polling, such as how long the call waits for a message to arrive in the
queue before returning. " ).
        DATA(lt_messages) = oo_result->get_messages( ).
        MESSAGE 'Message received from SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsoverlimit.
        MESSAGE 'Maximum number of in-flight messages reached.' TYPE 'E'.
    ENDTRY.
```

- API에 대한 세부 정보는 AWS SDK for SAP ABAP API 참조의 ReceiveMessage를 참조하세요.

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
```

```
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.receiveMessage(
            input: ReceiveMessageInput(
                maxNumberOfMessages: maxMessages,
                queueUrl: url
            )
        )

        guard let messages = output.messages else {
            print("No messages received.")
            return
        }

        for message in messages {
            print("Message ID:     \(message.messageId ?? "<unknown>")")
            print("Receipt handle: \(message.receiptHandle ?? "<unknown>")")
            print(message.body ?? "<body missing>")
            print("---")
        }
```

- API 세부 정보는 SDK for Swift API 참조의 ReceiveMessage를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

CLI로 **RemovePermission** 사용

다음 코드 예시는 `RemovePermission`의 사용 방법을 보여 줍니다.

CLI

AWS CLI

권한을 제거하는 방법

이 예시에서는 지정된 대기열에서 지정된 레이블이 있는 권한을 제거합니다.

명령:

```
aws sqs remove-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue
```

출력:

```
None.
```

- API 세부 정보는 AWS CLI 명령 참조의 RemovePermission을 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제는 지정된 대기열에서 지정된 레이블이 있는 권한 설정을 제거합니다.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 RemovePermission을 참조하세요.

Tools for PowerShell V5

예제 1: 이 예제는 지정된 대기열에서 지정된 레이블이 있는 권한 설정을 제거합니다.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 RemovePermission을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **SendMessage** 함께 사용

다음 코드 예시는 SendMessage의 사용 방법을 보여 줍니다.

작업 예시는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- S3를 사용하여 대용량 메시지 관리
- 메시지 일괄 전송 및 수신

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열을 생성하고 메시지를 전송합니다.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.SQS;
using Amazon.SQS.Model;

public class CreateSendExample
{
    // Specify your AWS Region (an example Region is shown).
    private static readonly string QueueName = "Example_Queue";
    private static readonly RegionEndpoint ServiceRegion =
RegionEndpoint.USWest2;
    private static IAmazonSQS client;

    public static async Task Main()
    {
        client = new AmazonSQSClient(ServiceRegion);
        var createQueueResponse = await CreateQueue(client, QueueName);

        string queueUrl = createQueueResponse.QueueUrl;
```

```
            Dictionary<string, MessageAttributeValue> messageAttributes = new
 Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
 StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
 StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl,
 messageBody, messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS
 client, string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");
```

```
            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 SendMessage를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Send a message to an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageBody: A message body.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::sendMessage(const Aws::String &queueUrl,
                              const Aws::String &messageBody,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SendMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetMessageBody(messageBody);

    const Aws::SQS::Model::SendMessageOutcome outcome =
 sqsClient.SendMessage(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully sent message to " << queueUrl <<
                  std::endl;
    }
    else {
        std::cerr << "Error sending message to " << queueUrl << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
```

```
    return outcome.IsSuccess();
}
```

- API에 대한 세부 정보는 AWS SDK for C++ API 참조의 SendMessage를 참조하세요.

CLI

AWS CLI

메시지 전송

이 예시에서는 지정된 메시지 본문, 지연 기간 및 메시지 속성이 설정된 메시지를 지정된 대기열로 보냅니다.

명령:

```
aws sqs send-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --message-body "Information about the largest city in Any Region." --delay-seconds 10 --message-attributes file://send-message.json
```

입력 파일(send-message.json):

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

출력:

```
{
```

```
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- API 세부 정보는 AWS CLI 명령 참조의 SendMessage를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

`SendMessage` 작업의 두 가지 예는 다음과 같습니다.

- 본문 및 지연이 포함된 메시지 전송
- 본문 및 메시지 속성이 포함된 메시지 전송

본문과 지연이 포함된 메시지를 전송합니다.

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SendMessages {
```

```
    public static void main(String[] args) {
        final String usage = """

                Usage:    <queueName> <message>

                Where:
                   queueName - The name of the queue.
                   message - The message to send.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String queueName = args[0];
        String message = args[1];
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();
        sendMessage(sqsClient, queueName, message);
        sqsClient.close();
    }

    public static void sendMessage(SqsClient sqsClient, String queueName, String
 message) {
        try {
            CreateQueueRequest request = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();
            sqsClient.createQueue(request);

            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                    .queueName(queueName)
                    .build();

            String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            SendMessageRequest sendMsgRequest = SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody(message)
                    .delaySeconds(5)
                    .build();

            sqsClient.sendMessage(sendMsgRequest);
```

```
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

본문 및 메시지 속성이 포함된 메시지를 전송합니다.

```
    /**
     * <p>This method demonstrates how to add message attributes to a message.
     * Each attribute must specify a name, value, and data type. You use a Java
 Map to supply the attributes. The map's
     * key is the attribute name, and you specify the map's entry value using a
 builder that includes the attribute
     * value and data type.</p>
     *
     * <p>The data type must start with one of "String", "Number" or "Binary".
 You can optionally
     * define a custom extension by using a "." and your extension.</p>
     *
     * <p>The SQS Developer Guide provides more information on @see <a
     * href="https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-message-metadata.html#sqs-message-attributes">message
     * attributes</a>.</p>
     *
     * @param thumbailPath Filesystem path of the image.
     * @param queueUrl     URL of the SQS queue.
     */
    static void sendMessageWithAttributes(Path thumbailPath, String queueUrl) {
        Map<String, MessageAttributeValue> messageAttributeMap;
        try {
            messageAttributeMap = Map.of(
                    "Name", MessageAttributeValue.builder()
                            .stringValue("Jane Doe")
                            .dataType("String").build(),
                    "Age", MessageAttributeValue.builder()
                            .stringValue("42")
                            .dataType("Number.int").build(),
                    "Image", MessageAttributeValue.builder()
```

```
.binaryValue(SdkBytes.fromByteArray(Files.readAllBytes(thumbailPath)))
                            .dataType("Binary.jpg").build()
            );
        } catch (IOException e) {
            LOGGER.error("An I/O exception occurred reading thumbnail image: {}",
e.getMessage(), e);
            throw new RuntimeException(e);
        }

        SendMessageRequest request = SendMessageRequest.builder()
                .queueUrl(queueUrl)
                .messageBody("Hello SQS")
                .messageAttributes(messageAttributeMap)
                .build();
        try {
            SendMessageResponse sendMessageResponse =
SQS_CLIENT.sendMessage(request);
            LOGGER.info("Message ID: {}", sendMessageResponse.messageId());
        } catch (SqsException e) {
            LOGGER.error("Exception occurred sending message: {}",
e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 SendMessage를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에 메시지를 전송합니다.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";
```

```
const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
    MessageBody:
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 SendMessage를 참조하십시오.

SDK for JavaScript (v2)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 대기열에 메시지를 전송합니다.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- 자세한 정보는 AWS SDK for JavaScript 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for JavaScript API 참조의 SendMessage를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun sendMessages(
    queueUrlVal: String,
    message: String,
) {
    println("Sending multiple messages")
    println("\nSend message")
    val sendRequest =
        SendMessageRequest {
            queueUrl = queueUrlVal
            messageBody = message
            delaySeconds = 10
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessage(sendRequest)
        println("A single message was successfully sent.")
    }
}

suspend fun sendBatchMessages(queueUrlVal: String?) {
    println("Sending multiple messages")

    val msg1 =
        SendMessageBatchRequestEntry {
            id = "id1"
            messageBody = "Hello from msg 1"
        }
```

```
    val msg2 =
        SendMessageBatchRequestEntry {
            id = "id2"
            messageBody = "Hello from msg 2"
        }

    val sendMessageBatchRequest =
        SendMessageBatchRequest {
            queueUrl = queueUrlVal
            entries = listOf(msg1, msg2)
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessageBatch(sendMessageBatchRequest)
        println("Batch message were successfully sent.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 SendMessage를 참조하십시오.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 지정된 속성과 메시지 본문이 포함된 메시지를 메시지 전송이 10초 동안 지연된 상태로 지정된 대기열로 보냅니다.

```
$cityAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)
```

```
Send-SQSMessage -DelayInSeconds 10 -MessageAttributes $messageAttributes -
MessageBody "Information about the largest city in Any Region." -QueueUrl
 https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
MD5OfMessageAttributes              MD5OfMessageBody                    MessageId

----------------------              ----------------                    ---------

1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V4)의 SendMessage를 참조하세요.

Tools for PowerShell V5

예제 1: 이 예제에서는 지정된 속성과 메시지 본문이 포함된 메시지를 메시지 전송이 10초 동안 지연된 상태로 지정된 대기열로 보냅니다.

```
$cityAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Send-SQSMessage -DelayInSeconds 10 -MessageAttributes $messageAttributes -
MessageBody "Information about the largest city in Any Region." -QueueUrl
 https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

출력:

```
MD5OfMessageAttributes              MD5OfMessageBody                    MessageId

----------------------              ----------------                    ---------
```

```
1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- API 세부 정보는 AWS Tools for PowerShell Cmdlet 참조(V5)의 SendMessage를 참조하세요.

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def send_message(queue, message_body, message_attributes=None):
    """
    Send a message to an Amazon SQS queue.

    :param queue: The queue that receives the message.
    :param message_body: The body text of the message.
    :param message_attributes: Custom attributes of the message. These are key-
value
                               pairs that can be whatever you want.
    :return: The response from SQS that contains the assigned message ID.
    """
    if not message_attributes:
        message_attributes = {}

    try:
        response = queue.send_message(
            MessageBody=message_body, MessageAttributes=message_attributes
        )
    except ClientError as error:
        logger.exception("Send message failed: %s", message_body)
        raise error
    else:
        return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의SendMessage를 참조하십시오.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param message_body [String] The contents of the message to be sent.
# @return [Boolean] true if the message was sent; otherwise, false.
# @example
#   exit 1 unless message_sent?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     'This is my message.'
#   )
def message_sent?(sqs_client, queue_url, message_body)
  sqs_client.send_message(
    queue_url: queue_url,
    message_body: message_body
  )
  true
rescue StandardError => e
  puts "Error sending message: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
```

```
  region = 'us-west-2'
  queue_name = 'my-queue'
  message_body = 'This is my message.'

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Sending a message to the queue named '#{queue_name}'..."

  if message_sent?(sqs_client, queue_url, message_body)
    puts 'Message sent.'
  else
    puts 'Message not sent.'
  end
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- API 세부 정보는 AWS SDK for Ruby API 참조의 SendMessage를 참조하십시오.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
async fn send(client: &Client, queue_url: &String, message: &SQSMessage) ->
 Result<(), Error> {
    println!("Sending message to queue with URL: {}", queue_url);
```

```
    let rsp = client
        .send_message()
        .queue_url(queue_url)
        .message_body(&message.body)
        // If the queue is FIFO, you need to set .message_deduplication_id
        // and message_group_id or configure the queue for
 ContentBasedDeduplication.
        .send()
        .await?;

    println!("Send message to the queue: {:#?}", rsp);

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의SendMessage를 참조하십시오.

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    TRY.
        oo_result = lo_sqs->sendmessage(              " oo_result is returned for
 testing purposes. "
           iv_queueurl = iv_queue_url
           iv_messagebody = iv_message ).
        MESSAGE 'Message sent to SQS queue.' TYPE 'I'.
      CATCH /aws1/cx_sqsinvalidmsgconts.
        MESSAGE 'Message contains non-valid characters.' TYPE 'E'.
      CATCH /aws1/cx_sqsunsupportedop.
        MESSAGE 'Operation not supported.' TYPE 'E'.
    ENDTRY.
```

- API에 대한 세부 정보는 AWS SDK for SAP ABAP API 참조의 SendMessage를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **SendMessageBatch** 함께 사용

다음 코드 예시는 SendMessageBatch의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 메시지 일괄 전송 및 수신

CLI

AWS CLI

여러 메시지를 배치로 전송하는 방법

이 예시에서는 지정된 메시지 본문, 지연 기간 및 메시지 속성이 설정된 메시지 2개를 지정된 대기열로 보냅니다.

명령:

```
aws sqs send-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-batch.json
```

입력 파일(send-message-batch.json):

```
[
  {
    "Id": "FuelReport-0001-2015-09-16T140731Z",
        "MessageBody": "Fuel report for account 0001 on 2015-09-16 at 02:07:31
 PM.",
        "DelaySeconds": 10,
        "MessageAttributes": {
          "SellerName": {
            "DataType": "String",
```

```
                "StringValue": "Example Store"
      },
          "City": {
        "DataType": "String",
        "StringValue": "Any City"
      },
          "Region": {
            "DataType": "String",
                "StringValue": "WA"
      },
          "PostalCode": {
            "DataType": "String",
                "StringValue": "99065"
          },
          "PricePerGallon": {
            "DataType": "Number",
                "StringValue": "1.99"
      }
        }
  },
  {
    "Id": "FuelReport-0002-2015-09-16T140930Z",
        "MessageBody": "Fuel report for account 0002 on 2015-09-16 at 02:09:30
PM.",
        "DelaySeconds": 10,
        "MessageAttributes": {
          "SellerName": {
            "DataType": "String",
                "StringValue": "Example Fuels"
      },
          "City": {
        "DataType": "String",
        "StringValue": "North Town"
      },
          "Region": {
            "DataType": "String",
                "StringValue": "WA"
      },
          "PostalCode": {
            "DataType": "String",
                "StringValue": "99123"
          },
          "PricePerGallon": {
            "DataType": "Number",
```

```
                "StringValue": "1.87"
      }
        }
  }
]
```

출력:

```
{
  "Successful": [
    {
      "MD5OfMessageBody": "203c4a38...7943237e",
      "MD5OfMessageAttributes": "10809b55...baf283ef",
      "Id": "FuelReport-0001-2015-09-16T140731Z",
      "MessageId": "d175070c-d6b8-4101-861d-adeb3EXAMPLE"
    },
    {
      "MD5OfMessageBody": "2cf0159a...c1980595",
      "MD5OfMessageAttributes": "55623928...ae354a25",
      "Id": "FuelReport-0002-2015-09-16T140930Z",
      "MessageId": "f9b7d55d-0570-413e-b9c5-a9264EXAMPLE"
    }
  ]
}
```

- API 세부 정보는 AWS CLI 명령 참조의 SendMessageBatch를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
            SendMessageBatchRequest sendMessageBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)
```

```
.entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello
from msg 1").build(),

SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
2").delaySeconds(10)
                                    .build())
                    .build();
            sqsClient.sendMessageBatch(sendMessageBatchRequest);
```

- API에 대한 세부 정보는 AWS SDK for Java 2.x API 참조의 SendMessageBatch를 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 지정된 속성 및 메시지 본문이 포함된 메시지 2개를 지정된 대기열로 보냅니다. 첫 번째 메시지의 경우 15초, 두 번째 메시지의 경우 10초 동안 전송이 지연됩니다.

```
$student1NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1NameAttributeValue.DataType = "String"
$student1NameAttributeValue.StringValue = "John Doe"

$student1GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1GradeAttributeValue.DataType = "Number"
$student1GradeAttributeValue.StringValue = "89"

$student2NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2NameAttributeValue.DataType = "String"
$student2NameAttributeValue.StringValue = "Jane Doe"

$student2GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2GradeAttributeValue.DataType = "Number"
$student2GradeAttributeValue.StringValue = "93"

$message1 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message1.DelaySeconds = 15
$message1.Id = "FirstMessage"
$message1.MessageAttributes.Add("StudentName", $student1NameAttributeValue)
$message1.MessageAttributes.Add("StudentGrade", $student1GradeAttributeValue)
```

```
$message1.MessageBody = "Information about John Doe's grade."

$message2 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message2.DelaySeconds = 10
$message2.Id = "SecondMessage"
$message2.MessageAttributes.Add("StudentName", $student2NameAttributeValue)
$message2.MessageAttributes.Add("StudentGrade", $student2GradeAttributeValue)
$message2.MessageBody = "Information about Jane Doe's grade."

Send-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -Entry $message1, $message2
```

출력:

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- API 세부 정보는 Cmdlet 참조(V4)의 SendMessageBatch를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제에서는 지정된 속성 및 메시지 본문이 포함된 메시지 2개를 지정된 대기열로 보냅니다. 첫 번째 메시지의 경우 15초, 두 번째 메시지의 경우 10초 동안 전송이 지연됩니다.

```
$student1NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1NameAttributeValue.DataType = "String"
$student1NameAttributeValue.StringValue = "John Doe"

$student1GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1GradeAttributeValue.DataType = "Number"
$student1GradeAttributeValue.StringValue = "89"

$student2NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2NameAttributeValue.DataType = "String"
$student2NameAttributeValue.StringValue = "Jane Doe"

$student2GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2GradeAttributeValue.DataType = "Number"
$student2GradeAttributeValue.StringValue = "93"
```

```
$message1 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message1.DelaySeconds = 15
$message1.Id = "FirstMessage"
$message1.MessageAttributes.Add("StudentName", $student1NameAttributeValue)
$message1.MessageAttributes.Add("StudentGrade", $student1GradeAttributeValue)
$message1.MessageBody = "Information about John Doe's grade."

$message2 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message2.DelaySeconds = 10
$message2.Id = "SecondMessage"
$message2.MessageAttributes.Add("StudentName", $student2NameAttributeValue)
$message2.MessageAttributes.Add("StudentGrade", $student2GradeAttributeValue)
$message2.MessageBody = "Information about Jane Doe's grade."

Send-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -Entry $message1, $message2
```

출력:

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- API 세부 정보는 Cmdlet 참조(V5)의 SendMessageBatch를 참조하세요. AWS Tools for PowerShell

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def send_messages(queue, messages):
    """
```

```
    Send a batch of messages in a single request to an SQS queue.
    This request may return overall success even when some messages were not
sent.
    The caller must inspect the Successful and Failed lists in the response and
    resend any failed messages.

    :param queue: The queue to receive the messages.
    :param messages: The messages to send to the queue. These are simplified to
                     contain only the message body and attributes.
    :return: The response from SQS that contains the list of successful and
failed
             messages.
    """
    try:
        entries = [
            {
                "Id": str(ind),
                "MessageBody": msg["body"],
                "MessageAttributes": msg["attributes"],
            }
            for ind, msg in enumerate(messages)
        ]
        response = queue.send_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info(
                    "Message sent: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Failed to send: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
    except ClientError as error:
        logger.exception("Send messages failed to queue: %s", queue)
        raise error
    else:
        return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 SendMessageBatch를 참조하십시오.

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param entries [Hash] The contents of the messages to be sent,
#   in the correct format.
# @return [Boolean] true if the messages were sent; otherwise, false.
# @example
#   exit 1 unless messages_sent?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     [
#       {
#         id: 'Message1',
#         message_body: 'This is the first message.'
#       },
#       {
#         id: 'Message2',
#         message_body: 'This is the second message.'
#       }
#     ]
#   )
def messages_sent?(sqs_client, queue_url, entries)
```

```
  sqs_client.send_message_batch(
    queue_url: queue_url,
    entries: entries
  )
  true
rescue StandardError => e
  puts "Error sending messages: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  entries = [
    {
      id: 'Message1',
      message_body: 'This is the first message.'
    },
    {
      id: 'Message2',
      message_body: 'This is the second message.'
    }
  ]

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Sending messages to the queue named '#{queue_name}'..."

  if messages_sent?(sqs_client, queue_url, entries)
    puts 'Messages sent.'
  else
    puts 'Messages not sent.'
  end
end
```

- API에 대한 세부 정보는 AWS SDK for Ruby API 참조의 SendMessageBatch를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **SetQueueAttributes** 함께 사용

다음 코드 예시는 SetQueueAttributes의 사용 방법을 보여 줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 대기열에 메시지 게시

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

주제에 대한 대기열의 정책 속성을 설정합니다.

```
    /// <summary>
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string
topicArn, string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
```

```
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                                $"\"aws:SourceArn\":
\"{topicArn}\"" +
                                            "}" +
                                        "}" +
                                "}]" +
                            "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 SetQueueAttributes를 참조하십시오.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
//! Set the value for an attribute in an Amazon Simple Queue Service (Amazon SQS)
 queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param attributeName: An attribute name enum.
  \param attribute: The attribute value as a string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setQueueAttributes(const Aws::String &queueURL,
                                     Aws::SQS::Model::QueueAttributeName
 attributeName,
                                     const Aws::String &attribute,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(queueURL);
    request.AddAttributes(
            attributeName,
            attribute);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
 sqsClient.SetQueueAttributes(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully set the attribute  " <<

 Aws::SQS::Model::QueueAttributeNameMapper::GetNameForQueueAttributeName(
                         attributeName)
                  << " with value " << attribute << " in queue " <<
                  queueURL << "." << std::endl;
    }
    else {
        std::cout << "Error setting attribute for  queue " <<
                  queueURL << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }

    return outcome.IsSuccess();
}
```

배달 못한 편지 대기열을 구성합니다.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Connect an Amazon Simple Queue Service (Amazon SQS) queue to an associated
//! dead-letter queue.
/*!
  \param srcQueueUrl: An Amazon SQS queue URL.
  \param deadLetterQueueARN: The Amazon Resource Name (ARN) of an Amazon SQS
 dead-letter queue.
  \param maxReceiveCount: The max receive count of a message before it is sent to
 the dead-letter queue.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setDeadLetterQueue(const Aws::String &srcQueueUrl,
                                     const Aws::String &deadLetterQueueARN,
                                     int maxReceiveCount,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String redrivePolicy = MakeRedrivePolicy(deadLetterQueueARN,
 maxReceiveCount);

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(srcQueueUrl);
    request.AddAttributes(
            Aws::SQS::Model::QueueAttributeName::RedrivePolicy,
            redrivePolicy);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
            sqsClient.SetQueueAttributes(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully set dead letter queue for queue  " <<
                  srcQueueUrl << " to " << deadLetterQueueARN << std::endl;
    }
    else {
        std::cerr << "Error setting dead letter queue for queue " <<
```

```
                    srcQueueUrl << ": " << outcome.GetError().GetMessage() <<
                    std::endl;
    }

    return outcome.IsSuccess();
}

//! Make a redrive policy for a dead-letter queue.
/*!
  \param queueArn: An Amazon SQS ARN for the dead-letter queue.
  \param maxReceiveCount: The max receive count of a message before it is sent to
 the dead-letter queue.
  \return Aws::String: Policy as JSON string.
 */
Aws::String MakeRedrivePolicy(const Aws::String &queueArn, int maxReceiveCount) {
    Aws::Utils::Json::JsonValue redrive_arn_entry;
    redrive_arn_entry.AsString(queueArn);

    Aws::Utils::Json::JsonValue max_msg_entry;
    max_msg_entry.AsInteger(maxReceiveCount);

    Aws::Utils::Json::JsonValue policy_map;
    policy_map.WithObject("deadLetterTargetArn", redrive_arn_entry);
    policy_map.WithObject("maxReceiveCount", max_msg_entry);

    return policy_map.View().WriteReadable();
}
```

긴 폴링을 사용하도록 Amazon SQS 대기열을 구성합니다.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Set the wait time for an Amazon Simple Queue Service (Amazon SQS) queue poll.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param pollTimeSeconds: The receive message wait time in seconds.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setQueueLongPollingAttribute(const Aws::String &queueURL,
```

```
                                                const Aws::String
 &pollTimeSeconds,
                                                const
 Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(queueURL);
    request.AddAttributes(
            Aws::SQS::Model::QueueAttributeName::ReceiveMessageWaitTimeSeconds,
            pollTimeSeconds);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
 sqsClient.SetQueueAttributes(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated long polling time for queue " <<
                  queueURL << " to " << pollTimeSeconds << std::endl;
    }
    else {
        std::cout << "Error updating long polling time for queue " <<
                  queueURL << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 SetQueueAttributes를 참조하십시오.

CLI

AWS CLI

대기열 속성을 설정하는 방법

이 예시에서는 지정된 대기열을 전송 지연 10초, 최대 메시지 크기 128KB(128KB * 1,024바이트), 메시지 보존 기간 3일(3일 * 24시간 * 60분 * 60초), 메시지 수신 대기 시간 20초, 기본 가시성 제한 시간 60초로 설정합니다. 또한 이 예시에서는 최대 메시지 수신 개수가 1,000개인 지정된 Dead Letter Queue(DLQ)를 연결합니다.

명령:

```
aws sqs set-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-
attributes.json
```

입력 파일(set-queue-attributes.json):

```
{
  "DelaySeconds": "10",
  "MaximumMessageSize": "131072",
  "MessageRetentionPeriod": "259200",
  "ReceiveMessageWaitTimeSeconds": "20",
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "VisibilityTimeout": "60"
}
```

출력:

```
None.
```

- API 세부 정보는 AWS CLI 명령 참조의 SetQueueAttributes를 참조하세요.

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy
 to an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages
 to the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
```

```
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string
```

- API 세부 정보는 AWS SDK for Go API 참조의 SetQueueAttributes를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

사용자 지정 KMS 키를 사용하여 서버 측 암호화(SSE)를 사용하도록 Amazon SQS를 구성합니다.

```
    public static void addEncryption(String queueName, String kmsMasterKeyAlias)
 {
```

```
        SqsClient sqsClient = SqsClient.create();

        GetQueueUrlRequest urlRequest = GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build();

        GetQueueUrlResponse getQueueUrlResponse;
        try {
            getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest);
        } catch (QueueDoesNotExistException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        }
        String queueUrl = getQueueUrlResponse.queueUrl();


        Map<QueueAttributeName, String> attributes = Map.of(
                QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias,
                QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140" //
Set the data key reuse period to 140 seconds.
        );                                                                  //
This is how long SQS can reuse the data key before requesting a new one from
KMS.

        SetQueueAttributesRequest attRequest =
SetQueueAttributesRequest.builder()
                .queueUrl(queueUrl)
                .attributes(attributes)
                .build();
        try {
            sqsClient.setQueueAttributes(attRequest);
            LOGGER.info("The attributes have been applied to {}", queueName);
        } catch (InvalidAttributeNameException | InvalidAttributeValueException
e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        } finally {
            sqsClient.close();
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 SetQueueAttributes를 참조하십시오.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

긴 폴링을 사용하도록 Amazon SQS 대기열을 구성합니다.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
```

```
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

배달 못한 편지 대기열을 구성합니다.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 SetQueueAttributes를 참조하세요.

PowerShell

Tools for PowerShell V4

예제 1: 이 예제에서는 SNS 주제에 대기열을 구독하는 정책을 설정하는 방법을 보여줍니다. 메시지를 주제에 게시하면 구독 중인 대기열에 메시지가 전송됩니다.

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
          }
      }
    }
  ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

예제 2: 이 예제는 지정된 대기열에 대해 지정된 속성을 설정합니다.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 Cmdlet 참조(V4)의 SetQueueAttributes를 참조하세요. AWS Tools for PowerShell

Tools for PowerShell V5

예제 1: 이 예제에서는 SNS 주제에 대기열을 구독하는 정책을 설정하는 방법을 보여줍니다. 메시지를 주제에 게시하면 구독 중인 대기열에 메시지가 전송됩니다.

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
          }
      }
    }
  ]
}
"@
```

```
# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

예제 2: 이 예제는 지정된 대기열에 대해 지정된 속성을 설정합니다.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API 세부 정보는 Cmdlet 참조(V5)의 SetQueueAttributes를 참조하세요. AWS Tools for PowerShell

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        do {
            _ = try await sqsClient.setQueueAttributes(
                input: SetQueueAttributesInput(
                    attributes: [
                        "MaximumMessageSize": "\(maxSize)"
                    ],
                    queueUrl: url
                )
            )
        } catch _ as AWSSQS.InvalidAttributeValue {
            print("Invalid maximum message size: \(maxSize) kB.")
        }
```

- API 세부 정보는 SDK for Swift API 참조의 SetQueueAttributes를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDKs를 사용한 Amazon SQS 시나리오

다음 코드 예제에서는 Amazon SQS AWS SDKs에서 일반적인 시나리오를 구현하는 방법을 보여줍니다. 이러한 시나리오에서는 Amazon SQS 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여줍니다. 각 시나리오에는 전체 소스 코드에 대한 링크가 포함되어 있습니다. 여기에서 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

시나리오는 컨텍스트에 맞는 서비스 작업을 이해하는 데 도움이 되도록 중급 수준의 경험을 대상으로 합니다.

예시

- Amazon SQS를 사용하여 메시지를 보내고 검색하는 웹 애플리케이션 생성
- Step Functions를 사용하여 메신저 애플리케이션 생성
- Amazon Textract 탐색기 애플리케이션 생성
- AWS SDK를 사용하여 FIFO Amazon SNS 주제 생성 및 게시
- AWS SDK를 사용하여 Amazon Rekognition을 사용하여 비디오에서 사람 및 객체 감지
- AWS SDK와 함께 Amazon S3를 사용하여 대규모 Amazon SQS 메시지 관리 Amazon S3
- AWS SDK를 사용하여 Amazon S3 이벤트 알림 수신 및 처리
- AWS SDK를 사용하여 Amazon SNS 메시지를 Amazon SQS 대기열에 게시
- AWS SDK를 사용하여 Amazon SQS로 메시지 배치 전송 및 수신
- .NET용 AWS 메시지 처리 프레임워크를 사용하여 Amazon SQS 메시지 게시 및 수신
- Amazon SQS Java 메시징 라이브러리를 사용하여 Amazon SQS용 Java Message Service(JMS) 인터페이스로 작업
- AWS SDK를 사용하여 대기열 태그 및 Amazon SQS 작업

Amazon SQS를 사용하여 메시지를 보내고 검색하는 웹 애플리케이션 생성

다음 코드 예제는 Amazon SQS를 사용하여 메시징 애플리케이션을 생성하는 방법을 보여줍니다.

Java

SDK for Java 2.x

Amazon SQS API를 사용하여 메시지를 보내고 검색하는 Spring REST API를 개발하는 방법을 보여 줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK for Kotlin

Amazon SQS API를 사용하여 메시지를 보내고 검색하는 Spring REST API를 개발하는 방법을 보여 줍니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예시에서 사용되는 서비스

- Amazon Comprehend
- Amazon SQS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Step Functions를 사용하여 메신저 애플리케이션 생성

다음 코드 예제에서는 데이터베이스 테이블에서 메시지 레코드를 검색하는 AWS Step Functions 메신저 애플리케이션을 생성하는 방법을 보여줍니다.

Python

SDK for Python(Boto3)

AWS SDK for Python (Boto3) 와 함께를 사용하여 Amazon DynamoDB 테이블에서 메시지 레코드를 검색하고 Amazon Simple Queue Service(Amazon SQS)를 통해 보내는 메신저 애플리케이션을 AWS Step Functions 생성하는 방법을 보여줍니다. 상태 시스템은 AWS Lambda 함수와 통합되어 데이터베이스에 전송되지 않은 메시지가 있는지 스캔합니다.

- Amazon DynamoDB 테이블에서 메시지 레코드를 검색하고 업데이트하는 상태 머신을 생성합니다.
- 상태 머신 정의를 업데이트하여 메시지를 Amazon Simple Queue Service(Amazon SQS)에도 전송합니다.
- 상태 머신의 실행을 시작하고 중지합니다.
- 서비스 통합을 사용하여 상태 머신에서 Lambda, DynamoDB 및 Amazon SQS에 연결합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon Textract 탐색기 애플리케이션 생성

다음 코드 예제에서는 대화형 애플리케이션을 통해 Amazon Textract 출력을 탐색하는 방법을 보여줍니다.

JavaScript

JavaScript용 SDK(v3)

AWS SDK for JavaScript 를 사용하여 Amazon Textract를 사용하여 문서 이미지에서 데이터를 추출하고 대화형 웹 페이지에 표시하는 React 애플리케이션을 구축하는 방법을 보여줍니다. 이

예제는 웹 브라우저에서 실행되며 자격 증명을 위해 인증된 Amazon Cognito 자격 증명이 필요합니다. 이 애플리케이션은 스토리지로 Amazon Simple Storage Service (Amazon S3)를 사용하고 알림을 위해 Amazon Simple Notification Service(Amazon SNS) 주제를 구독하는 Amazon Simple Queue Service(Amazon SQS) 대기열을 폴링합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python(Boto3)

Amazon Textract와 AWS SDK for Python (Boto3) 함께를 사용하여 문서 이미지에서 텍스트, 양식 및 테이블 요소를 감지하는 방법을 보여줍니다. 입력 이미지와 Amazon Textract 출력은 탐지된 요소를 탐색할 수 있는 Tkinter 애플리케이션에 표시됩니다.

- 문서 이미지를 Amazon Textract에 제출하고 감지된 요소의 출력을 탐색합니다.
- Amazon Textract로 직접, 또는 Amazon Simple Storage Service (Amazon S3) 버킷을 통해 이미지를 제출합니다.
- 비동기식 API를 사용하여 작업이 완료되면 Amazon Simple Notification Service(Amazon SNS) 주제에 알림을 게시하는 작업을 시작합니다.
- Amazon Simple Queue Service(Amazon SQS) 대기열에서 작업 완료 메시지를 폴링하고 결과를 표시합니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Cognito 자격 증명
- Amazon S3
- Amazon SNS

- Amazon SQS
- Amazon Textract

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 FIFO Amazon SNS 주제 생성 및 게시

다음 코드 예제는 FIFO Amazon SNS 주제를 생성하고 거기에 게시하는 방법을 보여줍니다.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예에서는

- Amazon SNS FIFO 주제 1개, Amazon SQS FIFO 대기열 2개, 표준 대기열 1개를 생성합니다.
- 대기열에서 주제를 구독하고 해당 주제에 메시지를 게시합니다.

테스트에서는 각 대기열에 대한 메시지 수신 여부를 확인합니다. 또한 전체 예제에서는 액세스 정책을 추가하는 것을 보여주고 마지막에 리소스를 삭제합니다.

```
public class PriceUpdateExample {
    public final static SnsClient snsClient = SnsClient.create();
    public final static SqsClient sqsClient = SqsClient.create();

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName>
 <analyticsQueueName>\n\n" +
            "Where:\n" +
```

```
            "   fifoTopicName - The name of the FIFO topic that you want to
create. \n\n" +
            "   wholesaleQueueARN - The name of a SQS FIFO queue that will be
created for the wholesale consumer. \n\n"
            +
            "   retailQueueARN - The name of a SQS FIFO queue that will created
for the retail consumer. \n\n" +
            "   analyticsQueueARN - The name of a SQS standard queue that will be
created for the analytics consumer. \n\n";
        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        final String fifoTopicName = args[0];
        final String wholeSaleQueueName = args[1];
        final String retailQueueName = args[2];
        final String analyticsQueueName = args[3];

        // For convenience, the QueueData class holds metadata about a queue:
ARN, URL,
        // name and type.
        List<QueueData> queues = List.of(
            new QueueData(wholeSaleQueueName, QueueType.FIFO),
            new QueueData(retailQueueName, QueueType.FIFO),
            new QueueData(analyticsQueueName, QueueType.Standard));

        // Create queues.
        createQueues(queues);

        // Create a topic.
        String topicARN = createFIFOTopic(fifoTopicName);

        // Subscribe each queue to the topic.
        subscribeQueues(queues, topicARN);

        // Allow the newly created topic to send messages to the queues.
        addAccessPolicyToQueuesFINAL(queues, topicARN);

        // Publish a sample price update message with payload.
        publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}",
"Consumables");

        // Clean up resources.
```

```
        deleteSubscriptions(queues);
        deleteQueues(queues);
        deleteTopic(topicARN);
    }

    public static String createFIFOTopic(String topicName) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = Map.of(
                "FifoTopic", "true",
                "ContentBasedDeduplication", "false",
                "FifoThroughputScope", "MessageGroup");

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            String topicArn = response.topicArn();
            System.out.println("The topic ARN is" + topicArn);

            return topicArn;

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void subscribeQueues(List<QueueData> queues, String topicARN) {
        queues.forEach(queue -> {
            SubscribeRequest subscribeRequest = SubscribeRequest.builder()
                .topicArn(topicARN)
                .endpoint(queue.queueARN)
                .protocol("sqs")
                .build();

            // Subscribe to the endpoint by using the SNS service client.
            // Only Amazon SQS queues can receive notifications from an Amazon
SNS FIFO
            // topic.
```

```
            SubscribeResponse subscribeResponse =
snsClient.subscribe(subscribeRequest);
            System.out.println("The queue [" + queue.queueARN + "] subscribed to
the topic [" + topicARN + "]");
            queue.subscriptionARN = subscribeResponse.subscriptionArn();
        });
    }

    public static void publishPriceUpdate(String topicArn, String payload, String
groupId) {

        try {
            // Create and publish a message that updates the wholesale price.
            String subject = "Price Update";
            String dedupId = UUID.randomUUID().toString();
            String attributeName = "business";
            String attributeValue = "wholesale";

            MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
                .dataType("String")
                .stringValue(attributeValue)
                .build();

            Map<String, MessageAttributeValue> attributes = new HashMap<>();
            attributes.put(attributeName, msgAttValue);
            PublishRequest pubRequest = PublishRequest.builder()
                .topicArn(topicArn)
                .subject(subject)
                .message(payload)
                .messageGroupId(groupId)
                .messageDeduplicationId(dedupId)
                .messageAttributes(attributes)
                .build();

            final PublishResponse response = snsClient.publish(pubRequest);
            System.out.println(response.messageId());
            System.out.println(response.sequenceNumber());
            System.out.println("Message was published to " + topicArn);

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 항목을 참조하세요.
  - CreateTopic
  - 게시
  - Subscribe

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon FIFO 주제를 생성하고, Amazon SQS FIFO 및 표준 대기열에서 주제를 구독하고, 해당 주제에 메시지를 게시합니다.

```
def usage_demo():
    """Shows how to subscribe queues to a FIFO topic."""
    print("-" * 88)
    print("Welcome to the `Subscribe queues to a FIFO topic` demo!")
    print("-" * 88)

    sns = boto3.resource("sns")
    sqs = boto3.resource("sqs")
    fifo_topic_wrapper = FifoTopicWrapper(sns)
    sns_wrapper = SnsWrapper(sns)

    prefix = "sqs-subscribe-demo-"
    queues = set()
    subscriptions = set()

    wholesale_queue = sqs.create_queue(
        QueueName=prefix + "wholesale.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
```

```
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(wholesale_queue)
    print(f"Created FIFO queue with URL: {wholesale_queue.url}.")

    retail_queue = sqs.create_queue(
        QueueName=prefix + "retail.fifo",
        Attributes={
            "MaximumMessageSize": str(4096),
            "ReceiveMessageWaitTimeSeconds": str(10),
            "VisibilityTimeout": str(300),
            "FifoQueue": str(True),
            "ContentBasedDeduplication": str(True),
        },
    )
    queues.add(retail_queue)
    print(f"Created FIFO queue with URL: {retail_queue.url}.")

    analytics_queue = sqs.create_queue(QueueName=prefix + "analytics",
Attributes={})
    queues.add(analytics_queue)
    print(f"Created standard queue with URL: {analytics_queue.url}.")

    topic = fifo_topic_wrapper.create_fifo_topic("price-updates-topic.fifo")
    print(f"Created FIFO topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        fifo_topic_wrapper.add_access_policy(q, topic.attributes["TopicArn"])

    print(f"Added access policies for topic: {topic.attributes['TopicArn']}.")

    for q in queues:
        sub = fifo_topic_wrapper.subscribe_queue_to_topic(
            topic, q.attributes["QueueArn"]
        )
        subscriptions.add(sub)

    print(f"Subscribed queues to topic: {topic.attributes['TopicArn']}.")

    input("Press Enter to publish a message to the topic.")
```

```
    message_id = fifo_topic_wrapper.publish_price_update(
        topic, '{"product": 214, "price": 79.99}', "Consumables"
    )

    print(f"Published price update with message ID: {message_id}.")

    # Clean up the subscriptions, queues, and topic.
    input("Press Enter to clean up resources.")
    for s in subscriptions:
        sns_wrapper.delete_subscription(s)

    sns_wrapper.delete_topic(topic)

    for q in queues:
        fifo_topic_wrapper.delete_queue(q)

    print(f"Deleted subscriptions, queues, and topic.")

    print("Thanks for watching!")
    print("-" * 88)



class FifoTopicWrapper:
    """Encapsulates Amazon SNS FIFO topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def create_fifo_topic(self, topic_name):
        """
        Create a FIFO topic.
        Topic names must be made up of only uppercase and lowercase ASCII
 letters,
        numbers, underscores, and hyphens, and must be between 1 and 256
 characters long.
        For a FIFO topic, the name must end with the .fifo suffix.

        :param topic_name: The name for the topic.
        :return: The new topic.
        """
```

```
        try:
            topic = self.sns_resource.create_topic(
                Name=topic_name,
                Attributes={
                    "FifoTopic": str(True),
                    "ContentBasedDeduplication": str(False),
                    "FifoThroughputScope": "MessageGroup",
                },
            )
            logger.info("Created FIFO topic with name=%s.", topic_name)
            return topic
        except ClientError as error:
            logger.exception("Couldn't create topic with name=%s!", topic_name)
            raise error


    @staticmethod
    def add_access_policy(queue, topic_arn):
        """
        Add the necessary access policy to a queue, so
        it can receive messages from a topic.

        :param queue: The queue resource.
        :param topic_arn: The ARN of the topic.
        :return: None.
        """
        try:
            queue.set_attributes(
                Attributes={
                    "Policy": json.dumps(
                        {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Sid": "test-sid",
                                    "Effect": "Allow",
                                    "Principal": {"AWS": "*"},
                                    "Action": "SQS:SendMessage",
                                    "Resource": queue.attributes["QueueArn"],
                                    "Condition": {
                                        "ArnLike": {"aws:SourceArn": topic_arn}
                                    },
                                }
                            ],
```

```
                        }
                    )
                }
            )
            logger.info("Added trust policy to the queue.")
        except ClientError as error:
            logger.exception("Couldn't add trust policy to the queue!")
            raise error


    @staticmethod
    def subscribe_queue_to_topic(topic, queue_arn):
        """
        Subscribe a queue to a topic.

        :param topic: The topic resource.
        :param queue_arn: The ARN of the queue.
        :return: The subscription resource.
        """
        try:
            subscription = topic.subscribe(
                Protocol="sqs",
                Endpoint=queue_arn,
            )
            logger.info("The queue is subscribed to the topic.")
            return subscription
        except ClientError as error:
            logger.exception("Couldn't subscribe queue to topic!")
            raise error


    @staticmethod
    def publish_price_update(topic, payload, group_id):
        """
        Compose and publish a message that updates the wholesale price.

        :param topic: The topic to publish to.
        :param payload: The message to publish.
        :param group_id: The group ID for the message.
        :return: The ID of the message.
        """
        try:
            att_dict = {"business": {"DataType": "String", "StringValue":
 "wholesale"}}
```

```
            dedup_id = uuid.uuid4()
            response = topic.publish(
                Subject="Price Update",
                Message=payload,
                MessageAttributes=att_dict,
                MessageGroupId=group_id,
                MessageDeduplicationId=str(dedup_id),
            )
            message_id = response["MessageId"]
            logger.info("Published message to topic %s.", topic.arn)
        except ClientError as error:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise error
        return message_id


    @staticmethod
    def delete_queue(queue):
        """
        Removes an SQS queue. When run against an AWS account, it can take up to
        60 seconds before the queue is actually deleted.

        :param queue: The queue to delete.
        :return: None
        """
        try:
            queue.delete()
            logger.info("Deleted queue with URL=%s.", queue.url)
        except ClientError as error:
            logger.exception("Couldn't delete queue with URL=%s!", queue.url)
            raise error
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하세요.
  - CreateTopic
  - 게시
  - Subscribe

SAP ABAP

SDK for SAP ABAP API

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

FIFO 주제를 생성하고, 주제에 대한 Amazon SQS FIFO 대기열을 구독하고, Amazon SNS 주제에 메시지를 게시합니다.

```
    " Creates a FIFO topic. "
    DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>tt_topicattributesmap.
    DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>ts_topicattributesmap_maprow.
    ls_tpc_attributes-key = 'FifoTopic'.
    ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsmap_w( iv_value =
 'true' ).
    INSERT ls_tpc_attributes INTO TABLE lt_tpc_attributes.

    TRY.
        DATA(lo_create_result) = lo_sns->createtopic(
               iv_name = iv_topic_name
               it_attributes = lt_tpc_attributes ).
        DATA(lv_topic_arn) = lo_create_result->get_topicarn( ).
        ov_topic_arn = lv_topic_arn.                                    "
 ov_topic_arn is returned for testing purposes. "
        MESSAGE 'FIFO topic created' TYPE 'I'.
      CATCH /aws1/cx_snstopiclimitexcdex.
        MESSAGE 'Unable to create more topics. You have reached the maximum
 number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
 SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
 to an SNS FIFO topic. "
    TRY.
```

```
        DATA(lo_subscribe_result) = lo_sns->subscribe(
               iv_topicarn = lv_topic_arn
               iv_protocol = 'sqs'
               iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.                      "
 ov_subscription_arn is returned for testing purposes. "
        MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
      CATCH /aws1/cx_snssubscriptionlmte00.
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
 number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
 'String'

 iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.

        DATA(lo_result) = lo_sns->publish(
             iv_topicarn = lv_topic_arn
             iv_message = 'The price of your mobile plan has been increased from
 $19 to $23'
             iv_subject = 'Changes to mobile plan'
             iv_messagegroupid = 'Update-2'
             iv_messagededuplicationid = 'Update-2.1'
             it_messageattributes = lt_msg_attributes ).
        ov_message_id = lo_result->get_messageid( ).                    "
 ov_message_id is returned for testing purposes. "
        MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- API 세부 정보는 AWS SDK for SAP ABAP API 참조의 다음 주제를 참조하세요.
  - CreateTopic
  - 게시
  - Subscribe

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon Rekognition을 사용하여 비디오에서 사람 및 객체 감지

다음 코드 예제에서는 Amazon Rekognition을 사용하여 동영상에서 사람과 객체를 감지하는 방법을 보여줍니다.

Java

SDK for Java 2.x

Amazon Rekognition Java API를 사용하여 Amazon Simple Storage Service (Amazon S3) 버킷에 있는 동영상에서 얼굴과 객체를 감지하기 위한 앱을 만드는 방법을 보여줍니다. 이 앱은 Amazon Simple Email Service(Amazon SES)를 사용하여 결과와 함께 이메일 알림을 관리자에게 보냅니다.

전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하십시오.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK for Python(Boto3)

Amazon Rekognition을 사용하여 비동기식 감지 작업을 시작해 동영상의 얼굴, 객체 및 사람을 감지할 수 있습니다. 또한 이 예제에서는 작업이 완료되고 주제에 대한 Amazon Simple Queue Service(Amazon SQS) 대기열을 구독할 때 Amazon Simple Notification Service(Amazon SNS) 주제를 알리도록 Amazon Rekognition을 구성합니다. 대기열이 작업에 대한 메시지를 받으면 작업이 검색되고 결과가 출력됩니다.

이 예제는 GitHub에서 가장 잘 볼 수 있습니다. 전체 소스 코드와 설정 및 실행 방법에 대한 지침은 GitHub에서 전체 예제를 참조하세요.

이 예제에서 사용되는 서비스

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK와 함께 Amazon S3를 사용하여 대규모 Amazon SQS 메시지 관리 Amazon S3

다음 코드 예제에서는 Amazon SQS 확장 클라이언트 라이브러리를 사용하여 대규모 Amazon SQS 메시지로 작업하는 방법을 보여줍니다.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.BucketLifecycleConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ExpirationStatus;
import software.amazon.awssdk.services.s3.model.LifecycleExpiration;
import software.amazon.awssdk.services.s3.model.LifecycleRule;
import software.amazon.awssdk.services.s3.model.LifecycleRuleFilter;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import
 software.amazon.awssdk.services.s3.model.PutBucketLifecycleConfigurationRequest;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;

/**
 * Example of using Amazon SQS Extended Client Library for Java 2.x.
 */
public class SqsExtendedClientExample {
    private static final Logger logger =
 LoggerFactory.getLogger(SqsExtendedClientExample.class);

    private String s3BucketName;
```

```
    private String queueUrl;
    private final String queueName;
    private final S3Client s3Client;
    private final SqsClient sqsExtendedClient;
    private final int messageSize;

    /**
     * Constructor with default clients and message size.
     */
    public SqsExtendedClientExample() {
        this(S3Client.create(), 300000);
    }

    /**
     * Constructor with custom S3 client and message size.
     *
     * @param s3Client The S3 client to use
     * @param messageSize The size of the test message to create
     */
    public SqsExtendedClientExample(S3Client s3Client, int messageSize) {
        this.s3Client = s3Client;
        this.messageSize = messageSize;

        // Generate a unique bucket name.
        this.s3BucketName = UUID.randomUUID() + "-" +
                DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());

        // Generate a unique queue name.
        this.queueName = "MyQueue-" + UUID.randomUUID();

        // Configure the SQS extended client.
        final ExtendedClientConfiguration extendedClientConfig = new
 ExtendedClientConfiguration()
                .withPayloadSupportEnabled(s3Client, s3BucketName);

        this.sqsExtendedClient = new
 AmazonSQSExtendedClient(SqsClient.builder().build(), extendedClientConfig);
    }

    public static void main(String[] args) {
        SqsExtendedClientExample example = new SqsExtendedClientExample();
        try {
            example.setup();
            example.sendAndReceiveMessage();
```

```
        } finally {
            example.cleanup();
        }
    }

    /**
     * Send a large message and receive it back.
     *
     * @return The received message
     */
    public Message sendAndReceiveMessage() {
        try {
            // Create a large message.
            char[] chars = new char[messageSize];
            Arrays.fill(chars, 'x');
            String largeMessage = new String(chars);

            // Send the message.
            final SendMessageRequest sendMessageRequest =
SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody(largeMessage)
                    .build();

            sqsExtendedClient.sendMessage(sendMessageRequest);
            logger.info("Sent message of size: {}", largeMessage.length());

            // Receive and return the message.
            final ReceiveMessageResponse receiveMessageResponse =
sqsExtendedClient.receiveMessage(
                    ReceiveMessageRequest.builder().queueUrl(queueUrl).build());

            List<Message> messages = receiveMessageResponse.messages();
            if (messages.isEmpty()) {
                throw new RuntimeException("No messages received");
            }

            Message message = messages.getFirst();
            logger.info("\nMessage received.");
            logger.info("  ID: {}", message.messageId());
            logger.info("  Receipt handle: {}", message.receiptHandle());
            logger.info("  Message body size: {}", message.body().length());
            logger.info("  Message body (first 5 characters): {}",
message.body().substring(0, 5));
```

```
            return message;
        } catch (RuntimeException e) {
            logger.error("Error during message processing: {}", e.getMessage(),
e);
            throw e;
        }
    }
```

- 자세한 정보는 AWS SDK for Java 2.x 개발자 안내서를 참조하세요.
- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 항목을 참조하세요.
  - CreateBucket
  - PutBucketLifecycleConfiguration
  - ReceiveMessage
  - SendMessage

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon S3 이벤트 알림 수신 및 처리

다음 코드 예시에서는 객체 지향적 방식으로 S3 이벤트 알림을 사용하는 방법을 보여줍니다.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 예시에서는 Amazon SQS를 사용하여 S3 알림 이벤트를 처리하는 방법을 보여줍니다.

```
    /**
     * This method receives S3 event notifications by using an SqsAsyncClient.
```

```
    * After the client receives the messages it deserializes the JSON payload
 and logs them. It uses
    * the S3EventNotification class (part of the S3 event notification API for
 Java) to deserialize
    * the JSON payload and access the messages in an object-oriented way.
    *
    * @param queueUrl The URL of the AWS SQS queue that receives the S3 event
 notifications.
    * @see <a href="https://sdk.amazonaws.com/java/api/latest/software/amazon/
awssdk/eventnotifications/s3/model/package-summary.html">S3EventNotification
 API</a>.
    * <p>
    * To use S3 event notification serialization/deserialization to objects, add
 the following
    * dependency to your Maven pom.xml file.
    * <dependency>
    * <groupId>software.amazon.awssdk</groupId>
    * <artifactId>s3-event-notifications</artifactId>
    * <version><LATEST></version>
    * </dependency>
    * <p>
    * The S3 event notification API became available with version 2.25.11 of the
 Java SDK.
    * <p>
    * This example shows the use of the API with AWS SQS, but it can be used to
 process S3 event notifications
    * in AWS SNS or AWS Lambda as well.
    * <p>
    * Note: The S3EventNotification class does not work with messages routed
 through AWS EventBridge.
    */
   static void processS3Events(String bucketName, String queueUrl, String
 queueArn) {
       try {
           // Configure the bucket to send Object Created and Object Tagging
 notifications to an existing SQS queue.
           s3Client.putBucketNotificationConfiguration(b -> b
                   .notificationConfiguration(ncb -> ncb
                           .queueConfigurations(qcb -> qcb
                                   .events(Event.S3_OBJECT_CREATED,
 Event.S3_OBJECT_TAGGING)
                                   .queueArn(queueArn)))
                           .bucket(bucketName)
           ).join();
```

```
            triggerS3EventNotifications(bucketName);
            // Wait for event notifications to propagate.
            Thread.sleep(Duration.ofSeconds(5).toMillis());

            boolean didReceiveMessages = true;
            while (didReceiveMessages) {
                // Display the number of messages that are available in the
queue.
                sqsClient.getQueueAttributes(b -> b
                                .queueUrl(queueUrl)

.attributeNames(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)
                        ).thenAccept(attributeResponse ->
                                logger.info("Approximate number of messages in
the queue: {}",

attributeResponse.attributes().get(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)))
                        .join();

                // Receive the messages.
                ReceiveMessageResponse response = sqsClient.receiveMessage(b -> b
                        .queueUrl(queueUrl)
                ).get();
                logger.info("Count of received messages: {}",
response.messages().size());
                didReceiveMessages = !response.messages().isEmpty();

                // Create a collection to hold the received message for deletion
                // after we log the messages.
                HashSet<DeleteMessageBatchRequestEntry> messagesToDelete = new
HashSet<>();
                // Process each message.
                response.messages().forEach(message -> {
                    logger.info("Message id: {}", message.messageId());
                    // Deserialize JSON message body to a S3EventNotification
object
                    // to access messages in an object-oriented way.
                    S3EventNotification event =
S3EventNotification.fromJson(message.body());

                    // Log the S3 event notification record details.
                    if (event.getRecords() != null) {
                        event.getRecords().forEach(record -> {
```

```
                            String eventName = record.getEventName();
                            String key = record.getS3().getObject().getKey();
                            logger.info(record.toString());
                            logger.info("Event name is {} and key is {}",
eventName, key);
                        });
                    }
                    // Add logged messages to collection for batch deletion.
                    messagesToDelete.add(DeleteMessageBatchRequestEntry.builder()
                            .id(message.messageId())
                            .receiptHandle(message.receiptHandle())
                            .build());
                });
                // Delete messages.
                if (!messagesToDelete.isEmpty()) {

sqsClient.deleteMessageBatch(DeleteMessageBatchRequest.builder()
                            .queueUrl(queueUrl)
                            .entries(messagesToDelete)
                            .build()
                    ).join();
                }
            } // End of while block.
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - DeleteMessageBatch
  - GetQueueAttributes
  - PutBucketNotificationConfiguration
  - ReceiveMessage

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon SNS 메시지를 Amazon SQS 대기열에 게시

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 주제(FIFO 또는 비 FIFO)를 생성합니다.
- 필터 적용 옵션을 사용하여 여러 개의 대기열로 주제를 구독합니다.
- 주제에 메시지를 게시합니다.
- 대기열에서 받은 메시지를 폴링합니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
 "sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
```

```
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();
```

```
            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues scenario is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this scenario, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for
configuring the topic and the subscriptions for the 2 queues." +
```

```
                            $"\r\nYou can then post to the topic and see the
 results in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
                            $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                            $"\r\nYou can then post to the topic and see the
 results in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended
 to the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic,
 deduplication is supported." +
                                $"\r\nDeduplication IDs are either set in the
 message or automatically generated " +
                                $"\r\nfrom content using a hash function.\r\n" +
                                $"\r\nIf a message is successfully published to an
 SNS FIFO topic, any message " +
                                $"\r\npublished and determined to have the same
 deduplication ID, " +
                                $"\r\nwithin the five-minute deduplication
 interval, is accepted but not delivered.\r\n" +
```

```
                                $"\r\nFor more information about deduplication, " +
                                $"\r\nsee https://docs.aws.amazon.com/sns/latest/
dg/fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName,
 _useFifoTopic, _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
                          $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                          $"\r\nhas been created.\r\n");

        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
 Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS
 queue: ", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
 appended to the queue name.");
                }
```

```
                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must
be attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
```

```
    /// <returns>Async Task.</returns>
    public static async Task SetupFilters(int queueCount, string queueArn, string
 queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
                    "If you add a filter to this subscription, then only the
 filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
```

```
        }
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" :
 "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }
```

```
    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID
for this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this
message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
```

```
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?",
false);
        }
    }

    /// <summary>
    /// Poll for the published messages to see the results of the user's choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<List<Message>> PollForMessages(string queueUrl)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                          "\r\nPress any key to continue.");
        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
```

```
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl,
10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the
queue at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                              $"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message>
messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
```

```
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer =
 true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}
```

Amazon SQS 작업을 래핑하는 클래스를 만듭니다.

```
/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

    /// <summary>
    /// Constructor for the Amazon SQS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SQS client.</param>
    public SQSWrapper(IAmazonSQS amazonSQS)
    {
        _amazonSQSClient = amazonSQS;
    }

    /// <summary>
    /// Create a queue with a specific name.
    /// </summary>
    /// <param name="queueName">The name for the queue.</param>
    /// <param name="useFifoQueue">True to use a FIFO queue.</param>
    /// <returns>The url for the queue.</returns>
    public async Task<string> CreateQueueWithName(string queueName, bool
 useFifoQueue)
    {
        int maxMessage = 256 * 1024;
        var queueAttributes = new Dictionary<string, string>
        {
            {
                QueueAttributeName.MaximumMessageSize,
                maxMessage.ToString()
            }
        };

        var createQueueRequest = new CreateQueueRequest()
        {
            QueueName = queueName,
            Attributes = queueAttributes
```

```
        };

        if (useFifoQueue)
        {
            // Update the name if it is not correct for a FIFO queue.
            if (!queueName.EndsWith(".fifo"))
            {
                createQueueRequest.QueueName = queueName + ".fifo";
            }

            // Add an attribute for a FIFO queue.
            createQueueRequest.Attributes.Add(
                QueueAttributeName.FifoQueue, "true");
        }

        var createResponse = await _amazonSQSClient.CreateQueueAsync(
            new CreateQueueRequest()
            {
                QueueName = queueName
            });
        return createResponse.QueueUrl;
    }

    /// <summary>
    /// Get the ARN for a queue from its URL.
    /// </summary>
    /// <param name="queueUrl">The URL of the queue.</param>
    /// <returns>The ARN of the queue.</returns>
    public async Task<string> GetQueueArnByUrl(string queueUrl)
    {
        var getAttributesRequest = new GetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
        };

        var getAttributesResponse = await
_amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

        return getAttributesResponse.QueueARN;
    }

    /// <summary>
```

```
    /// Set the policy attribute of a queue for a topic.
    /// </summary>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="queueUrl">The url for the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> SetQueuePolicyForTopic(string queueArn, string
topicArn, string queueUrl)
    {
        var queuePolicy = "{" +
                                "\"Version\": \"2012-10-17\"," +
                                "\"Statement\": [{" +
                                     "\"Effect\": \"Allow\"," +
                                     "\"Principal\": {" +
                                         $"\"Service\": " +
                                             "\"sns.amazonaws.com\"" +
                                            "}," +
                                     "\"Action\": \"sqs:SendMessage\"," +
                                     $"\"Resource\": \"{queueArn}\"," +
                                      "\"Condition\": {" +
                                           "\"ArnEquals\": {" +
                                                $"\"aws:SourceArn\":
\"{topicArn}\"" +
                                            "}" +
                                        "}" +
                                "}]" +
                            "}";
        var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
            new SetQueueAttributesRequest()
            {
                QueueUrl = queueUrl,
                Attributes = new Dictionary<string, string>() { { "Policy",
queuePolicy } }
            });
        return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
```

```
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl,
 List<Message> messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
 _amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
```

```
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Amazon SNS 작업을 래핑하는 클래스를 만듭니다.

```
/// <summary>
/// Wrapper for Amazon Simple Notification Service (SNS) operations.
/// </summary>
public class SNSWrapper
{
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;

    /// <summary>
    /// Constructor for the Amazon SNS wrapper.
    /// </summary>
    /// <param name="amazonSQS">The injected Amazon SNS client.</param>
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)
    {
        _amazonSNSClient = amazonSNS;
    }

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
 attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
```

```
    /// <returns>The ARN of the new topic.</returns>
    public async Task<string> CreateTopicWithName(string topicName, bool
 useFifoTopic, bool useContentBasedDeduplication)
    {
        var createTopicRequest = new CreateTopicRequest()
        {
            Name = topicName,
        };

        if (useFifoTopic)
        {
            // Update the name if it is not correct for a FIFO topic.
            if (!topicName.EndsWith(".fifo"))
            {
                createTopicRequest.Name = topicName + ".fifo";
            }

            // Add the attributes from the method parameters.
            createTopicRequest.Attributes = new Dictionary<string, string>
            {
                { "FifoTopic", "true" }
            };
            if (useContentBasedDeduplication)
            {
                createTopicRequest.Attributes.Add("ContentBasedDeduplication",
 "true");
            }
        }

        var createResponse = await
 _amazonSNSClient.CreateTopicAsync(createTopicRequest);
        return createResponse.TopicArn;
    }

    /// <summary>
    /// Subscribe a queue to a topic with optional filters.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="useFifoTopic">The optional filtering policy for the
 subscription.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <returns>The ARN of the new subscription.</returns>
    public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
 filterPolicy, string queueArn)
```

```
    {
        var subscribeRequest = new SubscribeRequest()
        {
            TopicArn = topicArn,
            Protocol = "sqs",
            Endpoint = queueArn
        };

        if (!string.IsNullOrEmpty(filterPolicy))
        {
            subscribeRequest.Attributes = new Dictionary<string, string>
 { { "FilterPolicy", filterPolicy } };
        }

        var subscribeResponse = await
 _amazonSNSClient.SubscribeAsync(subscribeRequest);
        return subscribeResponse.SubscriptionArn;
    }

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
 and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</
param>
    /// <param name="attributeValue">The optional attribute value for the
 message.</param>
    /// <param name="deduplicationId">The optional deduplication ID for the
 message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>
    /// <returns>The ID of the message published.</returns>
    public async Task<string> PublishToTopicWithAttribute(
        string topicArn,
        string message,
        string? attributeName = null,
        string? attributeValue = null,
        string? deduplicationId = null,
        string? groupId = null)
    {
        var publishRequest = new PublishRequest()
        {
            TopicArn = topicArn,
```

```
            Message = message,
            MessageDeduplicationId = deduplicationId,
            MessageGroupId = groupId
        };

        if (attributeValue != null)
        {
            // Add the string attribute if it exists.
            publishRequest.MessageAttributes =
                new Dictionary<string, MessageAttributeValue>
                {
                    { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
                };
        }

        var publishResponse = await
_amazonSNSClient.PublishAsync(publishRequest);
        return publishResponse.MessageId;
    }


    /// <summary>
    /// Unsubscribe from a topic by a subscription ARN.
    /// </summary>
    /// <param name="subscriptionArn">The ARN of the subscription.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> UnsubscribeByArn(string subscriptionArn)
    {
        var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
            new UnsubscribeRequest()
            {
                SubscriptionArn = subscriptionArn
            });
        return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a topic by its topic ARN.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteTopicByArn(string topicArn)
    {
```

```
        var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
            new DeleteTopicRequest()
            {
                TopicArn = topicArn
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하세요.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
//! Workflow for messaging with topics and queues using Amazon SNS and Amazon
 SQS.
/*!
 \param clientConfig Aws client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::TopicsAndQueues::messagingWithTopicsAndQueues(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << "Welcome to messaging with topics and queues." << std::endl;
    printAsterisksLine();
    std::cout << "In this workflow, you will create an SNS topic and subscribe "
              << NUMBER_OF_QUEUES <<
              " SQS queues to the topic." << std::endl;
    std::cout
            << "You can select from several options for configuring the topic and
 the subscriptions for the "
            << NUMBER_OF_QUEUES << " queues." << std::endl;
    std::cout << "You can then post to the topic and see the results in the
 queues."
              << std::endl;

    Aws::SNS::SNSClient snsClient(clientConfiguration);

    printAsterisksLine();

    std::cout << "SNS topics can be configured as FIFO (First-In-First-Out)."
              << std::endl;
    std::cout
            << "FIFO topics deliver messages in order and support deduplication
 and message filtering."
            << std::endl;
    bool isFifoTopic = askYesNoQuestion(
            "Would you like to work with FIFO topics? (y/n) ");

    bool contentBasedDeduplication = false;
    Aws::String topicName;
    if (isFifoTopic) {
        printAsterisksLine();
        std::cout << "Because you have chosen a FIFO topic, deduplication is
 supported."
                  << std::endl;
        std::cout
                << "Deduplication IDs are either set in the message or
 automatically generated "
```

```
                << "from content using a hash function." << std::endl;
        std::cout
                << "If a message is successfully published to an SNS FIFO topic,
 any message "
                << "published and determined to have the same deduplication ID, "
                << std::endl;
        std::cout
                << "within the five-minute deduplication interval, is accepted
 but not delivered."
                << std::endl;
        std::cout
                << "For more information about deduplication, "
                << "see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-
dedup.html."
                << std::endl;
        contentBasedDeduplication = askYesNoQuestion(
                "Use content-based deduplication instead of entering a
 deduplication ID? (y/n) ");
    }

    printAsterisksLine();

    Aws::SQS::SQSClient sqsClient(clientConfiguration);
    Aws::Vector<Aws::String> queueURLS;
    Aws::Vector<Aws::String> subscriptionARNS;

    Aws::String topicARN;
    {
        topicName = askQuestion("Enter a name for your SNS topic. ");

        // 1.  Create an Amazon SNS topic, either FIFO or non-FIFO.
        Aws::SNS::Model::CreateTopicRequest request;

        if (isFifoTopic) {
            request.AddAttributes("FifoTopic", "true");
            if (contentBasedDeduplication) {
                request.AddAttributes("ContentBasedDeduplication", "true");
            }
            topicName = topicName + FIFO_SUFFIX;

            std::cout
                    << "Because you have selected a FIFO topic, '.fifo' must be
 appended to the topic name."
                    << std::endl;
```

```
        }

        request.SetName(topicName);

        Aws::SNS::Model::CreateTopicOutcome outcome =
snsClient.CreateTopic(request);

        if (outcome.IsSuccess()) {
            topicARN = outcome.GetResult().GetTopicArn();
            std::cout << "Your new topic with the name '" << topicName
                      << "' and the topic Amazon Resource Name (ARN) " <<
std::endl;
            std::cout << "'" << topicARN << "' has been created." << std::endl;

        }
        else {
            std::cerr << "Error with TopicsAndQueues::CreateTopic. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

            return false;
        }
    }

    printAsterisksLine();

    std::cout << "Now you will create " << NUMBER_OF_QUEUES
              << " SQS queues to subscribe to the topic." << std::endl;
    Aws::Vector<Aws::String> queueNames;
    bool filteringMessages = false;
    bool first = true;
    for (int i = 1; i <= NUMBER_OF_QUEUES; ++i) {
        Aws::String queueURL;
        Aws::String queueName;
        {
            printAsterisksLine();
            std::ostringstream ostringstream;
            ostringstream << "Enter a name for " << (first ? "an" : "the next")
```

```
                        << " SQS queue. ";
            queueName = askQuestion(ostringstream.str());

            // 2.  Create an SQS queue.
            Aws::SQS::Model::CreateQueueRequest request;
            if (isFifoTopic) {

request.AddAttributes(Aws::SQS::Model::QueueAttributeName::FifoQueue,
                                  "true");
                queueName = queueName + FIFO_SUFFIX;

                if (first) // Only explain this once.
                {
                    std::cout
                            << "Because you are creating a FIFO SQS queue,
'.fifo' must "
                            << "be appended to the queue name." << std::endl;
                }
            }

            request.SetQueueName(queueName);
            queueNames.push_back(queueName);

            Aws::SQS::Model::CreateQueueOutcome outcome =
                    sqsClient.CreateQueue(request);

            if (outcome.IsSuccess()) {
                queueURL = outcome.GetResult().GetQueueUrl();
                std::cout << "Your new SQS queue with the name '" << queueName
                          << "' and the queue URL " << std::endl;
                std::cout << "'" << queueURL << "' has been created." <<
std::endl;
            }
            else {
                std::cerr << "Error with SQS::CreateQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);
```

```
                return false;
            }
        }
        queueURLS.push_back(queueURL);

        if (first) // Only explain this once.
        {
            std::cout
                    << "The queue URL is used to retrieve the queue ARN, which is
"
                    << "used to create a subscription." << std::endl;
        }

        Aws::String queueARN;
        {
            // 3.  Get the SQS queue ARN attribute.
            Aws::SQS::Model::GetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);

request.AddAttributeNames(Aws::SQS::Model::QueueAttributeName::QueueArn);

            Aws::SQS::Model::GetQueueAttributesOutcome outcome =
                    sqsClient.GetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                const Aws::Map<Aws::SQS::Model::QueueAttributeName, Aws::String>
&attributes =
                        outcome.GetResult().GetAttributes();
                const auto &iter = attributes.find(
                        Aws::SQS::Model::QueueAttributeName::QueueArn);
                if (iter != attributes.end()) {
                    queueARN = iter->second;
                    std::cout << "The queue ARN '" << queueARN
                              << "' has been retrieved."
                              << std::endl;
                }
                else {
                    std::cerr
                            << "Error ARN attribute not returned by
GetQueueAttribute."
                            << std::endl;

                    cleanUp(topicARN,
                            queueURLS,
```

```
                            subscriptionARNS,
                            snsClient,
                            sqsClient);

                    return false;
                }
            }
            else {
                std::cerr << "Error with SQS::GetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        if (first) {
            std::cout
                    << "An IAM policy must be attached to an SQS queue, enabling
it to receive "
                       "messages from an SNS topic." << std::endl;
        }

        {
            // 4.  Set the SQS queue policy attribute with a policy enabling the
receipt of SNS messages.
            Aws::SQS::Model::SetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);
            Aws::String policy = createPolicyForQueue(queueARN, topicARN);
            request.AddAttributes(Aws::SQS::Model::QueueAttributeName::Policy,
                                  policy);

            Aws::SQS::Model::SetQueueAttributesOutcome outcome =
                    sqsClient.SetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                std::cout << "The attributes for the queue '" << queueName
                          << "' were successfully updated." << std::endl;
```

```
            }
            else {
                std::cerr << "Error with SQS::SetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        printAsterisksLine();

        {
            // 5.  Subscribe the SQS queue to the SNS topic.
            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);
            if (isFifoTopic) {
                if (first) {
                    std::cout << "Subscriptions to a FIFO topic can have
 filters."
                              << std::endl;
                    std::cout
                            << "If you add a filter to this subscription, then
 only the filtered messages "
                            << "will be received in the queue." << std::endl;
                    std::cout << "For information about message filtering, "
                              << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
                              << std::endl;
                    std::cout << "For this example, you can filter messages by a
 \""
                              << TONE_ATTRIBUTE << "\" attribute." << std::endl;
                }

                std::ostringstream ostringstream;
                ostringstream << "Filter messages for \"" << queueName
```

```
                            << "\"'s subscription to the topic \""
                            << topicName << "\"?  (y/n)";

                // Add filter if user answers yes.
                if (askYesNoQuestion(ostringstream.str())) {
                    Aws::String jsonPolicy = getFilterPolicyFromUser();
                    if (!jsonPolicy.empty()) {
                        filteringMessages = true;

                        std::cout << "This is the filter policy for this
subscription."
                                  << std::endl;
                        std::cout << jsonPolicy << std::endl;

                        request.AddAttributes("FilterPolicy", jsonPolicy);
                    }
                    else {
                        std::cout
                                << "Because you did not select any attributes, no
filter "
                                << "will be added to this subscription." <<
std::endl;
                    }
                }
            }  // if (isFifoTopic)
            Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

            if (outcome.IsSuccess()) {
                Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
                std::cout << "The queue '" << queueName
                          << "' has been subscribed to the topic '"
                          << "'" << topicName << "'" << std::endl;
                std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
                          << std::endl;
                subscriptionARNS.push_back(subscriptionARN);
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Subscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        first = false;
    }

    first = true;
    do {
        printAsterisksLine();

        // 6.  Publish a message to the SNS topic.
        Aws::SNS::Model::PublishRequest request;
        request.SetTopicArn(topicARN);
        Aws::String message = askQuestion("Enter a message text to publish.  ");
        request.SetMessage(message);
        if (isFifoTopic) {
            if (first) {
                std::cout
                        << "Because you are using a FIFO topic, you must set a
message group ID."
                        << std::endl;
                std::cout
                        << "All messages within the same group will be received
in the "
                        << "order they were published." << std::endl;
            }
            Aws::String messageGroupID = askQuestion(
                    "Enter a message group ID for this message. ");
            request.SetMessageGroupId(messageGroupID);
            if (!contentBasedDeduplication) {
                if (first) {
                    std::cout
                            << "Because you are not using content-based
deduplication, "
                            << "you must enter a deduplication ID." << std::endl;
                }
                Aws::String deduplicationID = askQuestion(
```

```
                        "Enter a deduplication ID for this message. ");
                request.SetMessageDeduplicationId(deduplicationID);
            }
        }

        if (filteringMessages && askYesNoQuestion(
                "Add an attribute to this message? (y/n) ")) {
            for (size_t i = 0; i < TONES.size(); ++i) {
                std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
            }
            int selection = askQuestionForIntRange(
                    "Enter a number for an attribute. ",
                    1, static_cast<int>(TONES.size()));
            Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
            messageAttributeValue.SetDataType("String");
            messageAttributeValue.SetStringValue(TONES[selection - 1]);
            request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
        }

        Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

        if (outcome.IsSuccess()) {
            std::cout << "Your message was successfully published." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::Publish. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

            return false;
        }

        first = false;
    } while (askYesNoQuestion("Post another message? (y/n) "));

    printAsterisksLine();

    std::cout << "Now the SQS queue will be polled to retrieve the messages."
```

```
              << std::endl;
    askQuestion("Press any key to continue...", alwaysTrueTest);

    for (size_t i = 0; i < queueURLS.size(); ++i) {
        // 7.  Poll an SQS queue for its messages.
        std::vector<Aws::String> messages;
        std::vector<Aws::String> receiptHandles;
        while (true) {
            Aws::SQS::Model::ReceiveMessageRequest request;
            request.SetMaxNumberOfMessages(10);
            request.SetQueueUrl(queueURLS[i]);

            // Setting WaitTimeSeconds to non-zero enables long polling.
            // For information about long polling, see
            // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
            request.SetWaitTimeSeconds(1);
            Aws::SQS::Model::ReceiveMessageOutcome outcome =
                    sqsClient.ReceiveMessage(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::SQS::Model::Message> &newMessages =
 outcome.GetResult().GetMessages();
                if (newMessages.empty()) {
                    break;
                }
                else {
                    for (const Aws::SQS::Model::Message &message: newMessages) {
                        messages.push_back(message.GetBody());
                        receiptHandles.push_back(message.GetReceiptHandle());
                    }
                }
            }
            else {
                std::cerr << "Error with SQS::ReceiveMessage. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);
```

```
                return false;
            }
        }

        printAsterisksLine();

        if (messages.empty()) {
            std::cout << "No messages were ";
        }
        else if (messages.size() == 1) {
            std::cout << "One message was ";
        }
        else {
            std::cout << messages.size() << " messages were ";
        }
        std::cout << "received by the queue '" << queueNames[i]
                  << "'." << std::endl;
        for (const Aws::String &message: messages) {
            std::cout << "  Message : '" << message << "'."
                      << std::endl;
        }

        // 8.  Delete a batch of messages from an SQS queue.
        if (!receiptHandles.empty()) {
            Aws::SQS::Model::DeleteMessageBatchRequest request;
            request.SetQueueUrl(queueURLS[i]);
            int id = 1; // Ids must be unique within a batch delete request.
            for (const Aws::String &receiptHandle: receiptHandles) {
                Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
                entry.SetId(std::to_string(id));
                ++id;
                entry.SetReceiptHandle(receiptHandle);
                request.AddEntries(entry);
            }

            Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
                    sqsClient.DeleteMessageBatch(request);

            if (outcome.IsSuccess()) {
                std::cout << "The batch deletion of messages was successful."
                          << std::endl;
            }
            else {
                std::cerr << "Error with SQS::DeleteMessageBatch. "
```

```
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }
    }

    return cleanUp(topicARN,
                   queueURLS,
                   subscriptionARNS,
                   snsClient,
                   sqsClient,
                   true); // askUser
}


bool AwsDoc::TopicsAndQueues::cleanUp(const Aws::String &topicARN,
                                      const Aws::Vector<Aws::String> &queueURLS,
                                      const Aws::Vector<Aws::String>
 &subscriptionARNS,
                                      const Aws::SNS::SNSClient &snsClient,
                                      const Aws::SQS::SQSClient &sqsClient,
                                      bool askUser) {
    bool result = true;
    printAsterisksLine();
    if (!queueURLS.empty() && askUser &&
        askYesNoQuestion("Delete the SQS queues? (y/n) ")) {

        for (const auto &queueURL: queueURLS) {
            // 9.  Delete an SQS queue.
            Aws::SQS::Model::DeleteQueueRequest request;
            request.SetQueueUrl(queueURL);

            Aws::SQS::Model::DeleteQueueOutcome outcome =
                    sqsClient.DeleteQueue(request);

            if (outcome.IsSuccess()) {
                std::cout << "The queue with URL '" << queueURL
```

```
                        << "' was successfully deleted." << std::endl;
            }
            else {
                std::cerr << "Error with SQS::DeleteQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }

        for (const auto &subscriptionARN: subscriptionARNS) {
            // 10. Unsubscribe an SNS subscription.
            Aws::SNS::Model::UnsubscribeRequest request;
            request.SetSubscriptionArn(subscriptionARN);

            Aws::SNS::Model::UnsubscribeOutcome outcome =
                    snsClient.Unsubscribe(request);

            if (outcome.IsSuccess()) {
                std::cout << "Unsubscribe of subscription ARN '" <<
subscriptionARN
                          << "' was successful." << std::endl;
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Unsubscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    printAsterisksLine();
    if (!topicARN.empty() && askUser &&
        askYesNoQuestion("Delete the SNS topic? (y/n) ")) {

        // 11. Delete an SNS topic.
        Aws::SNS::Model::DeleteTopicRequest request;
        request.SetTopicArn(topicARN);

        Aws::SNS::Model::DeleteTopicOutcome outcome =
snsClient.DeleteTopic(request);

        if (outcome.IsSuccess()) {
```

```
            std::cout << "The topic with ARN '" << topicARN
                      << "' was successfully deleted." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::DeleteTopicRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}

//! Create an IAM policy that gives an SQS queue permission to receive messages
 from an SNS topic.
/*!
 \sa createPolicyForQueue()
 \param queueARN: The SQS queue Amazon Resource Name (ARN).
 \param topicARN: The SNS topic ARN.
 \return Aws::String: The policy as JSON.
 */
Aws::String AwsDoc::TopicsAndQueues::createPolicyForQueue(const Aws::String
 &queueARN,
                                                           const Aws::String
 &topicARN) {
    std::ostringstream policyStream;
    policyStream << R"({
        "Statement": [
        {
            "Effect": "Allow",
                    "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "sqs:SendMessage",
                    "Resource": ")" << queueARN << R"(",
                    "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": ")" << topicARN << R"("
                }
            }
        }
        ]
    })";
```

```
    return policyStream.str();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 다음 항목을 참조하세요.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
 "strings"
```

```
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions
 so that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string,
 bool, bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or
 standard.\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.
\n" +
   "Deduplication IDs are either set in the message or are automatically
 generated\n" +
```

```
   "from content using a hash function. If a message is successfully published to
\n" +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted
\n" +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to
\n"+
   "the topic name.", FIFO_SUFFIX)
 }

 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN)
 \n"+
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS
 queue. ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
  }
 }
```

```
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string,
 topicArn string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
 if err != nil {
  panic(err)
 }
 log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
  "messages to it.")
 log.Println(strings.Repeat("-", 88))

 var filterPolicy map[string][]string
 if isFifoTopic {
  if ordinal == "first" {
   log.Println("Subscriptions to a FIFO topic can have filters.\n" +
    "If you add a filter to this subscription, then only the filtered messages\n"
 +
    "will be received in the queue.\n" +
    "For information about message filtering, see\n" +
    "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
    "For this example, you can filter messages by a \"tone\" attribute.")
  }

  wantFiltering := runner.questioner.AskBool(
   fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
    "from the %v topic? (y/n) ", queueName, topicName), "y")
```

```
  if wantFiltering {
   log.Println("You can filter messages by one or more of the following \"tone\"
 attributes.")

   var toneSelections []string
   askAboutTones := true
   for askAboutTones {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelections = append(toneSelections, ToneChoices[toneIndex])
    askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
 the filter? (y/n) ", "y")
   }
   log.Printf("Your subscription will be filtered to only pass the following
 tones: %v\n", toneSelections)
   filterPolicy = map[string][]string{TONE_KEY: toneSelections}
  }
 }

 subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
 filterPolicy)
 if err != nil {
  panic(err)
 }
 log.Printf("The queue %v is now subscribed to the topic %v with the subscription
 ARN %v.\n",
  queueName, topicName, subscriptionArn)

 return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn
 string, isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
 var message string
 var groupId string
 var dedupId string
 var toneSelection string
 publishMore := true
 for publishMore {
  groupId = ""
  dedupId = ""
  toneSelection = ""
  message = runner.questioner.Ask("Enter a message to publish: ")
  if isFifoTopic {
```

```
    log.Println("Because you are using a FIFO topic, you must set a message group
 ID.\n" +
     "All messages within the same group will be received in the order they were
 published.")
    groupId = runner.questioner.Ask("Enter a message group ID: ")
    if !contentBasedDeduplication {
     log.Println("Because you are not using content-based deduplication,\n" +
      "you must enter a deduplication ID.")
     dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
    }
   }
   if usingFilters {
    if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
     toneIndex := runner.questioner.AskChoice(
      "Enter the number of the tone you want to filter by:\n", ToneChoices)
     toneSelection = ToneChoices[toneIndex]
    }
   }

   err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId,
 TONE_KEY, toneSelection)
   if err != nil {
    panic(err)
   }
   log.Println(("Your message was published."))

   publishMore = runner.questioner.AskBool("Do you want to publish another
 messsage? (y/n) ", "y")
 }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
    break
```

```
    }
    messages = append(messages, currentMsgs...)
   }
   if len(messages) == 0 {
    log.Printf("No messages were received by queue %v.\n", queueUrl)
   } else if len(messages) == 1 {
    log.Printf("One message was received by queue %v:\n", queueUrl)

   } else {
    log.Printf("%v messages were received by queue %v:\n", len(messages),
  queueUrl)
   }
   for msgIndex, message := range messages {
    messageBody := MessageBody{}
    err := json.Unmarshal([]byte(*message.Body), &messageBody)
    if err != nil {
     panic(err)
    }
    log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
   }

   if len(messages) > 0 {
    log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
    err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
    if err != nil {
     panic(err)
    }
   }
  }
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
 the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
```

```
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to
 the\n"+
  "topic. You can select from several options for configuring the topic and the
\n"+
  "subscriptions for the queues. You can then post to the topic and see the
 results\n"+
  "in the queues.\n", queueCount)

 log.Println(strings.Repeat("-", 88))

 runner := ScenarioRunner{
  questioner: questioner,
  snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
  sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
 }
 resources.snsActor = runner.snsActor
 resources.sqsActor = runner.sqsActor

 topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
 runner.CreateTopic(ctx)
 resources.topicArn = topicArn
 log.Println(strings.Repeat("-", 88))

 log.Printf("Now you will create %v SQS queues and subscribe them to the topic.
\n", queueCount)
 ordinals := []string{"first", "next"}
 usingFilters := false
```

```
 for _, ordinal := range ordinals {
  queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl,
 topicName, topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }

 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(ctx, resources.queueUrls)

 log.Println(strings.Repeat("-", 88))

 wantCleanup := questioner.AskBool("Do you want to remove all AWS resources
 created for this scenario? (y/n) ", "y")
 if wantCleanup {
  log.Println("Cleaning up resources...")
  resources.Cleanup(ctx)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

이 예시에서 사용되는 S3 작업을 래핑하는 구조를 정의합니다.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
```

```
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
 an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent
 to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered
 according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string)
 error {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
 err)
 }
 return err
}
```

이 예시에서 사용되는 S3 작업을 래핑하는 구조를 정의합니다.

```
import (
 "context"
 "encoding/json"
```

```
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can
 specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
```

```
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string)
 (string, error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy
 to an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages
 to the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
```

```
    string(types.QueueAttributeNamePolicy): string(policyBytes),
   },
   QueueUrl: aws.String(queueUrl),
  })
  if err != nil {
   log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
  queueUrl, err)
  }
  return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
```

```
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 } else {
  messages = result.Messages
 }
 return messages, err
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of
 messages from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

리소스를 정리합니다.

```
import (
 "context"
 "fmt"
 "log"
 "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 topicArn  string
 queueUrls []string
 snsActor  *actions.SnsActions
 sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Println("Something went wrong during cleanup. Use the AWS Management
 Console\n" +
    "to remove any remaining resources that were created for this scenario.")
  }
 }()

 var err error
 if resources.topicArn != "" {
  log.Printf("Deleting topic %v.\n", resources.topicArn)
  err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
  if err != nil {
   panic(err)
  }
 }

 for _, queueUrl := range resources.queueUrls {
  log.Printf("Deleting queue %v.\n", queueUrl)
  err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
```

```
  if err != nil {
   panic(err)
  }
 }
}
```

- API 세부 정보는 AWS SDK for Go API 참조의 다음 주제를 참조하세요.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package com.example.sns;

import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
```

```
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.MessageAttributeValue;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import
 software.amazon.awssdk.services.sns.model.SetSubscriptionAttributesRequest;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.google.gson.JsonPrimitive;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
```

```
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Gives the user three options to choose from.
 * 2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
 * 3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
 * 4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
 * 5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
 * 6. Subscribes to the SQS queue.
 * 7. Publishes a message to the topic.
 * 8. Displays the messages.
 * 9. Deletes the received message.
 * 10. Unsubscribes from the topic.
 * 11. Deletes the SNS topic.
 */
public class SNSWorkflow {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage:\n" +
            "    <fifoQueueARN>\n\n" +
            "Where:\n" +
            "    accountId - Your AWS account Id value.";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();

        SqsClient sqsClient = SqsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();
```

```
        Scanner in = new Scanner(System.in);
        String accountId = args[0];
        String useFIFO;
        String duplication = "n";
        String topicName;
        String deduplicationID = null;
        String groupId = null;

        String topicArn;
        String sqsQueueName;
        String sqsQueueUrl;
        String sqsQueueArn;
        String subscriptionArn;
        boolean selectFIFO = false;

        String message;
        List<Message> messageList;
        List<String> filterList = new ArrayList<>();
        String msgAttValue = "";

        System.out.println(DASHES);
        System.out.println("Welcome to messaging with topics and queues.");
        System.out.println("In this scenario, you will create an SNS topic and
 subscribe an SQS queue to the topic.\n" +
            "You can select from several options for configuring the topic and
 the subscriptions for the queue.\n" +
            "You can then post to the topic and see the results in the queue.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("SNS topics can be configured as FIFO (First-In-First-
Out).\n" +
            "FIFO topics deliver messages in order and support deduplication and
 message filtering.\n" +
            "Would you like to work with FIFO topics? (y/n)");
        useFIFO = in.nextLine();
        if (useFIFO.compareTo("y") == 0) {
            selectFIFO = true;
            System.out.println("You have selected FIFO");
            System.out.println(" Because you have chosen a FIFO topic,
 deduplication is supported.\n" +
                "        Deduplication IDs are either set in the message or
 automatically generated from content using a hash function.\n"
```

```
                +
                "        If a message is successfully published to an SNS FIFO
 topic, any message published and determined to have the same deduplication ID,
\n"
                +
                "        within the five-minute deduplication interval, is
 accepted but not delivered.\n" +
                "        For more information about deduplication, see https://
docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.");

            System.out.println(
                "Would you like to use content-based deduplication instead of
 entering a deduplication ID? (y/n)");
            duplication = in.nextLine();
            if (duplication.compareTo("y") == 0) {
                System.out.println("Please enter a group id value");
                groupId = in.nextLine();
            } else {
                System.out.println("Please enter deduplication Id value");
                deduplicationID = in.nextLine();
                System.out.println("Please enter a group id value");
                groupId = in.nextLine();
            }
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a topic.");
        System.out.println("Enter a name for your SNS topic.");
        topicName = in.nextLine();
        if (selectFIFO) {
            System.out.println("Because you have selected a FIFO topic, '.fifo'
 must be appended to the topic name.");
            topicName = topicName + ".fifo";
            System.out.println("The name of the topic is " + topicName);
            topicArn = createFIFO(snsClient, topicName, duplication);
            System.out.println("The ARN of the FIFO topic is " + topicArn);

        } else {
            System.out.println("The name of the topic is " + topicName);
            topicArn = createSNSTopic(snsClient, topicName);
            System.out.println("The ARN of the non-FIFO topic is " + topicArn);

        }
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Create an SQS queue.");
        System.out.println("Enter a name for your SQS queue.");
        sqsQueueName = in.nextLine();
        if (selectFIFO) {
            sqsQueueName = sqsQueueName + ".fifo";
        }
        sqsQueueUrl = createQueue(sqsClient, sqsQueueName, selectFIFO);
        System.out.println("The queue URL is " + sqsQueueUrl);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the SQS queue ARN attribute.");
        sqsQueueArn = getSQSQueueAttrs(sqsClient, sqsQueueUrl);
        System.out.println("The ARN of the new queue is " + sqsQueueArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Attach an IAM policy to the queue.");

        // Define the policy to use. Make sure that you change the REGION if you
are
        // running this code
        // in a different region.
        String policy = """
        {
             "Statement": [
             {
                 "Effect": "Allow",
                         "Principal": {
                     "Service": "sns.amazonaws.com"
                 },
                 "Action": "sqs:SendMessage",
                         "Resource": "arn:aws:sqs:us-east-1:%s:%s",
                         "Condition": {
                     "ArnEquals": {
                         "aws:SourceArn": "arn:aws:sns:us-east-1:%s:%s"
                     }
                 }
             }
             ]
         }
```

```
        """.formatted(accountId, sqsQueueName, accountId, topicName);

        setQueueAttr(sqsClient, sqsQueueUrl, policy);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Subscribe to the SQS queue.");
        if (selectFIFO) {
            System.out.println(
                "If you add a filter to this subscription, then only the filtered
 messages will be received in the queue.\n"
                    +
                    "For information about message filtering, see https://
docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n"
                    +
                    "For this example, you can filter messages by a \"tone\"
 attribute.");
            System.out.println("Would you like to filter messages for " +
 sqsQueueName + "'s subscription to the topic "
                + topicName + "?  (y/n)");
            String filterAns = in.nextLine();
            if (filterAns.compareTo("y") == 0) {
                boolean moreAns = false;
                System.out.println("You can filter messages by one or more of the
 following \"tone\" attributes.");
                System.out.println("1. cheerful");
                System.out.println("2. funny");
                System.out.println("3. serious");
                System.out.println("4. sincere");
                while (!moreAns) {
                    System.out.println("Select a number or choose 0 to end.");
                    String ans = in.nextLine();
                    switch (ans) {
                        case "1":
                            filterList.add("cheerful");
                            break;
                        case "2":
                            filterList.add("funny");
                            break;
                        case "3":
                            filterList.add("serious");
                            break;
                        case "4":
                            filterList.add("sincere");
```

```
                            break;
                        default:
                            moreAns = true;
                            break;
                    }
                }
            }
        }
        subscriptionArn = subQueue(snsClient, topicArn, sqsQueueArn, filterList);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Publish a message to the topic.");
        if (selectFIFO) {
            System.out.println("Would you like to add an attribute to this
message?  (y/n)");
            String msgAns = in.nextLine();
            if (msgAns.compareTo("y") == 0) {
                System.out.println("You can filter messages by one or more of the
following \"tone\" attributes.");
                System.out.println("1. cheerful");
                System.out.println("2. funny");
                System.out.println("3. serious");
                System.out.println("4. sincere");
                System.out.println("Select a number or choose 0 to end.");
                String ans = in.nextLine();
                switch (ans) {
                    case "1":
                        msgAttValue = "cheerful";
                        break;
                    case "2":
                        msgAttValue = "funny";
                        break;
                    case "3":
                        msgAttValue = "serious";
                        break;
                    default:
                        msgAttValue = "sincere";
                        break;
                }

                System.out.println("Selected value is " + msgAttValue);
            }
            System.out.println("Enter a message.");
```

```
            message = in.nextLine();
            pubMessageFIFO(snsClient, message, topicArn, msgAttValue,
 duplication, groupId, deduplicationID);

        } else {
            System.out.println("Enter a message.");
            message = in.nextLine();
            pubMessage(snsClient, message, topicArn);
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Display the message. Press any key to continue.");
        in.nextLine();
        messageList = receiveMessages(sqsClient, sqsQueueUrl, msgAttValue);
        for (Message mes : messageList) {
            System.out.println("Message Id: " + mes.messageId());
            System.out.println("Full Message: " + mes.body());
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Delete the received message. Press any key to
 continue.");
        in.nextLine();
        deleteMessages(sqsClient, sqsQueueUrl, messageList);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Unsubscribe from the topic and delete the queue.
 Press any key to continue.");
        in.nextLine();
        unSub(snsClient, subscriptionArn);
        deleteSQSQueue(sqsClient, sqsQueueName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Delete the topic. Press any key to continue.");
        in.nextLine();
        deleteSNSTopic(snsClient, topicArn);

        System.out.println(DASHES);
        System.out.println("The SNS/SQS workflow has completed successfully.");
        System.out.println(DASHES);
```

```
    }

    public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
        try {
            DeleteTopicRequest request = DeleteTopicRequest.builder()
                .topicArn(topicArn)
                .build();

            DeleteTopicResponse result = snsClient.deleteTopic(request);
            System.out.println("Status was " +
result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteSQSQueue(SqsClient sqsClient, String queueName) {
        try {
            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build();

            String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            DeleteQueueRequest deleteQueueRequest = DeleteQueueRequest.builder()
                .queueUrl(queueUrl)
                .build();

            sqsClient.deleteQueue(deleteQueueRequest);
            System.out.println(queueName + " was successfully deleted.");

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void unSub(SnsClient snsClient, String subscriptionArn) {
        try {
            UnsubscribeRequest request = UnsubscribeRequest.builder()
                .subscriptionArn(subscriptionArn)
                .build();
```

```
            UnsubscribeResponse result = snsClient.unsubscribe(request);
            System.out.println("Status was " +
result.sdkHttpResponse().statusCode()
                + "\nSubscription was removed for " + request.subscriptionArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {
        try {
            List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();
            for (Message msg : messages) {
                DeleteMessageBatchRequestEntry entry =
DeleteMessageBatchRequestEntry.builder()
                    .id(msg.messageId())
                    .build();

                entries.add(entry);
            }

            DeleteMessageBatchRequest deleteMessageBatchRequest =
DeleteMessageBatchRequest.builder()
                .queueUrl(queueUrl)
                .entries(entries)
                .build();

            sqsClient.deleteMessageBatch(deleteMessageBatchRequest);
            System.out.println("The batch delete of messages was successful");

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static List<Message> receiveMessages(SqsClient sqsClient, String
queueUrl, String msgAttValue) {
        try {
            if (msgAttValue.isEmpty()) {
```

```
                ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(5)
                    .build();
                return
sqsClient.receiveMessage(receiveMessageRequest).messages();
            } else {
                // We know there are filters on the message.
                ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageAttributeNames(msgAttValue) // Include other message
attributes if needed.
                    .maxNumberOfMessages(5)
                    .build();

                return sqsClient.receiveMessage(receiveRequest).messages();
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static void pubMessage(SnsClient snsClient, String message, String
topicArn) {
        try {
            PublishRequest request = PublishRequest.builder()
                .message(message)
                .topicArn(topicArn)
                .build();

            PublishResponse result = snsClient.publish(request);
            System.out
                .println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

```
    }

    public static void pubMessageFIFO(SnsClient snsClient,
                                      String message,
                                      String topicArn,
                                      String msgAttValue,
                                      String duplication,
                                      String groupId,
                                      String deduplicationID) {

        try {
            PublishRequest request;
            // Means the user did not choose to use a message attribute.
            if (msgAttValue.isEmpty()) {
                if (duplication.compareTo("y") == 0) {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
                } else {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageDeduplicationId(deduplicationID)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
                }

            } else {
                Map<String, MessageAttributeValue> messageAttributes = new
HashMap<>();
                messageAttributes.put(msgAttValue,
MessageAttributeValue.builder()
                    .dataType("String")
                    .stringValue("true")
                    .build());

                if (duplication.compareTo("y") == 0) {
                    request = PublishRequest.builder()
                        .message(message)
                        .messageGroupId(groupId)
                        .topicArn(topicArn)
                        .build();
```

```
                } else {
                    // Create a publish request with the message and attributes.
                    request = PublishRequest.builder()
                        .topicArn(topicArn)
                        .message(message)
                        .messageDeduplicationId(deduplicationID)
                        .messageGroupId(groupId)
                        .messageAttributes(messageAttributes)
                        .build();
                }
            }

            // Publish the message to the topic.
            PublishResponse result = snsClient.publish(request);
            System.out
                .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    // Subscribe to the SQS queue.
    public static String subQueue(SnsClient snsClient, String topicArn, String
queueArn, List<String> filterList) {
        try {
            SubscribeRequest request;
            if (filterList.isEmpty()) {
                // No filter subscription is added.
                request = SubscribeRequest.builder()
                    .protocol("sqs")
                    .endpoint(queueArn)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

                SubscribeResponse result = snsClient.subscribe(request);
                System.out.println("The queue " + queueArn + " has been
subscribed to the topic " + topicArn + "\n" +
                    "with the subscription ARN " + result.subscriptionArn());
                return result.subscriptionArn();
            } else {
```

```
                request = SubscribeRequest.builder()
                    .protocol("sqs")
                    .endpoint(queueArn)
                    .returnSubscriptionArn(true)
                    .topicArn(topicArn)
                    .build();

                SubscribeResponse result = snsClient.subscribe(request);
                System.out.println("The queue " + queueArn + " has been
subscribed to the topic " + topicArn + "\n" +
                    "with the subscription ARN " + result.subscriptionArn());

                String attributeName = "FilterPolicy";
                Gson gson = new Gson();
                String jsonString = "{\"tone\": []}";
                JsonObject jsonObject = gson.fromJson(jsonString,
JsonObject.class);
                JsonArray toneArray = jsonObject.getAsJsonArray("tone");
                for (String value : filterList) {
                    toneArray.add(new JsonPrimitive(value));
                }

                String updatedJsonString = gson.toJson(jsonObject);
                System.out.println(updatedJsonString);
                SetSubscriptionAttributesRequest attRequest =
SetSubscriptionAttributesRequest.builder()
                    .subscriptionArn(result.subscriptionArn())
                    .attributeName(attributeName)
                    .attributeValue(updatedJsonString)
                    .build();

                snsClient.setSubscriptionAttributes(attRequest);
                return result.subscriptionArn();
            }

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    // Attach a policy to the queue.
```

```
   public static void setQueueAttr(SqsClient sqsClient, String queueUrl, String
policy) {
       try {
           Map<software.amazon.awssdk.services.sqs.model.QueueAttributeName,
String> attrMap = new HashMap<>();
           attrMap.put(QueueAttributeName.POLICY, policy);

           SetQueueAttributesRequest attributesRequest =
SetQueueAttributesRequest.builder()
               .queueUrl(queueUrl)
               .attributes(attrMap)
               .build();

           sqsClient.setQueueAttributes(attributesRequest);
           System.out.println("The policy has been successfully attached.");

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static String getSQSQueueAttrs(SqsClient sqsClient, String queueUrl) {
       // Specify the attributes to retrieve.
       List<QueueAttributeName> atts = new ArrayList<>();
       atts.add(QueueAttributeName.QUEUE_ARN);

       GetQueueAttributesRequest attributesRequest =
GetQueueAttributesRequest.builder()
           .queueUrl(queueUrl)
           .attributeNames(atts)
           .build();

       GetQueueAttributesResponse response =
sqsClient.getQueueAttributes(attributesRequest);
       Map<String, String> queueAtts = response.attributesAsStrings();
       for (Map.Entry<String, String> queueAtt : queueAtts.entrySet())
           return queueAtt.getValue();

       return "";
   }

   public static String createQueue(SqsClient sqsClient, String queueName,
Boolean selectFIFO) {
```

```
        try {
            System.out.println("\nCreate Queue");
            if (selectFIFO) {
                Map<QueueAttributeName, String> attrs = new HashMap<>();
                attrs.put(QueueAttributeName.FIFO_QUEUE, "true");
                CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .attributes(attrs)
                    .build();

                sqsClient.createQueue(createQueueRequest);
                System.out.println("\nGet queue url");
                GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
                return getQueueUrlResponse.queueUrl();
            } else {
                CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();

                sqsClient.createQueue(createQueueRequest);
                System.out.println("\nGet queue url");
                GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
                return getQueueUrlResponse.queueUrl();
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                .name(topicName)
                .build();
```

```
            result = snsClient.createTopic(request);
            return result.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createFIFO(SnsClient snsClient, String topicName, String
 duplication) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = new HashMap<>();
            if (duplication.compareTo("n") == 0) {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "false");
            } else {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "true");
            }

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            return response.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 항목을 참조하세요.
  - CreateQueue

- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이 시나리오의 진입점입니다.

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
```

```
};
```

위의 코드는 필요한 종속성을 제공하고 시나리오를 시작합니다. 다음 섹션에는 대부분의 예제가 포함되어 있습니다.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
 string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../../libs/prompter.js').Prompter} prompter
   * @param {import('../../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
```

```
    this.prompter = prompter;
    this.logger = logger;
  }

  async welcome() {
    await this.logger.log(MESSAGES.description);
  }

  async confirmFifo() {
    await this.logger.log(MESSAGES.snsFifoDescription);
    this.isFifo = await this.prompter.confirm({
      message: MESSAGES.snsFifoPrompt,
    });

    if (this.isFifo) {
      this.logger.logSeparator(MESSAGES.headerDedup);
      await this.logger.log(MESSAGES.deduplicationNotice);
      await this.logger.log(MESSAGES.deduplicationDescription);
      this.autoDedup = await this.prompter.confirm({
        message: MESSAGES.deduplicationPrompt,
      });
    }
  }

  async createTopic() {
    await this.logger.log(MESSAGES.creatingTopics);
    this.topicName = await this.prompter.input({
      message: MESSAGES.topicNamePrompt,
    });
    if (this.isFifo) {
      this.topicName += ".fifo";
      this.logger.logSeparator(MESSAGES.headerFifoNaming);
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.snsClient.send(
      new CreateTopicCommand({
        Name: this.topicName,
        Attributes: {
          FifoTopic: this.isFifo ? "true" : "false",
          ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
        },
      }),
    );
```

```
    this.topicArn = response.TopicArn;

    await this.logger.log(
      MESSAGES.topicCreatedNotice
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    const maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
```

```
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });

      await this.logger.log(
        MESSAGES.queueCreatedNotice
          .replace("${QUEUE_NAME}", queueName)
          .replace("${QUEUE_URL}", response.QueueUrl)
          .replace("${QUEUE_ARN}", Attributes.QueueArn),
      );
    }
  }

  async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
      const policy = JSON.stringify(
        {
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                Service: "sns.amazonaws.com",
              },
              Action: "sqs:SendMessage",
              Resource: queue.queueArn,
              Condition: {
                ArnEquals: {
                  "aws:SourceArn": this.topicArn,
                },
              },
            },
          ],
        },
        null,
        2,
      );

      if (index !== 0) {
        this.logger.logSeparator();
      }

      await this.logger.log(MESSAGES.attachPolicyNotice);
      console.log(policy);
```

```
      const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      });

      if (addPolicy) {
        await this.sqsClient.send(
          new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
              Policy: policy,
            },
          }),
        );
        queue.policy = policy;
      } else {
        await this.logger.log(
          MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
      /**
       * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
       */
      const subscribeParams = {
        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
      };
      let tones = [];

      if (this.isFifo) {
        if (index === 0) {
          await this.logger.log(MESSAGES.fifoFilterNotice);
        }
```

```
        tones = await this.prompter.checkbox({
          message: MESSAGES.fifoFilterSelect.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
          choices: toneChoices,
        });

        if (tones.length) {
          subscribeParams.Attributes = {
            FilterPolicyScope: "MessageAttributes",
            FilterPolicy: JSON.stringify({
              tone: tones,
            }),
          };
        }
      }

      const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
      );

      this.subscriptionArns.push(SubscriptionArn);

      await this.logger.log(
        MESSAGES.queueSubscribedNotice
          .replace("${QUEUE_NAME}", queue.queueName)
          .replace("${TOPIC_NAME}", this.topicName)
          .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
      );
    }
  }

  async publishMessages() {
    const message = await this.prompter.input({
      message: MESSAGES.publishMessagePrompt,
    });

    let groupId;
    let deduplicationId;
    let choices;

    if (this.isFifo) {
      await this.logger.log(MESSAGES.groupIdNotice);
```

```
      groupId = await this.prompter.input({
        message: MESSAGES.groupIdPrompt,
      });

      if (this.autoDedup === false) {
        await this.logger.log(MESSAGES.deduplicationIdNotice);
        deduplicationId = await this.prompter.input({
          message: MESSAGES.deduplicationIdPrompt,
        });
      }

      choices = await this.prompter.checkbox({
        message: MESSAGES.messageAttributesPrompt,
        choices: toneChoices,
      });
    }

    await this.snsClient.send(
      new PublishCommand({
        TopicArn: this.topicArn,
        Message: message,
        ...(groupId
          ? {
              MessageGroupId: groupId,
            }
          : {}),
        ...(deduplicationId
          ? {
              MessageDeduplicationId: deduplicationId,
            }
          : {}),
        ...(choices
          ? {
              MessageAttributes: {
                tone: {
                  DataType: "String.Array",
                  StringValue: JSON.stringify(choices),
                },
              },
            }
          : {}),
      }),
    );
```

```
    const publishAnother = await this.prompter.confirm({
      message: MESSAGES.publishAnother,
    });

    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
```

```
    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });

    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
```

```
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
      this.logger.logSeparator(MESSAGES.headerReceiveMessages);
      await this.receiveAndDeleteMessages();
    } catch (err) {
      console.error(err);
    } finally {
      await this.destroyResources();
    }
  }
}
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하세요.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
package com.example.sns

import aws.sdk.kotlin.services.sns.SnsClient
import aws.sdk.kotlin.services.sns.model.CreateTopicRequest
import aws.sdk.kotlin.services.sns.model.DeleteTopicRequest
import aws.sdk.kotlin.services.sns.model.PublishRequest
import aws.sdk.kotlin.services.sns.model.SetSubscriptionAttributesRequest
import aws.sdk.kotlin.services.sns.model.SubscribeRequest
import aws.sdk.kotlin.services.sns.model.UnsubscribeRequest
import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.model.CreateQueueRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequestEntry
import aws.sdk.kotlin.services.sqs.model.DeleteQueueRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueAttributesRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueUrlRequest
import aws.sdk.kotlin.services.sqs.model.Message
import aws.sdk.kotlin.services.sqs.model.QueueAttributeName
import aws.sdk.kotlin.services.sqs.model.ReceiveMessageRequest
import aws.sdk.kotlin.services.sqs.model.SetQueueAttributesRequest
import com.google.gson.Gson
import com.google.gson.JsonObject
import com.google.gson.JsonPrimitive
import java.util.Scanner

/**
Before running this Kotlin code example, set up your development environment,
including your AWS credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This Kotlin example performs the following tasks:

 1. Gives the user three options to choose from.
 2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
 3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
 4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
 5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
 6. Subscribes to the SQS queue.
 7. Publishes a message to the topic.
 8. Displays the messages.
 9. Deletes the received message.
```

```
 10. Unsubscribes from the topic.
 11. Deletes the SNS topic.
 */

val DASHES: String = String(CharArray(80)).replace("\u0000", "-")
suspend fun main() {
    val input = Scanner(System.`in`)
    val useFIFO: String
    var duplication = "n"
    var topicName: String
    var deduplicationID: String? = null
    var groupId: String? = null
    val topicArn: String?
    var sqsQueueName: String
    val sqsQueueUrl: String?
    val sqsQueueArn: String
    val subscriptionArn: String?
    var selectFIFO = false
    val message: String
    val messageList: List<Message?>?
    val filterList = ArrayList<String>()
    var msgAttValue = ""

    println(DASHES)
    println("Welcome to the AWS SDK for Kotlin messaging with topics and
 queues.")
    println(
        """
                In this scenario, you will create an SNS topic and subscribe an
 SQS queue to the topic.
                You can select from several options for configuring the topic and
 the subscriptions for the queue.
                You can then post to the topic and see the results in the queue.
        """.trimIndent(),
    )
    println(DASHES)

    println(DASHES)
    println(
        """
                SNS topics can be configured as FIFO (First-In-First-Out).
                FIFO topics deliver messages in order and support deduplication
 and message filtering.
                Would you like to work with FIFO topics? (y/n)
```

```
        """.trimIndent(),
    )
    useFIFO = input.nextLine()
    if (useFIFO.compareTo("y") == 0) {
        selectFIFO = true
        println("You have selected FIFO")
        println(
            """ Because you have chosen a FIFO topic, deduplication is supported.
        Deduplication IDs are either set in the message or automatically
 generated from content using a hash function.
        If a message is successfully published to an SNS FIFO topic, any message
 published and determined to have the same deduplication ID,
        within the five-minute deduplication interval, is accepted but not
 delivered.
        For more information about deduplication, see https://
docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.""",
        )

        println("Would you like to use content-based deduplication instead of
 entering a deduplication ID? (y/n)")
        duplication = input.nextLine()
        if (duplication.compareTo("y") == 0) {
            println("Enter a group id value")
            groupId = input.nextLine()
        } else {
            println("Enter deduplication Id value")
            deduplicationID = input.nextLine()
            println("Enter a group id value")
            groupId = input.nextLine()
        }
    }
    println(DASHES)

    println(DASHES)
    println("2. Create a topic.")
    println("Enter a name for your SNS topic.")
    topicName = input.nextLine()
    if (selectFIFO) {
        println("Because you have selected a FIFO topic, '.fifo' must be appended
 to the topic name.")
        topicName = "$topicName.fifo"
        println("The name of the topic is $topicName")
        topicArn = createFIFO(topicName, duplication)
        println("The ARN of the FIFO topic is $topicArn")
```

```
    } else {
        println("The name of the topic is $topicName")
        topicArn = createSNSTopic(topicName)
        println("The ARN of the non-FIFO topic is $topicArn")
    }
    println(DASHES)

    println(DASHES)
    println("3. Create an SQS queue.")
    println("Enter a name for your SQS queue.")
    sqsQueueName = input.nextLine()
    if (selectFIFO) {
        sqsQueueName = "$sqsQueueName.fifo"
    }
    sqsQueueUrl = createQueue(sqsQueueName, selectFIFO)
    println("The queue URL is $sqsQueueUrl")
    println(DASHES)

    println(DASHES)
    println("4. Get the SQS queue ARN attribute.")
    sqsQueueArn = getSQSQueueAttrs(sqsQueueUrl)
    println("The ARN of the new queue is $sqsQueueArn")
    println(DASHES)

    println(DASHES)
    println("5. Attach an IAM policy to the queue.")
    // Define the policy to use.
    val policy = """{
     "Statement": [
     {
         "Effect": "Allow",
                 "Principal": {
             "Service": "sns.amazonaws.com"
         },
         "Action": "sqs:SendMessage",
                 "Resource": "$sqsQueueArn",
                 "Condition": {
             "ArnEquals": {
                 "aws:SourceArn": "$topicArn"
             }
         }
     }
     ]
     }"""
```

```
    setQueueAttr(sqsQueueUrl, policy)
    println(DASHES)

    println(DASHES)
    println("6. Subscribe to the SQS queue.")
    if (selectFIFO) {
        println(
            """If you add a filter to this subscription, then only the filtered
 messages will be received in the queue.
For information about message filtering, see https://docs.aws.amazon.com/sns/
latest/dg/sns-message-filtering.html
For this example, you can filter messages by a "tone" attribute.""",
        )
        println("Would you like to filter messages for $sqsQueueName's
 subscription to the topic $topicName?  (y/n)")
        val filterAns: String = input.nextLine()
        if (filterAns.compareTo("y") == 0) {
            var moreAns = false
            println("You can filter messages by using one or more of the
 following \"tone\" attributes.")
            println("1. cheerful")
            println("2. funny")
            println("3. serious")
            println("4. sincere")
            while (!moreAns) {
                println("Select a number or choose 0 to end.")
                val ans: String = input.nextLine()
                when (ans) {
                    "1" -> filterList.add("cheerful")
                    "2" -> filterList.add("funny")
                    "3" -> filterList.add("serious")
                    "4" -> filterList.add("sincere")
                    else -> moreAns = true
                }
            }
        }
    }
    subscriptionArn = subQueue(topicArn, sqsQueueArn, filterList)
    println(DASHES)

    println(DASHES)
    println("7. Publish a message to the topic.")
    if (selectFIFO) {
        println("Would you like to add an attribute to this message?  (y/n)")
```

```
        val msgAns: String = input.nextLine()
        if (msgAns.compareTo("y") == 0) {
            println("You can filter messages by one or more of the following
\"tone\" attributes.")
            println("1. cheerful")
            println("2. funny")
            println("3. serious")
            println("4. sincere")
            println("Select a number or choose 0 to end.")
            val ans: String = input.nextLine()
            msgAttValue = when (ans) {
                "1" -> "cheerful"
                "2" -> "funny"
                "3" -> "serious"
                else -> "sincere"
            }
            println("Selected value is $msgAttValue")
        }
        println("Enter a message.")
        message = input.nextLine()
        pubMessageFIFO(message, topicArn, msgAttValue, duplication, groupId,
deduplicationID)
    } else {
        println("Enter a message.")
        message = input.nextLine()
        pubMessage(message, topicArn)
    }
    println(DASHES)

    println(DASHES)
    println("8. Display the message. Press any key to continue.")
    input.nextLine()
    messageList = receiveMessages(sqsQueueUrl, msgAttValue)
    if (messageList != null) {
        for (mes in messageList) {
            println("Message Id: ${mes.messageId}")
            println("Full Message: ${mes.body}")
        }
    }
    println(DASHES)

    println(DASHES)
    println("9. Delete the received message. Press any key to continue.")
    input.nextLine()
```

```
    if (messageList != null) {
        deleteMessages(sqsQueueUrl, messageList)
    }
    println(DASHES)

    println(DASHES)
    println("10. Unsubscribe from the topic and delete the queue. Press any key
 to continue.")
    input.nextLine()
    unSub(subscriptionArn)
    deleteSQSQueue(sqsQueueName)
    println(DASHES)

    println(DASHES)
    println("11. Delete the topic. Press any key to continue.")
    input.nextLine()
    deleteSNSTopic(topicArn)
    println(DASHES)

    println(DASHES)
    println("The SNS/SQS workflow has completed successfully.")
    println(DASHES)
}

suspend fun deleteSNSTopic(topicArnVal: String?) {
    val request = DeleteTopicRequest {
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was deleted")
    }
}

suspend fun deleteSQSQueue(queueNameVal: String) {
    val getQueueRequest = GetQueueUrlRequest {
        queueName = queueNameVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val queueUrlVal = sqsClient.getQueueUrl(getQueueRequest).queueUrl
        val deleteQueueRequest = DeleteQueueRequest {
            queueUrl = queueUrlVal
```

```
        }

        sqsClient.deleteQueue(deleteQueueRequest)
        println("$queueNameVal was successfully deleted.")
    }
}

suspend fun unSub(subscripArn: String?) {
    val request = UnsubscribeRequest {
        subscriptionArn = subscripArn
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for $subscripArn")
    }
}

suspend fun deleteMessages(queueUrlVal: String?, messages: List<Message>) {
    val entriesVal: MutableList<DeleteMessageBatchRequestEntry> = mutableListOf()
    for (msg in messages) {
        val entry = DeleteMessageBatchRequestEntry {
            id = msg.messageId
        }
        entriesVal.add(entry)
    }

    val deleteMessageBatchRequest = DeleteMessageBatchRequest {
        queueUrl = queueUrlVal
        entries = entriesVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteMessageBatch(deleteMessageBatchRequest)
        println("The batch delete of messages was successful")
    }
}

suspend fun receiveMessages(queueUrlVal: String?, msgAttValue: String):
 List<Message>? {
    if (msgAttValue.isEmpty()) {
        val request = ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }
```

```
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            return sqsClient.receiveMessage(request).messages
        }
    } else {
        val receiveRequest = ReceiveMessageRequest {
            queueUrl = queueUrlVal
            waitTimeSeconds = 1
            maxNumberOfMessages = 5
        }
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            return sqsClient.receiveMessage(receiveRequest).messages
        }
    }
}

suspend fun pubMessage(messageVal: String?, topicArnVal: String?) {
    val request = PublishRequest {
        message = messageVal
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}

suspend fun pubMessageFIFO(
    messageVal: String?,
    topicArnVal: String?,
    msgAttValue: String,
    duplication: String,
    groupIdVal: String?,
    deduplicationID: String?,
) {
    // Means the user did not choose to use a message attribute.
    if (msgAttValue.isEmpty()) {
        if (duplication.compareTo("y") == 0) {
            val request = PublishRequest {
                message = messageVal
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }
```

```
            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        } else {
            val request = PublishRequest {
                message = messageVal
                messageDeduplicationId = deduplicationID
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        }
    } else {
        val messAttr = aws.sdk.kotlin.services.sns.model.MessageAttributeValue {
            dataType = "String"
            stringValue = "true"
        }

        val mapAtt: Map<String,
aws.sdk.kotlin.services.sns.model.MessageAttributeValue> =
            mapOf(msgAttValue to messAttr)
        if (duplication.compareTo("y") == 0) {
            val request = PublishRequest {
                message = messageVal
                messageGroupId = groupIdVal
                topicArn = topicArnVal
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        } else {
            // Create a publish request with the message and attributes.
            val request = PublishRequest {
                topicArn = topicArnVal
                message = messageVal
                messageDeduplicationId = deduplicationID
                messageGroupId = groupIdVal
```

```
                messageAttributes = mapAtt
            }

            SnsClient { region = "us-east-1" }.use { snsClient ->
                val result = snsClient.publish(request)
                println(result.messageId.toString() + " Message sent.")
            }
        }
    }
}

// Subscribe to the SQS queue.
suspend fun subQueue(topicArnVal: String?, queueArnVal: String, filterList:
 List<String?>): String? {
    val request: SubscribeRequest
    if (filterList.isEmpty()) {
        // No filter subscription is added.
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
            println(
                "The queue " + queueArnVal + " has been subscribed to the topic "
 + topicArnVal + "\n" +
                    "with the subscription ARN " + result.subscriptionArn,
            )
            return result.subscriptionArn
        }
    } else {
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
```

```
            println("The queue $queueArnVal has been subscribed to the topic
 $topicArnVal with the subscription ARN ${result.subscriptionArn}")

            val attributeNameVal = "FilterPolicy"
            val gson = Gson()
            val jsonString = "{\"tone\": []}"
            val jsonObject = gson.fromJson(jsonString, JsonObject::class.java)
            val toneArray = jsonObject.getAsJsonArray("tone")
            for (value: String? in filterList) {
                toneArray.add(JsonPrimitive(value))
            }

            val updatedJsonString: String = gson.toJson(jsonObject)
            println(updatedJsonString)
            val attRequest = SetSubscriptionAttributesRequest {
                subscriptionArn = result.subscriptionArn
                attributeName = attributeNameVal
                attributeValue = updatedJsonString
            }

            snsClient.setSubscriptionAttributes(attRequest)
            return result.subscriptionArn
        }
    }
}

suspend fun setQueueAttr(queueUrlVal: String?, policy: String) {
    val attrMap: MutableMap<String, String> = HashMap()
    attrMap[QueueAttributeName.Policy.toString()] = policy

    val attributesRequest = SetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributes = attrMap
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.setQueueAttributes(attributesRequest)
        println("The policy has been successfully attached.")
    }
}

suspend fun getSQSQueueAttrs(queueUrlVal: String?): String {
    val atts: MutableList<QueueAttributeName> = ArrayList()
    atts.add(QueueAttributeName.QueueArn)
```

```
    val attributesRequest = GetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributeNames = atts
    }
    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.getQueueAttributes(attributesRequest)
        val mapAtts = response.attributes
        if (mapAtts != null) {
            mapAtts.forEach { entry ->
                println("${entry.key} : ${entry.value}")
                return entry.value
            }
        }
    }
    return ""
}

suspend fun createQueue(queueNameVal: String?, selectFIFO: Boolean): String? {
    println("\nCreate Queue")
    if (selectFIFO) {
        val attrs = mutableMapOf<String, String>()
        attrs[QueueAttributeName.FifoQueue.toString()] = "true"

        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
            attributes = attrs
        }

        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("\nGet queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    } else {
        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
        }
```

```
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("Get queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    }
}

suspend fun createSNSTopic(topicName: String?): String? {
    val request = CreateTopicRequest {
        name = topicName
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn
    }
}

suspend fun createFIFO(topicName: String?, duplication: String): String? {
    val topicAttributes: MutableMap<String, String> = HashMap()
    if (duplication.compareTo("n") == 0) {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "false"
    } else {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "true"
    }

    val topicRequest = CreateTopicRequest {
        name = topicName
        attributes = topicAttributes
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.createTopic(topicRequest)
        return response.topicArn
    }
```

```
}
```

- API 세부 정보는 AWS SDK for Kotlin API reference의 다음 주제를 참조하세요.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Swift

SDK for Swift

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import ArgumentParser
import AWSClientRuntime
import AWSSNS
import AWSSQS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
```

```
        commandName: "queue-scenario",
        abstract: """
        This example interactively demonstrates how to use Amazon Simple
        Notification Service (Amazon SNS) and Amazon Simple Queue Service
        (Amazon SQS) together to publish and receive messages using queues.
        """,
        discussion: """
        Supports filtering using a "tone" attribute.
        """
    )

    /// Prompt for an input string. Only non-empty strings are allowed.
    ///
    /// - Parameter prompt: The prompt to display.
    ///
    /// - Returns: The string input by the user.
    func stringRequest(prompt: String) -> String {
        var str: String?

        while str == nil {
            print(prompt, terminator: "")
            str = readLine()

            if str != nil && str?.count == 0 {
                str = nil
            }
        }

        return str!
    }

    /// Ask a yes/no question.
    ///
    /// - Parameter prompt: A prompt string to print.
    ///
    /// - Returns: `true` if the user answered "Y", otherwise `false`.
    func yesNoRequest(prompt: String) -> Bool {
        while true {
            let answer = stringRequest(prompt: prompt).lowercased()
            if answer == "y" || answer == "n" {
                return answer == "y"
            }
        }
    }
```

```
    /// Display a menu of options then request a selection.
    ///
    /// - Parameters:
    ///   - prompt: A prompt string to display before the menu.
    ///   - options: An array of strings giving the menu options.
    ///
    /// - Returns: The index number of the selected option or 0 if no item was
    ///   selected.
    func menuRequest(prompt: String, options: [String]) -> Int {
        let numOptions = options.count

        if numOptions == 0 {
            return 0
        }

        print(prompt)

        for (index, value) in options.enumerated() {
            print("(\(index)) \(value)")
        }

        repeat {
            print("Enter your selection (0 - \(numOptions-1)): ", terminator: "")
            if let answer = readLine() {
                guard let answer = Int(answer) else {
                    print("Please enter the number matching your selection.")
                    continue
                }

                if answer >= 0 && answer < numOptions {
                    return answer
                } else {
                    print("Please enter the number matching your selection.")
                }
            }
        } while true
    }

    /// Ask the user too press RETURN. Accepts any input but ignores it.
    ///
    /// - Parameter prompt: The text prompt to display.
    func returnRequest(prompt: String) {
        print(prompt, terminator: "")
```

```
        _ = readLine()
    }

    var attrValues = [
        "<none>",
        "cheerful",
        "funny",
        "serious",
        "sincere"
    ]

    /// Ask the user to choose one of the attribute values to use as a filter.
    ///
    /// - Parameters:
    ///   - message: A message to display before the menu of values.
    ///   - attrValues: An array of strings giving the values to choose from.
    ///
    /// - Returns: The string corresponding to the selected option.
    func askForFilter(message: String, attrValues: [String]) -> String? {
        print(message)
        for (index, value) in attrValues.enumerated() {
            print("  [\(index)] \(value)")
        }

        var answer: Int?
        repeat {
            answer = Int(stringRequest(prompt: "Select an value for the 'tone'
 attribute or 0 to end: "))
        } while answer == nil || answer! < 0 || answer! > attrValues.count + 1

        if answer == 0 {
            return nil
        }
        return attrValues[answer!]
    }

    /// Prompts the user for filter terms and constructs the attribute
    /// record that specifies them.
    ///
    /// - Returns: A mapping of "FilterPolicy" to a JSON string representing
    ///   the user-defined filter.
    func buildFilterAttributes() -> [String:String] {
        var attr: [String:String] = [:]
        var filterString = ""
```

```
        var first = true

        while let ans = askForFilter(message: "Choose a value to apply to the
'tone' attribute.",
                                     attrValues: attrValues) {
            if !first {
                filterString += ","
            }
            first = false

            filterString += "\"\(ans)\""
        }

        let filterJSON = "{ \"tone\": [\(filterString)]}"
        attr["FilterPolicy"] = filterJSON

        return attr
    }
    /// Create a queue, returning its URL string.
    ///
    /// - Parameters:
    ///   - prompt: A prompt to ask for the queue name.
    ///   - isFIFO: Whether or not to create a FIFO queue.
    ///
    /// - Returns: The URL of the queue.
    func createQueue(prompt: String, sqsClient: SQSClient, isFIFO: Bool) async
throws -> String? {
        repeat {
            var queueName = stringRequest(prompt: prompt)
            var attributes: [String: String] = [:]

            if isFIFO {
                queueName += ".fifo"
                attributes["FifoQueue"] = "true"
            }

            do {
                let output = try await sqsClient.createQueue(
                    input: CreateQueueInput(
                        attributes: attributes,
                        queueName: queueName
                    )
                )
```

```
                guard let url = output.queueUrl else {
                    return nil
                }

                return url
            } catch _ as QueueDeletedRecently {
                print("You need to use a different queue name. A queue by that
name was recently deleted.")
                continue
            }
        } while true
    }

    /// Return the ARN of a queue given its URL.
    ///
    /// - Parameter queueUrl: The URL of the queue for which to return the
    ///   ARN.
    ///
    /// - Returns: The ARN of the specified queue.
    func getQueueARN(sqsClient: SQSClient, queueUrl: String) async throws ->
String? {
        let output = try await sqsClient.getQueueAttributes(
            input: GetQueueAttributesInput(
                attributeNames: [.queuearn],
                queueUrl: queueUrl
            )
        )

        guard let attributes = output.attributes else {
            return nil
        }

        return attributes["QueueArn"]
    }

    /// Applies the needed policy to the specified queue.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
    ///   - queueUrl: The queue to apply the policy to.
    ///   - queueArn: The ARN of the queue to apply the policy to.
    ///   - topicArn: The topic that should have access via the policy.
    ///
    /// - Throws: Errors from the SQS `SetQueueAttributes` action.
```

```
func setQueuePolicy(sqsClient: SQSClient, queueUrl: String,
                    queueArn: String, topicArn: String) async throws {
    _ = try await sqsClient.setQueueAttributes(
        input: SetQueueAttributesInput(
            attributes: [
                "Policy":
                    """
                    {
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Principal": {
                                    "Service": "sns.amazonaws.com"
                                },
                                "Action": "sqs:SendMessage",
                                "Resource": "\(queueArn)",
                                "Condition": {
                                    "ArnEquals": {
                                        "aws:SourceArn": "\(topicArn)"
                                    }
                                }
                            }
                        ]
                    }
                    """

            ],
            queueUrl: queueUrl
        )
    )
}

/// Receive the available messages on a queue, outputting them to the
/// screen. Returns a dictionary you pass to DeleteMessageBatch to delete
/// all the received messages.
///
/// - Parameters:
///   - sqsClient: The Amazon SQS client to use.
///   - queueUrl: The SQS queue on which to receive messages.
///
/// - Throws: Errors from `SQSClient.receiveMessage()`
///
/// - Returns: An array of SQSClientTypes.DeleteMessageBatchRequestEntry
///   items, each describing one received message in the format needed to
```

```
    ///   delete it.
    func receiveAndListMessages(sqsClient: SQSClient, queueUrl: String) async
throws
                                ->
[SQSClientTypes.DeleteMessageBatchRequestEntry] {
        let output = try await sqsClient.receiveMessage(
            input: ReceiveMessageInput(
                maxNumberOfMessages: 10,
                queueUrl: queueUrl
            )
        )

        guard let messages = output.messages else {
            print("No messages received.")
            return []
        }

        var deleteList: [SQSClientTypes.DeleteMessageBatchRequestEntry] = []

        // Print out all the messages that were received, including their
        // attributes, if any.

        for message in messages {
            print("Message ID:     \(message.messageId ?? "<unknown>")")
            print("Receipt handle: \(message.receiptHandle ?? "<unknown>")")
            print("Message JSON:   \(message.body ?? "<body missing>")")

            if message.receiptHandle != nil {
                deleteList.append(
                    SQSClientTypes.DeleteMessageBatchRequestEntry(
                        id: message.messageId,
                        receiptHandle: message.receiptHandle
                    )
                )
            }
        }

        return deleteList
    }

    /// Delete all the messages in the specified list.
    ///
    /// - Parameters:
    ///   - sqsClient: The Amazon SQS client to use.
```

```
   ///   - queueUrl: The SQS queue to delete messages from.
   ///   - deleteList: A list of `DeleteMessageBatchRequestEntry` objects
   ///     describing the messages to delete.
   ///
   /// - Throws: Errors from `SQSClient.deleteMessageBatch()`.
   func deleteMessageList(sqsClient: SQSClient, queueUrl: String,
                          deleteList:
[SQSClientTypes.DeleteMessageBatchRequestEntry]) async throws {
       let output = try await sqsClient.deleteMessageBatch(
           input: DeleteMessageBatchInput(entries: deleteList, queueUrl:
queueUrl)
       )

       if let failed = output.failed {
           print("\(failed.count) errors occurred deleting messages from the
queue.")
           for message in failed {
               print("---> Failed to delete message \(message.id ?? "<unknown
ID>") with error: \(message.code ?? "<unknown>") (\(message.message ?? "..."))")
           }
       }
   }

   /// Called by ``main()`` to run the bulk of the example.
   func runAsync() async throws {
       let rowOfStars = String(repeating: "*", count: 75)

       print("""
             \(rowOfStars)
             Welcome to the cross-service messaging with topics and queues
example.
             In this workflow, you'll create an SNS topic, then create two SQS
             queues which will be subscribed to that topic.

             You can specify several options for configuring the topic, as well
as
             the queue subscriptions. You can then post messages to the topic
and
             receive the results on the queues.
             \(rowOfStars)\n
             """
       )

       // 0. Create SNS and SQS clients.
```

```
        let snsConfig = try await SNSClient.SNSClientConfiguration(region:
 region)
        let snsClient = SNSClient(config: snsConfig)

        let sqsConfig = try await SQSClient.SQSClientConfiguration(region:
 region)
        let sqsClient = SQSClient(config: sqsConfig)

        // 1. Ask the user whether to create a FIFO topic. If so, ask whether
        //    to use content-based deduplication instead of requiring a
        //    deduplication ID.

        let isFIFO = yesNoRequest(prompt: "Do you want to create a FIFO topic (Y/
N)? ")
        var isContentBasedDeduplication = false

        if isFIFO {
            print("""
                  \(rowOfStars)
                  Because you've chosen to create a FIFO topic, deduplication is
                  supported.

                  Deduplication IDs are either set in the message or are
 automatically
                  generated from the content using a hash function.

                  If a message is successfully published to an SNS FIFO topic,
 any
                  message published and found to have the same deduplication ID
                  (within a five-minute deduplication interval), is accepted but
                  not delivered.

                  For more information about deduplication, see:
                  https://docs.aws.amazon.com/sns/latest/dg/fifo-message-
dedup.html.
                  """
            )

            isContentBasedDeduplication = yesNoRequest(
                prompt: "Use content-based deduplication instead of entering a
 deduplication ID (Y/N)? ")
            print(rowOfStars)
        }
```

```
        var topicName = stringRequest(prompt: "Enter the name of the topic to
create: ")

        // 2. Create the topic. Append ".fifo" to the name if FIFO was
        //    requested, and set the "FifoTopic" attribute to "true" if so as
        //    well. Set the "ContentBasedDeduplication" attribute to "true" if
        //    content-based deduplication was requested.

        if isFIFO {
            topicName += ".fifo"
        }

        print("Topic name: \(topicName)")

        var attributes = [
            "FifoTopic": (isFIFO ? "true" : "false")
        ]

        // If it's a FIFO topic with content-based deduplication, set the
        // "ContentBasedDeduplication" attribute.

        if isContentBasedDeduplication {
            attributes["ContentBasedDeduplication"] = "true"
        }

        // Create the topic and retrieve the ARN.

        let output = try await snsClient.createTopic(
            input: CreateTopicInput(
                attributes: attributes,
                name: topicName
            )
        )

        guard let topicArn = output.topicArn else {
            print("No topic ARN returned!")
            return
        }

        print("""
              Topic '\(topicName) has been created with the
              topic ARN \(topicArn)."
              """
```

```
        )

        print(rowOfStars)

        // 3. Create an SQS queue. Append ".fifo" to the name if one of the
        //    FIFO topic configurations was chosen, and set "FifoQueue" to
        //    "true" if the topic is FIFO.

        print("""
              Next, you will create two SQS queues that will be subscribed
              to the topic you just created.\n
              """
        )

        let q1Url = try await createQueue(prompt: "Enter the name of the first
queue: ",
                                          sqsClient: sqsClient, isFIFO: isFIFO)
        guard let q1Url else {
            print("Unable to create queue 1!")
            return
        }

        // 4. Get the SQS queue's ARN attribute using `GetQueueAttributes`.

        let q1Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q1Url)

        guard let q1Arn else {
            print("Unable to get ARN of queue 1!")
            return
        }
        print("Got queue 1 ARN: \(q1Arn)")

        // 5. Attach an AWS IAM policy to the queue using
        //    `SetQueueAttributes`.

        try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q1Url,
                                 queueArn: q1Arn, topicArn: topicArn)

        // 6. Subscribe the SQS queue to the SNS topic. Set the topic ARN in
        //    the request. Set the protocol to "sqs". Set the queue ARN to the
        //    ARN just received in step 5. For FIFO topics, give the option to
        //    apply a filter. A filter allows only matching messages to enter
        //    the queue.
```

```
        var q1Attributes: [String:String]? = nil

        if isFIFO {
            print(
                """

                If you add a filter to this subscription, then only the filtered
 messages will
                be received in the queue. For information about message
 filtering, see
                https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html
                For this example, you can filter messages by a 'tone' attribute.

                """
            )

            let subPrompt = """
                Would you like to filter messages for the first queue's
 subscription to the
                topic \(topicName) (Y/N)?
                """
            if (yesNoRequest(prompt: subPrompt)) {
                q1Attributes = buildFilterAttributes()
            }
        }

        let sub1Output = try await snsClient.subscribe(
            input: SubscribeInput(
                attributes: q1Attributes,
                endpoint: q1Arn,
                protocol: "sqs",
                topicArn: topicArn
            )
        )

        guard let q1SubscriptionArn = sub1Output.subscriptionArn else {
            print("Invalid subscription ARN returned for queue 1!")
            return
        }

        // 7. Repeat steps 3-6 for the second queue.
```

```
        let q2Url = try await createQueue(prompt: "Enter the name of the second
queue: ",
                                  sqsClient: sqsClient, isFIFO: isFIFO)

        guard let q2Url else {
            print("Unable to create queue 2!")
            return
        }

        let q2Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q2Url)

        guard let q2Arn else {
            print("Unable to get ARN of queue 2!")
            return
        }
        print("Got queue 2 ARN: \(q2Arn)")

        try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q2Url,
                                 queueArn: q2Arn, topicArn: topicArn)

        var q2Attributes: [String:String]? = nil

        if isFIFO {
            let subPrompt = """
                Would you like to filter messages for the second queue's
subscription to the
                topic \(topicName) (Y/N)?
                """
            if (yesNoRequest(prompt: subPrompt)) {
                q2Attributes = buildFilterAttributes()
            }
        }

        let sub2Output = try await snsClient.subscribe(
            input: SubscribeInput(
                attributes: q2Attributes,
                endpoint: q2Arn,
                protocol: "sqs",
                topicArn: topicArn
            )
        )

        guard let q2SubscriptionArn = sub2Output.subscriptionArn else {
            print("Invalid subscription ARN returned for queue 1!")
```

```
            return
        }

        // 8. Let the user publish messages to the topic, asking for a message
        //    body for each message. Handle the types of topic correctly (SEE
        //    MVP INFORMATION AND FIX THESE COMMENTS!!!

        print("\n\(rowOfStars)\n")

        var first = true

        repeat {
            var publishInput = PublishInput(
                topicArn: topicArn
            )

            publishInput.message = stringRequest(prompt: "Enter message text to
 publish: ")

            // If using a FIFO topic, a message group ID must be set on the
            // message.

            if isFIFO {
                if first {
                    print("""
                        Because you're using a FIFO topic, you must set a message
                        group ID. All messages within the same group will be
                        received in the same order in which they were published.
\n
                        """
                    )
                }
                publishInput.messageGroupId = stringRequest(prompt: "Enter a
 message group ID for this message: ")

                if !isContentBasedDeduplication {
                    if first {
                        print("""
                              Because you're not using content-based
 deduplication, you
                              must enter a deduplication ID. If other messages
 with the
                              same deduplication ID are published within the same
                              deduplication interval, they will not be delivered.
```

```
                                """
                        )
                    }
                    publishInput.messageDeduplicationId = stringRequest(prompt:
"Enter a deduplication ID for this message: ")
                }
            }

            // Allow the user to add a value for the "tone" attribute if they
            // wish to do so.

            var messageAttributes: [String:SNSClientTypes.MessageAttributeValue]
= [:]
            let attrValSelection = menuRequest(prompt: "Choose a tone to apply to
this message.", options: attrValues)

            if attrValSelection != 0 {
                let val = SNSClientTypes.MessageAttributeValue(dataType:
"String", stringValue: attrValues[attrValSelection])
                messageAttributes["tone"] = val
            }

            publishInput.messageAttributes = messageAttributes

            // Publish the message and display its ID.

            let publishOutput = try await snsClient.publish(input: publishInput)

            guard let messageID = publishOutput.messageId else {
                print("Unable to get the published message's ID!")
                return
            }

            print("Message published with ID \(messageID).")
            first = false

            // 9. Repeat step 8 until the user says they don't want to post
            //    another.

        } while (yesNoRequest(prompt: "Post another message (Y/N)? "))

        // 10. Display a list of the messages in each queue by using
        //     `ReceiveMessage`. Show at least the body and the attributes.
```

```
        print(rowOfStars)
        print("Contents of queue 1:")
        let q1DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q1Url)
        print("\n\nContents of queue 2:")
        let q2DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q2Url)
        print(rowOfStars)

        returnRequest(prompt: "\nPress return to clean up: ")

        // 11. Delete the received messages using `DeleteMessageBatch`.

        print("Deleting the messages from queue 1...")
        try await deleteMessageList(sqsClient: sqsClient, queueUrl: q1Url,
deleteList: q1DeleteList)
        print("\nDeleting the messages from queue 2...")
        try await deleteMessageList(sqsClient: sqsClient, queueUrl: q2Url,
deleteList: q2DeleteList)

        // 12. Unsubscribe and delete both queues.

        print("\nUnsubscribing from queue 1...")
        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(subscriptionArn: q1SubscriptionArn)
        )

        print("Unsubscribing from queue 2...")
        _ = try await snsClient.unsubscribe(
            input: UnsubscribeInput(subscriptionArn: q2SubscriptionArn)
        )

        print("Deleting queue 1...")
        _ = try await sqsClient.deleteQueue(
            input: DeleteQueueInput(queueUrl: q1Url)
        )

        print("Deleting queue 2...")
        _ = try await sqsClient.deleteQueue(
            input: DeleteQueueInput(queueUrl: q2Url)
        )

        // 13. Delete the topic.
```

```
        print("Deleting the SNS topic...")
        _ = try await snsClient.deleteTopic(
            input: DeleteTopicInput(topicArn: topicArn)
        )
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API 세부 정보는 AWS SDK for Swift API 참조의 다음 주제를 참조하세요.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 Amazon SQS로 메시지 배치 전송 및 수신

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Amazon SQS 대기열을 생성합니다.
- 대기열에 메시지를 일괄 전송합니다.
- 대기열에서 메시지를 일괄 수신합니다.
- 대기열에서 메시지 배치를 삭제합니다.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

다음 예제와 같이에서 두 가지 접근 방식을 사용하여 Amazon SQS에서 배치 메시지 작업을 처리할 수 있습니다 AWS SDK for Java 2.x.

SendRecvBatch.java는 명시적 배치 작업을 사용합니다. 메시지 배치를 수동으로 생성하고 `sendMessageBatch()` 및를 `deleteMessageBatch()` 직접 호출합니다. 또한 실패한 메시지를 포함하여 배치 응답을 처리합니다. 이 접근 방식을 사용하면 배치 크기 조정 및 오류 처리를 완벽하게 제어할 수 있습니다. 그러나 일괄 처리 로직을 관리하려면 더 많은 코드가 필요합니다.

SimpleProducerConsumer.java는 자동 요청 일괄 처리를 위해 상위 수준 `SqsAsyncBatchManager` 라이브러리를 사용합니다. 표준 클라이언트와 동일한 메서드 서명으로 개별 `sendMessage()` 및 `deleteMessage()` 호출을 수행합니다. SDK는 이러한 호출을 자동으로 버퍼링하고 배치 작업으로 전송합니다. 이 접근 방식은 배치 성능 이점을 제공하면서 코드 변경을 최소화해야 합니다.

배치 구성 및 오류 처리를 세밀하게 제어해야 하는 경우 명시적 배치 처리를 사용합니다. 코드 변경을 최소화하면서 성능을 최적화하려면 자동 일괄 처리를 사용합니다.

SendRecvBatch.java - 메시지와 함께 명시적 배치 작업을 사용합니다.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.BatchResultErrorEntry;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchResultEntry;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.MessageAttributeValue;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResultEntry;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
```

```
/**
 * This code demonstrates basic message operations in Amazon Simple Queue Service
 (Amazon SQS).
 */

public class SendRecvBatch {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SendRecvBatch.class);
    private static final SqsClient sqsClient = SqsClient.create();


    public static void main(String[] args) {
        usageDemo();
    }
    /**
     * Send a batch of messages in a single request to an SQS queue.
     * This request may return overall success even when some messages were not
 sent.
     * The caller must inspect the Successful and Failed lists in the response
 and
     * resend any failed messages.
     *
     * @param queueUrl  The URL of the queue to receive the messages.
     * @param messages  The messages to send to the queue. Each message contains
 a body and attributes.
     * @return The response from SQS that contains the list of successful and
 failed messages.
     */
    public static SendMessageBatchResponse sendMessages(
            String queueUrl, List<MessageEntry> messages) {

        try {
            List<SendMessageBatchRequestEntry> entries = new ArrayList<>();

            for (int i = 0; i < messages.size(); i++) {
                MessageEntry message = messages.get(i);
                entries.add(SendMessageBatchRequestEntry.builder()
                        .id(String.valueOf(i))
                        .messageBody(message.getBody())
                        .messageAttributes(message.getAttributes())
                        .build());
            }
```

```
            SendMessageBatchRequest sendBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)
                    .entries(entries)
                    .build();

            SendMessageBatchResponse response =
sqsClient.sendMessageBatch(sendBatchRequest);

            if (!response.successful().isEmpty()) {
                for (SendMessageBatchResultEntry resultEntry :
response.successful()) {
                    LOGGER.info("Message sent: {}: {}", resultEntry.messageId(),

messages.get(Integer.parseInt(resultEntry.id())).getBody());
                }
            }

            if (!response.failed().isEmpty()) {
                for (BatchResultErrorEntry errorEntry : response.failed()) {
                    LOGGER.warn("Failed to send: {}: {}", errorEntry.id(),

messages.get(Integer.parseInt(errorEntry.id())).getBody());
                }
            }

            return response;

        } catch (SqsException e) {
            LOGGER.error("Send messages failed to queue: {}", queueUrl, e);
            throw e;
        }
    }

    /**
     * Receive a batch of messages in a single request from an SQS queue.
     *
     * @param queueUrl   The URL of the queue from which to receive messages.
     * @param maxNumber  The maximum number of messages to receive (capped at 10
by SQS).
     *                   The actual number of messages received might be less.
     * @param waitTime   The maximum time to wait (in seconds) before returning.
When
```

```
    *                   this number is greater than zero, long polling is used.
This
    *                   can result in reduced costs and fewer false empty
responses.
    * @return The list of Message objects received. These each contain the body
    *         of the message and metadata and custom attributes.
    */
   public static List<Message> receiveMessages(String queueUrl, int maxNumber,
int waitTime) {
       try {
           ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                   .queueUrl(queueUrl)
                   .maxNumberOfMessages(maxNumber)
                   .waitTimeSeconds(waitTime)
                   .messageAttributeNames("All")
                   .build();

           List<Message> messages =
sqsClient.receiveMessage(receiveRequest).messages();

           for (Message message : messages) {
               LOGGER.info("Received message: {}: {}", message.messageId(),
message.body());
           }

           return messages;

       } catch (SqsException e) {
           LOGGER.error("Couldn't receive messages from queue: {}", queueUrl,
e);
           throw e;
       }
   }

   /**
    * Delete a batch of messages from a queue in a single request.
    *
    * @param queueUrl  The URL of the queue from which to delete the messages.
    * @param messages  The list of messages to delete.
    * @return The response from SQS that contains the list of successful and
failed
    *         message deletions.
    */
```

```
   public static DeleteMessageBatchResponse deleteMessages(String queueUrl,
List<Message> messages) {
       try {
           List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();

           for (int i = 0; i < messages.size(); i++) {
               entries.add(DeleteMessageBatchRequestEntry.builder()
                       .id(String.valueOf(i))
                       .receiptHandle(messages.get(i).receiptHandle())
                       .build());
           }

           DeleteMessageBatchRequest deleteRequest =
DeleteMessageBatchRequest.builder()
                   .queueUrl(queueUrl)
                   .entries(entries)
                   .build();

           DeleteMessageBatchResponse response =
sqsClient.deleteMessageBatch(deleteRequest);

           if (!response.successful().isEmpty()) {
               for (DeleteMessageBatchResultEntry resultEntry :
response.successful()) {
                   LOGGER.info("Deleted {}",
messages.get(Integer.parseInt(resultEntry.id())).receiptHandle());
               }
           }

           if (!response.failed().isEmpty()) {
               for (BatchResultErrorEntry errorEntry : response.failed()) {
                   LOGGER.warn("Could not delete {}",
messages.get(Integer.parseInt(errorEntry.id())).receiptHandle());
               }
           }

           return response;

       } catch (SqsException e) {
           LOGGER.error("Couldn't delete messages from queue {}", queueUrl, e);
           throw e;
       }
   }
```

```
    /**
     * Helper class to represent a message with body and attributes.
     */
    public static class MessageEntry {
        private final String body;
        private final Map<String, MessageAttributeValue> attributes;

        public MessageEntry(String body, Map<String, MessageAttributeValue>
attributes) {
            this.body = body;
            this.attributes = attributes != null ? attributes : new HashMap<>();
        }

        public String getBody() {
            return body;
        }

        public Map<String, MessageAttributeValue> getAttributes() {
            return attributes;
        }
    }

    /**
     * Shows how to:
     * * Read the lines from a file and send the lines in
     *   batches of 10 as messages to a queue.
     * * Receive the messages in batches until the queue is empty.
     * * Reassemble the lines of the file and verify they match the original
file.
     */
    public static void usageDemo() {
        LOGGER.info("-".repeat(88));
        LOGGER.info("Welcome to the Amazon Simple Queue Service (Amazon SQS)
demo!");
        LOGGER.info("-".repeat(88));

        String queueUrl = null;
        try {
            // Create a queue for the demo.
            String queueName = "sqs-usage-demo-message-wrapper-" +
System.currentTimeMillis();
            CreateQueueRequest createRequest = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();
```

```
            queueUrl = sqsClient.createQueue(createRequest).queueUrl();
            LOGGER.info("Created queue: {}", queueUrl);

            try (InputStream inputStream =
 SendRecvBatch.class.getResourceAsStream("/log4j2.xml");
                 BufferedReader reader = new BufferedReader(new
 InputStreamReader(inputStream))) {

                List<String> lines = reader.lines().toList();

                // Send file lines in batches.
                int batchSize = 10;
                LOGGER.info("Sending file lines in batches of {} as messages.",
 batchSize);

                for (int i = 0; i < lines.size(); i += batchSize) {
                    List<MessageEntry> messageBatch = new ArrayList<>();

                    for (int j = i; j < Math.min(i + batchSize, lines.size()); j+
+) {
                        String line = lines.get(j);
                        if (line == null || line.trim().isEmpty()) {
                            continue; // Skip empty lines.
                        }

                        Map<String, MessageAttributeValue> attributes = new
 HashMap<>();
                        attributes.put("line", MessageAttributeValue.builder()
                                .dataType("String")
                                .stringValue(String.valueOf(j))
                                .build());

                        messageBatch.add(new MessageEntry(lines.get(j),
 attributes));
                    }

                    sendMessages(queueUrl, messageBatch);
                    System.out.print(".");
                    System.out.flush();
                }

                LOGGER.info("\nDone. Sent {} messages.", lines.size());

                // Receive and process messages.
```

```
                LOGGER.info("Receiving, handling, and deleting messages in
batches of {}.", batchSize);
                String[] receivedLines = new String[lines.size()];
                boolean moreMessages = true;

                while (moreMessages) {
                    List<Message> receivedMessages = receiveMessages(queueUrl,
batchSize, 5);

                    for (Message message : receivedMessages) {
                        int lineNumber =
Integer.parseInt(message.messageAttributes().get("line").stringValue());
                        receivedLines[lineNumber] = message.body();
                    }

                    if (!receivedMessages.isEmpty()) {
                        deleteMessages(queueUrl, receivedMessages);
                    } else {
                        moreMessages = false;
                    }
                }

                LOGGER.info("\nDone.");

                // Verify that all lines were received correctly.
                boolean allLinesMatch = true;
                for (int i = 0; i < lines.size(); i++) {
                    String originalLine = lines.get(i);
                    String receivedLine = receivedLines[i] == null ? "" :
receivedLines[i];

                    if (!originalLine.equals(receivedLine)) {
                        allLinesMatch = false;
                        break;
                    }
                }

                if (allLinesMatch) {
                    LOGGER.info("Successfully reassembled all file lines!");
                } else {
                    LOGGER.info("Uh oh, some lines were missed!");
                }
            }
        } catch (SqsException e) {
```

```
            LOGGER.error("SQS operation failed", e);
        } catch (RuntimeException | IOException e) {
            LOGGER.error("Unexpected runtime error during demo", e);
        } finally {
            // Clean up by deleting the queue if it was created.
            if (queueUrl != null) {
                try {
                    DeleteQueueRequest deleteQueueRequest =
 DeleteQueueRequest.builder()
                            .queueUrl(queueUrl)
                            .build();
                    sqsClient.deleteQueue(deleteQueueRequest);
                    LOGGER.info("Deleted queue: {}", queueUrl);
                } catch (SqsException e) {
                    LOGGER.error("Failed to delete queue: {}", queueUrl, e);
                }
            }
        }

        LOGGER.info("Thanks for watching!");
        LOGGER.info("-".repeat(88));
    }
 }
```

SimpleProducerConsumer.java - 메시지의 자동 일괄 처리를 사용합니다.

```
package com.example.sqs;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsAsyncClient;
import software.amazon.awssdk.services.sqs.batchmanager.SqsAsyncBatchManager;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageResponse;
import software.amazon.awssdk.core.exception.SdkException;
```

```
import java.math.BigInteger;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Demonstrates the AWS SDK for Java 2.x Automatic Request Batching API for
 Amazon SQS.
 *
 * This example showcases the high-level SqsAsyncBatchManager library that
 provides
 * efficient batching and buffering for SQS operations. The batch manager offers
 * methods that directly mirror SqsAsyncClient methods—sendMessage,
 changeMessageVisibility,
 * deleteMessage, and receiveMessage—making it a drop-in replacement with minimal
 code changes.
 *
 * Key features of the SqsAsyncBatchManager:
 * - Automatic batching: The SDK automatically buffers individual requests and
 sends them
 *   as batches when maxBatchSize (default: 10) or sendRequestFrequency (default:
 200ms)
 *   thresholds are reached
 * - Familiar API: Method signatures match SqsAsyncClient exactly, requiring no
 learning curve
 * - Background optimization: The batch manager maintains internal buffers and
 handles
 *   batching logic transparently
 * - Asynchronous operations: All methods return CompletableFuture for non-
blocking execution
 *
 * Performance benefits demonstrated:
 * - Reduced API calls: Multiple individual requests are consolidated into single
 batch operations
 * - Lower costs: Fewer API calls result in reduced SQS charges
 * - Higher throughput: Batch operations process more messages per second
 * - Efficient resource utilization: Fewer network round trips and better
 connection reuse
 *
 * This example compares:
```

```
 * 1. Single-message operations using SqsAsyncClient directly
 * 2. Batch operations using SqsAsyncBatchManager with identical method calls
 *
 * Usage patterns:
 * - Set batch size to 1 to use SqsAsyncClient for baseline performance
 measurement
 * - Set batch size > 1 to use SqsAsyncBatchManager for optimized batch
 processing
 * - Monitor real-time throughput metrics to observe performance improvements
 *
 * Prerequisites:
 * - AWS SDK for Java 2.x version 2.28.0 or later
 * - An existing SQS queue
 * - Valid AWS credentials configured
 *
 * The program displays real-time metrics showing the dramatic performance
 difference
 * between individual operations and automatic batching.
 */
public class SimpleProducerConsumer {

    // The maximum runtime of the program.
    private final static int MAX_RUNTIME_MINUTES = 60;
    private final static Logger log =
 LoggerFactory.getLogger(SimpleProducerConsumer.class);

    /**
     * Runs the SQS batching demonstration with user-configured parameters.
     *
     * Prompts for queue name, thread counts, batch size, message size, and
 runtime.
     * Creates producer and consumer threads to demonstrate batching performance.
     *
     * @param args command line arguments (not used)
     * @throws InterruptedException if thread operations are interrupted
     */
    public static void main(String[] args) throws InterruptedException {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the number of producers: ");
```

```
        final int producerCount = input.nextInt();

        System.out.print("Enter the number of consumers: ");
        final int consumerCount = input.nextInt();

        System.out.print("Enter the number of messages per batch: ");
        final int batchSize = input.nextInt();

        System.out.print("Enter the message size in bytes: ");
        final int messageSizeByte = input.nextInt();

        System.out.print("Enter the run time in minutes: ");
        final int runTimeMinutes = input.nextInt();

        // Create SQS async client and batch manager for all operations.
        // The SqsAsyncBatchManager is created from the SqsAsyncClient using the
        // batchManager() factory method, which provides default batching
configuration.
        // This high-level library automatically handles request buffering and
batching
        // while maintaining the same method signatures as SqsAsyncClient.
        final SqsAsyncClient sqsAsyncClient = SqsAsyncClient.create();
        final SqsAsyncBatchManager batchManager = sqsAsyncClient.batchManager();

        final String queueUrl =
sqsAsyncClient.getQueueUrl(GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build()).join().queueUrl();

        // The flag used to stop producer, consumer, and monitor threads.
        final AtomicBoolean stop = new AtomicBoolean(false);

        // Start the producers.
        final AtomicInteger producedCount = new AtomicInteger();
        final Thread[] producers = new Thread[producerCount];
        for (int i = 0; i < producerCount; i++) {
            if (batchSize == 1) {
                producers[i] = new Producer(sqsAsyncClient, queueUrl,
messageSizeByte,
                        producedCount, stop);
            } else {
                producers[i] = new BatchProducer(batchManager, queueUrl,
batchSize,
                        messageSizeByte, producedCount, stop);
```

```
            }
            producers[i].start();
        }

        // Start the consumers.
        final AtomicInteger consumedCount = new AtomicInteger();
        final Thread[] consumers = new Thread[consumerCount];
        for (int i = 0; i < consumerCount; i++) {
            if (batchSize == 1) {
                consumers[i] = new Consumer(sqsAsyncClient, queueUrl,
consumedCount, stop);
            } else {
                consumers[i] = new BatchConsumer(batchManager, queueUrl,
batchSize,
                        consumedCount, stop);
            }
            consumers[i].start();
        }

        // Start the monitor thread.
        final Thread monitor = new Monitor(producedCount, consumedCount, stop);
        monitor.start();

        // Wait for the specified amount of time then stop.
        Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runTimeMinutes,
                MAX_RUNTIME_MINUTES)));
        stop.set(true);

        // Join all threads.
        for (int i = 0; i < producerCount; i++) {
            producers[i].join();
        }

        for (int i = 0; i < consumerCount; i++) {
            consumers[i].join();
        }

        monitor.interrupt();
        monitor.join();

        // Close resources
        batchManager.close();
        sqsAsyncClient.close();
    }
```

```
    /**
     * Creates a random string of approximately the specified size in bytes.
     *
     * @param sizeByte the target size in bytes for the generated string
     * @return a random string encoded in base-32
     */
    private static String makeRandomString(int sizeByte) {
        final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
        new Random().nextBytes(bs);
        bs[0] = (byte) ((bs[0] | 64) & 127);
        return new BigInteger(bs).toString(32);
    }

    /**
     * Sends messages individually using SqsAsyncClient for baseline performance
measurement.
     *
     * This producer demonstrates traditional single-message operations without
batching.
     * Each sendMessage() call results in a separate API request to SQS,
providing
     * a performance baseline for comparison with the batch operations.
     *
     * The sendMessage() method signature is identical to
SqsAsyncBatchManager.sendMessage(),
     * showing how the high-level batching library maintains API compatibility
while
     * adding automatic optimization behind the scenes.
     */
    private static class Producer extends Thread {
        final SqsAsyncClient sqsAsyncClient;
        final String queueUrl;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        /**
         * Creates a producer thread for single-message operations.
         *
         * @param sqsAsyncClient the SQS client for sending messages
         * @param queueUrl the URL of the target queue
         * @param messageSizeByte the size of messages to generate
         * @param producedCount shared counter for tracking sent messages
```

```
         * @param stop shared flag to signal thread termination
         */
        Producer(SqsAsyncClient sqsAsyncClient, String queueUrl, int
messageSizeByte,
                 AtomicInteger producedCount, AtomicBoolean stop) {
            this.sqsAsyncClient = sqsAsyncClient;
            this.queueUrl = queueUrl;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /**
         * Continuously sends messages until the stop flag is set.
         *
         * Uses SqsAsyncClient.sendMessage() directly, resulting in one API call
per message.
         * This approach provides baseline performance metrics for comparison
with batching.
         * Each call blocks until the individual message is sent, demonstrating
traditional
         * one-request-per-operation behavior.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    sqsAsyncClient.sendMessage(SendMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .messageBody(theMessage)
                            .build()).join();
                    producedCount.incrementAndGet();
                }
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                // Handle both SdkException and CompletionException from async
operations.
                // If this unlikely condition occurs, stop.
                log.error("Producer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
```

```
    * Sends messages using SqsAsyncBatchManager for automatic request batching
and optimization.
    *
    * This producer demonstrates the AWS SDK for Java 2.x high-level batching
library.
    * The SqsAsyncBatchManager automatically buffers individual sendMessage()
calls and
    * sends them as batches when thresholds are reached:
    * - maxBatchSize: Maximum 10 messages per batch (default)
    * - sendRequestFrequency: 200ms timeout before sending partial batches
(default)
    *
    * Key advantages of the batching approach:
    * - Identical API: batchManager.sendMessage() has the same signature as
sqsAsyncClient.sendMessage()
    * - Automatic optimization: No code changes needed to benefit from batching
    * - Transparent buffering: The SDK handles batching logic internally
    * - Reduced API calls: Multiple messages sent in single batch requests
    * - Lower costs: Fewer API calls result in reduced SQS charges
    * - Higher throughput: Batch operations process significantly more messages
per second
    */
   private static class BatchProducer extends Thread {
       final SqsAsyncBatchManager batchManager;
       final String queueUrl;
       final int batchSize;
       final AtomicInteger producedCount;
       final AtomicBoolean stop;
       final String theMessage;

       /**
        * Creates a producer thread for batch operations.
        *
        * @param batchManager the batch manager for efficient message sending
        * @param queueUrl the URL of the target queue
        * @param batchSize the number of messages to send per batch
        * @param messageSizeByte the size of messages to generate
        * @param producedCount shared counter for tracking sent messages
        * @param stop shared flag to signal thread termination
        */
       BatchProducer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                     int messageSizeByte, AtomicInteger producedCount,
                     AtomicBoolean stop) {
```

```
            this.batchManager = batchManager;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /**
         * Continuously sends batches of messages using the high-level batching
library.
         *
         * Notice how batchManager.sendMessage() uses the exact same method
signature
         * and request builder pattern as SqsAsyncClient.sendMessage(). This
demonstrates
         * the drop-in replacement capability of the SqsAsyncBatchManager.
         *
         * The SDK automatically:
         * - Buffers individual sendMessage() calls internally
         * - Groups them into batch requests when thresholds are met
         * - Sends SendMessageBatchRequest operations to SQS
         * - Returns individual CompletableFuture responses for each message
         *
         * This transparent batching provides significant performance
improvements
         * without requiring changes to application logic or error handling
patterns.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    // Send multiple messages using the high-level batch manager.
                    // Each batchManager.sendMessage() call uses identical syntax
to
                    // sqsAsyncClient.sendMessage(), demonstrating API
compatibility.
                    // The SDK automatically buffers these calls and sends them
as
                    // batch operations when maxBatchSize (10) or
sendRequestFrequency (200ms)
                    // thresholds are reached, significantly improving
throughput.
                    for (int i = 0; i < batchSize; i++) {
```

```
                        CompletableFuture<SendMessageResponse> future =
batchManager.sendMessage(
                                SendMessageRequest.builder()
                                        .queueUrl(queueUrl)
                                        .messageBody(theMessage)
                                        .build());

                        // Handle the response asynchronously
                        future.whenComplete((response, throwable) -> {
                            if (throwable == null) {
                                producedCount.incrementAndGet();
                            } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
                                      !(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {
                                log.error("BatchProducer: Failed to send
message", throwable);
                            }
                            // Ignore CancellationException and executor shutdown
errors - expected during shutdown
                        });
                    }

                    // Small delay to allow batching to occur
                    Thread.sleep(10);
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                log.error("BatchProducer interrupted: " + e.getMessage());
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                log.error("BatchProducer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * Receives and deletes messages individually using SqsAsyncClient for
baseline measurement.
     *
     * This consumer demonstrates traditional single-message operations without
batching.
     * Each receiveMessage() and deleteMessage() call results in separate API
requests,
```

```
     * providing a performance baseline for comparison with batch operations.
     *
     * The method signatures are identical to SqsAsyncBatchManager methods:
     * - receiveMessage() matches batchManager.receiveMessage()
     * - deleteMessage() matches batchManager.deleteMessage()
     *
     * This API consistency allows easy migration to the high-level batching
 library.
     */
    private static class Consumer extends Thread {
        final SqsAsyncClient sqsAsyncClient;
        final String queueUrl;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        /**
         * Creates a consumer thread for single-message operations.
         *
         * @param sqsAsyncClient the SQS client for receiving messages
         * @param queueUrl the URL of the source queue
         * @param consumedCount shared counter for tracking processed messages
         * @param stop shared flag to signal thread termination
         */
        Consumer(SqsAsyncClient sqsAsyncClient, String queueUrl, AtomicInteger
 consumedCount,
                 AtomicBoolean stop) {
            this.sqsAsyncClient = sqsAsyncClient;
            this.queueUrl = queueUrl;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Continuously receives and deletes messages using traditional single-
request operations.
         *
         * Uses SqsAsyncClient methods directly:
         * - receiveMessage(): One API call per receive operation
         * - deleteMessage(): One API call per delete operation
         *
         * This approach demonstrates the baseline performance without batching
 optimization.
         * Compare these method calls with the identical signatures used in
 BatchConsumer
```

```
        * to see how the high-level batching library maintains API
compatibility.
        */
       public void run() {
           try {
               while (!stop.get()) {
                   try {
                       final ReceiveMessageResponse result =
sqsAsyncClient.receiveMessage(
                               ReceiveMessageRequest.builder()
                                       .queueUrl(queueUrl)
                                       .build()).join();

                       if (!result.messages().isEmpty()) {
                           final Message m = result.messages().get(0);
                           // Note: deleteMessage() signature identical to
batchManager.deleteMessage()

sqsAsyncClient.deleteMessage(DeleteMessageRequest.builder()
                                   .queueUrl(queueUrl)
                                   .receiptHandle(m.receiptHandle())
                                   .build()).join();
                           consumedCount.incrementAndGet();
                       }
                   } catch (SdkException |
java.util.concurrent.CompletionException e) {
                       log.error(e.getMessage());
                   }
               }
           } catch (SdkException | java.util.concurrent.CompletionException e) {
               // Handle both SdkException and CompletionException from async
operations.
               // If this unlikely condition occurs, stop.
               log.error("Consumer: " + e.getMessage());
               System.exit(1);
           }
       }
   }

   /**
    * Receives and deletes messages using SqsAsyncBatchManager for automatic
optimization.
    *
```

```
    * This consumer demonstrates the AWS SDK for Java 2.x high-level batching
library
    * for message consumption. The SqsAsyncBatchManager provides two key
optimizations:
    *
    * 1. Receive optimization: Maintains an internal buffer of messages fetched
in the
    *    background, so receiveMessage() calls return immediately from the
buffer
    * 2. Delete batching: Automatically buffers deleteMessage() calls and sends
them
    *    as DeleteMessageBatchRequest operations when thresholds are reached
    *
    * Key features:
    * - Identical API: receiveMessage() and deleteMessage() have the same
signatures
    *   as SqsAsyncClient methods, making this a true drop-in replacement
    * - Background fetching: The batch manager continuously fetches messages to
keep
    *   the internal buffer populated, reducing receive latency
    * - Automatic delete batching: Individual deleteMessage() calls are buffered
and
    *   sent as batch operations (up to 10 per batch, 200ms frequency)
    * - Transparent optimization: No application logic changes needed to benefit
    *
    * Performance benefits:
    * - Reduced API calls through automatic batching of delete operations
    * - Lower latency for receives due to background message buffering
    * - Higher overall throughput with fewer network round trips
    */
   private static class BatchConsumer extends Thread {
       final SqsAsyncBatchManager batchManager;
       final String queueUrl;
       final int batchSize;
       final AtomicInteger consumedCount;
       final AtomicBoolean stop;

       /**
        * Creates a consumer thread for batch operations.
        *
        * @param batchManager the batch manager for efficient message processing
        * @param queueUrl the URL of the source queue
        * @param batchSize the maximum number of messages to receive per batch
        * @param consumedCount shared counter for tracking processed messages
```

```
         * @param stop shared flag to signal thread termination
         */
        BatchConsumer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                      AtomicInteger consumedCount, AtomicBoolean stop) {
            this.batchManager = batchManager;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Continuously receives and deletes messages using the high-level
batching library.
         *
         * Demonstrates the key advantage of SqsAsyncBatchManager: identical
method signatures
         * with automatic optimization. Notice how:
         *
         * - batchManager.receiveMessage() uses the same syntax as
sqsAsyncClient.receiveMessage()
         * - batchManager.deleteMessage() uses the same syntax as
sqsAsyncClient.deleteMessage()
         *
         * Behind the scenes, the batch manager:
         * 1. Maintains an internal message buffer populated by background
fetching
         * 2. Returns messages immediately from the buffer (reduced latency)
         * 3. Automatically batches deleteMessage() calls into
DeleteMessageBatchRequest operations
         * 4. Sends batch deletes when maxBatchSize (10) or sendRequestFrequency
(200ms) is reached
         *
         * This provides significant performance improvements with zero code
changes
         * compared to traditional SqsAsyncClient usage patterns.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    // Receive messages using the high-level batch manager.
                    // This call uses identical syntax to
sqsAsyncClient.receiveMessage()
```

```
                    // but benefits from internal message buffering for improved
performance.
                    final ReceiveMessageResponse result =
batchManager.receiveMessage(
                            ReceiveMessageRequest.builder()
                                    .queueUrl(queueUrl)
                                    .maxNumberOfMessages(Math.min(batchSize, 10))
                                    .build()).join();

                    if (!result.messages().isEmpty()) {
                        final List<Message> messages = result.messages();

                        // Delete messages using the batch manager.
                        // Each deleteMessage() call uses identical syntax to
SqsAsyncClient
                        // but the SDK automatically buffers these calls and
sends them
                        // as DeleteMessageBatchRequest operations for optimal
performance.
                        for (Message message : messages) {
                            CompletableFuture<DeleteMessageResponse> future =
batchManager.deleteMessage(
                                    DeleteMessageRequest.builder()
                                            .queueUrl(queueUrl)

.receiptHandle(message.receiptHandle())
                                            .build());

                            future.whenComplete((response, throwable) -> {
                                if (throwable == null) {
                                    consumedCount.incrementAndGet();
                                } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
                                            !(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {
                                    log.error("BatchConsumer: Failed to delete
message", throwable);
                                }
                                // Ignore CancellationException and executor
shutdown errors - expected during shutdown
                            });
                        }
                    }
```

```
                    // Small delay to prevent tight polling
                    Thread.sleep(10);
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                log.error("BatchConsumer interrupted: " + e.getMessage());
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                // Handle both SdkException and CompletionException from async
operations.
                // If this unlikely condition occurs, stop.
                log.error("BatchConsumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * Displays real-time throughput statistics every second.
     *
     * This thread logs the current count of produced and consumed messages
     * to help you monitor the performance comparison.
     */
    private static class Monitor extends Thread {
        private final AtomicInteger producedCount;
        private final AtomicInteger consumedCount;
        private final AtomicBoolean stop;

        /**
         * Creates a monitoring thread that displays throughput statistics.
         *
         * @param producedCount shared counter for messages sent
         * @param consumedCount shared counter for messages processed
         * @param stop shared flag to signal thread termination
         */
        Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
                AtomicBoolean stop) {
            this.producedCount = producedCount;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Logs throughput statistics every second until stopped.
         *
```

```
         * Displays the current count of produced and consumed messages
         * to help monitor the performance comparison between batching
 strategies.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    Thread.sleep(1000);
                    log.info("produced messages = " + producedCount.get()
                            + ", consumed messages = " + consumedCount.get());
                }
            } catch (InterruptedException e) {
                // Allow the thread to exit.
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon SQS 메시지 함수를 래핑하는 함수를 생성합니다.

```
import logging
import sys

import boto3
from botocore.exceptions import ClientError

import queue_wrapper

logger = logging.getLogger(__name__)
sqs = boto3.resource("sqs")

def send_messages(queue, messages):
    """
    Send a batch of messages in a single request to an SQS queue.
    This request may return overall success even when some messages were not
 sent.
    The caller must inspect the Successful and Failed lists in the response and
    resend any failed messages.

    :param queue: The queue to receive the messages.
    :param messages: The messages to send to the queue. These are simplified to
                     contain only the message body and attributes.
    :return: The response from SQS that contains the list of successful and
 failed
             messages.
    """
    try:
        entries = [
            {
                "Id": str(ind),
                "MessageBody": msg["body"],
                "MessageAttributes": msg["attributes"],
            }
            for ind, msg in enumerate(messages)
        ]
        response = queue.send_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info(
                    "Message sent: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
```

```
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Failed to send: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
    except ClientError as error:
        logger.exception("Send messages failed to queue: %s", queue)
        raise error
    else:
        return response



def receive_messages(queue, max_number, wait_time):
    """
    Receive a batch of messages in a single request from an SQS queue.

    :param queue: The queue from which to receive messages.
    :param max_number: The maximum number of messages to receive. The actual
 number
                        of messages received might be less.
    :param wait_time: The maximum time to wait (in seconds) before returning.
 When
                       this number is greater than zero, long polling is used.
 This
                       can result in reduced costs and fewer false empty
 responses.
    :return: The list of Message objects received. These each contain the body
             of the message and metadata and custom attributes.
    """
    try:
        messages = queue.receive_messages(
            MessageAttributeNames=["All"],
            MaxNumberOfMessages=max_number,
            WaitTimeSeconds=wait_time,
        )
        for msg in messages:
            logger.info("Received message: %s: %s", msg.message_id, msg.body)
    except ClientError as error:
        logger.exception("Couldn't receive messages from queue: %s", queue)
        raise error
    else:
```

```
        return messages



def delete_messages(queue, messages):
    """
    Delete a batch of messages from a queue in a single request.

    :param queue: The queue from which to delete the messages.
    :param messages: The list of messages to delete.
    :return: The response from SQS that contains the list of successful and
 failed
             message deletions.
    """
    try:
        entries = [
            {"Id": str(ind), "ReceiptHandle": msg.receipt_handle}
            for ind, msg in enumerate(messages)
        ]
        response = queue.delete_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info("Deleted %s",
 messages[int(msg_meta["Id"])].receipt_handle)
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Could not delete %s",
 messages[int(msg_meta["Id"])].receipt_handle
                )
    except ClientError:
        logger.exception("Couldn't delete messages from queue %s", queue)
    else:
        return response
```

래퍼 함수를 사용하여 메시지를 일괄적으로 보내고 받을 수 있습니다.

```
def usage_demo():
    """
    Shows how to:
```

```
    * Read the lines from this Python file and send the lines in
      batches of 10 as messages to a queue.
    * Receive the messages in batches until the queue is empty.
    * Reassemble the lines of the file and verify they match the original file.
    """

    def pack_message(msg_path, msg_body, msg_line):
        return {
            "body": msg_body,
            "attributes": {
                "path": {"StringValue": msg_path, "DataType": "String"},
                "line": {"StringValue": str(msg_line), "DataType": "String"},
            },
        }

    def unpack_message(msg):
        return (
            msg.message_attributes["path"]["StringValue"],
            msg.body,
            int(msg.message_attributes["line"]["StringValue"]),
        )

    print("-" * 88)
    print("Welcome to the Amazon Simple Queue Service (Amazon SQS) demo!")
    print("-" * 88)

    queue = queue_wrapper.create_queue("sqs-usage-demo-message-wrapper")

    with open(__file__) as file:
        lines = file.readlines()

    line = 0
    batch_size = 10
    received_lines = [None] * len(lines)
    print(f"Sending file lines in batches of {batch_size} as messages.")
    while line < len(lines):
        messages = [
            pack_message(__file__, lines[index], index)
            for index in range(line, min(line + batch_size, len(lines)))
        ]
        line = line + batch_size
        send_messages(queue, messages)
        print(".", end="")
        sys.stdout.flush()
```

```
    print(f"Done. Sent {len(lines) - 1} messages.")

    print(f"Receiving, handling, and deleting messages in batches of
{batch_size}.")
    more_messages = True
    while more_messages:
        received_messages = receive_messages(queue, batch_size, 2)
        print(".", end="")
        sys.stdout.flush()
        for message in received_messages:
            path, body, line = unpack_message(message)
            received_lines[line] = body
        if received_messages:
            delete_messages(queue, received_messages)
        else:
            more_messages = False
    print("Done.")

    if all([lines[index] == received_lines[index] for index in
range(len(lines))]):
        print(f"Successfully reassembled all file lines!")
    else:
        print(f"Uh oh, some lines were missed!")

    queue.delete()

    print("Thanks for watching!")
    print("-" * 88)
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하십시오.
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

.NET용 AWS 메시지 처리 프레임워크를 사용하여 Amazon SQS 메시지 게시 및 수신

다음 코드 예제에서는 .NET용 AWS 메시지 처리 프레임워크를 사용하여 Amazon SQS 메시지를 게시하고 수신하는 애플리케이션을 생성하는 방법을 보여줍니다.

.NET

SDK for .NET

.NET용 AWS 메시지 처리 프레임워크에 대한 자습서를 제공합니다. 자습서에서는 사용자가 Amazon SQS 메시지와 메시지를 수신하는 명령줄 애플리케이션을 게시할 수 있는 웹 애플리케이션을 생성합니다.

설정 및 실행 방법에 대한 전체 소스 코드와 지침은 AWS SDK for .NET 개발자 안내서의 전체 자습서와 GitHub의 예제를 참조하세요.

이 예시에서 사용되는 서비스

- Amazon SQS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon SQS Java 메시징 라이브러리를 사용하여 Amazon SQS용 Java Message Service(JMS) 인터페이스로 작업

다음 코드 예제에서는 Amazon SQS Java 메시징 라이브러리를 사용하여 JMS 인터페이스로 작업하는 방법을 보여줍니다.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

다음 예제는 표준 Amazon SQS 대기열에서 작동하며 다음을 포함합니다.

- 문자 메시지 전송.
- 메시지를 동기식으로 수신합니다.
- 메시지를 비동기적으로 수신합니다.
- CLIENT_ACKNOWLEDGE 모드를 사용하여 메시지 수신.
- UNORDERED_ACKNOWLEDGE 모드를 사용하여 메시지 수신.
- Spring을 사용하여 종속성을 주입합니다.
- 다른 예제에서 사용하는 일반적인 메서드를 제공하는 유틸리티 클래스입니다.

Amazon SQS에서 JMS를 사용하는 방법에 대한 자세한 내용은 Amazon SQS 개발자 안내서를 참조하세요.

문자 메시지 전송.

```
   /**
    * This method establishes a connection to a standard Amazon SQS queue using
the Amazon SQS
    * Java Messaging Library and sends text messages to it. It uses JMS (Java
Message Service) API
    * with automatic acknowledgment mode to ensure reliable message delivery,
and automatically
    * manages all messaging resources.
    *
    * @throws JMSException If there is a problem connecting to or sending
messages to the queue
    */
   public static void doSendTextMessage() throws JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
```

```
                SqsClient.create()
        );

        // Create the connection in a try-with-resources statement so that it's
closed automatically.
        try (SQSConnection connection = connectionFactory.createConnection()) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session that uses the JMS auto-acknowledge mode.
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));

            createAndSendMessages(session, producer);
        } // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed");
    }

    /**
     * This method reads text input from the keyboard and sends each line as a
separate message
     * to a standard Amazon SQS queue using the Amazon SQS Java Messaging
Library. It continues
     * to accept input until the user enters an empty line, using JMS (Java
Message Service) API to
     * handle the message delivery.
     *
     * @param session The JMS session used to create messages
     * @param producer The JMS message producer used to send messages to the
queue
     */
    private static void createAndSendMessages(Session session, MessageProducer
producer) {
        BufferedReader inputReader = new BufferedReader(
                new InputStreamReader(System.in, Charset.defaultCharset()));

        try {
            String input;
            while (true) {
                LOGGER.info("Enter message to send (leave empty to exit): ");
```

```
                input = inputReader.readLine();
                if (input == null || input.isEmpty()) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
                LOGGER.info("Send message {}", message.getJMSMessageID());
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
            LOGGER.error("Failed reading input: {}", e.getMessage(), e);
        } catch (JMSException e) {
            LOGGER.error("Failed sending message: {}", e.getMessage(), e);
        }
    }
```

메시지를 동기식으로 수신합니다.

```
    /**
     * This method receives messages from a standard Amazon SQS queue using the
Amazon SQS Java
     * Messaging Library. It creates a connection to the queue using JMS (Java
Message Service),
     * waits for messages to arrive, and processes them one at a time. The method
handles all
     * necessary setup and cleanup of messaging resources.
     *
     * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
     */
    public static void doReceiveMessageSync() throws JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create a connection.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
```

```
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session.
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
            MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

            connection.start();

            receiveMessages(consumer);
        }  // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed");
    }

    /**
     * This method continuously checks for new messages from a standard Amazon
SQS queue using
     * the Amazon SQS Java Messaging Library. It waits up to 20 seconds for each
message, processes
     * it using JMS (Java Message Service), and confirms receipt. The method
stops checking for
     * messages after 20 seconds of no activity.
     *
     * @param consumer The JMS message consumer that receives messages from the
queue
     */
    private static void receiveMessages(MessageConsumer consumer) {
        try {
            while (true) {
                LOGGER.info("Waiting for messages...");
                // Wait 1 minute for a message
                Message message =
consumer.receive(Duration.ofSeconds(20).toMillis());
                if (message == null) {
                    LOGGER.info("Shutting down after 20 seconds of silence.");
                    break;
                }
                SqsJmsExampleUtils.handleMessage(message);
                message.acknowledge();
                LOGGER.info("Acknowledged message {}",
message.getJMSMessageID());
            }
```

```
        } catch (JMSException e) {
            LOGGER.error("Error receiving from SQS: {}", e.getMessage(), e);
        }
    }
```

메시지를 비동기적으로 수신합니다.

```
    /**
     * This method sets up automatic message handling for a standard Amazon SQS
queue using the
     * Amazon SQS Java Messaging Library. It creates a listener that processes
messages as soon
     * as they arrive using JMS (Java Message Service), runs for 5 seconds, then
cleans up all
     * messaging resources.
     *
     * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
     */
    public static void doReceiveMessageAsync() throws JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create a connection.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session.
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

            try {
                // Create a consumer for the queue.
                MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));
```

```
                // Provide an implementation of the MessageListener interface,
which has a single 'onMessage' method.
                // We use a lambda expression for the implementation.
                consumer.setMessageListener(message -> {
                    try {
                        SqsJmsExampleUtils.handleMessage(message);
                        message.acknowledge();
                    } catch (JMSException e) {
                        LOGGER.error("Error processing message: {}",
e.getMessage());
                    }
                });
                // Start receiving incoming messages.
                connection.start();
                LOGGER.info("Waiting for messages...");
            } catch (JMSException e) {
                throw new RuntimeException(e);
            }
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }  // The connection closes automatically. This also closes the session.
        LOGGER.info( "Connection closed" );
    }
```

CLIENT_ACKNOWLEDGE 모드를 사용하여 메시지 수신.

```
    /**
     * This method demonstrates how message acknowledgment affects message
processing in a standard
     * Amazon SQS queue using the Amazon SQS Java Messaging Library. It sends
messages to the queue,
     * then shows how JMS (Java Message Service) client acknowledgment mode
handles both explicit
     * and implicit message confirmations, including how acknowledging one
message can automatically
     * acknowledge previous messages.
     *
     * @throws JMSException If there is a problem with the messaging operations
     */
```

```
   public static void doReceiveMessagesSyncClientAcknowledge() throws
JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create the connection in a try-with-resources statement so that it's
closed automatically.
       try (SQSConnection connection = connectionFactory.createConnection() ) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);

           // Create a session with client acknowledge mode.
           Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

           // Create a producer and consumer.
           MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
           MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

           // Open the connection.
           connection.start();

           // Send two text messages.
           sendMessage(producer, session, "Message 1");
           sendMessage(producer, session, "Message 2");

           // Receive a message and don't acknowledge it.
           receiveMessage(consumer, false);

           // Receive another message and acknowledge it.
           receiveMessage(consumer, true);

           // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue,
           LOGGER.info("Waiting for visibility timeout...");
           try {
               Thread.sleep(TIME_OUT_MILLIS);
```

```
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, but none will be
available. This is because in
                CLIENT_ACKNOWLEDGE mode, when we acknowledged the second message,
all previous messages were
                automatically acknowledged as well. Therefore, although we never
directly acknowledged the first
                message, it was implicitly acknowledged when we confirmed the
second one. */
            receiveMessage(consumer, true);
        } // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed.");

    }


    /**
     * Sends a text message using the specified JMS MessageProducer and Session.
     *
     * @param producer    The JMS MessageProducer used to send the message
     * @param session     The JMS Session used to create the text message
     * @param messageText The text content to be sent in the message
     * @throws JMSException If there is an error creating or sending the message
     */
    private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
        // Create a text message and send it.
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives and processes a message from a JMS queue using the specified
consumer.
     * The method waits for a message until the configured timeout period is
reached.
     * If a message is received, it is logged and optionally acknowledged based
on the
     * acknowledge parameter.
```

```
    *
    * @param consumer    The JMS MessageConsumer used to receive messages from
the queue
    * @param acknowledge Boolean flag indicating whether to acknowledge the
message.
    *                    If true, the message will be acknowledged after
processing
    * @throws JMSException If there is an error receiving, processing, or
acknowledging the message
    */
   private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSException {
       // Receive a message.
       Message message = consumer.receive(TIME_OUT_MILLIS);

       if (message == null) {
           LOGGER.info("Queue is empty!");
       } else {
           // Since this queue has only text messages, cast the message object
and print the text.
           LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
message).getText(), acknowledge);

           // Acknowledge the message if asked.
           if (acknowledge) message.acknowledge();
       }
   }
```

UNORDERED_ACKNOWLEDGE 모드를 사용하여 메시지 수신.

```
   /**
    * Demonstrates message acknowledgment behavior in UNORDERED_ACKNOWLEDGE mode
with Amazon SQS JMS.
    * In this mode, each message must be explicitly acknowledged regardless of
receive order.
    * Unacknowledged messages return to the queue after the visibility timeout
expires,
    * unlike CLIENT_ACKNOWLEDGE mode where acknowledging one message
acknowledges all previous messages.
    *
    * @throws JMSException         If a JMS-related error occurs during message
operations
```

```
    */
   public static void doReceiveMessagesUnorderedAcknowledge() throws
JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create the connection in a try-with-resources statement so that it's
closed automatically.
       try( SQSConnection connection = connectionFactory.createConnection() ) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);

           // Create a session with unordered acknowledge mode.
           Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

           // Create the producer and consumer.
           MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
           MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

           // Open a connection.
           connection.start();

           // Send two text messages.
           sendMessage(producer, session, "Message 1");
           sendMessage(producer, session, "Message 2");

           // Receive a message and don't acknowledge it.
           receiveMessage(consumer, false);

           // Receive another message and acknowledge it.
           receiveMessage(consumer, true);

           // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue.
           LOGGER.info("Waiting for visibility timeout...");
           try {
```

```
                Thread.sleep(TIME_OUT_MILLIS);
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, and we'll get the
first message again. This occurs
                because in UNORDERED_ACKNOWLEDGE mode, each message requires its
own separate acknowledgment.
                Since we only acknowledged the second message, the first message
remains in the queue for
                redelivery. */
            receiveMessage(consumer, true);

            LOGGER.info("Connection closed.");
        } // The connection closes automatically. This also closes the session.
    }

    /**
     * Sends a text message to an Amazon SQS queue using JMS.
     *
     * @param producer    The JMS MessageProducer for the queue
     * @param session     The JMS Session for message creation
     * @param messageText The message content
     * @throws JMSException If message creation or sending fails
     */
    private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
        // Create a text message and send it.
        producer.send(session.createTextMessage(messageText));
    }
    /**
     * Synchronously receives a message from an Amazon SQS queue using the JMS
API
     * with an acknowledgment parameter.
     *
     * @param consumer    The JMS MessageConsumer for the queue
     * @param acknowledge If true, acknowledges the message after receipt
     * @throws JMSException If message reception or acknowledgment fails
     */
```

```
    private static void receiveMessage(MessageConsumer consumer, boolean
 acknowledge) throws JMSException {
        // Receive a message.
        Message message = consumer.receive(TIME_OUT_MILLIS);

        if (message == null) {
            LOGGER.info("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object
 and print the text.
            LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
 message).getText(), acknowledge);

            // Acknowledge the message if asked.
            if (acknowledge) message.acknowledge();
        }
    }
```

Spring을 사용하여 종속성을 주입합니다.

```
package com.example.sqs.jms.spring;

import com.amazon.sqs.javamessaging.SQSConnection;
import com.example.sqs.jms.SqsJmsExampleUtils;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.NoSuchBeanDefinitionException;
import org.springframework.context.support.FileSystemXmlApplicationContext;

import java.io.File;
import java.net.URL;
import java.util.concurrent.TimeUnit;

/**
 * Demonstrates how to send and receive messages using the Amazon SQS Java
 Messaging Library
 * with Spring Framework integration. This example connects to a standard Amazon
 SQS message
 * queue using Spring's dependency injection to configure the connection and
 messaging components.
```

```
 * The application uses the JMS (Java Message Service) API to handle message
 operations.
 */
public class SpringExample {
    private static final Integer POLLING_SECONDS = 15;
    private static final String SPRING_XML_CONFIG_FILE =
 "SpringExampleConfiguration.xml.txt";
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SpringExample.class);

    /**
     * Demonstrates sending and receiving messages through a standard Amazon SQS
 message queue
     * using Spring Framework configuration. This method loads connection
 settings from an XML file,
     * establishes a messaging session using the Amazon SQS Java Messaging
 Library, and processes
     * messages using JMS (Java Message Service) operations. If the queue doesn't
 exist, it will
     * be created automatically.
     *
     * @param args Command line arguments (not used)
     */
    public static void main(String[] args) {

        URL resource =
 SpringExample.class.getClassLoader().getResource(SPRING_XML_CONFIG_FILE);
        File springFile = new File(resource.getFile());
        if (!springFile.exists() || !springFile.canRead()) {
            LOGGER.error("File " + SPRING_XML_CONFIG_FILE + " doesn't exist or
 isn't readable.");
            System.exit(1);
        }

        try (FileSystemXmlApplicationContext context =
                     new FileSystemXmlApplicationContext("file://" +
 springFile.getAbsolutePath())) {

            Connection connection;
            try {
                connection = context.getBean(Connection.class);
            } catch (NoSuchBeanDefinitionException e) {
                LOGGER.error("Can't find the JMS connection to use: " +
 e.getMessage(), e);
```

```
                System.exit(2);
                return;
            }

            String queueName;
            try {
                queueName = context.getBean("queueName", String.class);
            } catch (NoSuchBeanDefinitionException e) {
                LOGGER.error("Can't find the name of the queue to use: " +
e.getMessage(), e);
                System.exit(3);
                return;
            }
            try {
                if (connection instanceof SQSConnection) {
                    SqsJmsExampleUtils.ensureQueueExists((SQSConnection)
connection, queueName, SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);
                }
                // Create the JMS session.
                Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

                SqsJmsExampleUtils.sendTextMessage(session, queueName);
                MessageConsumer consumer =
session.createConsumer(session.createQueue(queueName));

                receiveMessages(consumer);
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        }   // Spring context autocloses. Managed Spring beans that implement
AutoClosable, such as the
        // 'connection' bean, are also closed.
        LOGGER.info("Context closed");
    }

    /**
     * Continuously checks for and processes messages from a standard Amazon SQS
message queue
     * using the Amazon SQS Java Messaging Library underlying the JMS API. This
method waits for incoming messages,
     * processes them when they arrive, and acknowledges their receipt using JMS
(Java Message
```

```
     * Service) operations. The method will stop checking for messages after 15
 seconds of
     * inactivity.
     *
     * @param consumer The JMS message consumer used to receive messages from the
 queue
     */
    private static void receiveMessages(MessageConsumer consumer) {
        try {
            while (true) {
                LOGGER.info("Waiting for messages...");
                // Wait 15 seconds for a message.
                Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(POLLING_SECONDS));
                if (message == null) {
                    LOGGER.info("Shutting down after {} seconds of silence.",
 POLLING_SECONDS);
                    break;
                }
                SqsJmsExampleUtils.handleMessage(message);
                message.acknowledge();
                LOGGER.info("Message acknowledged.");
            }
        } catch (JMSException e) {
            LOGGER.error("Error receiving from SQS.", e);
        }
    }
}
```

스프링 빈 정의.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
        xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-3.0.xsd
        ">
    <!-- Define the AWS Region -->
    <bean id="region" class="software.amazon.awssdk.regions.Region" factory-
method="of">
```

```
        <constructor-arg value="us-east-1"/>
    </bean>

    <bean id="credentialsProviderBean"
 class="software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider"
          factory-method="create"/>

    <bean id="clientBuilder"
 class="software.amazon.awssdk.services.sqs.SqsClient" factory-method="builder"/>

    <bean id="regionSetClientBuilder" factory-bean="clientBuilder" factory-
method="region">
        <constructor-arg ref="region"/>
    </bean>

    <!-- Configure the Builder with Credentials Provider -->
    <bean id="sqsClient" factory-bean="regionSetClientBuilder" factory-
method="credentialsProvider">
        <constructor-arg ref="credentialsProviderBean"/>
    </bean>

    <bean id="providerConfiguration"
 class="com.amazon.sqs.javamessaging.ProviderConfiguration">
        <property name="numberOfMessagesToPrefetch" value="5"/>
    </bean>

    <bean id="connectionFactory"
 class="com.amazon.sqs.javamessaging.SQSConnectionFactory">
        <constructor-arg ref="providerConfiguration"/>
        <constructor-arg ref="clientBuilder"/>
    </bean>

    <bean id="connection"
          factory-bean="connectionFactory"
          factory-method="createConnection"
          init-method="start"
          destroy-method="close"/>

    <bean id="queueName" class="java.lang.String">
        <constructor-arg value="SQSJMSClientExampleQueue"/>
    </bean>
</beans>
```

다른 예제에서 사용하는 일반적인 메서드를 제공하는 유틸리티 클래스입니다.

```
package com.example.sqs.jms;

import com.amazon.sqs.javamessaging.AmazonSQSMessagingClientWrapper;
import com.amazon.sqs.javamessaging.ProviderConfiguration;
import com.amazon.sqs.javamessaging.SQSConnection;
import com.amazon.sqs.javamessaging.SQSConnectionFactory;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.exception.SdkException;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;

import java.time.Duration;
import java.util.Base64;
import java.util.Map;

/**
 * This utility class provides helper methods for working with Amazon Simple
 Queue Service (Amazon SQS)
 * through the Java Message Service (JMS) interface. It contains common
 operations for managing message
 * queues and handling message delivery.
 */
public class SqsJmsExampleUtils {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SqsJmsExampleUtils.class);
    public static final Long QUEUE_VISIBILITY_TIMEOUT = 5L;

    /**
     * This method verifies that a message queue exists and creates it if
 necessary. The method checks for
     * an existing queue first to optimize performance.
     *
     * @param connection The active connection to the messaging service
     * @param queueName The name of the queue to verify or create
     * @param visibilityTimeout The duration in seconds that messages will be
 hidden after being received
     * @throws JMSException If there is an error accessing or creating the queue
     */
```

```
   public static void ensureQueueExists(SQSConnection connection, String
queueName, Long visibilityTimeout) throws JMSException {
       AmazonSQSMessagingClientWrapper client =
connection.getWrappedAmazonSQSClient();

      /* In most cases, you can do this with just a 'createQueue' call, but
'getQueueUrl'
      (called by 'queueExists') is a faster operation for the common case where
the queue
      already exists. Also, many users and roles have permission to call
'getQueueUrl'
      but don't have permission to call 'createQueue'.
      */
       if( !client.queueExists(queueName) ) {
           CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
                   .queueName(queueName)
                   .attributes(Map.of(QueueAttributeName.VISIBILITY_TIMEOUT,
String.valueOf(visibilityTimeout)))
                   .build();
           client.createQueue( createQueueRequest );
       }
   }

   /**
    * This method sends a simple text message to a specified message queue. It
handles all necessary
    * setup for the message delivery process.
    *
    * @param session The active messaging session used to create and send the
message
    * @param queueName The name of the queue where the message will be sent
    */
   public static void sendTextMessage(Session session, String queueName) {
       // Rest of implementation...

       try {
           MessageProducer producer =
session.createProducer( session.createQueue( queueName) );
           Message message = session.createTextMessage("Hello world!");
           producer.send(message);
       } catch (JMSException e) {
           LOGGER.error( "Error receiving from SQS", e );
       }
   }
```

```
   /**
    * This method processes incoming messages and logs their content based on
the message type.
    * It supports text messages, binary data, and Java objects.
    *
    * @param message The message to be processed and logged
    * @throws JMSException If there is an error reading the message content
    */
   public static void handleMessage(Message message) throws JMSException {
       // Rest of implementation...
       LOGGER.info( "Got message {}", message.getJMSMessageID() );
       LOGGER.info( "Content: ");
       if(message instanceof TextMessage txtMessage) {
           LOGGER.info( "\t{}", txtMessage.getText() );
       } else if(message instanceof BytesMessage byteMessage){
           // Assume the length fits in an int - SQS only supports sizes up to
256k so that
           // should be true
           byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
           byteMessage.readBytes(bytes);
           LOGGER.info( "\t{}", Base64.getEncoder().encodeToString( bytes ) );
       } else if( message instanceof ObjectMessage) {
           ObjectMessage objMessage = (ObjectMessage) message;
           LOGGER.info( "\t{}", objMessage.getObject() );
       }
   }

   /**
    * This method sets up automatic message processing for a specified queue. It
creates a listener
    * that will receive and handle incoming messages without blocking the main
program.
    *
    * @param session The active messaging session
    * @param queueName The name of the queue to monitor
    * @param connection The active connection to the messaging service
    */
   public static void receiveMessagesAsync(Session session, String queueName,
Connection connection) {
       // Rest of implementation...
       try {
           // Create a consumer for the queue.
```

```
            MessageConsumer consumer =
session.createConsumer(session.createQueue(queueName));
            // Provide an implementation of the MessageListener interface, which
has a single 'onMessage' method.
            // We use a lambda expression for the implementation.
            consumer.setMessageListener(message -> {
                try {
                    SqsJmsExampleUtils.handleMessage(message);
                    message.acknowledge();
                } catch (JMSException e) {
                    LOGGER.error("Error processing message: {}", e.getMessage());
                }
            });
            // Start receiving incoming messages.
            connection.start();
        } catch (JMSException e) {
            throw new RuntimeException(e);
        }
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }


    /**
     * This method performs cleanup operations after message processing is
complete. It receives
     * any messages in the specified queue, removes the message queue and closes
all
     * active connections to prevent resource leaks.
     *
     * @param queueName The name of the queue to be removed
     * @param visibilityTimeout The duration in seconds that messages are hidden
after being received
     * @throws JMSException If there is an error during the cleanup process
     */
    public static void cleanUpExample(String queueName, Long visibilityTimeout)
throws JMSException {
        LOGGER.info("Performing cleanup.");

        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
```

```
                SqsClient.create()
        );

        try (SQSConnection connection = connectionFactory.createConnection() ) {
            ensureQueueExists(connection, queueName, visibilityTimeout);
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            receiveMessagesAsync(session, queueName, connection);

            SqsClient sqsClient =
connection.getWrappedAmazonSQSClient().getAmazonSQSClient();
            try {
                String queueUrl = sqsClient.getQueueUrl(b ->
b.queueName(queueName)).queueUrl();
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue deleted: {}", queueUrl);
            } catch (SdkException e) {
                LOGGER.error("Error during SQS operations: ", e);
            }
        }
        LOGGER.info("Clean up: Connection closed");
    }

    /**
     * This method creates a background task that sends multiple messages to a
specified queue
     * after waiting for a set time period. The task operates independently to
ensure efficient
     * message processing without interrupting other operations.
     *
     * @param queueName The name of the queue where messages will be sent
     * @param secondsToWait The number of seconds to wait before sending messages
     * @param numMessages The number of messages to send
     * @param visibilityTimeout The duration in seconds that messages remain
hidden after being received
     * @return A task that can be executed to send the messages
     */
    public static Runnable sendAMessageAsync(String queueName, Long
secondsToWait, Integer numMessages, Long visibilityTimeout) {
        return () -> {
            try {
                Thread.sleep(Duration.ofSeconds(secondsToWait).toMillis());
            } catch (InterruptedException e) {
```

```
                Thread.currentThread().interrupt();
                throw new RuntimeException(e);
            }
            try {
                SQSConnectionFactory connectionFactory = new
 SQSConnectionFactory(
                        new ProviderConfiguration(),
                        SqsClient.create()
                );
                try (SQSConnection connection =
 connectionFactory.createConnection()) {
                    ensureQueueExists(connection, queueName, visibilityTimeout);
                    Session session = connection.createSession(false,
 Session.CLIENT_ACKNOWLEDGE);
                    for (int i = 1; i <= numMessages; i++) {
                        MessageProducer producer =
 session.createProducer(session.createQueue(queueName));
                        producer.send(session.createTextMessage("Hello World " +
 i + "!"));
                    }
                }
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        };
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - CreateQueue
  - DeleteQueue

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 사용하여 대기열 태그 및 Amazon SQS 작업

다음 코드 예제에서는 Amazon SQS에서 태그 지정 작업을 수행하는 방법을 보여줍니다.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

다음 예제에서는 대기열에 대한 태그를 생성하고, 태그를 나열하고, 태그를 제거합니다.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.ListQueueTagsResponse;
import software.amazon.awssdk.services.sqs.model.QueueDoesNotExistException;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.util.Map;
import java.util.UUID;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials. For more
 * information, see the <a href="https://docs.aws.amazon.com/sdk-for-java/latest/
developer-guide/get-started.html">AWS
 * SDK for Java Developer Guide</a>.
 */
public class TagExamples {
    static final SqsClient sqsClient = SqsClient.create();
    static final String queueName = "TagExamples-queue-" +
 UUID.randomUUID().toString().replace("-", "").substring(0, 20);
    private static final Logger LOGGER =
 LoggerFactory.getLogger(TagExamples.class);

    public static void main(String[] args) {
        final String queueUrl;
        try {
            queueUrl = sqsClient.createQueue(b ->
 b.queueName(queueName)).queueUrl();
```

```
            LOGGER.info("Queue created. The URL is: {}", queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because queue was not created.");
            throw new RuntimeException(e);
        }
        try {
            addTags(queueUrl);
            listTags(queueUrl);
            removeTags(queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because of an error in a method.");
        } finally {
            try {
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue successfully deleted. Program ending.");
                sqsClient.close();
            } catch (RuntimeException e) {
                LOGGER.error("Program ending.");
            } finally {
                sqsClient.close();
            }
        }
    }

    /** This method demonstrates how to use a Java Map to a tag a aueue.
     * @param queueUrl The URL of the queue to tag.
     */
    public static void addTags(String queueUrl) {
        // Build a map of the tags.
        final Map<String, String> tagsToAdd = Map.of(
                "Team", "Development",
                "Priority", "Beta",
                "Accounting ID", "456def");

        try {
            // Add tags to the queue using a Consumer<TagQueueRequest.Builder>
parameter.
            sqsClient.tagQueue(b -> b
                    .queueUrl(queueUrl)
                    .tags(tagsToAdd)
            );
        } catch (QueueDoesNotExistException e) {
            LOGGER.error("Queue does not exist: {}", e.getMessage(), e);
            throw new RuntimeException(e);
```

```
        }
    }

    /** This method demonstrates how to view the tags for a queue.
     * @param queueUrl The URL of the queue whose tags you want to list.
     */
    public static void listTags(String queueUrl) {
        ListQueueTagsResponse response;
        try {
            // Call the listQueueTags method with a
 Consumer<ListQueueTagsRequest.Builder> parameter that creates a
 ListQueueTagsRequest.
            response = sqsClient.listQueueTags(b -> b
                    .queueUrl(queueUrl));
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }

        // Log the tags.
        response.tags()
                .forEach((k, v) ->
                        LOGGER.info("Key: {} -> Value: {}", k, v));
    }

    /**
     * This method demonstrates how to remove tags from a queue.
     * @param queueUrl The URL of the queue whose tags you want to remove.
     */
    public static void removeTags(String queueUrl) {
        try {
            // Call the untagQueue method with a
 Consumer<UntagQueueRequest.Builder> parameter.
            sqsClient.untagQueue(b -> b
                    .queueUrl(queueUrl)
                    .tagKeys("Accounting ID") // Remove a single tag.
            );
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하세요.
  - ListQueueTags
  - TagQueue
  - UntagQueue

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon SQS용 서버리스 예제

다음 코드 예제에서는 Amazon SQS를 AWS SDKs와 함께 사용하는 방법을 보여줍니다.

예시

- Amazon SQS 트리거에서 간접적으로 Lambda 함수 간접 호출
- Amazon SQS 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

Amazon SQS 트리거에서 간접적으로 Lambda 함수 간접 호출

다음 코드 예제에서는 SQS 대기열에서 메시지를 받아 트리거된 이벤트를 수신하는 Lambda 함수를 구현하는 방법을 보여줍니다. 함수는 이벤트 파라미터에서 메시지를 검색하고 각 메시지의 내용을 로깅합니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

.NET을 사용하여 Lambda로 SQS 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Go를 사용하여 Lambda로 SQS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
```

```
 lambda.Start(handler)
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Java를 사용하여 Lambda로 SQS 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
```

```
        }

    }
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda로 SQS 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

TypeScript를 사용하여 Lambda로 SQS 이벤트 사용

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

PHP를 사용하여 Lambda로 SQS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity
```

```
use Bref\Context\Context;
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Python을 사용하여 Lambda로 SQS 이벤트를 사용합니다.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Ruby를 사용하여 Lambda로 SQS 이벤트를 사용합니다.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
```

```
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SQS 이벤트를 사용합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
 record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
```

```
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon SQS 트리거로 Lambda 함수에 대한 배치 항목 실패 보고

다음 코드 예제에서는 SQS 대기열에서 이벤트를 수신하는 Lambda 함수에 대한 부분 배치 응답을 구현하는 방법을 보여줍니다. 이 함수는 응답으로 배치 항목 실패를 보고하고 나중에 해당 메시지를 다시 시도하도록 Lambda에 신호를 보냅니다.

.NET

SDK for .NET

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

.NET을 사용하여 Lambda로 SQS 배치 항목 실패 보고

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
namespace sqsSample;
```

```
public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

Go

SDK for Go V2

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Go를 사용하여 Lambda로 SQS 배치 항목 실패 보고

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {

  if /* Your message processing condition here */ {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
```

```
 lambda.Start(handler)
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Java를 사용하여 Lambda로 SQS 배치 항목 실패 보고

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
 SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {

        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<SQSBatchResponse.BatchItemFailure>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(message.getMessageId()));
            }
```

```
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

JavaScript를 사용하여 Lambda에서 SQS 배치 항목 실패를 보고합니다.

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

TypeScript를 사용하여 Lambda로 SQS 배치 항목 실패를 보고합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
 from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

PHP

SDK for PHP

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

PHP를 사용하여 Lambda로 SQS 배치 항목 실패 보고

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php
```

```
use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Python을 사용하여 Lambda로 SQS 배치 항목 실패 보고

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                # process message
            except Exception as e:
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

Ruby

SDK for Ruby

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Ruby를 사용하여 Lambda로 SQS 배치 항목 실패를 보고합니다.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
  end
end
```

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. 서버리스 예제 리포지토리에서 전체 예제를 찾아보고 설정 및 실행 방법을 알아봅니다.

Rust를 사용하여 Lambda로 SQS 배치 항목 실패를 보고합니다.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
 Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
    run(service_fn(function_handler)).await
}
```

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK에서 Amazon SQS 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon SQS 문제 해결

이 주제에서는 Amazon SQS 콘솔, Amazon SQS API 또는 Amazon SQS와 함께 기타 도구를 사용할 때 발생할 수 있는 일반적인 오류 및 문제에 대한 문제 해결 조언을 제공합니다 Amazon SQS. 여기에 나열되지 않은 문제를 발견하는 경우 이 페이지의 [Feedback] 버튼을 사용하여 해당 문제를 보고할 수 있습니다.

문제 해결 조언과 일반적인 지원 질문에 대한 답변은 AWS 지식 센터를 참조하세요.

주제

- Amazon SQS의 액세스 거부 오류 문제 해결
- Amazon SQS API 오류 문제 해결
- Amazon SQS Dead Letter Queue(DLQ) 및 DLQ 리드라이브 문제 해결
- Amazon SQS의 FIFO 제한 문제 해결
- Amazon SQS ReceiveMessage API 직접 호출에 대해 반환되지 않은 메시지 문제 해결
- Amazon SQS 네트워크 오류 문제 해결
- AWS X-Ray를 사용한 Amazon Simple Queue Service 대기열 문제 해결

Amazon SQS의 액세스 거부 오류 문제 해결

다음 주제에서는 Amazon SQS API 직접 호출의 가장 일반적인 `AccessDenied`의 원인 또는 `AccessDeniedException` 오류를 다룹니다. 이러한 오류를 해결하는 방법에 대한 자세한 내용은 AWS 지식 센터 안내서의 Amazon SQS API 직접 호출에서 “AccessDenied” 또는 “AccessDeniedException” 오류를 해결하려면 어떻게 해야 하나요?를 참조하세요.

오류 메시지 예제:

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to
        the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

– 또는 -

```
An error occurred (KMS.AccessDeniedException) when calling the SendMessage
        operation: User: arn:aws:iam::xxxxx:user/xxxx is not authorized to perform:
        kms:GenerateDataKey on resource: arn:aws:kms:us-east-1:xxxx:key/xxxx with an
 explicit
```

```
        deny.
```

Amazon SQS 대기열 정책 및 IAM 정책

요청자에게 Amazon SQS 작업을 수행할 수 있는 적절한 권한이 있는지 확인하려면 다음을 수행합니다.

- Amazon SQS API 직접 호출을 수행하는 IAM 보안 주체를 식별합니다. IAM 보안 주체가 동일한 계정에서 생성된 경우 Amazon SQS 대기열 정책 또는 AWS Identity and Access Management(AWS IAM) 정책에 작업에 대한 액세스를 명시적으로 허용하는 권한이 포함되어야 합니다.
- 보안 주체가 IAM 엔터티인 경우:
  - AWS Management Console오른쪽 상단을 확인하거나 `aws sts get-caller-identity` 명령을 사용하여 IAM 사용자 또는 역할을 식별할 수 있습니다.
  - 요청을 수행하는 IAM 사용자 또는 역할과 관련된 IAM 정책을 확인합니다. 다음 방법 중 하나를 사용할 수 있습니다.
    - IAM 정책 시뮬레이터로 IAM 정책을 테스트합니다.
    - 다양한 IAM 정책 유형을 검토합니다.
  - 필요한 경우 IAM 사용자 정책을 편집합니다.
  - 대기열 정책을 확인하고 필요한 경우 편집합니다.
- 보안 주체가 AWS 서비스인 경우 Amazon SQS 대기열 정책에서 액세스를 명시적으로 허용해야 합니다.
- 보안 주체가 교차 계정 보안 주체인 경우 Amazon SQS 대기열 정책과 IAM 정책 모두 액세스를 명시적으로 허용해야 합니다.
- 정책이 조건 요소를 사용하는 경우 조건이 액세스를 제한하는지 확인합니다.

> **Important**
>
> 두 정책 중 명시적 거부는 명시적 허용을 재정의합니다. 다음과 같이 Amazon SQS 정책의 기본 예제를 소개합니다.

AWS Key Management Service 권한

Amazon SQS 대기열에 고객 관리형 서버 측 암호화(SSE)가 켜져 있는 AWS KMS key경우 생산자와 소비자 모두에게 권한을 부여해야 합니다. 대기열이 암호화되었는지 확인하려면

GetQueueAttributes API KmsMasterKeyId 속성을 사용하거나 암호화의 대기열 콘솔에서 확인할 수 있습니다.

- 생산자에게 필요한 권한:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

- 소비자에 필요한 권한:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- 교차 계정 액세스에 필요한 권한:

```
{
"Effect": "Allow",
"Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

다음 옵션 중 하나를 선택하여 Amazon SQS 대기열에 대한 암호화를 활성화합니다.

- SSE-Amazon SQS(Amazon SQS 서비스에서 생성 및 관리하는 암호화 키)
- AWS 관리형 기본 키 (alias/aws/sqs)

- 고객 관리형 키

그러나 AWS관리형 KMS 키를 사용하는 경우 기본 키 정책을 수정할 수 없습니다. 따라서 다른 서비스 및 교차 계정에 대한 액세스를 제공하려면 고객 관리형 키를 사용합니다. 이렇게 하면 키 정책을 편집할 수 있습니다.

VPC 엔드포인트 정책

Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트를 통해 Amazon SQS에 액세스하는 경우 Amazon SQS VPC 엔드포인트 정책에서 액세스를 허용해야 합니다. Amazon SQS에 대한 Amazon VPC 엔드포인트 정책을 생성하여 다음을 지정할 수 있습니다.

1. 작업을 수행할 수 있는 위탁자.
2. 수행할 수 있는 작업.
3. 작업을 수행할 수 있는 리소스.

다음 예제에서 VPC 엔드포인트 정책은 IAM 사용자 *MyUser*가 Amazon SQS 대기열 *MyQueue*에 메시지를 전송할 수 있도록 지정합니다. 다른 작업, IAM 사용자 및 Amazon SQS 리소스는 VPC 엔드포인트를 통한 액세스가 거부됩니다.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

조직 서비스 제어 정책

가 조직에 AWS 계정 속한 경우 AWS Organizations 정책은 사용자가 Amazon SQS 대기열에 액세스하지 못하도록 차단할 수 있습니다. 기본적으로 AWS Organizations 정책은 Amazon SQS에 대한 요청을 차단하지 않습니다. 그러나 Amazon SQS 대기열에 대한 액세스를 차단하도록 AWS Organizations 정책이 구성되지 않았는지 확인합니다. AWS Organizations 정책을 확인하는 방법에 대한 지침은 AWS Organizations 사용 설명서의 모든 정책 나열을 참조하세요.

Amazon SQS API 오류 문제 해결

다음 주제에서는 Amazon SQS API 직접 호출 시 반환되는 가장 일반적인 오류와 문제를 해결하는 방법을 다룹니다.

QueueDoesNotExist 오류

이 오류는 Amazon SQS 서비스가 Amazon SQS 작업에 대해 언급된 대기열을 찾을 수 없을 때 반환됩니다.

가능한 원인 및 완화 조치:

- 잘못된 리전: Amazon SQS 클라이언트 구성을 검토하여 클라이언트에서 올바른 리전을 구성했는지 확인합니다. 클라이언트에서 리전을 구성하지 않으면 SDK 또는가 구성 파일 또는 환경 변수에서 리전을 AWS CLI 선택합니다. SDK가 구성 파일에서 리전을 찾지 못하면 SDK는 기본적으로 리전을 us-east-1로 설정합니다.
- 대기열이 최근에 삭제되었을 수 있음: API 직접 호출이 이루어지기 전에 대기열이 삭제된 경우 API 직접 호출이 이 오류를 반환합니다. 오류가 발생하기 전에 CloudTrail에서 DeleteQueue 작업을 확인합니다.
- 권한 문제: 요청하는 AWS Identity and Access Management (IAM) 사용자 또는 역할에 필요한 권한이 없는 경우 다음 오류가 발생할 수 있습니다.

  ```
  The specified queue does not exist or you do not have access to it.
  ```

  권한을 확인하고 올바른 권한으로 API를 직접 호출합니다.

`QueueDoesNotExist` 오류 문제 해결에 대한 자세한 내용은 AWS 지식 센터 안내서의 Amazon SQS 대기열에 대한 API 직접 호출 시 QueueDoesNotExist 오류를 해결하려면 어떻게 해야 하나요?를 참조하세요.

InvalidAttributeValue 오류

이 오류는 Amazon SQS 대기열 리소스 정책 또는 정책이나 보안 주체가 잘못된 속성을 업데이트하면 반환됩니다.

가능한 원인 및 완화 조치:

- 잘못된 리소스 정책: 리소스 정책에 모든 필수 필드가 있는지 확인합니다. 자세한 내용은 IAM JSON 정책 요소 참조 및 IAM 정책 검증을 참조하세요. IAM 정책 생성기를 사용하여 Amazon SQS 리소스 정책을 생성하고 테스트할 수도 있습니다. 정책이 JSON 형식인지 확인합니다.
- 잘못된 보안 주체: `Principal` 요소가 리소스 정책에 존재하고 값이 유효한지 확인합니다. Amazon SQS 리소스 정책 `Principal` 요소에 IAM 엔터티가 포함된 경우 정책을 사용하기 전에 엔터티가 존재하는지 확인합니다. Amazon SQS는 리소스 정책을 검증하고 IAM 엔터티를 확인합니다. IAM 엔터티가 없으면 오류가 발생합니다. IAM 엔터티를 확인하려면 `GetRole` 및 `GetUser` API를 사용합니다.

`InvalidAttributeValue` 오류 해결 방법에 대한 자세한 내용은 AWS 지식 센터 안내서의 Amazon SQS 대기열에 대한 API 직접 호출 시 QueueDoesNotExist 오류를 해결하려면 어떻게 해야 하나요?를 참조하세요.

ReceiptHandle 오류

`DeleteMessage` API를 직접 호출하면 수신 핸들이 잘못되었거나 만료된 경우 `ReceiptHandleIsInvalid` 또는 `InvalidParameterValue` 오류가 반환될 수 있습니다.

- ReceiptHandleIsInvalid 오류: 수신 핸들이 잘못된 경우 이 예와 유사한 오류가 발생합니다.

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
 The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- InvalidParameterValue 오류: 수신 핸들이 만료되면 이 예와 유사한 오류가 발생합니다.

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:
 Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The
 receipt handle has expired.
```

가능한 원인 및 완화 조치:

수신 핸들은 수신된 모든 메시지에 대해 생성되며 표시 제한 시간에만 유효합니다. 표시 제한 시간이 만료되면 메시지가 소비자 대기열에 표시됩니다. 소비자로부터 메시지를 다시 받으면 새 수신 핸들이 수신됩니다. 잘못되거나 만료된 수신 핸들 오류를 방지하려면 올바른 수신 핸들을 사용하여 Amazon SQS 대기열 표시 제한 시간 내에 메시지를 삭제합니다.

`ReceiptHandle` 오류 해결 방법에 대한 자세한 내용은 AWS 지식 센터 안내서의 Amazon SQS DeleteMessage API 직접 호출을 사용할 때 'ReceiptHandleIsInvalid' 및 'InvalidParameterValue' 오류를 해결하려면 어떻게 해야 하나요?를 참조하세요.

Amazon SQS Dead Letter Queue(DLQ) 및 DLQ 리드라이브 문제 해결

다음 주제에서는 Amazon SQS DLQ 및 DLQ 리드라이브 문제의 가장 일반적인 원인과 문제를 해결하는 방법을 다룹니다. 자세한 내용은 AWS 지식 센터 가이드의 Amazon SQS DLQ 리드라이브 문제를 해결하려면 어떻게 해야 하나요?를 참조하세요.

DLQ 문제

일반적인 DLQ 문제와 이를 해결하는 방법에 대해 알아봅니다.

콘솔을 사용하여 메시지를 보면 메시지가 DLQ(Dead Letter Queue)로 이동할 수 있음

Amazon SQS는 콘솔에서 메시지를 보는 것을 해당 대기열의 리드라이브 정책을 위반하는 것으로 간주합니다. 따라서 콘솔에서 해당 대기열의 리드라이브 정책에 지정된 횟수로 메시지를 보는 경우 이 메시지는 해당 대기열의 DLQ(Dead Letter Queue)로 이동됩니다.

이 동작을 조정하려면 다음 중 하나를 수행할 수 있습니다.

- 해당 대기열의 리드라이브 정책에 맞게 [Maximum Receives] 설정을 늘립니다.
- 콘솔에서 해당 대기열의 메시지를 보는 것을 피합니다.

배달 못한 편지 대기열에 대한 **`NumberOfMessagesSent`** 및 **`NumberOfMessagesReceived`**가 일치하지 않음

메시지를 배달 못한 편지 대기열에 수동으로 전송하는 경우, 이것은 `NumberOfMessagesSent` 측정치로 캡처됩니다. 그러나 처리 실패로 인해 메시지가 배달 못한 편지 대기열로 전송되는 경우에는 이 지표로 캡처되지 않습니다. 따라서 `NumberOfMessagesSent` 및 `NumberOfMessagesReceived` 값이 다를 수 있습니다.

배달 못한 편지 대기열 리드라이브 생성 및 구성

Dead Letter Queue(DLQ) 리드라이브를 사용하려면 Amazon SQS가 Dead Letter Queue(DLQ)에서 메시지를 수신하고 대상 대기열로 메시지를 보낼 수 있도록 적절한 권한을 설정해야 합니다. 올바른 권한

이 없는 경우 Dead Letter Queue(DLQ) 리드라이브 작업이 실패할 수 있습니다. 메시지 리드라이브 작업의 상태를 보고 문제를 해결하고 다시 시도할 수 있습니다.

표준 및 FIFO 대기열 메시지 실패 처리

표준 대기열은 보존 기간이 만료될 때까지 메시지를 계속 처리합니다. 이 연속 처리는 사용하지 않는 메시지로 인해 대기열이 차단될 가능성을 최소화합니다. 소비자가 반복적으로 많은 수의 메시지 삭제에 실패하면 비용이 증가하고 하드웨어의 부하가 늘어날 수 있습니다. 비용을 줄이려면 실패한 메시지를 Dead Letter Queue(DLQ)로 이동합니다.

표준 대기열에도 처리 중 메시지가 많을 수 있습니다. 대부분의 메시지를 소비하지 않고 Dead Letter Queue(DLQ)로 보내지도 않는다면 메시지의 처리 속도가 느려질 수 있습니다. 대기열의 효율성을 유지하려면 애플리케이션이 메시지 처리를 올바르게 수행하는지 확인하세요.

FIFO 대기열은 메시지 그룹의 메시지를 순서에 따라 소비함으로써 한 번만 처리되도록 합니다. 이렇게 하면 대기열을 가로막고 있는 메시지가 제대로 처리되거나 Dead Letter Queue(DLQ)로 이동될 때까지 첫 번째 메시지 그룹은 사용할 수 없는 상태가 되지만, 소비자는 다른 메시지 그룹으로부터 정렬된 메시지를 계속 가져올 수 있습니다.

또한 FIFO 대기열을 사용하면 처리 중 메시지의 수를 줄일 수 있습니다. 메시지로 인해 FIFO 대기열이 차단되는 것을 방지하려면 애플리케이션이 메시지 처리를 올바르게 수행하는지 확인하세요.

자세한 내용은 Amazon SQS 메시지 할당량 및 Amazon SQS 모범 사례 섹션을 참조하세요.

DLQ 리드라이브 문제

일반적인 DLQ 리드라이브 문제와 이를 해결하는 방법에 대해 알아봅니다.

AccessDenied 권한 문제

AWS Identity and Access Management (IAM) 엔터티에 필요한 권한이 없기 때문에 DLQ 리드라이브가 실패할 때 `AccessDenied` 오류가 발생합니다.

오류 메시지 예:

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

DLQ 리드라이브 요청을 수행하려면 다음 API 권한이 필요합니다.

메시지 리드라이브를 시작하려면:

- 배달 못한 편지 대기열 권한:

  - `sqs:StartMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt` - 배달 못한 편지 대기열 또는 원본 소스 대기열이 암호화되는 경우.
- 대상 대기열 권한:
  - `sqs:SendMessage`
  - `kms:GenerateDataKey` - 대상 대기열이 암호화되는 경우.
  - `kms:Decrypt` - 대상 대기열이 암호화되는 경우.

진행 중인 메시지 리드라이브를 취소하려면:

- 배달 못한 편지 대기열 권한:
  - `sqs:CancelMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt` - 배달 못한 편지 대기열 또는 원본 소스 대기열이 암호화되는 경우.

메시지 이동 상태를 표시하려면:

- 배달 못한 편지 대기열 권한:
  - `sqs:ListMessageMoveTasks`
  - `sqs:GetQueueAttributes`

`NonExistentQueue` 오류

`NonExistentQueue` 오류는 Amazon SQS 소스 대기열이 없거나 삭제되었을 때 발생합니다. 확인 후 현재 있는 Amazon SQS 대기열로 다시 이동합니다.

오류 메시지 예:

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource 오류

다음 시나리오로 DLQ 리드라이브를 시작하려고 하면 `CouldNotDetermineMessageSource` 오류가 발생합니다.

- `SendMessage` API를 사용하여 DLQ로 직접 전송되는 Amazon SQS 메시지입니다.
- DLQ가 구성된 Amazon Simple Notification Service(Amazon SNS) 주제 또는 AWS Lambda 함수의 메시지입니다.

이 오류를 해결하려면 리드라이브를 시작할 때 사용자 지정 대상으로 리드라이브를 선택합니다. 그런 다음 Amazon SQS 대기열 ARN을 입력하여 모든 메시지를 DLQ에서 대상 대기열로 이동합니다.

오류 메시지 예:

```
Failed: CouldNotDetermineMessageSource
```

Amazon SQS의 FIFO 제한 문제 해결

기본적으로 FIFO 대기열은 `SendMessage`, `ReceiveMessage` 및 `DeleteMessage` API 작업별 초당 300개의 트랜잭션을 지원합니다. 300TPS 이상의 요청은 대기열의 메시지를 사용할 수 있더라도 `ThrottlingException` 오류가 발생합니다. 이를 완화하기 위해 다음 방법을 사용할 수 있습니다.

- Amazon SQS에서 FIFO 대기열의 높은 처리량 활성화.
- Amazon SQS API 배치 작업 `SendMessageBatch`, `DeleteMessageBatch` 및 `ChangeMessageVisibilityBatch`를 사용하여 API 작업별 초당 최대 3,000개의 메시지로 TPS 제한을 늘리고 비용을 절감할 수 있습니다. `ReceiveMessage` API의 경우 트랜잭션당 최대 10개의 메시지를 수신하도록 `MaxNumberofMessages` 파라미터를 설정합니다. 자세한 내용은 Amazon SQS 배치 작업 단원을 참조하십시오.
- 처리량이 높은 FIFO 대기열의 경우 권장 사항에 따라 파티션 사용률을 최적화합니다. 동일한 메시지 그룹 ID를 가진 메시지를 일괄로 전송합니다. 메시지를 삭제하거나 동일한 `ReceiveMessage` API 요청의 수신 핸들이 있는 배치에서 메시지 표시 제한 시간 값을 변경합니다.
- 고유 `MessageGroupId` 값 수를 늘립니다. 이를 통해 FIFO 대기열 파티션 간에 균등하게 배포할 수 있습니다. 자세한 내용은 Amazon SQS 메시지 그룹 ID 사용을 참조하세요.

자세한 내용은 AWS 지식 센터 가이드의 Amazon SQS FIFO 대기열이 다른 메시지 그룹의 모든 메시지 또는 메시지를 반환하지 않는 이유는 무엇인가요?를 참조하세요.

Amazon SQS ReceiveMessage API 직접 호출에 대해 반환되지 않은 메시지 문제 해결

다음 주제에서는 Amazon SQS 메시지가 소비자에게 반환되지 않는 가장 일반적인 원인과 문제를 해결하는 방법을 다룹니다. 자세한 내용은 AWS 지식 센터 안내서의 Amazon SQS 대기열에서 메시지를 수신할 수 없는 이유를 참조하세요.

빈 대기열

대기열이 비어 있는지 확인하려면 긴 폴링을 사용하여 `ReceiveMessage` API를 직접 호출합니다. `ApproximateNumberOfMessagesVisible`, `ApproximateNumberOfMessagesNotVisible` 및 `ApproximateNumberOfMessagesDelayed` CloudWatch 지표를 사용할 수도 있습니다. 모든 지표 값이 몇 분 동안 0으로 설정된 경우 대기열은 비어 있는 것으로 간주됩니다.

처리 중 한도에 도달

긴 폴링을 사용하고 대기열의 처리 중 한도(기본적으로 FIFO의 경우 20,000, 표준의 경우 120,000)가 위반되는 경우 Amazon SQS는 할당량 한도를 초과하는 오류 메시지를 반환하지 않습니다.

메시지 지연

Amazon SQS 대기열이 지연 대기열로 구성되거나 메시지 타이머와 함께 메시지가 전송된 경우 지연 시간이 끝날 때까지 메시지가 표시되지 않습니다. 대기열이 지연 대기열로 구성되어 있는지 확인하려면 `GetQueueAttributes` API `DelaySeconds` 속성을 사용하거나 전송 지연의 대기열 콘솔에서 사용합니다. `ApproximateNumberOfMessagesDelayed` CloudWatch 지표를 확인하여 메시지가 지연되는지 확인합니다.

메시지 처리 중

다른 소비자가 메시지를 폴링한 경우 메시지는 표시 제한 시간 동안 처리 중이거나 보이지 않습니다. 추가 폴링은 빈 수신을 반환할 수 있습니다. ApproximateNumberOfMessagesVisible CloudWatch 지표를 확인하여 수신 가능한 메시지 수를 파악합니다. FIFO 대기열의 경우 메시지 그룹 ID가 있는 메시지가 처리 중이면 메시지를 삭제하거나 메시지가 표시되지 않는 한 더 이상 메시지가 반환되지 않습니다. 이는 메시지 순서가 FIFO 대기열의 메시지 그룹 수준에서 유지되기 때문입니다.

폴링 메서드

짧은 폴링을 사용하는 경우(WaitTimeSeconds는 0) Amazon SQS는 서버의 하위 집합을 샘플링하고 해당 서버의 메시지만 반환합니다. 따라서 메시지를 수신할 수 있는 상태에서도 메시지를 받지 못할 수 있습니다. 후속 폴링 요청은 메시지를 반환합니다.

긴 폴링을 사용하는 경우 Amazon SQS는 모든 서버를 폴링하고 사용 가능한 메시지를 하나 이상 수집한 후 지정된 최대 수까지 응답을 전송합니다. ReceiveMessage WaitTimeSeconds 값이 너무 낮으면 사용 가능한 메시지가 모두 수신되지 않을 수 있습니다.

Amazon SQS 네트워크 오류 문제 해결

다음 주제에서는 Amazon SQS 네트워크 문제의 가장 일반적인 원인과 문제를 해결하는 방법을 다룹니다.

ETIMEOUT error

클라이언트가 Amazon SQS 엔드포인트에 대한 TCP 연결을 설정할 수 없는 경우 ETIMEOUT 오류가 발생합니다.

문제 해결:

- 네트워크 연결 확인

  `telnet`과 같은 명령을 실행하여 Amazon SQS에 대한 네트워크 연결을 테스트합니다.

  Example: telnet sqs.us-east-1.amazonaws.com 443
- 네트워크 설정 확인
  - 로컬 방화벽 규칙, 경로 및 액세스 제어 목록(ACL)에서 사용하는 포트의 트래픽을 허용하는지 확인합니다.
  - 보안 그룹 아웃바운드(송신) 규칙은 포트 80 또는 443에 대한 트래픽을 허용해야 합니다.
  - 네트워크 ACL 아웃바운드(송신) 규칙은 TCP 포트 80 또는 443에 대한 트래픽을 허용해야 합니다.
  - 네트워크 ACL 인바운드(수신) 규칙은 TCP 포트 1024-65535의 트래픽을 허용해야 합니다.
  - 퍼블릭 인터넷에 연결하는 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스에는 인터넷 연결이 있어야 합니다.
- Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트

Amazon VPC 엔드포인트를 통해 Amazon SQS에 액세스하는 경우 엔드포인트 보안 그룹은 포트 443에서 클라이언트 보안 그룹에 대한 인바운드 트래픽을 허용해야 합니다. VPC 엔드포인트의 서브넷과 연결된 네트워크 ACL에는 다음 구성이 있어야 합니다.

- 네트워크 ACL 아웃바운드(송신) 규칙은 TCP 포트 1024-65535(휘발성 포트)의 트래픽을 허용해야 합니다.
- 네트워크 ACL 인바운드(수신) 규칙은 포트 443의 트래픽을 허용해야 합니다.

또한 Amazon SQS VPC 엔드포인트 AWS Identity and Access Management (IAM) 정책은 액세스를 허용해야 합니다. 다음 예제에서 VPC 엔드포인트 정책은 IAM 사용자 *MyUser*가 Amazon SQS 대기열 *MyQueue*에 메시지를 전송할 수 있도록 지정합니다. 다른 작업, IAM 사용자 및 Amazon SQS 리소스는 VPC 엔드포인트를 통한 액세스가 거부됩니다.

```
{
    "Statement": [{
        "Action": ["sqs:SendMessage"],
        "Effect": "Allow",
        "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
        "Principal": {
            "AWS": "arn:aws:iam:123456789012:user/MyUser"
        }
    }]
}
```

UnknownHostException error

UnknownHostException 오류는 호스트 IP 주소를 확인할 수 없을 때 발생합니다.

문제 해결:

nslookup 유틸리티를 사용하여 호스트 이름과 연결된 IP 주소를 반환합니다.

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI 또는 SDK for Python 레거시 엔드포인트:

```
nslookup <region>.queue.amazonaws.com
```

실패한 출력을 받은 경우 AWS 지식 센터 가이드의 DNS는 어떻게 작동하며 부분 또는 간헐적 DNS 실패 문제를 해결하려면 어떻게 해야 하나요?의 지침을 따르세요.

유효한 출력을 받은 경우 애플리케이션 수준 문제일 수 있습니다. 애플리케이션 수준 문제를 해결하려면 다음 방법을 시도해 보세요.

- 애플리케이션을 다시 시작합니다.
- Java 애플리케이션에 잘못된 DNS 캐시가 없는지 확인합니다. 가능하면 DNS TTL을 준수하도록 애플리케이션을 구성합니다. 자세한 내용은 DNS 이름 조회를 위한 JVM TTL 설정을 참조하세요.

네트워크 오류 해결 방법에 대한 자세한 내용은 AWS 지식 센터 안내서의 Amazon SQS 'ETIMEOUT' 및 'UnknownHostException' 연결 오류 문제를 해결하려면 어떻게 해야 하나요?를 참조하세요.

AWS X-Ray를 사용한 Amazon Simple Queue Service 대기열 문제 해결

AWS X-Ray 는 애플리케이션이 처리하는 요청에 대한 데이터를 수집하고 데이터를 보고 필터링하여 잠재적 문제와 최적화 기회를 식별할 수 있습니다. 애플리케이션에 대한 추적된 요청의 경우 요청, 응답 및 애플리케이션이 다운스트림 AWS 리소스, 마이크로서비스, 데이터베이스 및 HTTP 웹 APIs.

Amazon SQS를 통해 AWS X-Ray 추적 헤더를 전송하려면 다음 중 하나를 수행할 수 있습니다.

- `X-Amzn-Trace-Id` 트레이스 헤더를 사용합니다.
- `AWSTraceHeader` 메시지 시스템 속성을 사용합니다.

오류 및 지연 시간에 대한 데이터를 수집하려면 AWS X-Ray SDK를 사용하여 `AmazonSQS` 클라이언트를 계측해야 합니다.

AWS X-Ray 콘솔을 사용하여 Amazon SQS와 애플리케이션이 사용하는 다른 서비스 간의 연결 맵을 볼 수 있습니다. 또한 콘솔을 사용하여 평균 대기 시간 및 실패율과 같은 메트릭을 볼 수 있습니다. 자세한 정보는 AWS X-Ray 개발자 안내서의 Amazon SQS 및 AWS X-Ray를 참조하세요.

Amazon SQS의 보안

이 섹션에서는 Amazon SQS 보안, 인증 및 액세스 제어, Amazon SQS 액세스 정책 언어에 대한 정보를 제공합니다.

주제

- Amazon SQS의 데이터 보호
- Amazon SQS의 Identity and Access Management
- Amazon SQS의 로깅 및 모니터링
- Amazon SQS에 대한 규정 준수 확인
- Amazon SQS의 복원성
- Amazon SQS의 인프라 보안
- Amazon SQS 보안 모범 사례

Amazon SQS의 데이터 보호

AWS 공동 책임 모델 Amazon Simple Queue Service의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- AWS 암호화 솔루션과 내부의 모든 기본 보안 제어를 사용합니다 AWS 서비스.

- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 Amazon SQS 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

다음 단원에서는 Amazon SQS의 데이터 보호에 대해 설명합니다.

Amazon SQS의 데이터 암호화

데이터 보호란 전송 중(Amazon SQS 안팎으로 데이터가 이동 중)과 유휴 시(Amazon SQS 데이터 센터의 디스크에 데이터가 저장된 동안) 데이터를 보호하는 것을 말합니다. Secure Sockets Layer(SSL) 또는 클라이언트 측 암호화를 사용하여 전송 중인 데이터를 보호할 수 있습니다. 기본적으로 Amazon SQS는 디스크 암호화를 사용하여 메시지와 파일을 저장합니다. 메시지를 데이터 센터의 암호화된 파일 시스템에 저장하기 전에 먼저 메시지를 암호화하도록 Amazon SQS에 요청하여 저장 데이터를 보호할 수 있습니다. Amazon SQS에서는 최적화된 데이터 암호화를 위해 SSE 사용을 권장합니다.

주제

- Amazon SQS에서 저장 시 암호화
- Amazon SQS 키 관리

Amazon SQS에서 저장 시 암호화

서버 측 암호화(SSE)를 사용하면 암호화된 대기열에서 민감한 데이터를 전송할 수 있습니다. SSE는 SQS 관리형 암호화 키(SSE-SQS) 또는 AWS Key Management Service (SSE-KMS)에서 관리되는 키를 사용하여 대기열의 메시지 내용을 보호합니다. 를 사용한 SSE 관리에 대한 자세한 내용은 다음을 AWS Management Console참조하세요.

- 대기열에 대해 SSE-SQS 구성(콘솔)
- 대기열에 대해 SSE-KMS 구성(콘솔)

AWS SDK for Java (및 `CreateQueue`, `SetQueueAttributes`및 `GetQueueAttributes` 작업)을 사용하여 SSE를 관리하는 방법에 대한 자세한 내용은 다음 예제를 참조하세요.

- Amazon SQS 대기열에서 서버 측 암호화 사용
- 에 대한 KMS 권한 구성 AWS 서비스

Amazon SQS가 메시지를 수신하면 SSE가 메시지를 즉시 암호화합니다. 메시지는 암호화된 형식으로 저장되며 Amazon SQS는 권한 있는 소비자에게 전송된 경우에만 메시지를 해독합니다.

> **Important**
>
> SSE를 사용할 수 있는 대기열에 대한 모든 요청에서는 HTTPS 및 서명 버전 4를 사용해야 합니다.
> 기본 키(Amazon SQS용AWS 관리형 KMS 키)를 사용하는 암호화된 대기열은 다른에서 Lambda 함수를 호출할 수 없습니다 AWS 계정. Amazon SQS
> AWS Security Token Service `AssumeRole` 작업을 사용하여 Amazon SQS에 알림을 보낼 수 있는 AWS 서비스의 일부 기능은 SSE와 호환되지만 표준 대기열에서만 작동합니다.
>
> - Auto Scaling 수명 주기 후크
> - AWS Lambda 배달 못한 편지 대기열
>
> 기타 서비스의 암호화 대기열과의 호환성에 관한 정보는 AWS 서비스에 대한 KMS 권한 구성 및 서비스 설명서를 참조하십시오.

AWS KMS 는 안전하고 가용성이 높은 하드웨어와 소프트웨어를 결합하여 클라우드에 맞게 확장된 키 관리 시스템을 제공합니다. Amazon SQS AWS KMS를와 함께 사용하면 메시지 데이터를 암호화하는 데이터 키도 암호화되고 보호되는 데이터와 함께 저장됩니다.

AWS KMS를 사용하면 다음과 같은 이점이 있습니다.

- AWS KMS keys를 직접 생성하고 관리할 수 있습니다.
- 각 계정 및 리전에 고유한 Amazon SQS용 AWS 관리형 KMS 키를 사용할 수도 있습니다.
- AWS KMS 보안 표준은 암호화 관련 규정 준수 요구 사항을 충족하는 데 도움이 될 수 있습니다.

자세한 정보는 AWS Key Management Service개발자 안내서.의 AWS Key Management Service 란 무엇입니까?를 참조하세요.

암호화 범위

SSE는 Amazon SQS 대기열에서 메시지의 본문을 암호화합니다.

SSE는 다음을 암호화하지 않습니다.

- 대기열 메타데이터(대기열 이름과 속성)
- 메시지 메타데이터(메시지 ID, 타임스탬프 및 속성)
- 대기열당 측정치

메시지를 암호화하면 권한이 없는 사용자나 익명 사용자가 해당 내용을 사용할 수 없게 됩니다. SSE를 활성화하면 암호화된 대기열에 대한 익명 `SendMessage` 및 `ReceiveMessage` 요청이 거부됩니다. Amazon SQS 보안 모범 사례에서는 익명 요청을 사용하지 말 것을 권장합니다. Amazon SQS 대기열로 익명 요청을 보내려면 SSE를 비활성화해야 합니다. 다음과 같은 Amazon SQS의 정상 작동에는 영향을 주지 않습니다.

- 대기열 암호화가 활성화된 후 전송되는 경우에만 메시지가 암호화됩니다. Amazon SQS는 백로그된 메시지를 암호화하지 않습니다.
- 암호화된 메시지는 해당 대기열 암호화가 비활성화된 경우에만 암호화된 상태를 유지합니다.

메시지를 배달 못한 편지 대기열로 이동하는 것은 다음과 같이 해당 암호화에 영향을 주지 않습니다.

- Amazon SQS가 암호화된 소스 대기열에서 암호화되지 않은 DLQ(Dead Letter Queue)로 메시지를 이동하는 경우, 메시지는 암호화된 상태를 유지합니다.
- Amazon SQS가 암호화되지 않은 소스 대기열에서 암호화된 DLQ(Dead Letter Queue)로 메시지를 이동하는 경우, 메시지는 암호화되지 않은 상태를 유지합니다.

주요 용어

다음 주요 용어 설명은 SSE 기능을 보다 정확하게 이해하는 데 도움이 될 수 있습니다. 자세한 내용은 Amazon Simple Queue Service API 참조를 참조하세요.

데이터 키

Amazon SQS 메시지 내용의 암호화를 담당하는 키(DEK)입니다.

자세한 내용은 AWS Encryption SDK 개발자 안내서의 AWS Key Management Service 개발자 안내서에서 데이터 키를 참조하세요.

데이터 키 재사용 기간

Amazon SQS가 AWS KMS 다시 호출하기 전에 데이터 키를 재사용하여 메시지를 암호화하거나 복호화할 수 있는 초 단위의 시간 길이입니다. 초 단위의 정수로, 60초(1분)에서 86,400초(24시간) 사이입니다. 기본값은 300(5분)입니다. 자세한 내용은 데이터 키 재사용 기간 이해 단원을 참조하십시오.

> **Note**
>
> 드물지만 연결할 수 없는 경우 AWS KMS Amazon SQS는 연결이 다시 설정될 때까지 캐시된 데이터 키를 계속 사용합니다.

KMS 키 ID

계정 또는 다른 계정에서 AWS 관리형 KMS 키 또는 사용자 지정 KMS 키의 별칭, 별칭 ARN, 키 ID 또는 키 ARN입니다. Amazon SQS에 대한 AWS 관리형 KMS 키의 별칭은 항상 이지만 사용자 지정 KMS 키의 `alias/aws/sqs`별칭은 예를 들어 일 수 있습니다`alias/MyAlias`. 이러한 KMS 키를 사용하여 Amazon SQS 대기열의 메시지를 보호할 수 있습니다.

> **Note**
>
> 다음 사항에 유의하세요:
>
> - 사용자 지정 KMS 키를 지정하지 않으면 Amazon SQS는 Amazon SQS에 관리 AWS 형 KMS 키를 사용합니다.
> - AWS Management Console 를 사용하여 대기열 AWS KMS 에 대한 Amazon SQS의 AWS 관리형 KMS 키를 처음 지정하면가 Amazon SQS의 AWS 관리형 KMS 키를 생성합니다.
> - 또는 SSE가 활성화된 대기열에서 `SendMessage` 또는 `SendMessageBatch` 작업을 처음 사용할 때 Amazon SQS에 대한 AWS 관리형 KMS 키를 AWS KMS 생성합니다.

KMS 키를 생성하고, KMS 키 사용 방법을 제어하는 정책을 정의하고, 콘솔의 고객 관리형 키 섹션 또는 `CreateKey` AWS KMS 작업을 사용하여 KMS 키 사용을 감사할 AWS KMS 수 있습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 KMS 키 및 키 생성을 참조하세요.

KMS 키 식별자의 더 많은 예는 AWS Key Management Service API 참조의 KeyId를 참조하세요. KMS 키 식별자 찾기에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 키 ID 및 ARN 찾기를 참조하세요.

> **Important**
>
> 사용에 대한 추가 요금이 있습니다 AWS KMS. 자세한 내용은 AWS KMS 비용 추정 및 AWS Key Management Service 요금을 참조하세요.

봉투 암호화

암호화된 데이터의 보안은 부분적으로 암호를 해독할 수 있는 데이터 키를 보호하는 데 달려 있습니다. Amazon SQS는 KMS 키를 사용하여 데이터 키를 암호화하며, 암호화된 데이터 키는 암호화된 메시지와 함께 저장됩니다. KMS 키를 사용하여 데이터 키를 암호화하는 이러한 방법을 봉투 암호화라고 합니다.

자세한 내용은 AWS Encryption SDK 개발자 안내서에서 봉투 암호화를 참조하세요.

Amazon SQS 키 관리

Amazon SQS는 AWS Key Management Service (KMS)와 통합되어 서버 측 암호화(SSE)를 위한 KMS 키를 관리합니다. SSE 정보 및 키 관리 정의는 Amazon SQS에서 저장 시 암호화 섹션을 참조하세요. Amazon SQS는 KMS 키를 사용하여 메시지를 암호화하고 복호화하는 데이터 키를 검증하고 보호합니다. 다음 단원에서는 Amazon SQS 서비스에서 KMS 키 및 데이터 키를 사용하는 작업에 대해 설명합니다.

AWS KMS 권한 구성

모든 KMS 키에는 키 정책이 있어야 합니다. Amazon SQS용 AWS 관리형 KMS 키의 키 정책은 수정할 수 없습니다. 이 KMS 키의 정책에는 해당 계정의 모든 주체(Amazon SQS를 사용할 권한이 있는 주체)가 암호화된 대기열을 사용할 수 있는 권한이 포함되어 있습니다.

Amazon SQS는 다른 AWS 계정, IAM 사용자 또는 IAM 역할을 사용하는지 여부에 관계없이 자격 AWS 증명을 기반으로 호출자를 구분합니다. 또한 범위 지정 정책이 서로 다른 동일한 IAM 역할이 별도의 요청자로 처리됩니다. 그러나 IAM 역할 세션을 사용하는 경우 동일한 IAM 역할과 범위 지정 정책을 `RoleSessionName` 유지하면서 만 변경하면 고유한 요청자가 생성되지 않습니다. 따라서 AWS KMS 키 정책 보안 주체를 지정할 때는 `RoleSessionName` 차이에만 의존하지 마세요. 이러한 세션은 동일한 요청자로 취급되기 때문입니다.

또는 필요한 권한을 암호화된 메시지를 생산 및 소비하는 보안 주체에게 할당된 IAM 정책에 지정해도 됩니다. 자세한 내용은 AWS Key Management Service 개발자 안내서에서 AWS KMS로 IAM 정책 사용을 참조하세요.

> **Note**
>
> Amazon SQS에서 송수신하도록 전역 권한을 구성할 수 있지만는 IAM 정책의 `Resource` 섹션에서 특정 리전에 있는 KMS 키의 전체 ARN 이름을 명시적으로 지정해야 AWS KMS 합니다.

AWS 서비스에 대한 KMS 권한 구성

여러 AWS 서비스가 Amazon SQS 대기열로 이벤트를 전송할 수 있는 이벤트 소스 역할을 합니다. 이러한 이벤트 소스가 암호화된 대기열에서 작동하도록 허용하려면 고객 관리형 KMS 키를 생성하고 키 정책에 필요한 AWS KMS API 메서드를 사용할 수 있는 권한을 추가해야 합니다. 권한을 구성하려면 다음 단계를 수행합니다.

> **Warning**
>
> Amazon SQS 메시지를 암호화하기 위해 KMS 키를 변경할 때 이전 KMS 키로 암호화된 기존 메시지는 해당 키로 암호화된 상태로 유지된다는 점에 유의하세요. 이러한 메시지를 복호화하려면 이전 KMS 키를 유지하고 해당 키 정책이 Amazon SQS에 `kms:Decrypt` 및 `kms:GenerateDataKey`에 대한 권한을 부여하는지 확인해야 합니다. 새 메시지를 암호화하기 위해 새 KMS 키로 업데이트한 후에는 이전 KMS 키로 암호화된 모든 기존 메시지를 처리하고 대기열에서 제거한 다음 이전 KMS 키를 삭제하거나 비활성화해야 합니다

1. 고객 관리형 KMS 키를 생성합니다. 자세한 내용은 AWS Key Management Service 개발자 안내서에서 키 생성을 참조하세요.
2. AWS 서비스 이벤트 소스가 `kms:Decrypt` 및 `kms:GenerateDataKey` API 메서드를 사용하도록 허용하려면 KMS 키 정책에 다음 문을 추가합니다.

   JSON

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
```

```
            "Principal": {
               "Service": "service.amazonaws.com"
            },
            "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
            ],
            "Resource": "*"
         }]
}
```

위의 예에서 “service”를 이벤트 소스의 서비스 이름으로 바꿉니다. 이벤트 소스에는 다음 서비스가 포함됩니다.

이벤트 소스	서비스 이름
Amazon CloudWatch Events	`events.amazonaws.com`
Amazon S3 이벤트 알림	`s3.amazonaws.com`
Amazon SNS 주제 구독	`sns.amazonaws.com`

3. KMS 키의 ARN을 사용하여 기존 SSE 대기열을 구성합니다.
4. 암호화된 대기열의 ARN을 이벤트 소스에 제공합니다.

생산자에 대한 AWS KMS 권한 구성

데이터 키 재사용 기간이 만료되면 다음에 생산자가 `SendMessage` 또는 `SendMessageBatch`를 호출했을 때 `kms:Decrypt` 및 `kms:GenerateDataKey` 호출도 트리거됩니다. `kms:Decrypt` 호출은 새 데이터 키를 사용하기 전에 이 데이터 키의 무결성을 확인하기 위한 것입니다. 따라서 생산자에게 KMS 키의 `kms:Decrypt` 및 `kms:GenerateDataKey` 권한이 있어야 합니다.

생산자의 IAM 정책에 다음 문을 추가합니다. 키 리소스와 대기열 리소스에 올바른 ARN 값을 사용해야 합니다.

JSON

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
    ]
}
```

소비자에 대한 AWS KMS 권한 구성

데이터 키 재사용 기간이 만료되면 다음에 소비자가 `ReceiveMessage`를 호출했을 때 `kms:Decrypt` 호출도 트리거되며, 이는 새 데이터 키를 사용하기 전에 이 데이터 키의 무결성을 확인하기 위한 것입니다. 따라서 지정된 대기열에서 메시지 암호화에 사용한 KMS 키의 `kms:Decrypt` 권한이 소비자에게 있어야 합니다. 이 대기열이 DLQ(Dead Letter Queue)인 경우, 소비자는 소스 대기열에서 메시지 암호화에 사용한 KMS 키에 대한 `kms:Decrypt` 권한도 가지고 있어야 합니다. 소비자의 IAM 정책에 다음 문을 추가합니다. 키 리소스와 대기열 리소스에 올바른 ARN 값을 사용해야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
    ]
}
```

혼동된 대리자 보호를 사용하여 AWS KMS 권한 구성

키 정책문의 보안 주체가 AWS 서비스 보안 주체인 경우 `aws:SourceArn` 또는 `aws:SourceAccount` 전역 조건 키를 사용하여 혼동된 대리자 문제를 방지할 수 있습니다. 이러한 조건 키를 사용하려면 값을 암호화되는 리소스의 Amazon 리소스 이름(ARN)으로 설정합니다. 리소스의 ARN을 모르면 `aws:SourceAccount`를 대신 사용합니다.

이 KMS 키 정책에서는 계정별로 소유한 서비스의 특정 리소스가 Amazon SQS의 SSE 사용 중에 발생하는 `111122223333` 및 `Decrypt` 작업을 위해 KMS를 호출할 수 있도록 허용합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "<replaceable>service</replaceable>.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:<replaceable>service<::111122223333:resource"
                    ]
                }
            }
```

```
        }
    ]
}
```

SSE가 활성화된 Amazon SQS 대기열을 사용하는 경우 다음 서비스에서 `aws:SourceArn`을 지원합니다.

- Amazon SNS
- Amazon S3
- CloudWatch Events
- AWS Lambda
- CodeBuild
- Amazon Connect Customer Profiles
- AWS Auto Scaling
- Amazon Chime

데이터 키 재사용 기간 이해

데이터 키 재사용 기간은 Amazon SQS가 한 데이터 키를 재사용할 수 있는 최대 기간을 정의합니다. 데이터 키 재사용 기간이 종료되면 Amazon SQS에서 새 데이터 키를 생성합니다. 재사용 기간에 대한 다음 지침에 유의하십시오.

- 재사용 기간이 짧을수록 보안이 향상되지만에 대한 호출이 늘어나 프리 티어 이상의 요금이 발생할 AWS KMS수 있습니다.
- 데이터 키는 암호화 및 암호 해독에 대해 별도로 캐시되지만 재사용 기간은 데이터 키의 두 복사본 모두에 적용됩니다.
- 데이터 키 재사용 기간이 종료되면에 대한 다음 호출 `SendMessage` 또는 `SendMessageBatch` 일반적으로 메서드에 AWS KMS `GenerateDataKey` 대한 호출을 트리거하여 새 데이터 키를 가져옵니다. 또한 및에 대한 다음 호출`SendMessageReceiveMessage`은 각각에 대한 AWS KMS 호출을 트리거`Decrypt`하여 데이터 키를 사용하기 전에 데이터 키의 무결성을 확인합니다.
- 보안 주체(AWS 계정 또는 사용자)는 데이터 키를 공유하지 않습니다(고유 보안 주체가 보낸 메시지는 항상 고유 데이터 키를 얻음). 따라서에 대한 호출 볼륨 AWS KMS 은 데이터 키 재사용 기간 동안 사용 중인 고유 보안 주체 수의 배수입니다.

AWS KMS 비용 추정

비용을 예측하고 AWS 청구서를 더 잘 이해하려면 Amazon SQS가 KMS 키를 사용하는 빈도를 알고 싶을 수 있습니다.

> **Note**
> 다음 공식으로 예상되는 비용을 거의 정확하게 짐작할 수 있지만, Amazon SQS의 분산 특성상 실제 비용은 더 높을 수 있습니다.

대기열당 API 요청 수(R)를 계산하려면 다음 수식을 사용하세요.

```
R = (B / D) * (2 * P + C)
```

여기서 B는 청구 기간(초)입니다.

D는 데이터 키 재사용 기간(초)입니다.

P는 Amazon SQS 대기열로 전송하는 생성 보안 주체의 수입니다.

C는 Amazon SQS 대기열로부터 받는 소비 보안 주체의 수입니다.

> **Important**
> 일반적으로, 생산 보안 주체에서 발생하는 비용이 소비 보안 주체의 두 배입니다. 자세한 내용은 데이터 키 재사용 기간 이해 단원을 참조하십시오.
> 생산자와 소비자의 사용자가 서로 다른 경우, 비용이 증가합니다.

다음은 계산 예제입니다. 정확한 요금 정보는 AWS Key Management Service 요금을 참조하세요.

예제 1: 보안 주체 2개와 대기열 1개에 대한 AWS KMS API 호출 수 계산

이 예에서는 다음과 같이 가정합니다.

- 청구 기간은 1월 1일부터 31일까지입니다(2,678,400초).
- 데이터 키 재사용 기간은 5분(300초)로 설정되어 있습니다.
- 대기열은 1개입니다.
- 생산 보안 주체 1개와 소비 보안 주체 1개가 있습니다.

```
(2,678,400 / 300) * (2 * 1 + 1) = 26,784
```

예제 2: 여러 생산자 및 소비자와 2개의 대기열에 대한 AWS KMS API 호출 수 계산

이 예에서는 다음과 같이 가정합니다.

- 청구 기간은 2월 1일부터 28일까지입니다(2,419,200초).
- 데이터 키 재사용 기간은 24시간(86,400초)으로 설정되어 있습니다.
- 대기열은 2개입니다.
- 첫 번째 대기열에 생산 보안 주체 3개와 소비 보안 주체 1개가 있습니다.
- 두 번째 대기열에 생산 보안 주체 5개와 소비 보안 주체 2개가 있습니다.

```
(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532
```

AWS KMS 오류

Amazon SQS 및 작업 시 오류가 발생할 AWS KMS수 있습니다. 아래 참조에서 이러한 오류와 문제 해결 방법을 설명합니다.

- 일반적인 AWS KMS 오류
- AWS KMS 암호화 해제 오류
- AWS KMS GenerateDataKey 오류

Amazon SQS의 인터네트워크 트래픽 프라이버시

Amazon SQS용 Amazon Virtual Private Cloud(VPC) 엔드포인트는 VPC 내의 논리적 엔터티로서, Amazon SQS에만 연결을 허용합니다. VPC는 Amazon SQS로 요청을 라우팅하고, 응답을 다시 VPC로 라우팅합니다. 다음 섹션에서는 VPC 엔드포인트 작업 및 VPC 엔드포인트 정책 생성에 대해 설명합니다.

Amazon SQS용 Amazon Virtual Private Cloud 엔드포인트

Amazon VPC를 사용하여 AWS 리소스를 호스팅하는 경우 VPC와 Amazon SQS 간에 연결을 설정할 수 있습니다. 이 연결을 사용하여 퍼블릭 인터넷을 통하지 않고 Amazon SQS 대기열에 메시지를 보낼 수 있습니다.

Amazon VPC를 사용하면 사용자 지정 가상 네트워크에서 AWS 리소스를 시작할 수 있습니다. VPC를 사용하여 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 등의 네트워크 설정을 제어할 수 있습니다. VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

VPC를 Amazon SQS에 연결하려면 먼저 VPC를 다른 AWS 서비스에 연결할 수 있는 인터페이스 VPC 엔드포인트를 정의해야 합니다. 이 엔드포인트를 이용하면 인터넷 게이트웨이나 Network Address Translation(NAT) 인스턴스 또는 VPN 연결 없이도 Amazon SQS에 안정적이고 확장 가능하게 연결됩니다. 자세한 내용은 이 안내서의 자습서: Amazon Virtual Private Cloud에서 Amazon SQS 대기열로 메시지 보내기 및 예제 5: VPC 엔드포인트에서 온 액세스가 아닌 경우 액세스 거부와 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

> **Important**
>
> - Amazon Virtual Private Cloud는 HTTPS Amazon SQS 엔드포인트에서만 사용할 수 있습니다.
> - Amazon VPC에서 메시지를 보내도록 Amazon SQS를 구성할 때는 프라이빗 DNS를 활성화하고 듀얼 스택 엔드포인트에 `sqs.us-east-2.api.aws` 대해 `sqs.us-east-2.amazonaws.com` 또는 형식으로 엔드포인트를 지정해야 합니다.
> - Amazon SQS는 엔드포인트 서비스를 사용하여 PrivateLink를 통해 FIPS `com.amazonaws.region.sqs-fips` 엔드포인트도 지원합니다. 형식으로 FIPS 엔드포인트에 연결할 수 있습니다`sqs-fips.region.amazonaws.com`.
> - Amazon Virtual Private Cloud에서 듀얼 스택 엔드포인트를 사용하는 경우 IPv4 및 IPv6를 사용하여 요청이 전송됩니다.
> - 프라이빗 DNS는 `queue.amazonaws.com` 또는 `us-east-2.queue.amazonaws.com` 같은 레거시 엔드포인트를 지원하지 않습니다.

Amazon SQS용 Amazon VPC 엔드포인트 정책 생성

Amazon SQS에 대한 Amazon VPC 엔드포인트 정책을 생성하여 다음을 지정할 수 있습니다.

- 작업을 수행할 수 있는 위탁자.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

다음 예제 VPC 엔드포인트 정책은 `MyUser` 사용자가 Amazon SQS 대기열 `MyQueue`에 메시지를 게시할 수 있도록 지정합니다.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

다음 작업은 거부됩니다.

- `sqs:CreateQueue` 및 `sqs:DeleteQueue`와 같은 다른 Amazon SQS API 작업.
- 다른 사용자 및 규칙이 이 VPC 엔드포인트를 사용.
- `MyUser`가 다른 Amazon SQS 대기열에 메시지 전송.

> **Note**
>
> 사용자는 VPC 외부에서도 다른 Amazon SQS API 작업을 계속 사용할 수 있습니다. 자세한 내용은 예제 5: VPC 엔드포인트에서 온 액세스가 아닌 경우 액세스 거부 단원을 참조하십시오.

듀얼 스택(IPv4 및 IPv6) 엔드포인트를 사용하여 Amazon SQS에 연결

이중 스택 엔드포인트는 IPv4 트래픽과 IPv6 트래픽을 모두 지원합니다. 듀얼 스택 엔드포인트에 요청하면 엔드포인트 URL이 IPv4 또는 IPv6 주소로 확인됩니다. 듀얼 스택 및 FIPS 엔드포인트에 대한 자세한 내용은 SDK 참조 가이드를 참조하세요.

Amazon SQS는 리전 듀얼 스택 엔드포인트를 지원하므로 엔드포인트 이름의 일부로 AWS 리전을 지정해야 합니다. 듀얼 스택 엔드포인트 이름은 명명 규칙을 사용합니다`sqs.`*`Region`*`.amazonaws.com`. 예를 들어 `eu-west-1` 리전의 이중 스택 엔드포인트 이름은 `sqs.`*`eu-west-1`*`.amazonaws.com`입니다.

Amazon SQS 엔드포인트의 전체 목록은 AWS 일반 참조를 참조하세요.

Amazon SQS의 Identity and Access Management

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 도와주는 입니다. IAM 관리자는 어떤 사용자가 Amazon SQS 리소스를 사용할 수 있도록 인증(로그인)되고 권한이 부여(권한 있음)될 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 Amazon SQS에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 – Amazon SQS 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 Amazon SQS 기능을 사용하여 작업를 수행한다면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. Amazon SQS의 기능에 액세스할 수 없다면 Amazon Simple Queue Service ID 및 액세스 문제 해결 섹션을 참조하세요.

서비스 관리자 – 회사에서 Amazon SQS 리소스를 책임지고 있다면 Amazon SQS에 대한 완전한 액세스 권한이 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Amazon SQS 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 Amazon SQS에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon Simple Queue Service가 IAM으로 작동하는 방식 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 Amazon SQS에 대한 액세스 관리 정책 작성 방법을 자세히 알고 싶을 수도 있습니다. IAM에서 사용할 수 있는 Amazon SQS 자격 증명 기반 정책(identity-based policies)의 예제를 확인하시려면 자격 증명 기반 정책 예제 정책 모범 사례 섹션을 참조하세요.

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. IAM 사용자 또는 AWS 계정 루트 사용자 IAM 역할을 수임하여 로 인증(로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 자격 증명 AWS 으로에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션형 ID로 로그인

할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의에 로그인하는 방법을 AWS참조하세요. AWS 계정

AWS 프로그래밍 방식으로에 액세스하는 경우는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것이 AWS 좋습니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 이 자격 증명을 AWS 계정 루트 사용자라고 하며 계정을 생성하는 데 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스 에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 사용자 디렉터리, 웹 자격 증명 공급자, AWS Directory Service, Identity Center 디렉터리 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스 에 액세스하는 모든 사용자의 사용자입니다. 페더레이션 자격 증명에 액세스할 때 역할을 AWS 계정수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 및 애플리케이션에서 사용할 수 있도록 자체 ID 소스의 사용자 AWS 계정 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 IAM 역할(콘솔)로 전환할 AWS Management Console수 있습니다. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 AWS CLI 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.

- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 정책을 리소스에 직접 연결할 AWS 서비스수 있습니다(역할을 프록시로 사용하는 대신). 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.
- 교차 서비스 액세스 - 일부는 다른에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 완료하기 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 나타나 AWS 계정 며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결된 AWS 경우 권한을 정의하는의 객체입니다.는 보안 주체(사용자, 루트 사

용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console AWS CLI, 또는 API에서 역할 정보를 가져올 수 있습니다 AWS .

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF 및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - SCPs는의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다 AWS Organizations. AWS Organizations 는 기업이 소유한 여러 AWS 계정 을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각각을 포함하여 멤버 계정의 엔터티에 대한 권한을 제한합니다 AWS 계정 루트 사용자. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.
- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 AWS 계정 루트 사용자관계없이를 포함한 자격 증명에 대한 유효 권한에 영향을 미칠 수 있습니다. RCP를 AWS 서비스 지원하는 목록을 포함하여 조직 및 RCPs에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCPs)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon SQS의 액세스 관리 개요

모든 AWS 리소스는에서 소유하며 AWS 계정, 리소스를 생성하거나 액세스할 수 있는 권한은 권한 정책에 의해 관리됩니다. 계정 관리자는 IAM 자격 증명(즉, 사용자, 그룹, 역할)에 권한 정책을 연결할 수 있고, 일부 서비스(예: Amazon SQS)에서는 리소스에 대한 권한 정책 연결도 지원합니다.

> **Note**
> 계정 관리자 또는 관리자 사용자는 관리자 권한이 있는 사용자입니다. 자세한 설명은 IAM 사용자 가이드의 IAM 모범 사례 섹션을 참조하세요.

권한을 부여하려면 권한을 부여 받을 사용자, 권한 대상이 되는 리소스, 이 리소스에서 허용할 특정 작업을 지정합니다.

Amazon Simple Queue Service 리소스 및 작업

Amazon SQS에서 유일한 리소스는 대기열입니다. 정책에서 Amazon 리소스 이름(ARN)을 사용하여 정책이 적용되는 리소스를 식별합니다. 다음 리소스에는 연결된 고유한 ARN이 있습니다.

리소스 유형	ARN 형식
대기열	`arn:aws:sqs:` *`region`*`:`*`account_id`*` :`*`queue_name`

다음은 대기열 ARN 형식의 몇 가지 예입니다.

- AWS 계정 123456789012에 속하는 미국 동부(오하이오) 리전my_queue에 있는 라는 대기열의 ARN:

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- 다음은 Amazon SQS가 지원하는 여러 리전 각각에서 my_queue라고 하는 대기열의 ARN입니다.

```
arn:aws:sqs:*:123456789012:my_queue
```

- 다음은 * 또는 ?를 대기열 이름에서 와일드카드로 사용하는 ARN입니다. 다음 예제에서 ARN은 `my_prefix_`로 시작하는 모든 대기열과 일치합니다.

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

`GetQueueAttributes` 작업을 호출하여 기존 대기열의 ARN 값을 가져올 수 있습니다. `QueueArn` 속성 값은 대기열의 ARN입니다. ARN에 대한 자세한 내용은 IAM 사용 설명서의 IAM ARN 단원을 참조하세요.

Amazon SQS는 대기열 리소스를 사용하는 여러 작업을 제공합니다. 자세한 내용은 Amazon SQS API 권한: 작업 및 리소스 참조 단원을 참조하십시오.

리소스 소유권 이해

는 리소스를 생성한 사용자에 관계없이 계정에서 생성된 리소스를 AWS 계정 소유합니다. 특히, 리소스 소유자는 리소스 생성 요청을 인증하는 보안 주체 엔터티(즉, 루트 계정, 사용자 또는 IAM 역할)의 AWS 계정 입니다. 다음의 예제에서는 이러한 작동 방법을 설명합니다.

- 의 루트 계정 자격 증명을 사용하여 Amazon SQS 대기열을 AWS 계정 생성하는 경우 AWS 계정 는 리소스의 소유자입니다(Amazon SQS에서 리소스는 Amazon SQS 대기열).
- 에서 사용자를 생성하고 사용자에게 대기열을 생성할 수 있는 권한을 AWS 계정 부여하면 사용자가 대기열을 생성할 수 있습니다. 하지만 (사용자가 속한) AWS 계정 은 대기열 리소스를 소유합니다.
- 에서 Amazon SQS 대기열을 생성할 권한이 AWS 계정 있는 IAM 역할을 생성하는 경우 해당 역할을 수임할 수 있는 사람은 누구나 대기열을 생성할 수 있습니다. AWS 계정 (역할이 속한)가 대기열 리소스를 소유합니다.

리소스 액세스 관리

권한 정책은 계정에 부여된 권한을 설명합니다. 다음 섹션에서는 권한 정책을 생성하는 데 사용할 수 있는 옵션에 대해 설명합니다.

> **Note**
>
> 이 섹션에서는 Amazon SQS의 맥락에서 IAM을 사용하는 방법에 대해 설명하며, IAM 서비스에 대한 자세한 정보는 다루지 않습니다. IAM 설명서 전체 내용은 IAM 사용 설명서의 IAM이란

무엇인가요? 섹션을 참조하세요. IAM 정책 구문과 설명에 대한 자세한 내용은 IAM 사용 설명서의 AWS IAM 정책 참조 섹션을 참조하세요.

IAM 자격 증명에 연결된 정책을 자격 증명 기반 정책(IAM 정책)이라 하고, 리소스에 연결된 정책을 리소스 기반 정책이라고 합니다.

ID 기반 정책

사용자에게 Amazon SQS 대기열 권한을 부여하는 방법에는 Amazon SQS 정책 시스템을 사용하는 방법과 IAM 정책 시스템을 사용하는 방법이 있습니다. 두 시스템 중 하나 또는 둘 다를 사용하여 정책을 사용자 또는 역할에 연결할 수 있습니다. 대다수의 경우에 시스템을 사용하여 같은 결과를 얻을 수 있습니다. 예를 들어, 다음을 수행할 수 있습니다.

- 계정 내 사용자 또는 그룹에 권한 정책 연결 - Amazon SQS 대기열 생성 권한을 사용자에게 부여하려면, 권한 정책을 사용자 또는 사용자가 속한 그룹에 연결합니다.
- 다른의 사용자에게 권한 정책 연결 AWS 계정 - 다른의 사용자에게 권한 정책을 연결하여 사용자가 Amazon SQS 대기열과 상호 작용 AWS 계정 할 수 있도록 할 수 있습니다. 그러나 교차 계정 권한은 다음 작업에는 적용되지 않습니다.

  다음 작업에는 교차 계정 권한이 적용되지 않습니다.

  - `AddPermission`
  - `CancelMessageMoveTask`
  - `CreateQueue`
  - `DeleteQueue`
  - `ListMessageMoveTask`
  - `ListQueues`
  - `ListQueueTags`
  - `RemovePermission`
  - `SetQueueAttributes`
  - `StartMessageMoveTask`
  - `TagQueue`
  - `UntagQueue`

  이러한 작업에 대한 액세스 권한을 부여하려면 사용자가 Amazon SQS 대기열을 소유 AWS 계정 한 동일한에 속해야 합니다.

- 역할에 권한 정책 연결(교차 계정 권한 부여) - SQS 대기열에 교차 계정 권한을 부여하려면 IAM 정책과 리소스 기반 정책을 모두 결합해야 합니다.
  1. 계정 A(대기열을 소유)에서:
     - 리소스 기반 정책을 SQS 대기열에 연결합니다. 이 정책은 계정 B의 보안 주체(예: IAM 역할`ReceiveMessage`)에게 필요한 권한(예: `SendMessage`,)을 명시적으로 부여해야 합니다.
  2. 계정 A에서 IAM 역할을 생성합니다.
     - 계정 B 또는가 역할을 수임 AWS 서비스 하도록 허용하는 신뢰 정책입니다.

       > **Note**
       > AWS 서비스 (예: Lambda 또는 EventBridge)가 역할을 수임하도록 하려면 신뢰 정책에 서비스 보안 주체(예: lambda.amazonaws.com)를 지정합니다.

     - 수임된 역할에 대기열과 상호 작용할 수 있는 권한을 부여하는 자격 증명 기반 정책입니다.
  3. 계정 B에서 계정 A의 역할을 수임할 수 있는 권한을 부여합니다.

교차 계정 보안 주체를 허용하도록 대기열의 액세스 정책을 구성해야 합니다. IAM 자격 증명 기반 정책만으로는 SQS 대기열에 대한 교차 계정 액세스에 충분하지 않습니다.

IAM을 사용하여 권한을 위임하는 방법에 대한 자세한 설명은 IAM 사용자 가이드의 액세스 관리 섹션을 참조하세요.

Amazon SQS가 IAM 정책을 사용하는 동안에는 자체 정책 인프라를 보유합니다. 대기열과 함께 Amazon SQS 정책을 사용하여 대기열에 액세스할 수 있는 AWS 계정을 지정할 수 있습니다. 액세스 및 조건 유형을 지정할 수 있습니다(예: 2010년 12월 31일 이전에 요청한 경우 `SendMessage`, `ReceiveMessage`를 사용할 권한을 부여하는 조건). 권한을 부여할 수 있는 특정 작업은 전체 Amazon SQS 작업 목록의 하위 집합입니다. Amazon SQS 정책을 작성하고 `*`를 지정하여 "모든 Amazon SQS 작업을 허용"하도록 지정하면 이것은 사용자가 이 하위 집합에서 모든 작업을 수행할 수 있음을 의미합니다.

다음 다이어그램은 작업의 하위 집합과 관련된 이 기본 Amazon SQS 정책 중 하나의 개념을 보여줍니다. 정책은 용이며 `queue_xyz`계정 AWS 1 및 AWS 계정 2에 지정된 대기열에서 허용된 작업을 사용할 수 있는 권한을 부여합니다.

> **Note**
>
> 정책의 리소스는 로 지정됩니다. `123456789012/queue_xyz`여기서 `123456789012`는 대기열을 소유한 AWS 계정의 계정 ID입니다.

IAM 소개 및 사용자와 Amazon 리소스 이름(ARN)의 개념에서 SQS 정책에 대한 몇 가지 사항이 바뀌었습니다. 다음 다이어그램과 표에는 이 변경 사항에 대한 설명이 나와 있습니다.

1

다른 계정의 사용자에게 권한을 부여하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 자습서: IAM 역할을 사용하여 AWS 계정 간 액세스 권한 위임을 참조하세요.

2

*에 포함된 작업의 하위 집합이 확장되었습니다. 허용된 작업 목록에 대해서는 Amazon SQS API 권한: 작업 및 리소스 참조을 참조하십시오.

3

IAM 정책에서 리소스를 지정하는 기본 수단인 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정할 수 있습니다. Amazon SQS 대기열의 ARN 형식에 대한 정보는 Amazon Simple Queue Service 리소스 및 작업 섹션을 참조하세요.

예를 들어 이전 다이어그램의 Amazon SQS 정책에 따라 AWS 계정 1 또는 AWS 계정 2에 대한 보안 자격 증명을 보유한 사람은 누구나에 액세스할 수 있습니다queue_xyz. 또한, 자체 AWS 계정(ID 123456789012)에 있는 사용자 Bob과 Susan은 대기열에 액세스할 수 있습니다.

IAM 소개에 앞서 Amazon SQS는 대기열의 생성자에게 대기열에 대한 전체 제어권을 자동으로 부여합니다(즉, 이 대기열에서 가능한 Amazon SQS 작업 전체에 대한 액세스). 생성자가 AWS 보안 자격 증명을 사용하지 않을 경우 이것은 더 이상 적용되지 않습니다. 대기열 생성 권한을 가진 사용자는 누구나 생성된 대기열에서 어떤 작업이든 할 수 있도록 다른 Amazon SQS 작업을 사용할 권한이 있어야 합니다.

다음은 한 사용자에게 모든 Amazon SQS 작업을 사용하도록 허용하는 정책 예제이지만, 리터럴 문자열 bob_queue_로 시작하는 이름을 가진 대기열에만 적용됩니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
   }]
}
```

자세한 내용은 Amazon SQS에서 정책 사용 및 IAM 사용 설명서의 자격 증명(사용자, 그룹 및 역할) 섹션을 참조하세요.

정책 요소 지정: 작업, 효과, 리소스, 보안 주체

각 Amazon Simple Queue Service 리소스마다 서비스는 일련의 작업을 정의합니다. 이러한 작업에 대한 권한을 부여하기 위해 Amazon SQS는 정책에서 지정할 수 있는 작업을 정의합니다.

> **Note**
> 작업을 실시하려면 둘 이상의 작업에 대한 권한이 필요할 수 있습니다. 특정 작업에 대한 권한을 부여할 때 해당 작업이 허용되거나 거부되는 리소스도 식별합니다.

다음은 가장 기본적인 정책 요소입니다.

- 리소스 – 정책에서 Amazon 리소스 이름(ARN)을 사용하여 정책을 적용할 리소스를 식별합니다.
- 작업 - 작업 키워드를 사용하여 허용 또는 거부할 리소스 작업을 식별합니다. 예를 들어, `sqs:CreateQueue` 권한은 사용자에게 Amazon Simple Queue Service `CreateQueue` 작업 수행을 허용합니다.
- 결과 – 사용자가 특정 작업을 요청하는 경우의 결과를 지정합니다. 이는 허용 또는 거부 중에 하나가 될 수 있습니다. 리소스에 대한 액세스 권한을 명시적으로 부여하지 않으면 액세스가 암시적으로 거부됩니다. 한 리소스에 대한 액세스를 명시적으로 거부할 수도 있으며, 다른 정책에 따라 액세스 권한이 부여되더라도 사용자가 이 리소스에 액세스하지 못하도록 조치를 취할 수 있습니다.
- Principal – 자격 증명 기반 정책(IAM 정책)에서 정책이 연결되는 사용자는 암시적인 보안 주체입니다. 리소스 기반 정책의 경우, 사용자, 계정, 서비스 또는 권한의 수신자인 기타 개체를 지정합니다(리소스 기반 정책에만 해당).

Amazon SQS 정책 구문과 설명에 대한 자세한 내용은 IAM 사용 설명서의 AWS IAM 정책 참조 섹션을 참조하세요.

모든 Amazon Simple Queue Service 작업과 해당 작업이 적용되는 리소스를 보여주는 표는 Amazon SQS API 권한: 작업 및 리소스 참조 섹션을 참조하세요.

Amazon Simple Queue Service가 IAM으로 작동하는 방식

IAM을 사용하여 Amazon SQS에 대한 액세스를 관리하기 전에 Amazon SQS에서 사용할 수 있는 IAM 기능에 대해 알아봅니다.

Amazon SQS와 함께 사용할 수 있는 IAM 기능

IAM 기능	Amazon SQS 지원
ID 기반 정책	예

IAM 기능	Amazon SQS 지원
리소스 기반 정책	예
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	아니요
ABAC(정책 내 태그)	부분
임시 자격 증명	예
전달 액세스 세션(FAS)	예
서비스 역할	예
서비스 연결 역할	아니요

Amazon SQS 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방식을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

액세스 제어

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

> **Note**
>
> 모든가 계정의 사용자에게 자신의 권한을 위임할 AWS 계정 수 있다는 점을 이해하는 것이 중요합니다. 교차 계정 액세스를 사용하면 추가 사용자를 관리할 필요 없이 AWS 리소스에 대한 액세스를 공유할 수 있습니다. 교차 계정 액세스 사용에 대한 자세한 정보는 IAM 사용 설명서의 교차 계정 액세스 사용을 참조하세요.

Amazon SQS 사용자 지정 정책 내의 콘텐츠 간 권한 및 조건 키에 대한 자세한 내용은 Amazon SQS 사용자 지정 정책의 제한 사항 섹션을 참조하세요.

Amazon SQS의 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

Amazon SQS의 자격 증명 기반 정책 예

Amazon SQS 자격 증명 기반 정책 예제를 보려면 정책 모범 사례 섹션을 참조하세요.

Amazon SQS 내의 리소스 기반 정책

리소스 기반 정책 지원: 예

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 다른 경우 신뢰할 수 AWS 계정있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

Amazon SQS의 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Amazon SQS 작업의 목록을 보려면 서비스 권한 부여 참조의 Amazon Simple Queue Service에서 정의한 작업을 참조하세요.

Amazon SQS의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
sqs
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "sqs:action1",
      "sqs:action2"
         ]
```

Amazon SQS 자격 증명 기반 정책 예제를 보려면 정책 모범 사례 섹션을 참조하세요.

Amazon SQS의 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을

사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

Amazon SQS 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 권한 부여 참조의 Amazon Simple Queue Service에서 정의한 작업을 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon Simple Queue Service에서 정의한 리소스를 참조하세요.

Amazon SQS 자격 증명 기반 정책 예제를 보려면 정책 모범 사례 섹션을 참조하세요.

Amazon SQS의 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition` 요소를 지정하거나 단일 `Condition` 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 `AND` 작업을 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우는 논리적 `OR` 작업을 사용하여 조건을 AWS 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS 는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

Amazon SQS 조건 키 목록을 보려면 서비스 권한 부여 참조의 Amazon Simple Queue Service에 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon Simple Queue Service에서 정의한 리소스를 참조하세요.

Amazon SQS 자격 증명 기반 정책 예제를 보려면 정책 모범 사례 섹션을 참조하세요.

Amazon SQS의 ACL

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon SQS의 ABAC

ABAC 지원(정책의 태그): 부분적

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. 에서는 AWS이러한 속성을 태그라고 합니다. IAM 엔터티(사용자 또는 역할) 및 많은 AWS 리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

Amazon SQS에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

일부 AWS 서비스 는 임시 자격 증명을 사용하여 로그인할 때 작동하지 않습니다. 임시 자격 증명으로 AWS 서비스 작업하는를 포함한 추가 정보는 AWS 서비스 IAM 사용 설명서의 IAM으로 작업하는 섹션을 참조하세요.

사용자 이름과 암호를 제외한 방법을 AWS Management Console 사용하여에 로그인하는 경우 임시 자격 증명을 사용합니다. 예를 들어 회사의 SSO(Single Sign-On) 링크를 AWS 사용하여에 액세스하면 해당 프로세스가 임시 자격 증명을 자동으로 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을

전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다. 그런 다음 이러한 임시 자격 증명을 사용하여 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는 access AWS. AWS recommends에 액세스할 수 있습니다. 자세한 정보는 IAM의 임시 보안 자격 증명 섹션을 참조하세요.

Amazon SQS에 대한 액세스 세션 전달

전달 액세스 세션(FAS) 지원: 예

IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 완료하기 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

Amazon SQS의 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
> 서비스 역할에 대한 권한을 변경하면 Amazon SQS 기능이 중단될 수 있습니다. Amazon SQS에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

Amazon SQS의 서비스 연결 역할

서비스 링크 역할 지원: 아니요

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 나타나 AWS 계정 며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 Yes이(가) 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

AWS 관리형 정책에 대한 Amazon SQS 업데이트

사용자, 그룹 또는 역할에 권한을 추가할 때 정책을 직접 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 더욱 편리합니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하려면 시간과 전문 지식이 필요합니다. 빨리 시작하려면 AWS 관리형 정책을 사용할 수 있습니다. 이러한 정책은 일반적인 사용 사례에 적용되며 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 서비스는 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스는 때때로 AWS 관리형 정책에 추가 권한을 추가하여 새 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새 기능이 시작되거나 새 작업을 사용할 수 있게 될 때 AWS 관리형 정책을 업데이트할 가능성이 높습니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않으므로 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한는 여러 서비스에 걸쳐 있는 직무에 대한 관리형 정책을 AWS 지원합니다. 예를 들어 ReadOnlyAccess라는 이름의 AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스 권한을 제공합니다. 서비스가 새 기능을 시작하면는 새 작업 및 리소스에 대한 읽기 전용 권한을 AWS 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한AWS 관리형 정책을 참조하세요.

AWS 관리형 정책: AmazonSQSFullAccess

`AmazonSQSFullAccess` 정책을 Amazon SQS 자격 증명에 연결할 수 있습니다. 이 정책은 Amazon SQS에 대한 전체 액세스를 허용하는 권한을 부여합니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조에서 AmazonSQSFullAccess를 참조하세요.

AWS 관리형 정책: AmazonSQSReadOnlyAccess

`AmazonSQSReadOnlyAccess` 정책을 Amazon SQS 자격 증명에 연결할 수 있습니다. 이 정책은 Amazon SQS에 대한 읽기 전용 액세스를 허용하는 권한을 부여합니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조에서 AmazonSQSReadOnlyAccess를 참조하세요.

AWS 관리형 정책: SQSUnlockQueuePolicy

멤버 계정에 대해 모든 사용자의 Amazon SQS 대기열 액세스를 거부하도록 대기열 정책을 잘못 구성한 경우 `SQSUnlockQueuePolicy` AWS 관리형 정책을 사용하여 대기열을 잠금 해제할 수 있습니다.

모든 보안 주체가 Amazon SQS 대기열에 액세스하지 못하도록 잘못 구성된 대기열 정책을 제거하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS Organizations 멤버 계정에서 권한 있는 작업 수행을 참조하세요.

AWS 관리형 정책에 대한 Amazon SQS 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후부터 Amazon SQS의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 Amazon SQS 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
SQSUnlockQueuePolicy	Amazon SQS는 대기열을 잠금 해제하고 모든 보안 주체가 Amazon SQS 대기열에 액세스하는 것을 거부하는 잘못 구성된 대기열 정책을 제거하기 위해 `SQSUnlockQueuePolicy` 라는 새로운 AWS관리형 정책을 추가했습니다.	2024년 11월 15일
AmazonSQSReadOnlyAccess	Amazon SQS는 지정된 Amazon SQS 대기열과 연결된 모든 태그를 검색하는 `ListQueueTags` 작업을 추가했습니다. 이를 통해 조직 또는 메타데이터 목적으로 대기열에 할당된 키 값 페어를 볼 수 있습니다. 이 작업은 `ListQueueTags` API 작업과 연결되어 있습니다.	2024년 6월 20일

변경 사항	설명	날짜
AmazonSQSReadOnlyAccess	Amazon SQS는 특정 소스 대기열 아래에 가장 최근의 메시지 이동 작업(최대 10개)을 나열할 수 있는 새 작업을 추가했습니다. 이 작업은 `ListMessageMoveTasks` API 작업과 연결되어 있습니다.	2023년 6월 9일

Amazon Simple Queue Service ID 및 액세스 문제 해결

다음 정보를 사용하여 Amazon SQS 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

Amazon SQS에서 작업을 수행할 권한이 없음

작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음 예제 오류는 `mateojackson` 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `sqs:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 sqs:GetWidget on resource: my-example-widget
```

이 경우 Mateo의 정책은 `sqs:`*`GetWidget`* 작업을 사용하여 *`my-example-widget`* 리소스에 액세스하도록 허용하도록 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon SQS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 Amazon SQS에서 태스크를 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 iam:PassRole 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 Amazon SQS 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- Amazon SQS에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon Simple Queue Service가 IAM으로 작동하는 방식 섹션을 참조하세요.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 리소스에 대한 액세스 권한을 타사에 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

대기열을 잠금 해제하고 싶습니다.

가 조직에 AWS 계정 속한 경우 AWS Organizations 정책은 사용자가 Amazon SQS 리소스에 액세스하는 것을 차단할 수 있습니다. 기본적으로 AWS Organizations 정책은 Amazon SQS에 대한 요청을 차단하지 않습니다. 그러나 Amazon SQS 대기열에 대한 액세스를 차단하도록 AWS Organizations 정

책이 구성되지 않았는지 확인합니다. AWS Organizations 정책을 확인하는 방법에 대한 지침은 AWS Organizations 사용 설명서의 모든 정책 나열을 참조하세요.

또한 Amazon SQS 대기열에 대한 모든 사용자 액세스를 거부하도록 멤버 계정에 대한 대기열 정책을 잘못 구성한 경우 IAM에서 멤버 계정에 대한 권한 있는 세션을 시작하여 대기열을 잠금 해제할 수 있습니다. 권한 있는 세션을 시작하면 잘못 구성된 대기열 정책을 삭제하여 대기열에 대한 액세스 권한을 다시 얻을 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS Organizations 멤버 계정에 대한 권한 있는 작업 수행을 참조하세요.

Amazon SQS에서 정책 사용

이 주제에서는 계정 관리자가 IAM 자격 증명(사용자, 그룹, 역할)에 권한 정책을 연결할 수 있는 자격 증명 기반 정책 예제를 제시합니다.

> **Important**
>
> Amazon Simple Queue Service 리소스에 대한 액세스 관리를 위해 제공되는 기본 개념과 옵션 설명에 대한 소개 주제 부분을 먼저 읽어 보는 것이 좋습니다. 자세한 내용은 Amazon SQS의 액세스 관리 개요 단원을 참조하십시오.
> `ListQueues`를 제외한 모든 Amazon SQS 작업은 리소스 수준 권한을 지원합니다. 자세한 내용은 Amazon SQS API 권한: 작업 및 리소스 참조 단원을 참조하십시오.

Amazon SQS 및 IAM 정책 사용

사용자에게 Amazon SQS 리소스 권한을 부여하는 방법에는 Amazon SQS 정책 시스템(리소스 기반 정책)을 사용하는 방법과 IAM 정책 시스템(자격 증명 기반 정책)을 사용하는 방법이 있습니다. IAM 정책에서만 설정할 수 있는 리전 권한인 `ListQueues` 작업을 제외하고 하나 또는 두 가지 방법을 모두 사용할 수 있습니다.

예를 들어, 다음 다이어그램은 동일한 IAM 정책과 Amazon SQS 정책을 보여줍니다. IAM 정책은 AWS 계정queue_xyz에서 라는 대기열에 대한 Amazon SQS `ReceiveMessage` 및 `SendMessage` 작업에 대한 권한을 부여하며, 정책은 Bob과 Susan이라는 사용자에게 연결됩니다(Bob과 Susan은 정책에 명시된 권한을 가짐). 이 Amazon SQS 정책 역시 동일한 대기열에서 `ReceiveMessage` 및 `SendMessage` 작업에 대한 권한을 Bob과 Susan에게 부여합니다.

> **Note**
>
> 다음 예제는 조건 없는 간단한 정책을 보여줍니다. 두 정책 중 어느 쪽에서도 특정 조건을 지정하고 동일한 결과를 얻을 수 있습니다.

IAM 정책과 Amazon SQS 정책에는 한 가지 주요 차이점이 있습니다. Amazon SQS 정책 시스템을 사용하면 다른 AWS 계정에 권한을 부여할 수 있지만 IAM은 부여하지 않습니다.

사용자의 필요에 따라 두 시스템을 함께 사용하여 권한을 관리해야 합니다. 다음 예는 두 정책 시스템이 어떻게 연계하는지 보여줍니다.

- 첫 번째 예에서 Bob은 IAM 정책과 Amazon SQS 정책 모두를 자신의 계정에 적용했습니다. IAM 정책은 `queue_xyz`에서 `ReceiveMessage` 작업에 대한 계정 권한을 부여하지만, Amazon SQS 정책은 동일한 대기열에서 `SendMessage` 작업에 대한 계정 권한을 부여합니다. 다음 다이어그램에서 관련 개념을 설명합니다.

Bob이 `ReceiveMessage` 요청을 `queue_xyz`에 전송하면, IAM 정책은 이 작업을 허용합니다. Bob이 `SendMessage` 요청을 `queue_xyz`에 전송하면, Amazon SQS 정책은 이 작업을 허용합니다.

- 두 번째 예에서 Bob은 `queue_xyz`에 대한 액세스 권한을 침해하므로, 해당 대기열에 대한 전체 액세스 권한을 제거해야 합니다. 가장 쉬운 방법은 모든 대기열 작업에 대한 Bob의 액세스를 거부하는 정책을 추가하는 것입니다. 명시적 `deny`는 항상 `allow`를 무시하기 때문에 이 정책은 다른 2개 정책을 무시합니다. 정책 평가 로직에 대한 자세한 내용은 Amazon SQS 액세스 정책 언어와 함께 사용자 지정 정책 사용를 참조하십시오. 다음 다이어그램에서 관련 개념을 설명합니다.

Bob이 어떤 방식으로 대기열에 액세스하든 간에 거부하는 문을 Amazon SQS 정책에 추가할 수도 있습니다. 이것은 Bob의 대기열 액세스를 거부하는 IAM 정책을 추가하는 것과 동일한 효과가 있습

니다. Amazon SQS 작업 및 리소스에 대한 정책의 예는 Amazon SQS 정책의 기본 예제에서 확인하세요. Amazon SQS 정책 작성에 대한 자세한 내용은 Amazon SQS 액세스 정책 언어와 함께 사용자 지정 정책 사용 섹션을 참조하세요.

Amazon SQS 콘솔 사용에 필요한 권한

사용자가 Amazon SQS 콘솔로 작업하려면 사용자의 AWS 계정에서 Amazon SQS 대기열로 작업할 수 있는 최소 개수의 권한이 있어야 합니다. 예를 들어, 사용자에게 `ListQueues` 작업을 호출할 권한이 있어 대기열을 나열할 수 있거나 `CreateQueue` 작업을 호출할 권한이 있어 대기열을 생성할 수 있어야 합니다. Amazon SQS 권한 외에도 Amazon SQS 대기열에서 Amazon SNS 주제를 구독하려면 콘솔에 Amazon SNS 작업에 대한 권한도 있어야 합니다.

최소 필수 권한보다 더 제한적인 IAM 정책을 만들면 콘솔은 해당 IAM 정책에 연결된 사용자에 대해 의도대로 작동하지 않을 수 있습니다.

AWS CLI 또는 Amazon SQS 작업만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다.

Amazon SQS의 자격 증명 기반 정책 예

기본적으로 사용자 및 역할은 Amazon SQS 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 Amazon SQS에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 권한 부여 참조에서 Amazon Simple Queue Service에 대한 작업, 리소스 및 조건 키를 참조하세요.

> **Note**
>
> Amazon EC2 Auto Scaling의 수명 주기 후크를 구성할 때 Amazon SQS 대기열로 메시지를 전송하는 정책을 작성하지 않아도 됩니다. 자세한 설명은 Amazon EC2 사용자 가이드의 Amazon EC2 Auto Scaling 수명 주기 후크를 참조하세요.

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Amazon SQS 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Amazon SQS 콘솔 사용

Amazon Simple Queue Service 콘솔에 액세스하려면 최소한의 권한 집합이 있어야 합니다. 이러한 권한은에서 Amazon SQS 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 여전히 Amazon SQS 콘솔을 사용할 수 있도록 하려면 Amazon SQS `AmazonSQSReadOnlyAccess` AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

사용자가 대기열을 생성하도록 허용

다음 예제에서는 Bob이 모든 Amazon SQS 작업에 액세스할 수 있도록 정책을 생성하지만, 리터럴 문자열 `alice_queue_`로 시작하는 이름을 가진 대기열에만 해당됩니다.

Amazon SQS는 대기열 생성자에게 대기열 사용 권한을 자동으로 부여하지 않습니다. 따라서 IAM 정책의 `CreateQueue` 작업 외에도 모든 Amazon SQS 작업을 사용할 수 있는 권한을 Bob에게 명시적으로 부여해야 합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
   }]
}
```

개발자가 메시지를 공유 대기열에 작성하도록 허용

다음 예제에서는 개발자를 위한 그룹을 생성하고 그룹이 지정된에 속 AWS 계정 하고 이름이인 대기열에서만 Amazon SQS `SendMessage` 작업을 사용하도록 허용하는 정책을 연결합니다`MyCompanyQueue`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
   }]
}
```

SendMessage 대신에 *를 사용하여 공유 대기열에서 보안 주체에게 ChangeMessageVisibility, DeleteMessage, GetQueueAttributes, GetQueueUrl, ReceiveMessage 및 SendMessage 작업을 허용할 수 있습니다.

> **Note**
>
> *에는 다른 권한 유형에서 제공하는 액세스 권한이 포함되나 Amazon SQS는 권한을 개별적으로 고려합니다. 예를 들어, *에 SendMessage에서 제공하는 액세스 권한이 포함되어 있더라도 * 및 SendMessage 권한을 사용자에게 부여할 수 있습니다.
> 이 개념은 권한을 제거할 때도 적용됩니다. 보안 주체가 * 권한만 있는 경우, SendMessage 권한을 제거하도록 요청해도 보안 주체에게 everything-but 권한이 남는 것은 아닙니다. 그 대신 보안 주체가 명시적 SendMessage 권한을 소유하지 않았기 때문에 이 요청은 아무런 영향도 주지 않습니다. ReceiveMessage 권한만 보안 주체에게 남기려면 먼저 ReceiveMessage 권한을 추가한 후 * 권한을 제거합니다.

관리자가 일반 크기의 대기열을 가져오도록 허용

다음 예제에서는 관리자를 위한 그룹을 생성하고 그룹이 지정된 AWS 계정에 속한 모든 대기열과 함께 Amazon SQS GetQueueAttributes 작업을 사용하도록 허용하는 정책을 연결합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:GetQueueAttributes",
      "Resource": "*"
   }]
}
```

파트너가 메시지를 특정 대기열로 전송하도록 허용

Amazon SQS 정책 또는 IAM 정책을 사용하여 이 작업을 완료할 수 있습니다. 파트너에게이 있는 경우 Amazon SQS 정책을 사용하는 AWS 계정것이 더 쉬울 수 있습니다. 그러나 파트너의 회사에서 AWS 보안 자격 증명을 소유한 모든 사용자는 대기열에 메시지를 보낼 수 있습니다. 액세스를 특정 사용자나

애플리케이션으로 제한하려면, 파트너를 본인 회사의 사용자처럼 다루면서 Amazon SQS 정책이 아닌 IAM 정책을 사용해야 합니다.

이 예제에서는 다음 작업을 수행합니다.

1. 파트너사를 나타내는 WidgetCo라는 그룹을 생성합니다.
2. 파트너사에서 액세스 권한을 필요로 하는 특정 사용자나 애플리케이션을 위해 사용자를 생성합니다.
3. 사용자를 그룹에 추가합니다.
4. `WidgetPartnerQueue`라는 대기열에서만 `SendMessage` 작업에 유일하게 액세스할 수 있는 권한을 이 그룹에 부여하는 정책을 연결합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "sqs:SendMessage",
         "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
   }]
}
```

Amazon SQS 정책의 기본 예제

이 단원에서는 일반 Amazon SQS 사용 사례에 대한 정책 예제를 보여줍니다.

콘솔을 사용하여 정책을 사용자에게 연결할 때 각 정책의 효과를 확인할 수 있습니다. 처음에 사용자는 권한을 가지고 있지 않으며 콘솔에서 어떠한 작업도 수행할 수 없습니다. 정책을 사용자에게 연결하면 사용자가 콘솔에서 다양한 작업을 수행할 수 있는지 확인할 수 있습니다.

> **Note**
>
> 두 개의 브라우저 창을 사용하는 것이 좋습니다. 하나는 권한을 부여하고 다른 하나는 권한을 부여할 때 사용자의 자격 증명을 AWS Management Console 사용하여에 로그인하여 권한을 확인하는 것입니다.

예제 1: 하나의에 하나의 권한 부여 AWS 계정

다음 예제 정책은 미국 동부(오하1111222233333이오) 리전444455556666/queue1에 있는 라는 대기열에 대한 SendMessage 권한을 AWS 계정 번호에 부여합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_SendMessage",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue1"
   }]
}
```

예제 2: 한에 두 가지 권한 부여 AWS 계정

다음 예제 정책은 이라는 대기열에 대해 SendMessage 및 ReceiveMessage 권한을 AWS 계정 111122223333 모두 부여합니다444455556666/queue1.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_Send_Receive",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
```

```
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:*:444455556666:queue1"
   }]
}
```

예제 3: 2에 모든 권한 부여 AWS 계정

다음 예제 정책은 Amazon SQS가 미국 동부(111122223333오하이오444455556666) 리전123456789012/queue1에서 라는 대기열에 대한 공유 액세스를 허용하는 모든 작업을 사용할 수 있는 두 개의 서로 다른 AWS 계정 번호(및)를 부여합니다. Amazon SQS

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333",
            "444455556666"
         ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
   }]
}
```

예제 4: 역할과 사용자 이름에 교차 계정 권한 부여

다음 예제 정책은 Amazon SQS가 미국 동부(오하이오) 리전123456789012/queue1에 있는 라는 대기열에 대한 공유 액세스를 허용하는 모든 작업을 사용할 수 있는 111122223333 교차 계정 권한을 AWS 계정 및 번호 role1 username1 아래에 부여합니다.

다음 작업에는 교차 계정 권한이 적용되지 않습니다.

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTask
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "arn:aws:iam::111122223333:role/role1",
            "arn:aws:iam::111122223333:user/username1"
         ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
   }]
}
```

예제 5: 모든 사용자에게 권한 부여

다음 정책 예제에서는 111122223333/queue1이라는 대기열에 대한 ReceiveMessage 권한을 모든 사용자(익명 사용자)에게 부여합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_ReceiveMessage",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1"
   }]
}
```

예제 6: 모든 사용자에게 시간 제한적인 권한 부여

다음 정책 예제는 모든 사용자(익명 사용자)에게 111122223333/queue1라는 이름의 대기열에 대한 ReceiveMessage 권한을 부여하지만, 2009년 1월 31일에는 오후 12시(정오)부터 오후 3시까지만 가능합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_ReceiveMessage_TimeLimit",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "DateGreaterThan" : {
            "aws:CurrentTime":"2009-01-31T12:00Z"
         },
```

```
            "DateLessThan" : {
               "aws:CurrentTime":"2009-01-31T15:00Z"
            }
         }
      }]
}
```

예제 7: CIDR 범위의 모든 사용자에게 모든 권한 부여

다음 정책 예제에서는 모든 사용자(익명 사용자)에게 111122223333/queue1이라는 대기열에서 공유 가능한 모든 Amazon SQS 작업을 사용할 수 있는 권한을 부여하지만, 이것은 요청이 192.0.2.0/24 CIDR 범위에서 들어오는 경우에 한합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_AllActions_AllowlistIP",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.0.2.0/24"
         }
      }
   }]
}
```

예제 8: 서로 다른 CIDR 범위의 사용자에게 권한 허용(화이트리스트) 및 차단(블랙리스트)

다음의 정책 예제에는 문이 2개입니다.

- 첫 번째 문은 192.0.2.0/24 CIDR 범위(192.0.2.188 제외)에 있는 모든 사용자(익명 사용자)에게 111122223333/queue1이라는 대기열에서 SendMessage 작업을 사용할 수 있는 권한을 부여합니다.

- 두 번째 설명은 `12.148.72.0/23` CIDR 범위의 모든 사용자(익명 사용자)가 대기열을 사용하지 못하도록 차단합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_SendMessage_IPLimit",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.0.2.0/24"
         },
         "NotIpAddress" : {
            "aws:SourceIp":"192.0.2.188/32"
         }
      }
   }, {
      "Sid":"Queue1_AnonymousAccess_AllActions_IPLimit_Deny",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"12.148.72.0/23"
         }
      }
   }]
}
```

Amazon SQS 액세스 정책 언어와 함께 사용자 지정 정책 사용

AWS 계정 ID만을 기반으로 기본 권한(예: `SendMessage` 또는 `ReceiveMessage`)을 부여하려면 사용자 지정 정책을 작성할 필요가 없습니다. 대신 Amazon SQS `AddPermission` 작업을 사용합니다.

요청 시간 또는 요청자의 IP 주소와 같은 특정 조건에 따라 액세스를 허용하거나 거부하려면 사용자 지정 Amazon SQS 정책을 생성하고 SetQueueAttributes 작업을 사용하여 업로드해야 합니다.

주제

- Amazon SQS 액세스 제어 아키텍처
- Amazon SQS 액세스 제어 프로세스 워크플로
- Amazon SQS 액세스 정책 언어 주요 개념
- Amazon SQS 액세스 정책 언어 평가 로직
- Amazon SQS 액세스 정책 언어의 명시적 거부와 기본 거부 간의 관계
- Amazon SQS 사용자 지정 정책의 제한 사항
- 사용자 지정 Amazon SQS 액세스 정책 언어 예제

Amazon SQS 액세스 제어 아키텍처

다음 그림에서는 Amazon SQS 리소스의 액세스 제어에 대해 설명합니다.

1 리

소스 소유자

2

AWS 서비스에 포함된 리소스(예: Amazon SQS 대기열).

3

정책. 리소스당 하나의 정책을 가지는 것이 좋습니다. 이 AWS 서비스는 정책을 업로드하고 관리하는 데 사용하는 API를 제공합니다.

4

요청자 및 요청자가 AWS 서비스에서 수신하는 요청.

5

액세스 정책 언어 평가 코드. 이는 서비스 내에서 해당 정책을 기준으로 수신 요청을 AWS 평가하고 요청자가 리소스에 액세스할 수 있는지 여부를 결정하는 코드 세트입니다.

Amazon SQS 액세스 제어 프로세스 워크플로

다음 그림에서는 Amazon SQS 액세스 정책 언어를 사용한 일반적인 액세스 제어 워크플로를 설명합니다.

1

대기열에 대한 Amazon SQS 정책을 작성합니다.

2

정책을에 업로드합니다 AWS. AWS 서비스는 정책을 업로드하는 데 사용하는 API를 제공합니다. 예를

들어 Amazon SQS `SetQueueAttributes` 작업을 사용하여 특정 Amazon SQS 대기열에 대한 정책을 업로드합니다.

3

누군가가 Amazon SQS 대기열을 사용하기 위해 요청을 보냅니다.

4

Amazon SQS에서는 사용 가능한 모든 Amazon SQS 정책을 검사하고 어느 정책을 적용할 수 있는지 결정합니다.

5

Amazon SQS에서는 정책을 평가하고 요청자의 대기열 사용을 허용할지 여부를 결정합니다.

6

이를테면 정책 평가 결과에 따라 Amazon SQS가 `Access denied` 오류를 요청자에게 반환하거나 계속 요청을 처리합니다.

Amazon SQS 액세스 정책 언어 주요 개념

고유한 정책을 작성하려면 JSON과 많은 주요 개념에 대해 잘 알고 있어야 합니다.

허용

`allow`로 설정된 효과이(가) 있는 Statement의 결과.

작업

보안 주체가 일반적으로 AWS에 대한 요청을 수행할 권한이 있는 활동입니다.

Default-deny

허용 또는 Explicit-deny 설정이 없는 Statement의 결과입니다.

Condition

권한에 대한 제한 또는 세부 정보입니다. 일반적인 조건은 날짜 및 시간과 IP 주소와 관련되어 있습니다.

효과

정책의 Statement이(가) 평가 시간에 반환하는 결과. 정책 설명을 작성할 때 `deny` 또는 `allow` 값을 지정합니다. 정책 평가 시간에 Default-deny, 허용 및 Explicit-deny 결과가 나올 수 있습니다.

Explicit-deny

deny로 설정된 효과이(가) 있는 Statement의 결과.

평가

Amazon SQS에서 수신한 요청을 정책에 따라 거부할지 또는 허용할지 결정하는 데 이용하는 프로세스입니다.

발행자

리소스에 권한을 부여하기 위해 정책을 작성하는 사용자입니다. 발급자는 항상 리소스 소유자입니다. AWS 는 Amazon SQS 사용자가 소유하지 않은 리소스에 대한 정책을 생성하도록 허용하지 않습니다.

Key(키)

액세스 제한의 기준이 되는 구체적 특성입니다.

권한

Condition 및 Key(키)를 사용하여 리소스에 대한 액세스를 허용하거나 허용하지 않는 개념입니다.

정책

하나 이상의 설명에 대한 컨테이너 역할을 하는 문서입니다.

Amazon SQS는 사용자에게 리소스에 대한 액세스 권한을 부과할지 결정하는 정책을 사용합니다.

보안 주체

정책에서 권한을 수신하는 사용자입니다.

리소스

보안 주체 요청이 액세스하는 객체입니다.

Statement

단일 권한에 대한 공식적인 설명이며, 보다 폭넓은 정책 문서의 일환으로서 액세스 정책 언어로 작성됩니다.

요청자

리소스에 액세스 요청을 전송하는 사용자입니다.

Amazon SQS 액세스 정책 언어 평가 로직

평가 시점에 Amazon SQS는 리소스 소유자 이외의 다른 사람의 요청을 허용할지 또는 거부할지 결정합니다. 평가 로직에서는 몇 가지 기본 규칙을 따릅니다.

- 기본적으로, 본인을 제외한 사람이 리소스를 사용하겠다고 요청하면 모두 거부됩니다.
- 허용는 모든 Default-deny를 재정의합니다.
- Explicit-deny는 모든 허용을 재정의합니다.
- 정책이 평가되는 순서는 중요하지 않습니다.

다음 그림에서는 Amazon SQS가 액세스 권한에 대한 결정을 평가하는 방식에 대해 자세히 설명합니다.

1

결정은 기본 거부에서 시작합니다.

2

적용 코드에서 (리소스, 보안 주체, 작업, 조건을 고려하여) 해당 요청에 적용 가능한 모든 정책을 평가합니다. 적용 코드에서 정책을 평가하는 순서는 중요하지 않습니다.

3

적용 코드는 해당 요청에 적용할 수 있는 명시적 거부 명령을 찾습니다. 하나라도 찾으면 이 적용 코드는 거부 결정을 반환하고 프로세스는 종료됩니다.

4

명시적 거부 명령이 없을 경우 적용 코드는 해당 요청에 적용할 수 있는 허용 명령을 찾습니다. 하나라도 찾으면 적용 코드는 허용 결정을 반환하고 프로세스가 완료됩니다(서비스에서는 계속 요청을 처리함).

5

허용 명령이 없을 경우 최종 결정은 거부입니다. 명시적 거부 또는 허용이 없으므로 기본 거부로 간주됩니다.

Amazon SQS 액세스 정책 언어의 명시적 거부와 기본 거부 간의 관계

Amazon SQS 정책이 요청에 직접 적용되지 않으면 요청 결과는 Default-deny입니다. 예를 들어, 한 사용자가 Amazon SQS 사용 권한을 요청했지만 해당 사용자에게 적용되는 유일한 정책에서 DynamoDB를 사용할 수 있으면, 요청 결과는 기본 거부가 됩니다.

설명의 조건이 충족되지 않을 경우에도 요청 결과는 기본 거부가 됩니다. 명령문의 모든 조건이 충족되면 요청은 정책의 효과 요소 값에 따라 허용 또는 Explicit-deny로 처리됩니다. 정책에서는 어떤 조건이 충족되지 않을 때 해야 할 일을 지정하지 않으므로, 이러한 경우 기본 결과는 기본 거부가 됩니다. 예를 들어 남극 대륙에서 오는 요청을 차단하고 싶다면 남극 대륙에서 오지 않은 요청만 허용하는 정책 A1을 작성하면 됩니다. 다음 다이어그램은 Amazon SQS 정책을 보여줍니다.

사용자가 미국에서 요청을 보낼 경우 (남극 대륙에서 온 요청이 아니므로) 조건을 충족하고, 요청 결과는 허용이 됩니다. 그러나 사용자가 남극 대륙에서 요청을 보낸다면 조건이 충족되지 않으므로 요청 결과는 기본 거부가 됩니다. 이 결과를 명시적 거부로 변경하려면 남극 대륙에서 온 요청을 명시적으로 거부하는 정책 A2를 작성합니다. 다음 다이어그램은 이 정책을 보여줍니다.

사용자가 남극 대륙에서 요청을 보낸다면 조건이 충족되므로 요청 결과는 명시적 거부가 됩니다.

허용은 기본 거부를 덮어쓸 수 있지만 명시적 거부는 덮어쓸 수 없기 때문에 전자와 후자를 구분하는 것이 중요합니다. 예를 들어, 요청이 2010년 6월 1일에 도착하면 정책 B는 요청을 허용합니다. 다음 다이어그램은 이 정책을 정책 A1과 정책 A2 조합을 비교한 것입니다.

시나리오 1에서 이 정책은 2010년 6월 1일에 들어오는 요청을 허용하기 때문에 정책 A1의 결과는 기본 거부가 되고 정책 B의 결과는 허용이 됩니다. 정책 B의 허용은 정책 A1의 기본 거부보다 우선하므로 이 요청은 허용됩니다.

시나리오 2에서 정책 B2의 결과는 명시적 거부이고 정책 B의 결과는 허용입니다. 정책 A2의 명시적 거부는 정책 B의 허용보다 우선하고 이 요청은 거부됩니다.

Amazon SQS 사용자 지정 정책의 제한 사항

크로스 계정 액세스

다음 작업에는 교차 계정 권한이 적용되지 않습니다.

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTask
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

조건 키

현재 Amazon SQS는 다음과 같이 IAM에서 사용할 수 있는 조건 키의 제한된 하위 세트만 지원합니다. 자세한 내용은 Amazon SQS API 권한: 작업 및 리소스 참조 단원을 참조하십시오.

사용자 지정 Amazon SQS 액세스 정책 언어 예제

다음은 일반적인 Amazon SQS 액세스 정책의 예제입니다.

예제 1: 1개의 계정에 권한 부여

다음 Amazon SQS 정책 예제는 AWS 계정 444455556666이 소유한 queue2에 대해 전송 및 수신할 수 있는 권한을 AWS 계정 111122223333에 부여합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase1",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2"
   }]
}
```

예제 2: 1개 이상의 계정에 권한 부여

다음 예제 Amazon SQS 정책은 특정 기간 동안 계정이 소유한 대기열에 대한 액세스 AWS 계정 권한을 하나 이상 부여합니다. `AddPermission` 작업은 대기열에 대한 액세스 권한을 부여할 때 시간 제한 지정을 허용하지 않기 때문에 이 정책을 작성하고 `SetQueueAttributes` 작업을 사용하여 Amazon SQS에 업로드해야 합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase2",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333",
            "444455556666"
```

```
            ]
        },
        "Action": [
            "sqs:SendMessage",
            "sqs:ReceiveMessage"
        ],
        "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
        "Condition": {
            "DateLessThan": {
                "AWS:CurrentTime": "2009-06-30T12:00Z"
            }
        }
    }]
}
```

예제 3: Amazon EC2 인스턴스의 요청에 권한 부여

다음 예제 Amazon SQS 정책에서는 Amazon EC2 인스턴스에서 오는 요청에 대한 액세스를 부여합니다. 이 예제는 "예제 2: 1개 이상의 계정에 권한 부여" 예제를 보강한 것으로, 2009년 6월 30일 12시(정오)(UTC) 이전까지 액세스할 수 있고 액세스 IP 범위를 `203.0.113.0/24`로 제한합니다. `AddPermission` 작업은 대기열에 대한 액세스 권한을 부여할 때 IP 주소 제한 지정을 허용하지 않기 때문에 이 정책을 작성하고 `SetQueueAttributes` 작업을 사용하여 Amazon SQS에 업로드해야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Id": "UseCase3",
    "Statement" : [{
        "Sid": "1",
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "111122223333"
            ]
        },
        "Action": [
            "sqs:SendMessage",
            "sqs:ReceiveMessage"
        ],
```

```
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
      "Condition": {
         "DateLessThan": {
            "AWS:CurrentTime": "2009-06-30T12:00Z"
         },
         "IpAddress": {
            "AWS:SourceIp": "203.0.113.0/24"
         }
      }
   }]
}
```

예제 4: 특정 계정에 대한 액세스 거부

다음 예제 Amazon SQS 정책은 대기열에 대한 특정 AWS 계정 액세스를 거부합니다. 이 예제는 "예제 1: 1개의 계정에 권한 부여" 예제를 기반으로 합니다. 지정된에 대한 액세스를 거부합니다 AWS 계정. `AddPermission` 작업은 대기열에 대한 액세스 거부를 허용하지 않고 대기열에 대한 액세스 권한 부여만 허용하기 때문에 이 정책을 작성하고 `SetQueueAttributes` 작업을 사용하여 Amazon SQS에 업로드해야 합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase4",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Deny",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2"
   }]
}
```

예제 5: VPC 엔드포인트에서 온 액세스가 아닌 경우 액세스 거부

다음 예제 Amazon SQS 정책은 `queue1`에 대한 액세스를 제한합니다. 111122223333은 VPC 엔드포인트 ID `vpce-1a2b3c4d`(`aws:sourceVpce` 조건을 사용하여 지정됨)에서만 `SendMessage` 및 `ReceiveMessage` 작업을 수행할 수 있습니다. 자세한 내용은 Amazon SQS용 Amazon Virtual Private Cloud 엔드포인트 단원을 참조하십시오.

> **Note**
>
> - `aws:sourceVpce` 조건은 VPC 엔드포인트 리소스의 ARN을 요구하지 않고 VPC 엔드포인트 ID만 요구합니다.
> - 두 번째 문의 모든 Amazon SQS 작업(`sqs:*`)을 거부하도록 다음 예제를 수정하여 특정 VPC 엔드포인트에 대한 모든 작업을 제한할 수 있습니다. 그러나 그러한 정책 문은 모든 작업(대기열 권한 수정을 위한 관리 작업 포함)을 정책에 정의된 특정 VPC 엔드포인트를 통해서만 해야 하도록 규정하게 될 것이므로 앞으로 사용자가 대기열 권한을 수정하지 못하게 될 가능성이 있습니다.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase5",
   "Statement": [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
         "Resource": "arn:aws:sqs:us-east-2:111122223333:queue1"
      },
      {
         "Sid": "2",
```

```
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
               "sqs:SendMessage",
               "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:us-east-2:111122223333:queue1",
            "Condition": {
               "StringNotEquals": {
                  "aws:sourceVpce": "vpce-1a2b3c4d"
               }
            }
         }
      ]
   }
```

Amazon SQS에 대한 임시 보안 자격 증명 사용

자체 보안 자격 증명으로 사용자를 생성하는 것 외에도 IAM을 사용하면 모든 사용자에게 임시 보안 자격 증명을 부여하여 사용자가 AWS 서비스 및 리소스에 액세스할 수 있습니다. AWS 계정이 있는 사용자를 관리할 수 있습니다. 가 없는 시스템의 사용자 AWS 계정 (연동 사용자)를 관리할 수도 있습니다. 또한 AWS 리소스에 액세스하기 위해 생성한 애플리케이션도 "사용자"로 간주될 수 있습니다.

Amazon SQS에 요청할 때 이 임시 보안 자격 증명을 사용할 수 있습니다. API 라이브러리는 요청 인증 시 이 자격 증명을 사용하여 필요한 서명 값을 계산합니다. 만료된 자격 증명으로 요청을 보내면 Amazon SQS는 요청을 거부합니다.

> **Note**
>
> 임시 자격 증명을 기반으로 정책을 설정할 수 없습니다.

사전 조건

1. IAM을 사용하여 임시 보안 자격 증명 생성:
   - 보안 토큰
   - 액세스 키 ID
   - 비밀 액세스 키
2. 문자열을 준비하여 임시 액세스 키 ID와 보안 토큰으로 서명합니다.

3. 자체 보안 액세스 키가 아닌 임시 보안 액세스 키를 사용하여 쿼리 API 요청에 서명합니다.

> **Note**
>
> 서명된 쿼리 API 요청을 제출할 때 자체 액세스 키 ID가 아닌 임시 액세스 키 ID를 사용하여 보안 토큰을 포함시킵니다. 임시 보안 자격 증명에 대한 IAM 지원에 대한 자세한 내용은 IAM 사용 설명서의 AWS 리소스에 대한 임시 액세스 권한 부여를 참조하세요.

임시 보안 자격 증명을 사용하여 Amazon SQS 쿼리 API 작업을 호출하려면

1. 를 사용하여 임시 보안 토큰을 요청합니다 AWS Identity and Access Management. 자세한 내용은 IAM 사용 설명서의 IAM 사용자의 액세스를 활성화하기 위한 임시 보안 자격 증명 생성을 참조하세요.

   IAM은 보안 토큰, 액세스 키 ID 및 보안 액세스 키를 반환합니다.

2. 자체 액세스 키 ID가 아닌 임시 액세스 키 ID를 사용하여 쿼리를 준비하고 보안 토큰을 포함시킵니다. 자체 보안 액세스 키가 아닌 임시 보안 액세스 키를 사용하여 쿼리 요청에 서명합니다.

3. 임시 액세스 키 ID와 보안 토큰으로 서명된 쿼리 문자열을 제출합니다.

   다음 예는 Amazon SQS 요청을 인증하기 위해 임시 보안 자격 증명을 사용하는 방법을 보여줍니다. *AUTHPARAMS*의 구조는 API 요청의 서명에 따라 달라집니다. 자세한 내용은 Amazon Web Services 일반 참조의 AWS API 요청 서명을 참조하세요.

```
https://sqs.us-east-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Attribute.1.Name=VisibilityTimeout
&Attribute.1.Value=40
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

   다음 예제에서는 임시 보안 자격 증명을 사용하여 `SendMessageBatch` 작업을 통해 두 개의 메시지를 전송합니다.

```
https://sqs.us-east-2.amazonaws.com/
?Action=SendMessageBatch
&SendMessageBatchRequestEntry.1.Id=test_msg_001
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201
&SendMessageBatchRequestEntry.2.Id=test_msg_002
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202
&SendMessageBatchRequestEntry.2.DelaySeconds=60
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
&AWSAccessKeyId=AKIAI44QH8DHBEXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

최소 권한 정책이 있는 암호화된 Amazon SQS 대기열의 액세스 관리

Amazon SQS를 사용하여 AWS Key Management Service (KMS)와 통합된 서버 측 암호화(SSE)를 사용하여 애플리케이션 간에 민감한 데이터를 교환할 수 있습니다. Amazon SQS와를 통합 AWS KMS하면 Amazon SQS를 보호하는 키와 다른 AWS 리소스를 보호하는 키를 중앙에서 관리할 수 있습니다.

여러 AWS 서비스가 Amazon SQS로 이벤트를 보내는 이벤트 소스 역할을 할 수 있습니다. 이벤트 소스가 암호화된 Amazon SQS 대기열에 액세스할 수 있도록 하려면 고객 관리형 AWS KMS 키로 대기열을 구성해야 합니다. 그런 다음 키 정책을 사용하여 서비스가 필요한 AWS KMS API 메서드를 사용하도록 허용합니다. 또한 이 서비스에는 대기열이 이벤트를 보낼 수 있도록 액세스를 인증할 수 있는 권한이 필요합니다. Amazon SQS 대기열 및 해당 데이터에 대한 액세스를 제어하는 데 사용할 수 있는 리소스 기반 정책인 Amazon SQS 정책을 사용하여 이를 달성할 수 있습니다.

다음 섹션에서는 Amazon SQS 정책 및 AWS KMS 키 정책을 통해 암호화된 Amazon SQS 대기열에 대한 액세스를 제어하는 방법에 대한 정보를 제공합니다. 이 가이드의 정책은 최소 권한을 획득하는 데 도움이 됩니다.

또한 이 가이드에서는 리소스 기반 정책이 `aws:SourceArn`, `aws:SourceAccount`, `aws:PrincipalOrgID` 글로벌 IAM 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 해결하는 방법도 설명합니다.

주제

- 개요
- Amazon SQS에 대한 최소 권한 키 정책
- DLQ(Dead Letter Queue)에 대한 Amazon SQS 정책 명령문

- 교차 서비스 혼동된 대리자 문제 방지
- IAM Access Analyzer를 사용하여 크로스 계정 액세스를 검토할 수 있습니다.

개요

이 주제에서는 일반적인 사용 사례를 통해 키 정책과 Amazon SQS 대기열 정책을 구축하는 방법을 설명합니다. 이 사용 사례는 다음 이미지에 나와 있습니다.

이 예제에서 메시지 생산자는 Amazon Simple Notification Service(SNS) 주제로, 메시지를 암호화된 Amazon SQS 대기열로 팬아웃하도록 구성되어 있습니다. 메시지 소비자는 AWS Lambda 함수, Amazon Elastic Compute Cloud(EC2) 인스턴스 또는 AWS Fargate 컨테이너와 같은 컴퓨팅 서비스입니다. 그런 다음 실패한 메시지를 DLQ(Dead Letter Queue)로 보내도록 Amazon SQS 대기열을 구성합니다. DLQ는 소비되지 않은 메시지를 구분하여 처리가 실패한 이유를 확인할 수 있으므로 이 방법은 애플리케이션 또는 메시징 시스템을 디버깅하는 데 유용합니다. 이 주제에 정의된 솔루션에서는 Lambda 함수와 같은 컴퓨팅 서비스를 사용하여 Amazon SQS 대기열에 저장된 메시지를 처리합니다. 메시지 소비자가 Virtual Private Cloud(VPC)에 있는 경우 이 안내서에 포함된 `DenyReceivingIfNotThroughVPCE` 정책 명령문을 통해 메시지 수신을 특정 VPC로 제한할 수 있습니다.

> **Note**
>
> 이 안내서에는 정책 명령문 형식의 필수 IAM 권한만 포함되어 있습니다. 정책을 구성하려면 Amazon SQS 정책 또는 AWS KMS 키 정책에 문을 추가해야 합니다. 이 가이드에서는

Amazon SQS 대기열 또는 AWS KMS 키를 생성하는 방법에 대한 지침을 제공하지 않습니다. 이러한 리소스를 생성하는 방법에 대한 지침은 Amazon SQS 대기열 생성 및 키 생성을 참조하세요.
이 안내서에 정의된 Amazon SQS 정책은 메시지를 동일하거나 다른 Amazon SQS 대기열로 직접 리드라이브하는 것을 지원하지 않습니다.

Amazon SQS에 대한 최소 권한 키 정책

이 섹션에서는 Amazon SQS 대기열을 암호화하는 데 사용하는 고객 관리형 키에 AWS KMS 대한의 필수 최소 권한 권한을 설명합니다. 이러한 권한을 사용하면 최소 권한을 구현하면서 의도한 엔터티로만 액세스를 제한할 수 있습니다. 키 정책은 다음과 같은 정책 명령문으로 구성되어야 하며, 이에 대해서는 아래에서 자세히 설명합니다.

- AWS KMS 키에 대한 관리자 권한 부여
- 키 메타데이터에 대한 읽기 전용 액세스 권한 부여
- Amazon SNS에 대기열에 메시지를 게시할 수 있는 Amazon SNS KMS 권한 부여
- 소비자가 대기열에 있는 메시지를 해독할 수 있도록 허용

AWS KMS 키에 대한 관리자 권한 부여

AWS KMS 키를 생성하려면 AWS KMS 키를 배포하는 데 사용하는 IAM 역할에 AWS KMS 관리자 권한을 제공해야 합니다. 이러한 관리자 권한은 다음 `AllowKeyAdminPermissions` 정책 명령문에 정의되어 있습니다. 이 문을 AWS KMS 키 정책에 추가할 때 *<admin-role ARN>*을 AWS KMS 키를 배포하거나 AWS KMS 키를 관리하는 데 사용되는 IAM 역할의 Amazon 리소스 이름(ARN)으로 바꾸거나 둘 다 바꿔야 합니다. 이는 배포 파이프라인의 IAM 역할이거나 AWS 조직에서 조직의 관리자 역할일 수 있습니다.

```
{
  "Sid": "AllowKeyAdminPermissions",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<admin-role ARN>"
    ]
  },
  "Action": [
    "kms:Create*",
    "kms:Describe*",
```

```
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:TagResource",
    "kms:UntagResource",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

> **Note**
>
> AWS KMS 키 정책에서 `Resource` 요소의 값은 이어야 합니다. 즉`*`, "이 AWS KMS 키"입니다. 별표(`*`)는 AWS KMS 키 정책이 연결된 키를 식별합니다.

키 메타데이터에 대한 읽기 전용 액세스 권한 부여

다른 IAM 역할에 키 메타데이터에 대한 읽기 전용 액세스 권한을 부여하려면 키 정책에 `AllowReadAccessToKeyMetaData` 명령문을 추가합니다. 예를 들어 다음 문을 사용하면 감사 목적으로 계정의 모든 AWS KMS 키를 나열할 수 있습니다. 이 문은 AWS 루트 사용자에게 키 메타데이터에 대한 읽기 전용 액세스 권한을 부여합니다. 따라서 계정 내 모든 IAM 보안 주체는 자격 증명 기반 정책이 다음 명령문 `kms:Describe*`, `kms:Get*` 및 `kms:List*`에 나열된 권한을 가질 때 키 메타데이터에 액세스할 수 있습니다. *`<account-ID>`*를 자신의 정보로 바꿉니다.

```
{
  "Sid": "AllowReadAcesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",
```

```
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}
```

Amazon SNS에 대기열에 메시지를 게시할 수 있는 Amazon SNS KMS 권한 부여

Amazon SNS 주제가 암호화된 Amazon SQS 대기열에 메시지를 게시하도록 허용하려면 키 정책에 `AllowSNSToSendToSQS` 정책 명령문을 추가합니다. 이 문은 AWS KMS 키를 사용하여 Amazon SNS Amazon SQS에 부여합니다. *<account-ID>*를 자신의 정보로 바꿉니다.

> **Note**
>
> 문의 `Condition`는 동일한 AWS 계정의 Amazon SNS 서비스에 대한 액세스만 제한합니다.

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
}
```

소비자가 대기열에 있는 메시지를 해독할 수 있도록 허용

다음 `AllowConsumersToReceiveFromTheQueue` 명령문은 암호화된 Amazon SQS 대기열에서 수신한 메시지를 해독하는 데 필요한 권한을 Amazon SQS 메시지 소비자에게 부여합니다. 정책 명령문

을 첨부할 때 *<consumer's runtime role ARN>*을 메시지 소비자의 IAM 런타임 역할 ARN으로 대체하세요.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

Amazon SQS 정책에 대한 최소 권한

이 섹션에서는 이 안내서에서 다루는 사용 사례(예: Amazon SNS에서 Amazon SQS로)에 대한 최소 권한 Amazon SQS 대기열 정책을 안내합니다. 정의된 정책은 `Deny` 및 `Allow` 명령문을 모두 사용하여 의도하지 않은 액세스를 방지하도록 설계되었습니다. `Allow` 명령문은 의도한 엔터티 또는 엔터티에 대한 액세스 권한을 부여합니다. 이 `Deny` 명령문은 정책 조건 내에서 의도한 엔터티를 제외하면서 의도하지 않은 다른 엔터티가 Amazon SQS 대기열에 액세스하는 것을 방지합니다.

Amazon SQS 정책에는 다음과 같은 명령문이 포함되어 있으며, 이에 대해서는 아래에서 자세히 설명합니다.

- Amazon SQS 관리 권한 제한
- 지정된 조직의 Amazon SQS 대기열 작업을 제한합니다.
- 소비자에게 Amazon SQS 권한 부여
- 전송 중 암호화 적용
- 특정 Amazon SNS 주제로의 메시지 전송 제한
- (선택 사항) 메시지 수신을 특정 VPC 엔드포인트로 제한

Amazon SQS 관리 권한 제한

다음 `RestrictAdminQueueActions` 정책 명령문에서는 Amazon SQS 관리 권한을 대기열을 배포하거나 대기열을 관리하는 데 사용하는 IAM 역할 또는 두 가지 모두에 사용하는 역할로만 제한합니다.

*<placeholder values>*을 자체 정보로 바꾸세요. Amazon SQS 대기열을 배포하는 데 사용되는 IAM 역할의 ARN과 Amazon SQS 관리 권한이 있어야 하는 관리자 역할의 ARN을 지정합니다.

```
{
  "Sid": "RestrictAdminQueueActions",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes"
  ],
  "Resource": "<SQS Queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": [
        "arn:aws:iam::<account-id>:role/<deployment-role-name>",
        "<admin-role ARN>"
      ]
    }
  }
}
```

지정된 조직의 Amazon SQS 대기열 작업을 제한합니다.

외부 액세스(AWS 조직 외부 엔터티의 액세스)로부터 Amazon SQS 리소스를 보호하려면 다음 명령문을 사용하세요. 이 명령문은 Amazon SQS 대기열 액세스를 `Condition`에 지정된 조직으로 제한합니다. *<SQS queue ARN>*을 Amazon SQS 대기열을 배포하는 데 사용된 IAM 역할의 ARN으로 바꾸고, *<org-id>*를 조직 ID로 바꾸어야 합니다.

```
{
  "Sid": "DenyQueueActionsOutsideOrg",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:ChangeMessageVisibility",
```

```
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:PrincipalOrgID": [
        "<org-id>"
      ]
    }
  }
}
```

소비자에게 Amazon SQS 권한 부여

Amazon SQS 대기열에서 메시지를 수신하려면 메시지 소비자에게 필요한 권한을 제공해야 합니다. 다음 정책 명령문은 지정한 소비자에게 Amazon SQS 대기열의 메시지를 소비하는 데 필요한 권한을 부여합니다. Amazon SQS 정책에 이 명령문을 추가할 때 *<consumer's IAM runtime role ARN>*을 소비자가 사용하는 IAM 런타임 역할의 ARN으로 바꾸고, *<SQS queue ARN>*을 Amazon SQS 대기열을 배포하는 데 사용되는 IAM 역할의 ARN으로 바꾸어야 합니다.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": "<consumer's IAM execution role ARN>"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:GetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>"
}
```

다른 엔터티가 Amazon SQS 대기열에서 메시지를 수신하지 못하도록 하려면 Amazon SQS 대기열 정책에 `DenyOtherConsumersFromReceiving` 명령문을 추가하세요. 이 명령문은 메시지 소비를 지

정한 소비자로 제한하여 다른 소비자는 자격 증명 권한에 따라 액세스 권한이 있는 경우에도 액세스하지 못하도록 합니다. *<SQS queue ARN>*과 *<consumer's runtime role ARN>*을 자신의 정보로 바꾸어야 합니다.

```
{
  "Sid": "DenyOtherConsumersFromReceiving",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": "<consumer's execution role ARN>"
    }
  }
}
```

전송 중 암호화 적용

다음 `DenyUnsecureTransport` 정책 명령문은 소비자와 생산자가 보안 채널(TLS 연결)을 사용하여 Amazon SQS 대기열에서 메시지를 보내고 받도록 강제합니다. *<SQS queue ARN>*을 Amazon SQS 대기열을 배포하는 데 사용된 IAM 역할의 ARN으로 바꾸어야 합니다.

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
```

```
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}
```

특정 Amazon SNS 주제로의 메시지 전송 제한

다음 `AllowSNSToSendToTheQueue` 정책 명령문은 특정 Amazon SNS 주제가 Amazon SQS 대기열에 메시지를 전송하도록 허용합니다. *<SQS queue ARN>*을 Amazon SQS 대기열을 배포하는 데 사용된 IAM 역할의 ARN으로 바꾸고, *<SNS topic ARN>*을 Amazon SNS 주제 ARN으로 바꾸어야 합니다.

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

다음 `DenyAllProducersExceptSNSFromSending` 정책 명령문은 다른 생산자가 대기열에 메시지를 보내는 것을 방지합니다. *<SQS queue ARN>* 및 *<SNS topic ARN>*을 자신의 정보로 바꿉니다.

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
```

```
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(선택 사항) 메시지 수신을 특정 VPC 엔드포인트로 제한

메시지 수신을 특정 VPC 엔드포인트로만 제한하려면 Amazon SQS 대기열 정책에 다음 정책 명령문을 추가하세요. 이 명령문은 메시지가 원하는 VPC 엔드포인트에서 전송되지 않는 한 메시지 소비자가 대기열에서 메시지를 수신하지 못하도록 합니다. *<SQS queue ARN>*을 Amazon SQS 대기열을 배포하는 데 사용되는 IAM 역할의 ARN으로 바꾸고, *<vpce_id>*를 VPC 엔드포인트의 ID로 바꿉니다.

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

DLQ(Dead Letter Queue)에 대한 Amazon SQS 정책 명령문

명령문 ID로 식별되는 다음 정책 명령문을 DLQ 액세스 정책에 추가합니다.

- `RestrictAdminQueueActions`
- `DenyQueueActionsOutsideOrg`
- `AllowConsumersToReceiveFromTheQueue`
- `DenyOtherConsumersFromReceiving`
- `DenyUnsecureTransport`

DLQ 액세스 정책에 이전 정책 명령문을 추가하는 것 외에도 다음 섹션에 설명된 대로 메시지 전송을 Amazon SQS 대기열로 제한하는 명령문도 추가해야 합니다.

Amazon SQS 대기열로 메시지 전송 제한

동일한 계정의 Amazon SQS 대기열로만 액세스를 제한하려면 DLQ 대기열 정책에 다음 `DenyAnyProducersExceptSQS` 정책 명령문을 추가합니다. 이 명령문은 기본 대기열을 생성하기 전에 DLQ를 배포해야 하므로 메시지 전송을 특정 대기열로 제한하지 않습니다. 따라서 DLQ를 생성할 때 Amazon SQS ARN을 알 수 없습니다. 하나의 Amazon SQS 대기열로만 액세스를 제한해야 하는 경우, Amazon SQS 소스 대기열의 ARN을 사용하여(아는 경우) `Condition`의 `aws:SourceArn`을 수정합니다.

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

> **Important**
>
> 이 안내서에 정의된 Amazon SQS 대기열 정책은 `sqs:PurgeQueue` 작업을 특정 IAM 역할로 제한하지 않습니다. `sqs:PurgeQueue` 작업을 사용하면 Amazon SQS 대기열의 모든 메시지를 삭제할 수 있습니다. 또한 이 작업을 사용하여 Amazon SQS 대기열을 교체하지 않고도 메시지 형식을 변경할 수 있습니다. 애플리케이션을 디버깅할 때 Amazon SQS 대기열을 지워 오류가 있을 수 있는 메시지를 제거할 수 있습니다. 애플리케이션을 테스트할 때 Amazon SQS 대기열을 통해 대량의 메시지를 전송한 다음, 대기열을 제거하여 프로덕션에 들어가기 전에 새로 시작할 수 있습니다. 이 작업을 특정 역할로 제한하지 않는 이유는 Amazon SQS 대기열을 배포할 때 이 역할을 알지 못할 수 있기 때문입니다. 대기열을 제거하려면 이 권한을 역할의 ID 기반 정책에 추가해야 합니다.

교차 서비스 혼동된 대리자 문제 방지

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에 작업을 수행하도록 강요할 수 있는 보안 문제입니다. 이를 방지하기 위해는 타사(교차 계정이라고 함) 또는 기타 AWS 서비스(교차 서비스라고 함)에 계정의 리소스에 대한 액세스 권한을 제공하는 경우 계정을 보호하는 데 도움이 되는 도구를 AWS 제공합니다. 이 섹션의 정책 명령문은 교차 서비스 혼동된 대리자 문제를 방지하는 데 도움이 될 수 있습니다.

교차 서비스 가장은 한 서비스(호출하는 서비스)가 다른 서비스(호출되는 서비스)를 직접적으로 호출할 때 발생할 수 있습니다. 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이 문제를 방지하기 위해 이 게시물에 정의된 리소스 기반 정책은 `aws:SourceArn`, `aws:SourceAccount`, `aws:PrincipalOrgID` 글로벌 IAM 조건 컨텍스트 키를 사용합니다. 이렇게 하면 서비스가 AWS Organizations의 특정 리소스, 특정 계정 또는 특정 조직에 대해 갖는 권한이 제한됩니다.

IAM Access Analyzer를 사용하여 크로스 계정 액세스를 검토할 수 있습니다.

AWS IAM Access Analyzer를 사용하여 Amazon SQS 대기열 정책 및 AWS KMS 키 정책을 검토하고 Amazon SQS 대기열 또는 AWS KMS 키가 외부 엔터티에 액세스 권한을 부여할 때 알림을 받을 수 있습니다. IAM Access Analyzer는 신뢰 영역 외부의 엔터티와 공유되는 조직 및 계정의 리소스를 식별하는 데 도움이 됩니다. 이 신뢰 영역은 IAM Access Analyzer를 활성화할 때 지정하는 AWS Organizations 내의 AWS 계정 또는 조직일 수 있습니다.

IAM Access Analyzer는 로직 기반 추론을 사용하여 AWS 환경의 리소스 기반 정책을 분석하여 외부 보안 주체와 공유되는 리소스를 식별합니다. 신뢰 영역 외부에서 공유되는 리소스의 각 인스턴스에 대해 Access Analyzer는 결과를 생성합니다. 결과에는 액세스에 대한 정보와 액세스 권한이 부여되는 외부 보안 주체에 대한 정보가 포함됩니다. 결과를 검토하여 액세스가 의도한 안전한 액세스인지 또는 액세스가 의도하지 않은 보안 위험인지 확인할 수 있습니다. 의도하지 않은 액세스가 발생한 경우 영향을 받는 정책을 검토하고 수정합니다. AWS IAM Access Analyzer가 AWS 리소스에 대한 의도하지 않은 액세스를 식별하는 방법에 대한 자세한 내용은이 블로그 게시물을 참조하세요.

AWS IAM Access Analyzer에 대한 자세한 내용은 AWS IAM Access Analyzer 설명서를 참조하세요.

Amazon SQS API 권한: 작업 및 리소스 참조

액세스 제어를 설정하고 IAM 자격 증명에 연결할 수 있는 권한 정책을 작성할 때 다음 표를 참조로 사용할 수 있습니다. 는 각 Amazon Simple Queue Service 작업, 작업을 수행할 수 있는 권한을 부여할 수 있는 해당 작업, 권한을 부여할 수 있는 리소스가 포함되어 있습니다. AWS

정책의 `Action` 필드에서 작업을 지정하고, 정책의 `Resource` 필드에서 리소스 값을 지정합니다. 작업을 지정하려면 `sqs:` 접두사 다음에 작업 이름을 사용합니다(예: `sqs:CreateQueue`).

현재 Amazon SQS는 IAM에서 사용할 수 있는 전역 조건 문맥 키를 지원합니다.

Amazon Simple Queue Service API 및 작업에 필요한 권한

AddPermission

작업: `sqs:AddPermission`

리소스: `arn:aws:sqs:region:account_id:queue_name`

ChangeMessageVisibility

작업: `sqs:ChangeMessageVisibility`

리소스: `arn:aws:sqs:region:account_id:queue_name`

ChangeMessageVisibilityBatch

작업: `sqs:ChangeMessageVisibilityBatch`

리소스: `arn:aws:sqs:region:account_id:queue_name`

CreateQueue

작업: `sqs:CreateQueue`

리소스: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessage

작업: `sqs:DeleteMessage`

리소스: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessageBatch

작업: `sqs:DeleteMessageBatch`

리소스: `arn:aws:sqs:region:account_id:queue_name`

DeleteQueue

작업: `sqs:DeleteQueue`

리소스: `arn:aws:sqs:region:account_id:queue_name`

GetQueueAttributes

작업: `sqs:GetQueueAttributes`

리소스: `arn:aws:sqs:region:account_id:queue_name`

GetQueueUrl

작업: `sqs:GetQueueUrl`

리소스: `arn:aws:sqs:region:account_id:queue_name`

ListDeadLetterSourceQueues

작업: `sqs:ListDeadLetterSourceQueues`

리소스: `arn:aws:sqs:region:account_id:queue_name`

ListQueues

작업: `sqs:ListQueues`

리소스: `arn:aws:sqs:region:account_id:queue_name`

ListQueueTags

작업: `sqs:ListQueueTags`

리소스: `arn:aws:sqs:region:account_id:queue_name`

PurgeQueue

작업: `sqs:PurgeQueue`

리소스: `arn:aws:sqs:region:account_id:queue_name`

ReceiveMessage

작업: `sqs:ReceiveMessage`

리소스: `arn:aws:sqs:region:account_id:queue_name`

RemovePermission

작업: `sqs:RemovePermission`

리소스: `arn:aws:sqs:region:account_id:queue_name`

SendMessage 및 SendMessageBatch

작업: `sqs:SendMessage`

리소스: `arn:aws:sqs:region:account_id:queue_name`

SetQueueAttributes

작업: `sqs:SetQueueAttributes`

리소스: `arn:aws:sqs:region:account_id:queue_name`

TagQueue

작업: `sqs:TagQueue`

리소스: `arn:aws:sqs:region:account_id:queue_name`

UntagQueue

작업: `sqs:UntagQueue`

리소스: `arn:aws:sqs:region:account_id:queue_name`

Amazon SQS의 로깅 및 모니터링

Amazon Simple Queue Service는 사용자AWS CloudTrail, 역할 또는가 수행한 작업에 대한 레코드를 제공하는 서비스인와 통합됩니다 AWS 서비스. CloudTrail은 Amazon SQS에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 Amazon SQS 콘솔의 호출과 Amazon SQS API 작업에 대한 코드 호출이 포함됩니다. CloudTrail에서 수집한 정보를 사용하여 Amazon SQS에 수행된 요청, 요청이 수행된 IP 주소, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트 사용자로 했는지 사용자 보안 인증으로 했는지 여부.
- IAM Identity Center 사용자를 대신하여 요청이 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자에 대한 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화되며 CloudTrail 이벤트 기록에 자동으로 액세스할 수 있습니다. CloudTrail 이벤트 기록은 지난 90일 간 AWS 리전의 관리 이벤트에 대해 보기, 검색 및

다운로드가 가능하고, 수정이 불가능한 레코드를 제공합니다. 자세한 설명은 AWS CloudTrail 사용 설명서의 CloudTrail 이벤트 기록 작업을 참조하세요. Event history(이벤트 기록) 보기는 CloudTrail 요금이 부과되지 않습니다.

AWS 계정 지난 90일 동안의 이벤트를 지속적으로 기록하려면 추적 또는 CloudTrail Lake 이벤트 데이터 스토어를 생성합니다.

Amazon CloudWatch 경보

지정한 기간 동안 단일 지표를 모니터링하고 여러 기간 동안 정의된 임계값을 기준으로 지표 값을 기반으로 하나 이상의 작업을 수행합니다. 예를 들어 Amazon SNS 주제에 알림을 보내거나 Amazon SQS 대기열에 메시지를 보내는 작업을 트리거하도록 CloudWatch 경보를 구성할 수 있습니다. CloudWatch 경보는 단순히 특정 상태이기 때문에 작업을 수행하지 않습니다. 상태가 변경되고 정의된 기간 동안 해당 상태를 유지해야 합니다.

자세한 내용은 Amazon SQS 지표를 모니터링하여 CloudWatch 경보 생성 및 Amazon CloudWatch를 사용하여 배달 못한 편지 대기열에 대한 경보 생성 섹션을 참조하세요.

Amazon CloudWatch Logs

CloudWatch Logs로 로그를 전송하도록 메시지를 처리하는 애플리케이션 또는 Lambda 함수를 구성하여 Amazon SQS와 관련된 로그 파일을 모니터링, 저장 및 액세스합니다. 이러한 로그를 사용하여 메시지 처리를 분석하고, 문제를 디버깅하고, Amazon SQS 워크플로의 성능을 모니터링할 수 있습니다.

자세한 내용은 를 사용하여 Amazon Simple Queue Service API 호출 로깅 AWS CloudTrail 단원을 참조하십시오.

Amazon CloudWatch Events

Amazon CloudWatch Events를 사용하여 AWS 환경의 변경 사항 또는 특정 이벤트를 감지하고 Amazon SQS 대기열로 라우팅합니다. 이를 통해 이벤트 데이터를 캡처하거나, 워크플로를 트리거하거나, 나중에 처리할 이벤트를 저장할 수 있습니다.

자세한 내용은 이 가이드의 Amazon EventBridge를 사용하여 AWS 서비스에서 Amazon SQS로 알림 자동화 Amazon SQS 섹션과 Amazon EventBridge 사용 안내서의 EventBridge는 Amazon CloudWatch Events의 진화를 참조하세요.

AWS CloudTrail 로그

CloudTrail은 사용자, 역할 또는가 Amazon SQS에서 수행한 작업에 대한 세부 레코드를 캡처합니다 AWS 서비스. 이러한 로그를 사용하면 , `SendMessage` `ReceiveMessage`또는와 같은 API 호

출을 추적하고 요청을 수행한 사람`DeleteQueue`, 요청이 발생한 시간 및 발신 IP 주소와 같은 주요 세부 정보를 제공할 수 있습니다.

자세한 내용은 를 사용하여 Amazon Simple Queue Service API 호출 로깅 AWS CloudTrail 단원을 참조하십시오.

AWS Trusted Advisor

Trusted Advisor 는 AWS 고객에게 서비스를 제공하여 개발된 모범 사례를 사용하여 Amazon SQS 사용량을 최적화합니다. Amazon SQS 대기열을 검토하고 보안을 강화하고, 메시지 처리 신뢰성을 개선하고, 비용을 절감하기 위한 실행 가능한 권장 사항을 제공합니다. 예를 들어 배달 못한 편지 대기열을 활성화하거나 보안 작업을 보장하기 위해 대기열 액세스 정책을 개선하는 것이 좋습니다.

자세한 내용은지원 사용 설명서의 AWS Trusted Advisor를 참조하세요.

CloudTrail 추적

CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 를 사용하여 생성된 모든 추적 AWS Management Console 은 다중 리전입니다. AWS CLI를 사용하여 단일 리전 또는 다중 리전 추적을 생성할 수 있습니다. 계정의 모든에서 활동을 캡처하므로 다중 리전 추적 AWS 리전 을 생성하는 것이 좋습니다. 단일 리전 추적을 생성하는 경우 추적의 AWS 리전에 로깅된 이벤트만 볼 수 있습니다. 추적에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Creating a trail for your AWS 계정 및 Creating a trail for an organization을 참조하세요.

CloudTrail에서 추적을 생성하여 진행 중인 관리 이벤트의 사본 하나를 Amazon S3 버킷으로 무료로 전송할 수는 있지만, Amazon S3 스토리지 요금이 부과됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요. Amazon S3 요금에 대한 자세한 내용은 Amazon S3 요금을 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어

CloudTrail Lake를 사용하면 이벤트에 대해 SQL 기반 쿼리를 실행할 수 있습니다. CloudTrail Lake는 행 기반 JSON 형식의 기존 이벤트를 Apache ORC 형식으로 변환합니다. ORC는 빠른 데이터 검색에 최적화된 열 기반 스토리지 형식입니다. 이벤트는 이벤트 데이터 스토어로 집계되며, 이벤트 데이터 스토어는 고급 이벤트 선택기를 적용하여 선택한 기준을 기반으로 하는 변경 불가능한 이벤트 컬렉션입니다. 이벤트 데이터 스토어에 적용하는 선택기는 어떤 이벤트가 지속되고 쿼리할 수 있는지 제어합니다. CloudTrail Lake에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 AWS CloudTrail Lake 작업을 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어 및 쿼리에는 비용이 발생합니다. 이벤트 데이터 스토어를 생성할 때 이벤트 데이터 스토어에 사용할 요금 옵션을 선택합니다. 요금 옵션에 따라 이벤트 모으기

및 저장 비용과 이벤트 데이터 스토어의 기본 및 최대 보존 기간이 결정됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

를 사용하여 Amazon Simple Queue Service API 호출 로깅 AWS CloudTrail

CloudTrail을 사용하면 데이터 이벤트와 관리 이벤트라는 두 가지 이벤트 유형을 사용하여 Amazon SQS 작업을 로깅하고 모니터링할 수 있습니다. 이렇게 하면 계정에서 Amazon SQS 활동을 쉽게 추적하고 감사할 수 있습니다.

CloudTrail의 Amazon SQS 데이터 이벤트

데이터 이벤트는 리소스에서 또는 리소스에서 수행되는 리소스 작업에 대한 정보를 제공합니다(예: Amazon SQS 객체로 메시지 전송). 이를 데이터 영역 작업이라고도 합니다. 데이터 이벤트가 대량 활동인 경우도 있습니다. 기본적으로 CloudTrail은 데이터 이벤트를 로깅하지 않습니다. CloudTrail 이벤트 기록은 데이터 이벤트를 기록하지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

CloudTrail 콘솔 AWS CLI또는 CloudTrail API 작업을 사용하여 Amazon SQS 리소스 유형에 대한 데이터 이벤트를 로깅할 수 있습니다. 데이터 이벤트를 로깅하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Logging data events with the AWS Management Console 및 Logging data events with the AWS Command Line Interface를 참조하세요.

CloudTrail을 사용하여 Amazon SQS 데이터 이벤트를 로깅하려면 고급 이벤트 선택기를 사용하여 로깅하려는 특정 Amazon SQS 리소스 또는 작업을 구성해야 합니다. 리소스 유형을 포함하여 대기열 관련 작업을 `AWS::SQS::Queue` 캡처합니다. `eventName` (예: `SendMessage` 이벤트)와 같은 필터를 사용하여 로깅 기본 설정을 더욱 구체화할 수 있습니다. 자세한 내용은 CloudTrail API 참조의 AdvancedEventSelector를 참조하세요. CloudTrail

데이터 이벤트 유형(콘솔)	resources.type 값	CloudTrail에 로깅되는 데이터 API
Amazon SQS 대기열	`AWS::SQS::Queue`	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch

데이터 이벤트 유형(콘솔)	resources.type 값	CloudTrail에 로깅되는 데이터 API
		• GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

고급 이벤트 선택기를 사용하여 필드를 필터링하고 중요한 이벤트만 로깅합니다. 이러한 필드에 대한 자세한 내용은 AWS CloudTrail API 참조의 AdvancedFieldSelector 섹션을 참조하세요.

CloudTrail의 Amazon SQS 관리 이벤트

관리 이벤트는의 리소스에서 수행되는 관리 작업에 대한 정보를 제공합니다 AWS 계정. 이를 컨트롤 플레인 작업이라고도 합니다. 기본적으로 CloudTrail은 관리 이벤트를 로깅합니다.

Amazon SQS는 다음 컨트롤 플레인 작업을 관리 이벤트로 CloudTrail에 로깅합니다.

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTasks
- PurgeQueue
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Amazon SQS 이벤트 예제

이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청된 API 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 추적이 아니므로 이벤트가 특정 순서로 표시되지 않습니다.

다음 예제는 SendMessage 작업을 시연하는 CloudTrail 이벤트를 보여줍니다.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EXAMPLE_PRINCIPAL_ID",
    "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/SessionName",
    "accountId": "123456789012",
    "accessKeyId": "ACCESS_KEY_ID",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed",
        "accountId": "123456789012",
        "userName": "RoleToBeAssumed"
      },
      "attributes": {
        "creationDate": "2023-11-07T22:13:06Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2023-11-07T23:59:11Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessage",
  "awsRegion": "ap-southeast-4",
  "sourceIPAddress": "10.0.118.80",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/
linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/
pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue",
    "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageDeduplicationId": "MsgDedupIdSdk1ae1958f2-bbe8-4442-83e7-4916e3b035aa",
    "messageGroupId": "MsgGroupIdSdk16"
```

```
  },
  "responseElements": {
    "mD5OfMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
    "mD5OfMessageSystemAttributes": "f88f0587f951b7f5551f18ae699c3a9d",
    "messageId": "93bb6e2d-1090-416c-81b0-31eb1faa8cd8",
    "sequenceNumber": "18881790870905840128"
  },
  "requestID": "c4584600-fe8a-5aa3-a5ba-1bc42f055fae",
  "eventID": "98c735d8-70e0-4644-9432-b6ced4d791b1",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::SQS::Queue",
      "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sqs.ap-southeast-4.amazonaws.com"
  }
```

> **Note**
>
> `ListQueues` 작업은 특정 리소스에서 작동하지 않기 때문에 고유한 사례입니다. 따라서 ARN 필드에는 대기열 이름이 포함되지 않으며 대신 와일드카드(*)를 사용합니다.

CloudTrail 레코드 콘텐츠에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail record contents를 참조하세요.

CloudWatch를 사용한 Amazon SQS 쿼리 모니터링

Amazon SQS 및 Amazon CloudWatch는 통합되어 있으므로 CloudWatch를 사용하여 Amazon SQS 대기열 지표를 확인하고 분석할 수 있습니다. Amazon SQS 콘솔, CloudWatch 콘솔, AWS CLI 또는 CloudWatch API를 사용하여 대기열의 지표를 확인하고 분석할 수 있습니다. Amazon SQS 지표에 대해 CloudWatch 경보를 설정할 수도 있습니다.

Amazon SQS 대기열에 대한 CloudWatch 지표는 자동으로 수집되어 1분 간격으로 CloudWatch에 푸시됩니다. 이러한 지표는 활성화에 대한 CloudWatch 지침을 충족하는 모든 쿼리에서 수집됩니다. CloudWatch는 대기열에 메시지가 포함되어 있거나 대기열에 액세스하는 작업이 있을 경우 대기열을 최대 6시간 동안 활성 상태로 간주합니다.

Amazon SQS 대기열이 6시간 이상 비활성 상태인 경우 Amazon SQS 서비스는 절전 모드로 간주되어 CloudWatch 서비스에 대한 지표 전송을 중단합니다. Amazon SQS 대기열이 비활성 상태였던 기간 동안에는 Amazon SQS의 CloudWatch 지표에서 누락된 데이터 또는 0을 나타내는 데이터를 시각화할 수 없습니다.

> **Note**
>
> - 대기열에 대해 API를 직접 호출하는 사용자에게 권한이 없고 요청이 실패하면 Amazon SQS 대기열을 활성화할 수 있습니다.
> - Amazon SQS 콘솔은 대기열의 페이지가 열릴 때 `GetQueueAttributes` API 직접 호출을 수행합니다. `GetQueueAttributes` API 요청은 대기열을 활성화합니다.
> - 대기열이 비활성 상태에서 활성화되면 CloudWatch 지표에서 최대 15분의 지연이 발생합니다.
> - CloudWatch에서 보고되는 Amazon SQS 지표에는 요금이 부과되지 않습니다. 이것은 Amazon SQS 서비스의 일부로 제공됩니다.
> - CloudWatch 지표는 표준 대기열과 FIFO 대기열 모두에서 지원됩니다.

Amazon SQS의 CloudWatch 지표 액세스

Amazon SQS 및 Amazon CloudWatch는 통합되어 있으므로 CloudWatch를 사용하여 Amazon SQS 대기열 지표를 확인하고 분석할 수 있습니다. Amazon SQS 콘솔, CloudWatch 콘솔, AWS CLI 또는 CloudWatch API를 사용하여 대기열의 지표를 확인하고 분석할 수 있습니다. Amazon SQS 지표에 대해 CloudWatch 경보를 설정할 수도 있습니다.

Amazon SQS 콘솔 사용

Amazon SQS 콘솔을 사용하여 최대 10개의 Amazon SQS 대기열에 대한 지표에 액세스하고 분석할 수 있습니다.

1. Amazon SQS 콘솔에 로그인합니다.
2. 대기열 목록에서 측정치에 액세스할 대기열에 해당하는 상자를 선택합니다. 최대 10개의 측정치를 표시할 수 있습니다.

3. 모니터링 탭을 선택합니다.

   [SQS metrics] 섹션에 다양한 그래프가 표시됩니다.

4. 특정 그래프가 나타내는 정보를 이해하려면, 원하는 그래프 옆에 있는

   ℹ

   위에 마우스 포인터를 놓거나 Amazon SQS의 CloudWatch 지표 액세스 단원을 참조하십시오.

5. 동시에 모든 그래프의 시간 범위를 변경하려면, [Time Range]에서 원하는 시간 범위(예: [Last Hour])를 선택합니다.

6. 개별 그래프의 추가 통계를 보려면 그래프를 선택합니다.

7. Amazon SQS 모니터링 세부 정보 대화 상자에서 통계를 선택합니다(예: 합계). 지원되는 통계 목록은 Amazon SQS의 CloudWatch 지표 액세스 단원을 참조하십시오.

8. 개별 그래프가 표시되는 시간 범위와 시간 간격을 변경하려면(예: 시간 범위를 지난 5분이 아닌 지난 24시간으로 표시), 그래프의 대화 상자를 계속 표시한 상대로 [Time Range]에서 원하는 시간 범위(예: [Last 24 Hours])를 선택합니다. [Period]에서 원하는 기간을 지정된 시간 범위 내로 선택합니다(예: [1 Hour]). 그래프 확인이 끝나면, [Close]를 선택합니다.

9. (선택 사항) 추가 Amazon SQS 기능을 사용하려면, 모니터링 탭에서 모든 Amazon SQS 측정치 보기를 선택한 후 Amazon CloudWatch 콘솔 사용 절차의 지침을 따릅니다.

Amazon CloudWatch 콘솔 사용

CloudWatch 콘솔을 사용하여 Amazon SQS 지표에 액세스하고 분석합니다.

1. CloudWatch 콘솔에 로그인합니다.

2. 탐색 창에서 [Metrics]를 선택합니다.

3. [SQS] 측정치 네임스페이스를 선택합니다.

4. [Queue Metrics] 측정치 차원을 선택합니다.

5. 이제 Amazon SQS 지표를 검토할 수 있습니다.

- 지표를 정렬하려면 열 머리글을 사용합니다.
- 측정치를 그래프로 표시하려면 측정치 옆에 있는 확인란을 선택합니다.
- 지표로 필터링하려면 지표 이름을 선택한 후 검색에 추가를 선택합니다.

자세한 내용 및 추가 옵션은 Amazon CloudWatch 사용 설명서의 그래프 지표 및 Amazon CloudWatch 대시보드 사용을 참조하세요.

사용 AWS Command Line Interface

를 사용하여 Amazon SQS 지표에 액세스하려면 `get-metric-statistics` 명령을 AWS CLI실행합니다.

자세한 내용은 Amazon CloudWatch 사용 설명서의 지표에 대한 통계 얻기를 참조하세요.

CloudWatch API 사용

CloudWatch API를 사용하여 Amazon SQS 지표에 액세스하려면 `GetMetricStatistics` 작업을 사용합니다.

자세한 내용은 Amazon CloudWatch 사용 설명서의 지표에 대한 통계 얻기를 참조하세요.

Amazon SQS 지표를 모니터링하여 CloudWatch 경보 생성

CloudWatch를 사용하면 지표가 지정된 임계값에 도달하면 경보를 트리거할 수 있습니다. 예를 들어, `NumberOfMessagesSent` 지표에 대한 경보를 생성할 수 있습니다. 예를 들어 1시간 동안 `MyQueue` 대기열에 메시지 100개 이상이 전송된 경우 알림 이메일을 발송합니다. 자세한 정보는 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 경보 생성을 참조하세요.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/ CloudWatch 콘솔을 엽니다.
2. 알람를 선택한 다음 알람 생성을 선택합니다.
3. 경보 생성 대화 상자의 지표 선택 섹션에서 지표 찾아보기, SQS를 선택합니다.
4. SQS > 대기열 지표에서 경보를 설정할 QueueName과 지표 이름을 선택한 후 다음을 선택합니다. 사용 가능한 지표 목록은 Amazon SQS의 CloudWatch 지표 액세스 단원을 참조하십시오.

   다음 예에서는 `NumberOfMessagesSent` 대기열에 대한 `MyQueue` 지표 경보를 선택합니다. 전송된 메시지 수가 100개를 초과하면 경보가 트리거됩니다.
5. 경보 생성 대화 상자의 경보 정의 섹션에서 다음 작업을 수행합니다.

   a. 경보 임계값에서 경보의 이름 및 설명을 입력합니다.
   b. [is]를 [> 100]으로 설정합니다.
   c. 대상을 1 out of 1 datapoints(1 중 1 데이터 포인트)로 지정합니다.
   d. 경보 미리 보기에서 기간을 1시간으로 지정합니다.
   e. 통계를 표준, 합계로 지정합니다.
   f. 작업에서 이 경보가 발생할 경우 항상을 상태가 ALARM입니다.로 지정합니다.

경보 트리거 시 CloudWatch로부터 알림을 받으려면 기존 Amazon SNS 주제를 선택하거나 새 목록을 선택하여 이메일 주소(쉼표로 구분)를 입력합니다.

> **Note**
>
> 새 Amazon SNS 주제를 생성하는 경우 알림을 받기 전에 이메일 주소를 검증해야 합니다. 이메일 검증 전에 경보 상태가 변경되면 알림이 전송되지 않습니다.

6. 경보 생성을 선택합니다.

   경보가 생성됩니다.

Amazon SQS의 CloudWatch 지표 액세스

Amazon SQS는 CloudWatch에 다음 지표를 전송합니다.

> **Note**
>
> 일부 지표의 경우, Amazon SQS의 분산 아키텍처로 인해 그 결과는 거의 비슷합니다. 대부분의 경우 그 수는 대기열에 있는 메시지의 실제 개수에 근접해야 합니다.

Amazon SQS 지표

Amazon SQS는 `AWS/SQS` 네임스페이스 아래의 Amazon CloudWatch에 운영 지표를 자동으로 게시합니다. 이러한 지표는 대기열 상태 및 성능을 모니터링하는 데 도움이 됩니다. SQS의 분산 특성으로 인해 많은 값은 근사치이지만 대부분의 운영 결정에 충분히 정확합니다.

> **Note**
>
> - 모든 지표는 대기열이 활성 상태일 때만 음수가 아닌 값을 내보냅니다.
> - 일부 지표(예: `SentMessageSize`)는 하나 이상의 메시지가 전송될 때까지 내보내지지 않습니다.

지표	설명	단위	보고 동작	주요 참고 사항
`ApproximateAgeOfOldestMessage`	대기열에서 처리되지 않은 가장 오래된 메시지의 수명입니다.	초	대기열에 하나 이상의 활성 메시지가 포함된 경우 보고됩니다.	• 표준 대기열의 경우 메시지가 3회 이상 수신되고 삭제되지 않으면 SQS는 메시지를 대기열의 뒤쪽으로 이동합니다. 그런 다음 지표는 수신 임계값을 초과하지 않은 다음 메시지의 수명을 반영합니다. 이 재정렬은 리드라이브 정책이 있는 경우에도 발생합니다. • 독약 메시지(반복적으로 수신되었지만 삭제되지 않은 메시지)는 성공적으로 처리될 때까지이 지표에서 제외됩니다. • 를 초과한 후 메시지가 DLQ로 이동되면 `maxReceiveCount` 수명이 재설정됩니다. 이 경우 DLQ의 지표는 메시지가 원래 전송된 시간이 아니라 메시지가 이동된 시간을 반영합니다. • FIFO 대기열은 순서를 유지하기 위해 메시지를 재정렬하지 않습니다. 실패한 메시지는 삭제되거나 만료될 때까지 메시지 그룹을 차단합니다. DLQ가 구성된 경우 수신 임계값이 충족되면 메시지가 전송됩니다.
`ApproximateNumberOfGroupsWi`	FIFO 전용입니다. 하나 이상의 진행 중	개수	FIFO 대기열이 활성 상태	• 메시지는 소비자가 대기열에서 수신했지만 아직 삭제되

지표	설명	단위	보고 동작	주요 참고 사항
thInflighttMessages	인 메시지가 있는 메시지 그룹의 수입니다.		인 경우 보고됩니다.	거나 만료되지 않은 후 진행 중인 것으로 간주됩니다. • 이 지표는 FIFO 대기열 처리량 문제를 해결하고 최적화하는 데 도움이 됩니다. 값이 높으면 일반적으로 강력한 동시성을 나타냅니다. • 대기열에 큰 백로그가 있고 이 값이 여전히 낮은 경우 소비자 규모를 조정하거나 활성 메시지 그룹 수를 늘리는 것이 좋습니다. • 처리량 및 처리 중 제한은 섹션을 참조하세요Amazon SQS 할당량.
ApproximateNumberOfMessagesDelayed	대기열에서 지연되어 즉시 검색할 수 없는 메시지 수입니다.	개수	지연된 메시지가 대기열에 있는 경우 보고됩니다.	• 기본 지연으로 구성된 대기열과 DelaySeconds 파라미터로 전송된 개별 메시지에 적용됩니다. • 지연된 메시지는 지연 기간이 만료될 때까지 소비자에게 숨겨져 있으며, 이는 인식된 대기열 백로그 또는 처리량에 영향을 미칠 수 있습니다.

지표	설명	단위	보고 동작	주요 참고 사항
ApproximateNumberOfMessagesNotVisible	수신되었지만 아직 삭제되거나 만료되지 않은 진행 중인 메시지 수입니다.	개수	진행 중인 메시지가 있는 경우 보고됩니다.	• 메시지는 ReceiveMessage API를 통해 소비자에게 전송된 후 진행 중 상태로 전환됩니다. • 이러한 메시지는 표시 제한 시간 동안 다른 소비자에게 일시적으로 숨겨집니다. • 이 지표를 사용하여 메시지 처리 지연 또는 멈춘 소비자를 추적할 수 있습니다.
ApproximateNumberOfMessagesVisible	현재 검색 및 처리에 사용할 수 있는 메시지 수입니다.	개수	대기열이 활성 상태인 경우 보고됩니다.	• 대기열의 현재 처리 백로그를 반영합니다. • 누적할 수 있는 메시지 수에는 제한이 없지만 대기열의 구성된 보존 기간이 적용됩니다. • 값이 지속적으로 높으면 프로비저닝되지 않은 소비자 또는 멈춘 처리 로직을 나타낼 수 있습니다.

지표	설명	단위	보고 동작	주요 참고 사항
`NumberOfEmptyReceives` [1]	메시지를 반환하지 않은 ReceiveMessage API 호출 수입니다.	개수	수신 작업 중에 보고됩니다.	• 이 지표는 폴링 동작 또는 활용도가 낮은 소비자 인스턴스의 비효율성을 식별하는 데 도움이 될 수 있습니다. • 대기열이 비어 있거나, 소비자가 짧은 폴링을 사용하거나, 메시지가 생성된 것보다 빠르게 처리되는 경우 값이 높아질 수 있습니다. • 이는 대기열 상태를 정확하게 나타내는 지표가 아닙니다. 서비스 측 동작을 반영하며 재시도를 포함할 수 있습니다.
`NumberOfDeduplicatedSentMessages`	FIFO 전용입니다. 중복 제거되어 대기열에 추가되지 않은 전송된 메시지 수입니다.	개수	중복 `MessageDeduplicationId` 값 또는 콘텐츠가 감지되면 보고됩니다.	• SQS는 `MessageDeduplicationId` 또는 콘텐츠 기반 해싱(활성화된 경우)을 기반으로 메시지를 중복 제거합니다. • 값이 높으면 생산자가 5분 중복 제거 기간 내에 동일한 메시지를 반복적으로 전송하고 있음을 나타낼 수 있습니다. • 이 지표를 사용하여 중복 생산자 로직 문제를 해결하거나 중복 제거가 의도한 대로 작동하는지 확인합니다.

지표	설명	단위	보고 동작	주요 참고 사항

`NumberOfMessagesDeleted` [1]	대기열에서 성공적으로 삭제된 메시지 수입니다.	개수	유효한 수신 핸들을 사용하여 각 삭제 요청에 대해 보고됩니다.	• 이 지표는 동일한 메시지가 두 번 이상 삭제되더라도 성공한 모든 삭제 작업을 계산합니다.
				• higher-than-expected 값의 일반적인 이유는 다음과 같습니다.
				  • 표시 제한 시간이 만료되고 메시지가 다시 수신된 후 다른 수신 핸들을 사용하여 동일한 메시지를 여러 번 삭제합니다.
				  • 동일한 수신 핸들을 사용하여 중복 삭제합니다.이 핸들은 여전히 성공 상태를 반환하고 지표를 증가시킵니다.
				• 이 지표를 사용하여 메시지 처리 성공을 추적하지만 삭제된 고유한 메시지의 정확한 수로 취급하지는 않습니다.

지표	설명	단위	보고 동작	주요 참고 사항
NumberOfMessagesReceived 1	ReceiveMessage API에서 반환한 메시지 수입니다.	개수	수신 작업 중에 보고됩니다.	• 여기에는 가시성 제한 시간 만료로 인해 나중에 대기열로 반환되는 메시지를 포함하여 소비자에게 반환되는 모든 메시지가 포함됩니다. • 삭제되지 않으면 단일 메시지를 여러 번 수신할 수 있으며, 이로 인해이 지표가 전송된 메시지 수를 초과할 수 있습니다. • 이를 사용하여 소비자 활동을 추적하지만 처리된 고유 메시지 수로 취급하지 않습니다.
NumberOfMessagesSent 1	대기열에 성공적으로 추가된 메시지 수입니다.	개수	성공한 각 수동 전송에 대해 보고됩니다.	• DLQ를 직접 대상으로 하는 직접 호출을 포함하여 SendMessage 또는 에 대한 수동 호출이 계산SendMessageBatch 됩니다. • 를 초과한 후 DLQ로 자동으로 이동되는 메시지는이 지표에 포함되지 maxReceiveCount 않습니다. • 따라서 리드라이브 정책이 많은 메시지를 장면 뒤 DLQs로 이동하는 경우 NumberOfMessagesReceived NumberOfMessagesSent 가 보다 낮을 수 있습니다.

지표	설명	단위	보고 동작	주요 참고 사항
`SentMessageSize` [1]	대기열로 성공적으로 전송된 메시지의 크기입니다.	바이트	메시지가 하나 이상 전송될 때까지 방출되지 않습니다.	• 이 지표는 대기열이 첫 번째 메시지를 수신할 때까지 CloudWatch 콘솔에 표시되지 않습니다. • 이 지표를 사용하여 각 메시지의 크기를 바이트 단위로 추적합니다. 이는 페이로드 추세를 분석하거나 처리량 비용을 추정하는 데 유용합니다. • SQS의 최대 메시지 크기는 1MiB입니다.
`ApproximateNumberOfNoisyGroups`	공정 대기열에서 노이즈가 있는 것으로 간주되는 메시지 그룹의 수입니다. 시끄러운 메시지 그룹은 다중 테넌트 대기열의 시끄러운 이웃 테넌트를 나타냅니다.	개수	대기열이 활성 상태이면 음수가 아닌 값이 보고됩니다.	• 불균형한 리소스를 소비하는 메시지 그룹을 추적하여 다중 테넌트 환경에서 잠재적인 노이즈가 있는 이웃 문제를 식별하는 데 도움이 됩니다. • 이 지표를 사용하여 노이즈 그룹 수가 허용 임계값을 초과할 때 트리거되는 경보를 설정하여 잠재적인 대기열 공정성 문제를 나타냅니다.

지표	설명	단위	보고 동작	주요 참고 사항
`ApproximateNumberOfMessagesVisibleInQuietGroups`	노이즈가 있는 메시지 그룹의 메시지를 제외하고 표시되는 메시지 수입니다.	개수	대기열이 활성 상태이면 음수가 아닌 값이 보고됩니다.	• 노이즈가 많은 이웃의 메시지를 제외하고 표준 속도 메시지 그룹의 대기열 백로그에 대한 가시성을 제공합니다. • 노이즈가 많은 이웃의 영향을 필터링하여 일반적인 메시지 그룹의 실제 처리 백로그를 식별하는 데 도움이 됩니다.
`ApproximateNumberOfMessagesNotVisibleInQuietGroups`	노이즈가 있는 메시지 그룹의 메시지를 제외한 진행 중인 메시지 수입니다.	개수	대기열이 활성 상태이면 음수가 아닌 값이 보고됩니다.	• 잘 작동하는 메시지 그룹에서 진행 중인 메시지(처리 중이지만 아직 삭제되지 않음)를 추적합니다. • 이 지표를 사용하여 일반 메시지 그룹의 처리 처리량을 모니터링하고 노이즈가 많은 이웃으로 인해 발생하지 않는 처리 병목 현상을 감지할 수 있습니다.

지표	설명	단위	보고 동작	주요 참고 사항
ApproximateNumberOfMessagesDelayedInQuietGroups	지연되어 즉시 읽을 수 없는 노이즈가 있는 메시지 그룹의 메시지를 제외한 메시지 수입니다. 지연된 메시지는 대기열이 지연 대기열로 구성되거나 지연 파라미터와 함께 메시지가 전송되었을 때 발생합니다.	개수	대기열이 활성 상태이면 음수가 아닌 값이 보고됩니다.	• 정상 또는 예상 처리량 패턴이 있는 메시지 그룹의 지연된 메시지 백로그를 모니터링하는 데 도움이 됩니다(대량 또는 노이즈가 많은 그룹과 반대). • 일반적인 워크로드에 대한 향후 처리 요구 사항 및 용량 계획을 이해하는 데 유용합니다.
ApproximateAgeOfOldestMessageInQuietGroups	노이즈가 있는 메시지 그룹의 메시지를 제외하고 대기열에서 삭제되지 않은 가장 오래된 메시지의 수명입니다.	초	대기열이 활성 상태이면 음수가 아닌 값이 보고됩니다.	• SLA 규정 준수를 모니터링하고 처리량 패턴이 정상이거나 예상되는 메시지 그룹의 처리 병목 현상을 감지하는 데 사용됩니다(지표가 왜곡될 수 있는 대용량 또는 노이즈가 있는 메시지 그룹과 반대). • 이 지표를 사용하여 노이즈가 많은 이웃의 인위적으로 오래된 메시지를 무시하는 메시지 처리 제한 시간에 대한 경보를 설정합니다.

1 이러한 지표는 시스템 수준 활동을 반영하며 재시도, 중복 또는 지연된 메시지를 포함할 수 있습니다. 메시지 수명 주기 동작을 고려하지 않고 원시 수를 사용하여 실시간 대기열 상태를 추정하지 마세요.

배달 못한 편지 대기열(DLQs) 및 CloudWatch 지표

DLQs로 작업할 때는 Amazon SQS 지표가 어떻게 작동하는지 이해하는 것이 중요합니다.

- **`NumberOfMessagesSent`** -이 지표는 DLQs에서 다르게 작동합니다.
  - 수동 전송 - DLQ로 수동으로 전송된 메시지는이 지표에 의해 캡처됩니다.
  - 자동 리드라이브 - 처리 실패로 인해 DLQ로 자동으로 이동된 메시지는이 지표에 의해 캡처되지 않습니다. 따라서 `NumberOfMessagesSent` 및 `NumberOfMessagesReceived` 지표에 DLQs.
- DLQs에 대한 권장 지표 - DLQ의 상태를 모니터링하려면 `ApproximateNumberOfMessagesVisible` 지표를 사용합니다. 이 지표는 DLQ에서 현재 처리할 수 있는 메시지 수를 나타냅니다.

공정 대기열 및 CloudWatch 지표

공정 대기열을 사용하면 Amazon SQS는 다음과 같은 추가 지표를 내보냅니다.

- `ApproximateNumberOfNoisyGroups`
- `ApproximateNumberOfMessagesVisibleInQuietGroups`
- `ApproximateNumberOfMessagesNotVisibleInQuietGroups`
- `ApproximateNumberOfMessagesDelayedInQuietGroups`
- `ApproximateAgeOfOldestMessageInQuietGroups`

> **Note**
>
> 각 `QuietGroup` 지표는 동등한 표준 대기열 수준 `Approximate` 지표의 하위 집합이지만 노이즈가 많은 이웃 그룹의 메시지는 제외합니다.

노이즈 그룹

시끄러운 메시지 그룹은 다중 테넌트 대기열의 시끄러운 이웃 테넌트를 나타냅니다.

저소음 그룹

노이즈 그룹을 제외한 메시지 그룹입니다.

SQS 공정 대기열 동작 관찰

Amazon SQS 공정 대기열의 효과를 모니터링하기 위해 `Approximate..InQuietGroups` 지표를 표준 대기열 수준 지표와 비교할 수 있습니다. 특정 테넌트에 대한 트래픽이 급증하는 동안 일반 대기열 수준 지표에서 백로그 증가 또는 메시지 수명 증가를 확인할 수 있습니다. 그러나 저소음 그룹을 개별적으로 살펴보면 대부분의 소음이 없는 메시지 그룹 또는 테넌트가 영향을 받지 않음을 식별하고 영향을 받는 총 메시지 그룹 수의 추정치를 제공할 수 있습니다.

이러한 새로운 지표는 Amazon SQS 공정 대기열 동작에 대한 좋은 개요를 제공하지만 어떤 특정 테넌트가 로드를 유발하는지 이해하는 것이 유용할 수 있습니다. Amazon CloudWatch 기고자 인사이트를 사용하면 상위 N개 기고자, 총 고유 기고자 수 및 사용량에 대한 지표를 볼 수 있습니다. 이는 기존 지표를 내보낼 때 카디널리티가 높은 데이터(및 비용)로 이어질 수 있는 수천 개의 테넌트를 처리하는 시나리오에서 특히 유용합니다.

공정 대기열에 대한 구성 모니터링 예제는 GitHub의 샘플을 참조하세요.

Amazon SQS 지표 차원

CloudWatch의 Amazon SQS 지표는 단일 차원를 사용합니다**`QueueName`**. 모든 지표 데이터는 대기열 이름을 기준으로 그룹화되고 필터링됩니다.

모니터링 팁

주요 지표 및 CloudWatch 경보를 사용하여 SQS를 효과적으로 모니터링하여 대기열 백로그를 감지하고 성능을 최적화하며 서비스 제한을 유지합니다.

- 를 기반으로 CloudWatch 경보를 설정`ApproximateNumberOfMessagesVisible`하여 백로그 증가를 포착합니다.
- 모니터링`NumberOfEmptyReceives`하여 폴링 빈도를 조정하고 API 비용을 줄입니다.
- FIFO 대기열`ApproximateNumberOfGroupsWithInflightMessages`에서를 사용하여 처리량 제한을 진단합니다.
- SQS 할당량을 검토하여 지표 임계값 및 서비스 한도를 이해합니다.

Amazon SQS에 대한 규정 준수 확인

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 제공 범위 섹션을 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 Downloading Reports inDownloading AWS Artifact을 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 AWS 서비스 결정됩니다.는 규정 준수를 지원하기 위해 다음 리소스를 AWS 제공합니다.

- 보안 규정 준수 및 거버넌스 - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- HIPAA 적격 서비스 참조 - HIPAA 적격 서비스가 나열되어 있습니다. 모두 HIPAA 자격이 AWS 서비스 있는 것은 아닙니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 산업 및 위치에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에는 여러 프레임워크(미국 국립 표준 기술 연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI) 및 국제 표준화 기구(ISO) 포함)의 보안 제어에 대한 지침을 보호하고 AWS 서비스 매핑하는 모범 사례가 요약되어 있습니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 -이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub - 내 보안 상태에 대한 포괄적인 보기를 AWS 서비스 제공합니다 AWS. Security Hub는 보안 컨트롤을 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.
- Amazon GuardDuty - 의심스러운 악의적인 활동이 있는지 환경을 모니터링하여 사용자, AWS 계정 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.
- AWS Audit Manager - 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험과 규정 및 업계 표준 준수를 관리하는 방법을 간소화할 수 있습니다.

Amazon SQS의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며,이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결됩니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다. AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon SQS는 분산 대기열을 제공합니다.

분산 대기열

분산 메시징 시스템에는 분산 시스템의 구성 요소, 대기열(Amazon SQS 서버에 배포됨) 및 대기열의 메시지라는 세 가지 주요 부분이 있습니다.

다음 시나리오에서 시스템에는 여러 생산자(메시지를 대기열로 전송하는 구성 요소) 및 소비자(대기열의 메시지를 수신하는 구성 요소)가 있습니다. 대기열(메시지 A~E 유지)은 여러 Amazon SQS 서버에서 메시지를 중복으로 저장합니다.

Amazon SQS의 인프라 보안

관리형 서비스인 Amazon SQS는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다.

AWS 게시된 API 작업을 사용하여 네트워크를 통해 Amazon SQS에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.2 이상을 지원해야 합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 PFS(전달 완전 보안, Perfect Forward Secrecy)가 포함된 암호 제품군도 지원해야 합니다.

IAM 주체와 연결되어 있는 액세스 키 ID 및 보안 액세스 키를 사용해 요청에 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용해 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

이러한 API 작업은 모든 네트워크 위치에서 호출할 수 있지만, Amazon SQS는 원본 IP 주소를 기반으로 하는 제한을 포함할 수 있는 리소스 기반 액세스 정책을 지원합니다. Amazon SQS 정책을 사용하

여 특정 Amazon VPC 엔드포인트 또는 특정 VPC에서 액세스를 제어할 수도 있습니다. 이렇게 하면 네트워크 내의 특정 VPC에서만 지정된 Amazon SQS 대기열에 대한 AWS 네트워크 액세스가 효과적으로 격리됩니다. 자세한 내용은 예제 5: VPC 엔드포인트에서 온 액세스가 아닌 경우 액세스 거부 단원을 참조하십시오.

Amazon SQS 보안 모범 사례

AWS 는 Amazon SQS에 대한 다양한 보안 기능을 제공하며, 이는 자체 보안 정책의 맥락에서 검토해야 합니다. 다음은 Amazon SQS에 대한 예방 보안 모범 사례입니다.

> **Note**
>
> 제공된 구체적인 구현 지침은 일반적인 사용 사례 및 구현을 위한 것입니다. 특정 사용 사례, 아키텍처 및 위협 모델의 컨텍스트에서 이러한 모범 사례를 살펴보는 것이 좋습니다.

대기열에 대한 공개적 액세스 방지

인터넷상의 누군가가 Amazon SQS 대기열을 읽거나 쓸 수 있도록 명시적으로 요구하지 않는 한 대기열에 공개적으로 액세스할 수 없도록 해야 합니다(전 세계 모든 사람 또는 인증된 AWS 사용자가 액세스 가능).

- `Principal`이 `""`로 설정된 정책을 생성하지 마세요.
- 와일드카드(*)를 사용하지 마세요. 대신에 특정 사용자 또는 사용자의 이름을 지정합니다.

최소 권한 액세스 구현

권한을 부여할 때 권한을 받는 사용자, 권한이 있는 대기열 및 이러한 대기열에 허용하려는 특정 API 작업을 결정합니다. 최소 권한 구현은 보안 위험을 줄이고 오류 또는 악의적인 의도에 따른 영향을 줄이는 데 중요합니다.

최소 권한 부여에 대한 표준 보안 권고 사항을 따릅니다. 즉, 특정 작업을 수행하는 데 필요한 권한만 부여합니다. 보안 정책 조합을 사용하여 이를 구현할 수 있습니다.

Amazon SQS는 세 가지 유형의 사용자 계정 액세스를 필요로 하는 생산자-소비자 모델을 사용합니다.

- 관리자 - 대기열을 생성, 수정 및 삭제할 수 있습니다. 또한 관리자는 대기열 정책도 제어합니다.

- 생산자 - 대기열에 메시지를 보낼 수 있습니다.
- 소비자 - 대기열에서 보낸 메시지를 수신하고 삭제할 수 있습니다.

자세한 내용은 다음 단원을 참조하세요.

- Amazon SQS의 Identity and Access Management
- Amazon SQS API 권한: 작업 및 리소스 참조
- Amazon SQS 액세스 정책 언어와 함께 사용자 지정 정책 사용

Amazon SQS 액세스가 필요한 애플리케이션 및 AWS 서비스에 IAM 역할 사용

Amazon EC2와 같은 애플리케이션 또는 AWS 서비스가 Amazon SQS 대기열에 액세스하려면 AWS API 요청에 유효한 AWS 자격 증명을 사용해야 합니다. 이러한 자격 증명은 자동으로 교체되지 않으므로 자격 AWS 증명을 애플리케이션 또는 EC2 인스턴스에 직접 저장해서는 안 됩니다.

IAM 역할을 사용하여 Amazon SQS에 액세스해야 하는 애플리케이션이나 서비스의 임시 자격 증명을 관리해야 합니다. 역할을 사용하는 경우 장기 자격 증명(예: 사용자 이름, 암호 및 액세스 키)을와 같은 EC2 인스턴스 또는 AWS 서비스에 배포할 필요가 없습니다 AWS Lambda. 대신 역할은 애플리케이션이 다른 AWS 리소스를 호출할 때 사용할 수 있는 임시 권한을 제공합니다.

자세한 정보는 IAM 사용 설명서의 IAM 역할 및 역할에 대한 일반 시나리오: 사용자, 애플리케이션 및 서비스를 참조하세요.

서버 측 암호화 구현

데이터 유출 문제를 완화하려면 메시지를 저장하는 위치와 다른 위치에 저장된 키를 통해 메시지를 암호화하는 유휴 시 데이터 암호화를 사용합니다. 서버 측 암호화(SSE)는 저장된 데이터 암호화 기능을 제공합니다. Amazon SQS는 데이터를 저장할 때 메시지 수준에서 데이터를 암호화하고 사용자가 액세스할 때 메시지를 복호화합니다. SSE는에서 관리되는 키를 사용합니다 AWS Key Management Service. 요청을 인증하기만 하면 액세스 권한을 갖게 되며, 대기열의 암호화 여부와 관계없이 액세스 방식에는 차이가 없습니다.

자세한 내용은 Amazon SQS에서 저장 시 암호화 및 Amazon SQS 키 관리 섹션을 참조하세요.

전송 중인 데이터의 암호화 강제 시행

HTTPS(TLS)가 없으면 네트워크 기반 공격자가 중간자(man-in-the-middle)와 같은 공격을 사용하여 네트워크 트래픽을 도청하거나 조작할 수 있습니다. 요청에 SSL을 강제로 적용하도록 하려면 대기열 정책의 `aws:SecureTransport` 조건을 사용하여 HTTPS(TLS)를 통한 암호화된 연결만 허용합니다.

VPC 엔드포인트를 사용한 Amazon SQS 액세스 고려

상호 작용할 수 있어야 하지만 인터넷에 절대 노출되지 않아야 하는 대기열이 있는 경우, VPC 엔드포인트를 사용하여 특정 VPC 내의 호스트에 대한 액세스만 대기시킵니다. 대기열 정책을 사용하여 특정 엔드포인트 또는 특정 Amazon VPC에서 대기열에 대한 액세스를 제어할 수 있습니다.

Amazon SQS VPC 엔드포인트는 메시지에 대한 액세스를 제어하는 두 가지 방법을 제공합니다.

- 사용자는 특정 VPC 종단점을 통해 허용되는 요청, 사용자 또는 그룹을 제어할 수 있습니다.
- 대기열 정책을 사용하여 대기열에 액세스할 수 있는 VPC 또는 VPC 엔드포인트를 제어할 수 있습니다.

자세한 내용은 Amazon SQS용 Amazon Virtual Private Cloud 엔드포인트 및 Amazon SQS용 Amazon VPC 엔드포인트 정책 생성 섹션을 참조하세요.

Amazon SQS 관련 리소스

다음 표에는 이 서비스를 사용할 때 유용하게 참조할 수 있는 관련 리소스가 나열되어 있습니다.

리소스	설명
Amazon Simple Queue Service API 참조	작업, 파라미터, 데이터 유형 등의 설명 및 반환하는 오류 목록.
AWS CLI 명령 참조의 Amazon SQS	대기열 작업에 사용할 수 있는 AWS CLI 명령에 대한 설명입니다.
리전 및 엔드포인트	Amazon SQS 리전 및 엔드포인트에 대한 정보
제품 페이지	Amazon SQS에 대한 정보를 제공하는 기본 웹 페이지입니다.
토론 포럼	Amazon SQS와 관련된 기술적 질문에 대해 토론할 수 있는 개발자를 위한 커뮤니티 기반 포럼입니다.
AWS 프리미엄 지원 정보	AWS 인프라 서비스에서 애플리케이션을 구축하고 실행하는 데 도움이 되는 one-on-one 빠른 응답 지원 채널인 AWS Premium Support에 대한 정보를 제공하는 기본 웹 페이지입니다.

문서 기록

아래 표에는 2019년 1월 이후 Amazon Simple Queue Service 개발자 안내서의 중요한 변경 사항이 설명되어 있습니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

서비스 기능은 서비스를 사용할 수 있는 AWS 리전에 점진적으로 배포되는 경우가 있습니다. 이 문서는 첫 번째 릴리스에 대해서만 업데이트됩니다. 리전 가용성에 대한 정보를 제공하거나 후속 리전 롤아웃을 발표하지 않습니다. 서비스 기능의 리전 가용성과 업데이트에 대한 알림을 구독하는 방법에 대한 자세한 내용은 What's New with AWS?를 참조하세요.

변경 사항	설명	날짜
공정 대기열 지원	표준 대기열에 대한 공정 대기열 지원이 추가되었습니다.	2025년 7월 21일
듀얼 스택(IPv4 및 IPv6) 엔드포인트에 대한 지원 추가	Amazon SQS는 이제 듀얼 스택(IPv4 및 IPv6) 엔드포인트를 지원하므로 두 IP 프로토콜을 통해 대기열에 액세스할 수 있습니다.	2025년 4월 17일
모든 Amazon SQS APIs 대한 CloudTrail 통합	모든 Amazon SQS API에 CloudTrail 통합이 추가되었습니다. APIs	2025년 1월 10일
SQSUnlockQueuePolicy	Amazon SQS는 대기열을 잠금 해제하고 모든 보안 주체가 Amazon SQS 대기열에 액세스하는 것을 거부하는 잘못 구성된 대기열 정책을 제거하기 위해 라는 새로운 AWS관리형 정책을 추가했습니다. `SQSUnlockQueuePolicy`	2024년 11월 15일
AWS kms:Decrypt	Amazon SQS에는 더 이상 `SendMessage` API에 대한 `kms:Decrypt` 권한이 필요하지 않습니다. 이제 고객은 대	2024년 7월 24일

	기열을 암호화하는 데 사용되는 KMS 키에 대한 `kms:GenerateDataKey` 권한만 있으면 되지만, `ReceiveMessage` 를 직접 호출할 수 있는 `kms:Decrypt` 권한이 여전히 필요합니다.	
FIFO 지표 업데이트	Amazon SQS FIFO 지표에 `NumberOfDeduplicatedSentMessages` 및 `ApproximateNumberOfGroupsWithInflightMessages` 지원이 추가되었습니다.	2024년 7월 3일
AmazonSQSReadOnlyAccess 관리형 정책에서 지원되는 `ListQueueTags` 작업	AmazonSQSReadOnlyAccess 관리형 정책은 지정된 Amazon SQS 대기열과 연결된 모든 태그를 검색할 수 있도록 `ListQueueTags` 를 지원합니다.	2024년 5월 2일
AWS JSON 프로토콜	AWS JSON 프로토콜을 사용하여 API를 요청합니다.	2023년 7월 27일
Amazon SQS에 대한 AWS 관리형 정책 및 이러한 정책에 대한 업데이트를 설명하는 새 섹션	Amazon SQS는 특정 소스 대기열 아래에 가장 최근의 메시지 이동 작업(최대 10개)을 나열할 수 있는 새 작업을 추가했습니다. 이 작업은 `ListMessageMoveTasks` API 작업과 연결되어 있습니다.	2023년 6월 7일
API를 사용하여 배달 못한 편지 대기열 리드라이브	Amazon SQS API를 사용하여 배달 못한 편지 대기열 리드라이브를 구성합니다.	2023년 6월 7일

Amazon SQS에 대한 ABAC	유연하고 확장 가능한 액세스 권한을 위해 대기열 태그를 사용하는 ABAC(속성 기반 액세스 제어)입니다.	2022년 11월 10일
FIFO 높은 처리량 한도 증가	FIFO 높은 처리량 문서 최적화와 더불어 상용 리전의 FIFO 높은 처리량 모드에 대한 기본 할당량이 증가했습니다.	2022년 10월 20일
기본 서버 측 암호화(SSE)를 사용할 수 있음	기본적으로 SQS 소유 암호화(SSE-SQS)를 사용하는 서버 측 암호화(SSE)를 사용할 수 있습니다.	2022년 9월 26일
Amazon SQS 혼동된 대리인 보호 지원을 사용할 수 있음	혼동된 대리인 보호를 사용하면 요청에 새 헤더를 지정할 수 있으며, 이러한 헤더는 Amazon SQS 관리형 SSE를 사용할 때 KMS 정책의 조건과 비교하여 검토됩니다.	2021년 12월 29일
관리형 SSE를 사용할 수 있음	Amazon SQS 관리형 SSE(SSE-SQS)는 Amazon SQS 소유 암호화 키를 사용하여 메시지 대기열을 통해 전송되는 민감한 데이터를 보호하는 관리형 서버 측 암호화입니다.	2021년 11월 23일
배달 못한 편지 대기열 리드라이브를 사용할 수 있음	Amazon SQS는 표준 대기열에 대해 배달 못한 편지 대기열 리드라이브를 지원합니다.	2021년 11월 10일

FIFO 대기열의 메시지에 대해 높은 처리량을 사용할 수 있음	Amazon SQS FIFO 대기열의 높은 처리량은 FIFO 대기열의 메시지에 대해 더 많은 초당 트랜잭션(TPS) 수를 제공합니다. 처리량 할당량에 대한 자세한 내용은 메시지 관련 할당량을 참조하세요.	2021년 5월 27일
평가판 릴리스에서 FIFO 대기열의 메시지에 대해 높은 처리량을 사용할 수 있음	Amazon SQS FIFO 대기열의 높은 처리량은 평가판 릴리스에서 제공되며 변경될 수 있습니다. 이 기능은 FIFO 대기열에 있는 메시지에 대해 더 많은 초당 트랜잭션(TPS) 수를 제공합니다. 처리량 할당량에 대한 자세한 내용은 메시지 관련 할당량을 참조하세요.	2020년 12월 17일
새로운 Amazon SQS 콘솔 설계	개발 및 프로덕션 워크플로를 단순화하기 위해 Amazon SQS 콘솔에 새로운 사용자 경험이 추가되었습니다.	2020년 7월 8일
Amazon SQS가 listQueues 및 listDeadLetterSourceQueues에 대한 페이지 매김을 지원함	listQueues 또는 listDeadLetterSourceQueues 요청에서 반환할 최대 결과 수를 지정할 수 있습니다.	2020년 6월 22일
Amazon SQS는 GovCloud(미국) AWS 리전을 제외한 모든 리전에서 1분 Amazon CloudWatch 지표를 AWS 지원합니다.	이제 Amazon SQS에 대한 1분 CloudWatch 지표를 AWS GovCloud (US) 리전을 제외한 모든 리전에서 사용할 수 있습니다.	2020년 1월 9일

Amazon SQS가 1분 CloudWatch 지표를 지원함	Amazon SQS에 대한 1분 CloudWatch 지표는 현재 미국 동부(오하이오), 유럽(아일랜드), 유럽(스톡홀름) 및 아시아 태평양(도쿄) 리전에서만 사용할 수 있습니다.	2019년 11월 25일
AWS Lambda Amazon SQS FIFO 대기열에 대한 트리거 사용 가능	FIFO 대기열에 도착하는 메시지를 Lambda 함수 트리거로 구성할 수 있습니다.	2019년 11월 25일
중국 리전에서 Amazon SQS에 대한 서버 측 암호화(SSE)를 사용할 수 있음	Amazon SQS에 대한 SSE를 중국 리전에서 사용할 수 있습니다.	2019년 11월 13일
중동(바레인) 리전에서 FIFO 대기열을 사용할 수 있음	중동(바레인) 리전에서 FIFO 대기열을 사용할 수 있습니다.	2019년 10월 10일
Amazon SQS용 Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트는 AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 리전에서 사용할 수 있습니다. Amazon SQS	AWS GovCloud(미국 동부) 및 GovCloud(미국 AWS GovCloud 서부) 리전의 Amazon VPC에서 Amazon SQS 대기열로 메시지를 보낼 수 있습니다.	2019년 9월 5일

Amazon SQS를 사용하면 메시지 시스템 속성을 사용하여를 AWS X-Ray 사용하여 대기열 문제 해결 가능	X-Ray를 사용하여 Amazon SQS 대기열을 통해 전달되는 메시지의 문제를 해결할 수 있습니다. 이 릴리스는 `SendMessage` 및 `SendMessageBatch` API 작업에 대한 `MessageSystemAttribute` 요청 파라미터(이 파라미터를 사용하면 Amazon SQS를 통해 X-Ray 추적 헤더를 전송할 수 있음), `ReceiveMessage` API 작업에 대한 `AWSTraceHeader` 속성 및 `MessageSystemAttributeValue` 데이터 형식을 추가합니다.	2019년 8월 28일
생성 시 Amazon SQS 대기열에 태그를 지정할 수 있음	단일 Amazon SQS API 직접 호출, AWS SDK 함수 또는 AWS Command Line Interface (AWS CLI) 명령을 사용하여 동시에 대기열을 생성하고 태그를 지정할 수 있습니다. 또한 Amazon SQS는 `aws:TagKeys` 및 `aws:RequestTag` AWS Identity and Access Management (IAM) 키를 지원합니다.	2019년 8월 22일

Amazon SQS에 대한 임시 대기열 클라이언트를 지금 사용할 수 있음	임시 대기열을 사용하면 request-response와 같은 공통 메시지 패턴을 사용할 때 개발 시간 및 배포 비용을 줄일 수 있습니다. 임시 대기열 클라이언트를 사용하여 대량고속처리가 가능하고 비용 효율적인 애플리케이션 관리 임시 대기열을 생성할 수 있습니다.	2019년 7월 25일
Amazon SQS용 SSE는 AWS GovCloud(미국 동부) 리전에서 사용할 수 있습니다.	Amazon SQS에 대한 서버 측 암호화(SSE)는 AWS GovCloud(미국 동부) 리전에서 사용할 수 있습니다.	2019년 6월 20일
FIFO 대기열은 아시아 태평양(홍콩), 중국(베이징), AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 리전에서 사용할 수 있습니다.	FIFO 대기열은 아시아 태평양(홍콩), 중국(베이징), AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 리전에서 사용할 수 있습니다.	2019년 5월 15일
Amazon SQS에 대해 Amazon VPC 엔드포인트 정책을 사용할 수 있음	Amazon SQS에 대한 Amazon VPC 엔드포인트 정책을 생성할 수 있습니다.	2019년 4월 4일
유럽(스톡홀름) 및 중국(닝샤) 리전에서 FIFO 대기열을 사용할 수 있음	유럽(스톡홀름) 및 중국(닝샤) 리전에서 FIFO 대기열을 사용할 수 있습니다.	2019년 3월 14일

Amazon SQS를 사용할 수 있는 모든 리전에서 FIFO 대기열을 사용할 수 있음	미국 동부(오하이오), 미국 동부(버지니아 북부), 미국 서부(캘리포니아 북부), 미국 서부(오레곤), 아시아 태평양(뭄바이), 아시아 태평양(서울), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄), 캐나다(중부), 유럽(프랑크푸르트), 유럽(아일랜드), 유럽(런던), 유럽(파리) 및 남아메리카(상파울루) 리전에서 FIFO 대기열을 사용할 수 있습니다.	2019년 2월 7일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.