



Entwicklerhandbuch

Amazon Simple Queue Service



Amazon Simple Queue Service: Entwicklerhandbuch

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Marken und Handelsmarken von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, die geeignet ist, Kunden irrezuführen oder Amazon in irgendeiner Weise herabzusetzen oder zu diskreditieren. Alle anderen Marken, die nicht im Besitz von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist Amazon SQS?	1
Vorteile der Verwendung von Amazon SQS	1
Grundlegende Architektur	2
Verteilte Warteschlangen	2
Lebenszyklus einer Nachricht	2
Unterschiede zwischen Amazon SQS, Amazon MQ und Amazon SNS	4
Erste Schritte	6
Einrichtung	6
Schritt 1: Erstellen Sie ein AWS-Konto und IAM-Benutzer	6
Schritt 2: Erteilen programmgesteuerten Zugriffs	7
Schritt 3: Vorbereiten der Verwendung des Beispiel-Codes	9
Nächste Schritte	10
Die Amazon-SQS-Konsole verstehen	10
Warteschlangentypen	11
Implementierung von Anfrage-Antwort-Systemen in Amazon SQS	15
Erstellen einer Standard-Warteschlange	15
Erstellen einer Warteschlange	15
Senden einer Nachricht mithilfe einer Standardwarteschlange	18
Erstellen einer FIFO-Warteschlange	18
Erstellen einer Warteschlange	19
Senden einer Nachricht mithilfe einer FIFO-Warteschlange	22
Allgemeine Aufgaben	22
Verwalten einer Warteschlange	24
Eine Warteschlange bearbeiten	24
Empfangen und Löschen einer Nachricht	25
Bestätigen, dass eine Warteschlange leer ist	26
Löschen einer Warteschlange	28
Bereinigen einer Warteschlange	29
Standard-Warteschlangen	30
Amazon SQS SQS-Lieferung at-least-once	31
Warteschlangen- und Nachrichten-IDs	31
IDs für Standard-Warteschlangen	31
FIFO-Warteschlangen	33
Schlüsselbegriffe für FIFO-Warteschlangen	34

FIFO-Bereitstellungslogik	36
Senden von Nachrichten	36
Empfangen von Nachrichten	37
Mehrere Versuche	38
Zusätzliche Hinweise zum FIFO-Verhalten	39
Beispiele zum besseren Verständnis	39
Garantiert einmalige Verarbeitung	40
Wechseln von einer Standard- zu einer FIFO-Warteschlange	40
FIFO-Warteschlange und Lambda-Parallelitätsverhalten	42
Gruppierung von FIFO-Warteschlangennachrichten	43
Lambda-Parallelität mit FIFO-Warteschlangen	43
Beispiel für einen Anwendungsfall	43
Hoher Durchsatz für FIFO-Warteschlangen	44
Anwendungsfälle	44
Partitionen und Datenverteilung	45
Aktivierung eines hohen Durchsatzes für FIFO-Warteschlangen	48
Warteschlangen- und Nachrichten-IDs	49
IDs für FIFO-Warteschlangen	31
Zusätzliche Kennungen für FIFO-Warteschlangen	51
Kontingente	52
FIFO-Warteschlangenkontingente	52
Amazon-SQS-Kontingente	52
Standard-Warteschlangenkontingente	54
Nachrichtenkontingente	56
Richtlinienkontingente	63
Funktionen und Funktionen	65
Warteschlangen für unzustellbare Briefe	65
Verwenden von Richtlinien für Warteschlangen mit unzustellbaren Briefen	66
Grundlegendes zu den Aufbewahrungsfristen für Nachrichten in Warteschlangen mit unerlaubten Briefen	67
Konfigurieren einer Queue für unzustellbare Nachrichten	67
Konfigurieren eines Redrives einer Warteschlange für unzustellbare Nachrichten	68
CloudTrail Aktualisierungs- und Genehmigungsanforderungen	79
Alarmer für Warteschlangen mit unbestätigten Briefen mithilfe von Amazon erstellen CloudWatch	84
Nachrichten-Metadaten für Amazon SQS	84

Nachrichtenattribute	84
Nachrichtensystemattribute	88
Für die Nachrichtenverarbeitung erforderliche Ressourcen	89
Auflistung der Warteschlangenpaginierung	90
Kostenzuordnungs-Tags	90
Kurz- und Langabfragen	91
Abrufen von Nachrichten durch Kurzabfragen	92
Konsumieren von Nachrichten mithilfe von Langabfragen	93
Unterschiede zwischen Lang- und Kurzabfragen	94
Timeout für Sichtbarkeit	94
Anwendungsfälle mit Timeout bei der Sichtbarkeit	95
Einstellung und Anpassung des Sichtbarkeits-Timeouts	95
In Flugnachrichten und Kontingenten	96
Grundlegendes zum Sichtbarkeits-Timeout in Standard- und FIFO-Warteschlangen	96
Umgang mit Fehlern	97
Änderung und Beendigung des Sichtbarkeits-Timeouts	97
Best Practices	98
Faire Warteschlangen	99
Wie verwendet man faire Warteschlangen	101
Wann sollten faire Warteschlangen verwendet werden	101
Faire Kennzahlen für Warteschlangen CloudWatch	102
Wie funktionieren faire Warteschlangen	103
Verzögerungswarteschlangen	105
Temporäre Warteschlangen	106
Virtuelle Warteschlangen	107
Request-Response-Messaging-Muster (virtuelle Warteschlangen)	108
Beispielszenario: Verarbeiten einer Anmeldeanforderung	109
Bereinigen von Warteschlangen	111
Nachrichten-Timer	112
Zugreifen auf EventBridge Pipes	113
Verwalten großer Nachrichten	114
Verwendung der Extended Client Library für Java	115
Verwenden der Extended Client Library für Python	121
Konfiguration von Amazon SQS	125
ABAC für Amazon SQS	125
Was ist ABAC?	125

Weshalb sollte ich ABAC in Amazon SQS verwenden?	126
Markierungen für die Zugriffssteuerung	127
Erstellen von IAM-Benutzern und Amazon-SQS-Warteschlangen	127
Testen der attributbasierten Zugriffssteuerung	130
Konfiguration von Warteschlangenparametern	131
Konfiguration einer Zugriffsrichtlinie	134
Konfigurieren von SSE-SQS für eine Warteschlange	134
Konfigurieren von SSE-KMS für eine Warteschlange	136
Konfigurieren von Tags für eine Warteschlange	138
Abonnieren einer Warteschlange für ein Thema	138
Kontoübergreifende Abonnements	139
Regionsübergreifende Abonnements	140
Konfigurieren eines Lambda-Auslösers	141
Voraussetzungen	142
Automatisieren von Benachrichtigungen mit EventBridge	143
Nachrichtenattribute	143
Bewährte Methoden	145
Fehlerbehandlung und problematische Meldungen	145
Behandlung von Anforderungsfehlern in Amazon SQS	145
Erfassung problematischer Nachrichten in Amazon SQS	146
Einrichtung der Aufbewahrung von Warteschlangen für unzustellbare Briefe in Amazon SQS	146
Deduplizierung und Gruppierung von Nachrichten	147
Vermeidung inkonsistenter Nachrichtenverarbeitung in Amazon SQS	147
Verwenden der Nachrichtendeduplizierungs-ID	148
Verwenden der Nachrichtengruppen-ID	150
Verwenden der Empfangsanforderungsversuch-ID	153
Nachrichtenverarbeitung und Timing	153
Rechtzeitige Verarbeitung von Nachrichten in Amazon SQS	153
Einrichtung von Long Polling in Amazon SQS	154
Verwenden des entsprechenden Abfragemodus in Amazon SQS	155
Java-SDK-Beispiele	157
Verwenden der serverseitigen Verschlüsselung	157
Hinzufügen von SSE zu einer vorhandenen Warteschlange	157
Deaktivieren von SSE für eine Warteschlange	159
Erstellen einer Warteschlange mit SSE	159

Abrufen der SSE-Attribute	160
Konfigurieren von Tags	160
Auflisten von Tags	160
Hinzufügen oder Aktualisieren von Tags	161
Entfernen von Tags	162
Senden von Nachrichtenattributen	162
Definieren von Attributen	162
Senden einer Nachricht mit Attributen	164
Verwenden APIs	166
Abfrage-API-Anfragen mithilfe des AWS JSON-Protokolls stellen	167
Erstellen eines Endpunkts	167
Durchführen einer POST-Anforderung	168
Interpretieren von Amazon-SQS-JSON-API-Antworten	169
Amazon SQS AWS JSON-Protokoll FAQs	171
Abfrage-API-Anfragen mithilfe des AWS Abfrageprotokolls stellen	174
Erstellen eines Endpunkts	174
Durchführen einer GET-Anforderung	175
Durchführen einer POST-Anforderung	168
Interpretieren von Amazon-SQS-XML-API-Antworten	177
Authentifizieren von Anforderungen	178
Grundlegender Authentifizierungsprozess mit HMAC-SHA	179
Teil 1: Die Anforderung von dem Benutzer	180
Teil 2: Die Antwort von AWS	181
Stapelaktionen	182
Stapelverarbeitungsaktionen für Nachrichten	183
Aktivierung der clientseitigen Pufferung und der Batchverarbeitung von Anfragen mit Amazon SQS	184
Steigerung des Durchsatzes durch horizontale Skalierung und Action-Batching mit Amazon SQS	197
Arbeiten mit AWS SDKs	209
Verwenden von JMS	211
Voraussetzungen	211
Verwendung der Java-Messaging-Bibliothek	212
Erstellen einer JMS-Verbindung	213
Erstellen einer Amazon-SQS-Warteschlange	214
Synchrones Senden von Nachrichten	215

Synchrones Empfangen von Nachrichten	216
Asynchrones Empfangen von Nachrichten	218
Verwenden des Client-Bestätigungsmodus	219
Verwenden des ungeordneten Bestätigungsmodus	220
Verwenden des JMS-Clients mit anderen Amazon-SQS-Clients	221
Funktionierende Java-Beispiele für die Verwendung von JMS mit Standardwarteschlangen	222
ExampleConfiguration.java	223
TextMessageSender.java	225
SyncMessageReceiver.java	227
AsyncMessageReceiver.java	229
SyncMessageReceiverClientAcknowledge.java	231
SyncMessageReceiverUnorderedAcknowledge.java	235
SpringExampleConfiguration.xml	238
SpringExample.java	240
ExampleCommon.java	242
Unterstützte JMS 1.1-Implementierungen	244
Unterstützte gängige Schnittstellen	244
Unterstützte Nachrichtentypen	244
Unterstützte Nachrichtenbestätigungsmodi	244
JMS-definierte Kopfzeilen und reservierte Eigenschaften	245
Tutorials	246
Erstellen einer Amazon SQS SQS-Warteschlange mit CloudFormation	246
Senden einer Nachricht von einer VPC	248
Schritt 1: Erstellen eines Amazon EC2-Schlüsselpaares	249
Schritt 2: Ressourcen erstellen AWS	250
Schritt 3: Bestätigen, dass Ihre EC2-Instance nicht öffentlich zugänglich ist	251
Schritt 4: Erstellen eines Amazon-VPC-Endpunkts für Amazon SQS	252
Schritt 5: Senden einer Nachricht an Ihre Amazon-SQS-Warteschlange	253
Codebeispiele	255
Grundlagen	256
Hello Amazon SQS	257
Aktionen	270
Szenarien	445
Erstellen einer Nachrichtenanwendung	446
Erstellen einer Messenger-Anwendung	447
Erstellen Sie eine Amazon-Textextract-Explorer-Anwendung	448

Erstellen und veröffentlichen zu einem FIFO-Thema	449
Erkennen Sie Personen und Objekte in einem Video	462
Verwalten großer Nachrichten mit S3	463
Process-S3-Ereignisbenachrichtigungen	467
Veröffentlichen Sie Nachrichten in Warteschlangen	471
Senden und Empfangen von Stapeln von Nachrichten	608
Verwenden Sie das AWS Message Processing Framework für.NET mit Amazon SQS	640
Verwenden der Amazon SQS Java Messaging Library, um mit der JMS-Schnittstelle zu arbeiten	640
Arbeiten Sie mit Warteschlangen-Tags	663
Serverless-Beispiele	666
Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser	666
Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS- Auslöser	675
Fehlerbehebung	686
Fehler „Zugriff verweigert“	686
Amazon SQS SQS-Warteschlangenrichtlinie und IAM-Richtlinie	687
AWS Key Management Service (AWS KMS) Berechtigungen	688
VPC-Endpunktrichtlinie	689
Richtlinie zur Dienstkontrolle der Organisation	690
API-Fehler	690
QueueDoesNotExist Fehler	690
InvalidAttributeValue Fehler	691
ReceiptHandle Fehler	692
Probleme mit DLQ und DLQ Redrive	693
DLQ-Probleme	693
Probleme mit DLQ-Redrive	694
Probleme mit der FIFO-Drosselung	697
Nachrichten, die bei einem ReceiveMessage API-Aufruf nicht zurückgegeben wurden	697
Leere Warteschlange	698
Limit während des Fluges erreicht	698
Verzögerung der Nachricht	698
Die Nachricht ist im Flug	698
Methode der Umfrage	699
Netzwerkfehler	699
ETIMEOUT error	699

UnknownHostException error	700
Fehlerbehebung bei Warteschlangen mit X-Ray	701
Sicherheit	703
Datenschutz	703
Datenverschlüsselung	704
Richtlinie für den Datenverkehr zwischen Netzwerken	717
Verwenden Sie Dual-Stack-Endpunkte für die Konnektivität	719
Identity and Access Management	720
Zielgruppe	720
Authentifizierung mit Identitäten	720
Verwalten des Zugriffs mit Richtlinien	722
-Übersicht	724
So funktioniert Amazon Simple Notification Service mit IAM	731
AWS verwaltete Richtlinien	738
Fehlerbehebung	740
Verwenden von -Richtlinien	743
Protokollierung und Überwachung	793
Protokollieren von API-Aufrufen	797
Überwachen von Warteschlangen	801
Compliance-Validierung	825
Ausfallsicherheit	825
Verteilte Warteschlangen	825
Sicherheit der Infrastruktur	826
Best Practices	827
Sicherstellen, dass Warteschlangen nicht öffentlich zugänglich sind	827
Implementieren der geringstmöglichen Zugriffsrechte	827
Verwenden Sie IAM-Rollen für Anwendungen und AWS Services, die Amazon SQS SQS-	
Zugriff benötigen	828
Implementieren serverseitiger Verschlüsselung	828
Erzwingen der Verschlüsselung von Daten während der Übertragung	829
Erwägen der Verwendung von VPC-Endpunkten für den Zugriff auf Amazon SQS	829
Zugehörige Ressourcen	830
Dokumentationsverlauf	831
.....	dcccxl

What is Amazon Simple Queue Service?

Amazon Simple Queue Service (Amazon SQS) bietet eine sichere, dauerhafte und verfügbare gehostete Warteschlange, die es Ihnen ermöglicht, verteilte Softwaresysteme und -komponenten zu integrieren und zu entkoppeln. Amazon SQS bietet gängige Konstrukte, wie z. B. [Warteschlangen für unzustellbare Nachrichten](#) und [Kostenzuordnungs-Tags](#). Es bietet eine generische Webservice-API, auf die Sie mit jeder vom AWS SDK unterstützten Programmiersprache zugreifen können.

Vorteile der Verwendung von Amazon SQS

- Sicherheit – [Sie steuern](#), wer Mitteilungen an eine Warteschlange senden und Mitteilungen von einer Amazon-SQS-Warteschlange empfangen darf. Sie können wählen, ob Sie vertrauliche Daten übertragen möchten, indem Sie den Inhalt von Nachrichten in Warteschlangen schützen, indem Sie die standardmäßige serverseitige Verschlüsselung (SSE) von Amazon SQS oder benutzerdefinierte [SSE](#)-Schlüssel verwenden, die in AWS Key Management Service (AWS KMS) verwaltet werden.
- Dauerhaftigkeit – Um die Sicherheit Ihrer Nachrichten zu gewährleisten, speichert Amazon SQS sie auf mehreren Servern. [Standardwarteschlangen unterstützen die at-least-once Nachrichtenzustellung, und FIFO-Warteschlangen unterstützen die Verarbeitung von Nachrichten genau einmal und den Modus mit hohem Durchsatz.](#)
- Verfügbarkeit – Amazon SQS verwendet eine [redundante Infrastruktur](#) für den simultanen Zugriff auf Nachrichten und hohe Verfügbarkeit zum Erstellen und Verwenden von Nachrichten.
- Skalierbarkeit – Amazon SQS kann jede [gepufferte Anfrage](#) unabhängig verarbeiten und transparent skalieren, um sämtliche zunehmenden Lasten oder Spitzen ohne Bereitstellungsanweisungen zu verarbeiten.
- Zuverlässigkeit – Amazon SQS sperrt Ihre Nachrichten während der Verarbeitung, damit mehrere Produzenten und mehrere Konsumenten Nachrichten gleichzeitig senden und empfangen können.
- Anpassung – Ihre Warteschlangen müssen nicht genau übereinstimmen, sie können z. B. [Standardverzögerung für eine Warteschlange festlegen](#). Sie können den Inhalt von Nachrichten, die größer als 1 MiB sind, [mit Amazon Simple Storage Service \(Amazon S3\)](#) oder Amazon DynamoDB speichern, wobei Amazon SQS einen Zeiger auf das Amazon S3-Objekt hält, oder Sie können eine große Nachricht in kleinere Nachrichten aufteilen.

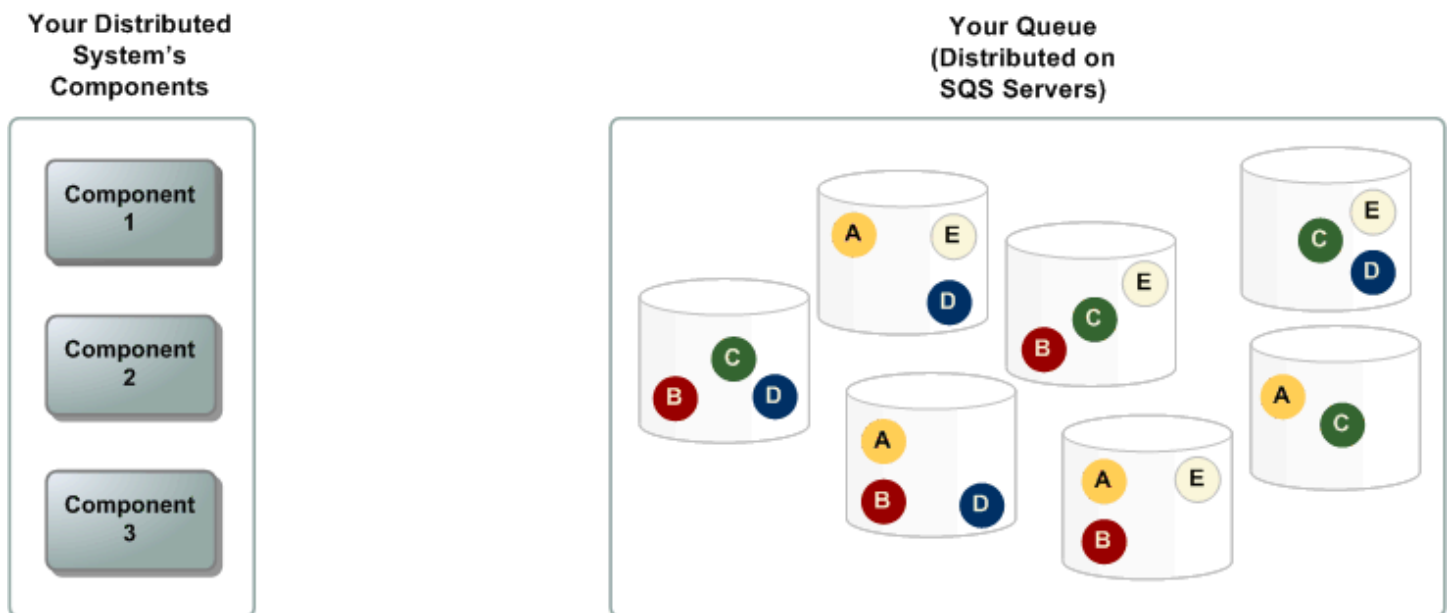
Grundlegende Amazon-SQS-Architektur

In diesem Abschnitt werden die Komponenten eines verteilten Messaging-Systems beschrieben und der Lebenszyklus einer Amazon SQS SQS-Nachricht erklärt.

Verteilte Warteschlangen

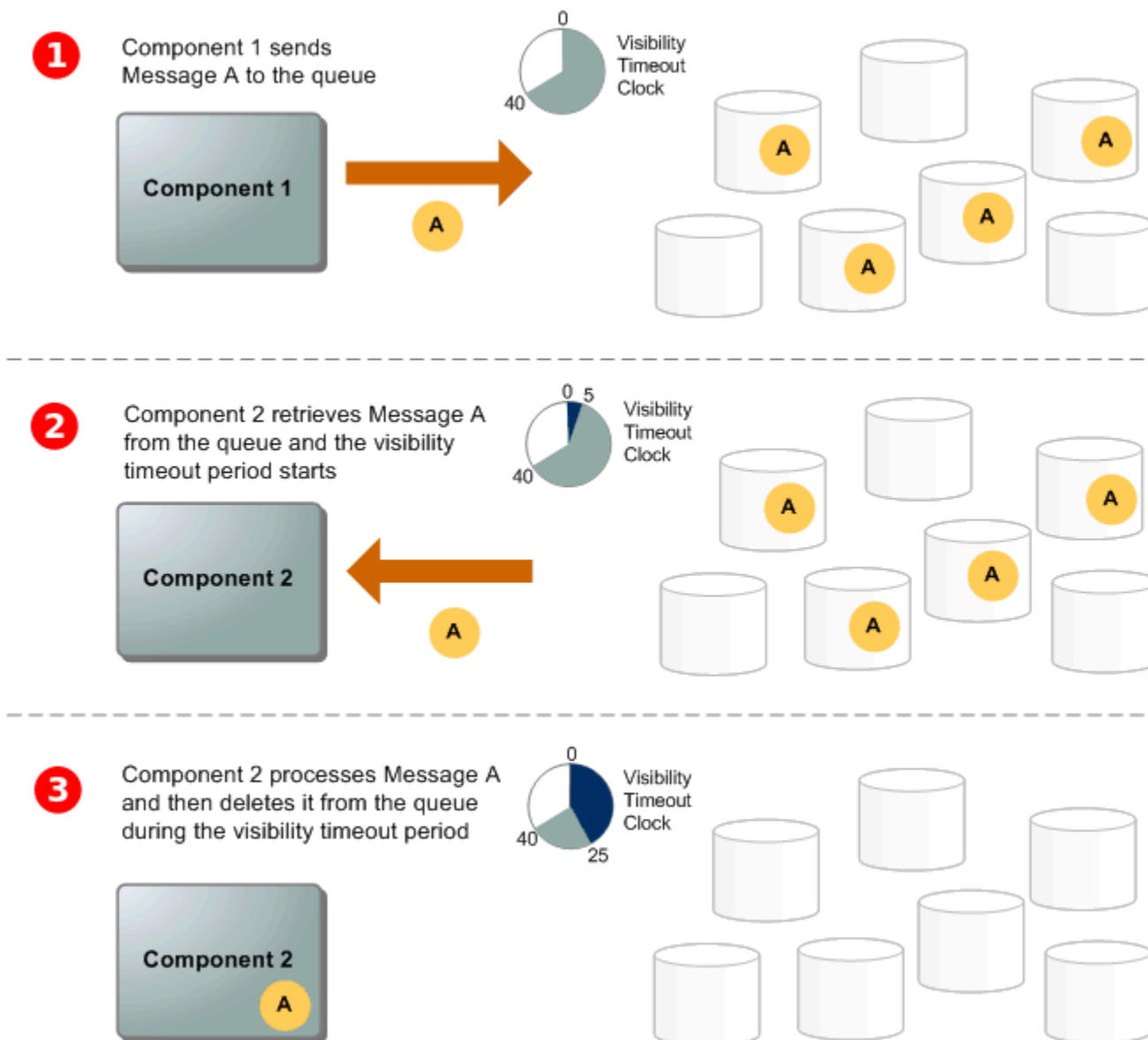
Ein verteiltes Messaging-System besteht aus drei Hauptteilen: den Komponenten Ihres verteilten Systems, Ihrer Warteschlange (verteilt auf Amazon SQS-Servern) und den Nachrichten in der Warteschlange.

Im folgenden Szenario hat Ihr System mehrere Produzenten (Komponenten, die Nachrichten an die Warteschlange senden) und Konsumenten (Komponenten, die Nachrichten aus der Warteschlange empfangen). Die Warteschlange (die die Nachrichten A bis E enthält) speichert die Nachrichten redundant auf mehreren Amazon-SQS-Servern.



Lebenszyklus einer Nachricht

Das folgende Szenario stellt den Lebenszyklus einer Amazon-SQS-Nachricht in einer Warteschlange von der Erstellung zur Löschung dar.



1 Producer (Komponente 1) sendet Nachricht A an eine Warteschlange, und die Nachricht wird redundant auf die Amazon SQS-Server verteilt.

Ein

2 ein Verbraucher (Komponente 2) bereit ist, Nachrichten zu verarbeiten, konsumiert er Nachrichten aus der Warteschlange, und Nachricht A wird zurückgegeben. Während Nachricht A verarbeitet wird, verbleibt sie in der Warteschlange und wird während der [Zeitbeschränkung für die Sichtbarkeit](#) nicht an nachfolgende Empfangsanforderungen zurückgegeben.

Wenn

3

Der

Verbraucher (Komponente 2) löscht Nachricht A aus der Warteschlange, um zu verhindern, dass die Nachricht empfangen und erneut verarbeitet wird, wenn das Sichtbarkeits-Timeout abgelaufen ist.

Note

Amazon SQS löscht automatisch Nachrichten, die länger als die maximale Aufbewahrungsdauer für Nachrichten in einer Warteschlange waren. Der Standardaufbewahrungszeitraum für Nachrichten beträgt 4 Tage. Sie können jedoch den Aufbewahrungszeitraum für Nachrichten mit der Aktion [SetQueueAttributes](#) von 60 Sekunden auf 1.209.600 Sekunden (14 Tage) festlegen.

Unterschiede zwischen Amazon SQS, Amazon MQ und Amazon SNS

Amazon SQS, [Amazon SNS](#) und [Amazon MQ](#) bieten hoch skalierbare und easy-to-use verwaltete Messaging-Dienste, die jeweils für bestimmte Rollen innerhalb verteilter Systeme konzipiert sind. Hier finden Sie einen erweiterten Überblick über die Unterschiede zwischen diesen Diensten:

Amazon SQS entkoppelt und skaliert verteilte Softwaresysteme und Komponenten als Warteschlangenservice. Es verarbeitet Nachrichten in der Regel über einen einzigen Abonnenten. Dies ist ideal für Workflows, bei denen Ordnung und Verlustprävention von entscheidender Bedeutung sind. Für eine breitere Verbreitung ermöglicht die Integration von Amazon SQS mit Amazon SNS ein [Fanout-Messaging-Muster, das](#) Nachrichten effektiv an mehrere Abonnenten gleichzeitig weiterleitet.

Amazon SNS ermöglicht es Verlagen, Nachrichten über Themen, die als Kommunikationskanäle dienen, an mehrere Abonnenten zu senden. Abonnenten erhalten veröffentlichte Nachrichten über einen unterstützten Endpunkttyp wie [Amazon SQS](#), [Amazon Data Firehose](#), [Lambda](#), HTTP, E-Mail, mobile Push-Benachrichtigungen und mobile Textnachrichten (SMS). Dieser Service ist ideal für Szenarien, in denen sofortige Benachrichtigungen erforderlich sind, z. B. für Benutzerinteraktionen in Echtzeit oder Alarmsysteme. Um Nachrichtenverlust zu verhindern, wenn Abonnenten offline sind, gewährleistet die Integration von Amazon SNS mit Amazon SQS SQS-Warteschlangennachrichten eine konsistente Zustellung.

Amazon MQ [eignet sich am besten für Unternehmen, die von herkömmlichen Message-Brokern migrieren möchten. Es unterstützt Standard-Messaging-Protokolle wie AMQP und MQTT sowie](#)

[Apache ActiveMQ und RabbitMQ](#). Es bietet Kompatibilität mit älteren Systemen, die ein stabiles, zuverlässiges Messaging benötigen, ohne dass eine umfangreiche Neukonfiguration erforderlich ist.

Die folgende Tabelle bietet einen Überblick über die Ressourcentypen der einzelnen Dienste:

Ressourcentyp	Amazon SNS	Amazon SQS	Amazon MQ
Synchron	Nein	Nein	Ja
Asynchron	Ja	Ja	Ja
Queues (Warteschlangen)	Nein	Ja	Ja
Messaging zwischen Publisher und Subscriber	Ja	Nein	Ja
Message Broker	Nein	Nein	Ja

Sowohl Amazon SQS als auch Amazon SNS werden für neue Anwendungen empfohlen, die von nahezu unbegrenzter Skalierbarkeit und Einfachheit profitieren können. APIs Sie bieten aufgrund ihrer Preisgestaltung im Allgemeinen kostengünstigere Lösungen für großvolumige Anwendungen. pay-as-you-go Wir empfehlen Amazon MQ für die Migration von Anwendungen von bestehenden Message Brokern, die auf Kompatibilität mit JMS oder Protokollen APIs wie Advanced Message Queuing Protocol (AMQP), MQTT und Simple Text Oriented Message Protocol (OpenWireSTOMP) angewiesen sind.

Erste Schritte mit Amazon SQS

Dieses Thema führt Sie durch die Verwendung der Amazon SQS SQS-Konsole zur Erstellung und Verwaltung von Standard- und FIFO-Warteschlangen. Sie erfahren, wie Sie in der Konsole navigieren, Warteschlangenattribute anzeigen und zwischen Warteschlangenarten unterscheiden. Zu den wichtigsten Aufgaben gehören das Senden, Empfangen und Konfigurieren von Nachrichten, das Anpassen von Parametern wie Sichtbarkeitszeitlimit und Nachrichtenaufbewahrung sowie die Verwaltung des Warteschlangenzugriffs mithilfe von Richtlinien.

Topics

- [Einrichten von Amazon SQS](#)
- [Die Amazon-SQS-Konsole verstehen](#)
- [Amazon-SQS-Warteschlangentypen](#)
- [Erstellen einer Amazon-SQS-Standard-Warteschlange und Senden einer Nachricht](#)
- [Erstellen einer Amazon-SQS-FIFO-Warteschlange und Senden einer Nachricht](#)
- [Allgemeine Aufgaben für die ersten Schritte mit Amazon SQS](#)

Einrichten von Amazon SQS

Bevor Sie Amazon SQS zum ersten Mal verwenden können, müssen Sie die folgenden Schritte ausführen:

Schritt 1: Erstellen Sie ein AWS-Konto und IAM-Benutzer

Um auf einen AWS Dienst zugreifen zu können, müssen Sie zunächst ein Amazon.com Konto erstellen [AWS-Konto](#), mit dem AWS Produkte verwendet werden können. Sie können Ihr verwendetes AWS-Konto , um Ihre Aktivitäts- und Nutzungsberichte einzusehen und die Authentifizierung und den Zugriff zu verwalten.

Um zu vermeiden, dass Ihr AWS-Konto Root-Benutzer für Amazon SQS-Aktionen verwendet wird, empfiehlt es sich, für jede Person, die Administratorzugriff auf Amazon SQS benötigt, einen IAM-Benutzer zu erstellen.

Melden Sie sich an für ein AWS-Konto

Um loszulegen AWS, benötigen Sie eine AWS-Konto. Informationen zum Erstellen eines AWS-Konto finden Sie unter [Erste Schritte mit einem AWS-Konto](#) im AWS -Kontenverwaltung Referenzhandbuch.

Schritt 2: Erteilen programmgesteuerten Zugriffs

Um Amazon SQS SQS-Aktionen zu verwenden (z. B. mit Java oder über AWS Command Line Interface), benötigen Sie eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel.

Note

Die Zugriffsschlüssel-ID und der geheime Zugriffsschlüssel sind spezifisch für AWS Identity and Access Management. Verwechseln Sie sie nicht mit Anmeldeinformationen für andere AWS Dienste, wie z. B. Amazon EC2 EC2-Schlüsselpaaren.

Benutzer benötigen programmgesteuerten Zugriff, wenn sie mit AWS außerhalb des Interagierens möchten. AWS-Managementkonsole Die Art und Weise, wie programmatischer Zugriff gewährt wird, hängt vom Benutzertyp ab, der zugreift. AWS

Um Benutzern programmgesteuerten Zugriff zu gewähren, wählen Sie eine der folgenden Optionen.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Empfohlen) Verwenden Sie Konsolenanmeldedaten als temporäre Anmeldeinformationen, um programmatische Anfragen an die AWS CLI, AWS SDKs oder APIs zu signieren. AWS	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen zum AWS CLI finden Sie unter Anmeldung für AWS lokale Entwicklung im AWS Command Line Interface Benutzerhandbuch. • Informationen zu AWS SDKs finden Sie unter

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
		Anmeldung für AWS lokale Entwicklung im Referenzhandbuch für AWS SDKs und Tools.
Mitarbeiteridentität (Benutzer, die in IAM Identity Center verwaltet werden)	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an die AWS CLI, AWS SDKs oder APIs zu signieren. AWS	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen zu den AWS CLI finden Sie unter Konfiguration der AWS CLI zu AWS IAM Identity Center verwendenden im AWS Command Line Interface Benutzerhandbuch. • Informationen zu AWS SDKs, Tools und AWS APIs finden Sie unter IAM Identity Center-Authentifizierung im Referenzhandbuch für AWS SDKs und Tools.
IAM	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an die AWS CLI, AWS SDKs oder APIs zu signieren. AWS	Folgen Sie den Anweisungen unter Verwenden temporärer Anmeldeinformationen mit AWS Ressourcen im IAM-Benutzerhandbuch.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Nicht empfohlen) Verwenden Sie langfristige Anmeldeinformationen, um programmatische Anfragen an die AWS CLI, AWS SDKs oder APIs zu signieren. AWS	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen dazu finden Sie unter Authentifizierung mithilfe von IAM-Benutzeranmeldedaten im Benutzerhandbuch. AWS CLI AWS Command Line Interface • Informationen zu AWS SDKs und Tools finden Sie unter Authentifizieren mit langfristigen Anmeldeinformationen im Referenzhandbuch für AWS SDKs und Tools. • Informationen zu AWS APIs finden Sie unter Verwaltung von Zugriffsschlüsseln für IAM-Benutzer im IAM-Benutzerhandbuch.

Schritt 3: Vorbereiten der Verwendung des Beispiel-Codes

Dieses Handbuch enthält Beispiele, die das AWS SDK for Java verwenden. Um den Beispiel-Code auszuführen, folgen Sie den Anweisungen zur Einrichtung unter [Erste Schritte mit AWS SDK für Java 2.0](#).

Sie können AWS Anwendungen in anderen Programmiersprachen wie GoJavaScript, Python und Ruby entwickeln. Weitere Informationen finden Sie unter [Tools, auf denen Sie aufbauen können AWS](#).

Note

Mit Tools wie AWS Command Line Interface (AWS CLI) oder Windows PowerShell können Sie Amazon SQS erkunden, ohne Code schreiben zu müssen. AWS CLI Beispiele finden Sie im [Abschnitt Amazon SQS](#) der AWS CLI Befehlsreferenz. PowerShell Windows-Beispiele finden Sie im Abschnitt Amazon Simple Queue Service der [AWS -Tools für PowerShell Cmdlet-Referenz](#).

Nächste Schritte

Sie sind jetzt bereit für [Erste Schritte](#) mit der Verwaltung von Amazon-SQS-Warteschlangen und -Nachrichten mithilfe der AWS-Managementkonsole.

Die Amazon-SQS-Konsole verstehen

Wenn Sie die Amazon SQS SQS-Konsole öffnen, wählen Sie im Navigationsbereich Warteschlangen aus. Die Seite Warteschlangen enthält Informationen zu allen Ihren Warteschlangen in der aktiven Region.

Jeder Warteschlangeneintrag enthält wichtige Informationen über die Warteschlange, einschließlich ihres Typs und ihrer wichtigsten Attribute. [Standardwarteschlangen](#), die für maximalen Durchsatz und optimale Nachrichtenreihenfolge optimiert sind, unterscheiden sich von [First-In-First-Out \(FIFO-\) Warteschlangen](#), bei denen die Reihenfolge und Eindeutigkeit von Nachrichten für Anwendungen, die eine strikte Nachrichtensequenzierung erfordern, Priorität eingeräumt wird.

Queues (2)			Edit	Delete	Send and receive messages	Actions ▼	Create queue
Search queues by prefix							
	Name ▲	Type ▼	Created ▼	Messages available ▼	Messages in flight ▼	Encryption ▼	Content-based deduplication ▼
☐	MyTestQueue	Standard	6/27/2022, 13:03:08 EDT	0	0	Disabled	-
☐	testFifo1.fifo	FIFO	6/27/2022, 13:03:41 EDT	0	0	Disabled	Disabled

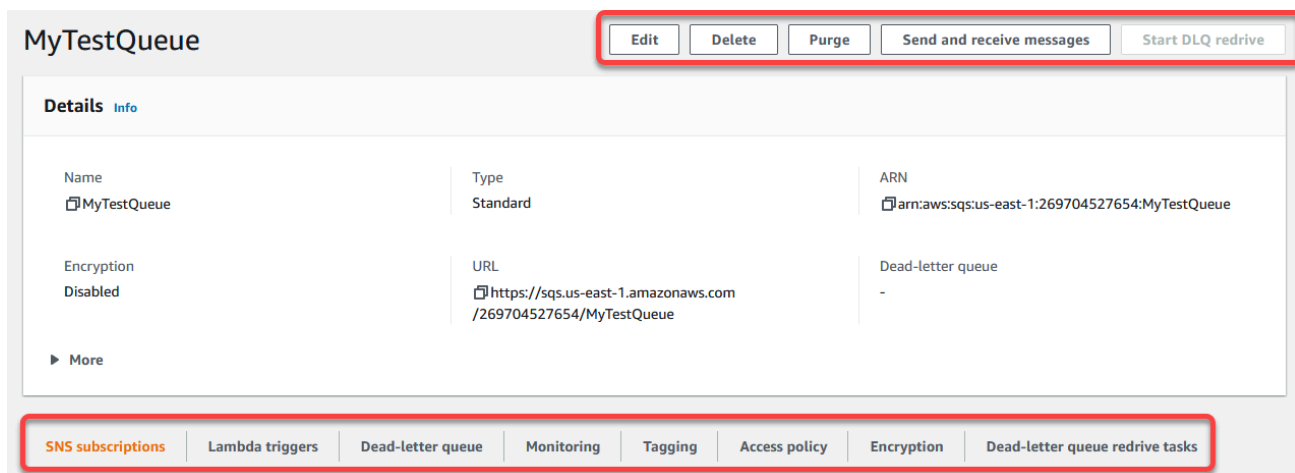
Interaktive Elemente und Aktionen

Auf der Seite Warteschlangen haben Sie mehrere Möglichkeiten, Ihre Warteschlangen zu verwalten:

1. Schnellaktionen — Neben jedem Warteschlangennamen bietet ein Dropdownmenü schnellen Zugriff auf häufig verwendete Aktionen wie das Senden von Nachrichten, das Anzeigen oder

Löschen von Nachrichten, das Konfigurieren von Triggern und das Löschen der Warteschlange selbst.

2. Detaillierte Ansicht und Konfiguration — Wenn Sie auf einen Warteschlangennamen klicken, wird die zugehörige Detailseite geöffnet, auf der Sie sich eingehender mit den Einstellungen und Konfigurationen der Warteschlange befassen können. Hier können Sie Parameter wie die Aufbewahrungsdauer von Nachrichten, das Zeitlimit für die Sichtbarkeit und die maximale Nachrichtengröße anpassen, um die Warteschlange an die Anforderungen Ihrer Anwendung anzupassen.



Regionsauswahl und Ressourcen-Tags

Stellen Sie sicher, dass Sie richtig sind AWS-Region , um effektiv auf Ihre Warteschlangen zuzugreifen und sie zu verwalten. Erwägen Sie außerdem, Ressourcen-Tags zu verwenden, um Ihre Warteschlangen zu organisieren und zu kategorisieren, was eine bessere Ressourcenverwaltung, Kostenzuweisung und Zugriffskontrolle in Ihrer AWS gemeinsamen Umgebung ermöglicht.

Durch die Nutzung der in der Amazon SQS SQS-Konsole angebotenen Funktionen können Sie Ihre Messaging-Infrastruktur effizient verwalten, die Warteschlangenleistung optimieren und eine zuverlässige Nachrichtenzustellung für Ihre Anwendungen sicherstellen.

Amazon-SQS-Warteschlangentypen

[Amazon SQS unterstützt zwei Arten von Warteschlangen: Standardwarteschlangen und FIFO-Warteschlangen.](#) Ermitteln Sie anhand der folgenden Tabelle, welche Warteschlange Ihren Anforderungen am besten entspricht.

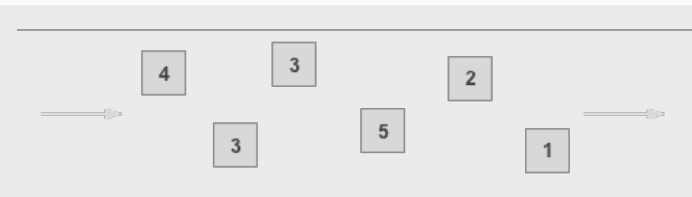
Standard-Warteschlangen	FIFO-Warteschlangen
Unbegrenzter Durchsatz — Standardwarteschlangen unterstützen eine sehr hohe, nahezu unbegrenzte Anzahl von API-Aufrufen pro Sekunde und Aktion (, SendMessage ReceiveMessage , oder). DeleteMessage Durch diesen hohen Durchsatz eignen sie sich ideal für Anwendungsfälle, bei denen große Nachrichtenmengen schnell verarbeitet werden müssen, z. B. Datenstreaming in Echtzeit oder umfangreiche Anwendungen. Standardwarteschlangen werden zwar automatisch mit der Nachfrage skaliert, es ist jedoch wichtig, die Nutzungsmuster zu überwachen, um eine optimale Leistung zu gewährleisten, insbesondere in Regionen mit höherer Arbeitslast. At-least-once Zustellung — garantierte at-least-once Zustellung, d. h. jede Nachricht wird mindestens einmal zugestellt. In einigen Fällen kann es jedoch vorkommen, dass eine Nachricht aufgrund von Wiederholungsversuchen oder Netzwerkverzögerungen mehrmals zugestellt wird. Sie sollten Ihre Anwendung so gestalten, dass sie potenzielle doppelte Nachrichten mithilfe idempotenter Operationen verarbeitet, die sicherstellen, dass die mehrfache Verarbeitung derselben Nachricht den Systemstatus nicht beeinträchtigt. Beste-Effort-Bestellung — Ermöglicht Bestellungen nach bestem Wissen. Das bedeutet, dass Amazon SQS zwar versucht, Nachrichten in der Reihenfolge zuzustellen, in der sie gesendet wurden, dies jedoch nicht garantiert. In einigen	Hoher Durchsatz — Wenn Sie Batching verwenden, verarbeiten FIFO-Warteschlangen bis zu 3.000 Nachrichten pro Sekunde pro API-Methode (, oder). SendMessageBatch ReceiveMessageBatch DeleteMessageBatch Dieser Durchsatz basiert auf 300 API-Aufrufen pro Sekunde, wobei jeder API-Aufruf einen Stapel von 10 Nachrichten verarbeitet. Wenn Sie den Hochdurchsatzmodus aktivieren, können Sie bis zu 30.000 Transaktionen pro Sekunde (TPS) skalieren und dabei die Reihenfolge innerhalb von Nachrichtengruppen lockerer gestalten. Ohne Batching unterstützen FIFO-Warteschlangen bis zu 300 API-Aufrufe pro Sekunde pro API-Methode (, oder). SendMessage ReceiveMessage DeleteMessage Wenn Sie mehr Durchsatz benötigen, können Sie über das AWS Support Center eine Erhöhung des Kontingents beantragen. Informationen zur Aktivierung des Hochdurchsatzmodus finden Sie unter Aktivierung eines hohen Durchsatzes für FIFO-Warteschlangen in Amazon SQS. Exactly-Once-Verarbeitung — FIFO-Warteschlangen stellen jede Nachricht einmal zu und halten sie verfügbar, bis Sie sie verarbeiten und löschen. Durch die Verwendung von Funktionen wie MessageDeduplicationId oder inhaltsbasierter Deduplizierung verhindern Sie doppelte Nachrichten, selbst wenn Sie es aufgrund von Netzwerkproblemen oder Timeouts erneut versuchen.

Standard-Warteschlangen

Fällen können Nachrichten nicht in der richtigen Reihenfolge ankommen, insbesondere bei hohem Durchsatz oder bei der Wiederherstellung nach einem Ausfall. Bei Anwendungen, bei denen die Reihenfolge der Nachrichtenverarbeitung entscheidend ist, sollten Sie die Logik der Neuordnung innerhalb der Anwendung regeln oder FIFO-Warteschlangen verwenden, um strikte Sortiergarantien zu gewährleisten.

Beständigkeit und Redundanz — Standardwarteschlangen sorgen für eine hohe Haltbarkeit, indem sie mehrere Kopien jeder Nachricht in mehreren Availability Zones speichern. AWS Dadurch wird sichergestellt, dass Nachrichten auch bei Infrastrukturausfällen nicht verloren gehen.

Sichtbarkeits-Timeout — Amazon SQS ermöglicht es Ihnen, ein Sichtbarkeits-Timeout zu konfigurieren, um zu kontrollieren, wie lange eine Nachricht nach dem Empfang verborgen bleibt. Dadurch wird sichergestellt, dass andere Verbraucher die Nachricht erst verarbeiten, wenn sie vollständig bearbeitet wurde oder der Timeout abgelaufen ist.



FIFO-Warteschlangen

First-in-first-out Zustellung — FIFO-Warteschlangen stellen sicher, dass Sie Nachrichten in der Reihenfolge erhalten, in der sie innerhalb der einzelnen Nachrichtengruppen gesendet wurden. Indem Sie Nachrichten auf mehrere Gruppen verteilen, können Sie sie parallel verarbeiten und gleichzeitig die Reihenfolge innerhalb der einzelnen Gruppen beibehalten.



Standard-Warteschlangen	FIFO-Warteschlangen
Verwenden Sie Standardwarteschlangen, um Daten zwischen Anwendungen zu senden, wenn der Durchsatz entscheidend ist, zum Beispiel: • Entkoppeln Sie Live-Benutzeranfragen von intensiver Hintergrundarbeit. Ermöglichen Sie es Benutzern, Medien schnell hochzuladen, während Sie Aufgaben wie das Ändern der Größe oder das Kodieren im Hintergrund bearbeiten, und sorgen Sie so für schnelle Reaktionszeiten, ohne das System zu überlasten. • Weisen Sie Aufgaben mehreren Worker-Knoten zu. Verteilen Sie eine große Anzahl von Anfragen zur Kreditkartenvalidierung auf mehrere Worker-Nodes und verarbeiten Sie doppelte Nachrichten mit idempotenten Operationen, um Verarbeitungsfehler zu vermeiden. • Batch-Nachrichten für die future Verarbeitung. Stellen Sie mehrere Einträge für Batch-Ergänzungen zu einer Datenbank in eine Warteschlange. Da die Reihenfolge der Nachrichten nicht garantiert werden kann, sollten Sie Ihr System so einrichten, dass es die out-of-order Verarbeitung bei Bedarf übernimmt.	Verwenden Sie FIFO-Warteschlangen, um Daten zwischen Anwendungen zu senden, wenn die Reihenfolge der Ereignisse wichtig ist, zum Beispiel: • Stellen Sie sicher, dass vom Benutzer eingegebene Befehle in der richtigen Reihenfolge ausgeführt werden. Dies ist ein wichtiger Anwendungsfall für FIFO-Warteschlangen, bei denen die Befehlsreihenfolge entscheidend ist. Wenn ein Benutzer beispielsweise eine Abfolge von Aktionen in einer Anwendung ausführt, stellen FIFO-Warteschlangen sicher, dass die Aktionen in derselben Reihenfolge verarbeitet werden, in der sie eingegeben wurden. • Zeigen Sie den richtigen Produktpreis an, indem Sie Preisänderungen in der richtigen Reihenfolge senden. FIFO-Warteschlangen stellen sicher, dass mehrere Preisaktualisierungen für ein Produkt eintreffen und nacheinander verarbeitet werden. Ohne FIFO könnte eine Preissenkung nach einer Preiserhöhung verarbeitet werden, was dazu führt, dass falsche Daten angezeigt werden. • Verhindern Sie, dass sich ein Student für einen Kurs anmeldet, bevor er sich für ein Konto registriert hat. Durch die Verwendung von FIFO-Warteschlangen stellen Sie sicher, dass der Registrierungsprozess in der richtigen Reihenfolge abläuft. Das System verarbeitet zuerst die Kontoregistrierung und dann die Kursanmeldung, wodurch verhindert

Standard-Warteschlangen	FIFO-Warteschlangen
	t wird, dass die Anmeldungsanfrage vorzeitig ausgeführt wird.

Implementierung von Anfrage-Antwort-Systemen in Amazon SQS

Beachten Sie beim Implementieren eines Request-Response- oder Remoteprozeduraufrufs (RPC)-Systems die folgenden bewährten Vorgehensweisen:

- Antwortwarteschlangen beim Start erstellen — Anstatt Antwortwarteschlangen pro Nachricht zu erstellen, sollten Sie sie beim Start pro Hersteller erstellen. Verwenden Sie ein Nachrichtenattribut mit Korrelations-ID, um Antworten effizient Anfragen zuzuordnen.
- Vermeiden Sie die gemeinsame Nutzung von Antwortwarteschlangen unter den Produzenten — Stellen Sie sicher, dass jeder Produzent seine eigene Antwortwarteschlange hat. Die gemeinsame Nutzung von Antwortwarteschlangen kann dazu führen, dass ein Produzent Antwortnachrichten erhält, die für einen anderen Hersteller bestimmt sind.

Weitere Informationen zum Implementieren der request-response-Muster mit dem Client temporärer Warteschlangen finden Sie unter [Request-Response-Messaging-Muster \(virtuelle Warteschlangen\)](#).

Erstellen einer Amazon-SQS-Standard-Warteschlange und Senden einer Nachricht

Mit der Amazon SQS SQS-Konsole können Sie eine [Standardwarteschlange](#) erstellen und Nachrichten senden. In diesem Thema werden auch bewährte Methoden hervorgehoben, darunter die Vermeidung vertraulicher Informationen in Warteschlangennamen und die Verwendung verwalteter serverseitiger Verschlüsselung.

Erstellen einer Standardwarteschlange mit der Amazon SQS SQS-Konsole

Important

Am 17. August 2022 wurde die serverseitige Verschlüsselung (SSE) standardmäßig auf alle Amazon-SQS-Warteschlangen angewendet.

Fügen Sie keine persönlich identifizierbare Informationen (PII) oder andere vertrauliche oder sensible Informationen in Warteschlangennamen hinzu. Warteschlangennamen sind für viele Amazon Web Services zugänglich, einschließlich Abrechnung und CloudWatch Protokollen. Warteschlangennamen sind nicht für private oder sensible Daten gedacht.

So erstellen Sie eine Amazon-SQS-Standard-Warteschlange

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie Create queue (Warteschlange erstellen) aus.
3. Für Typ ist standardmäßig der Standard-Warteschlangentyp festgelegt.

 Note

Sie können den Warteschlangentyp nicht ändern, nachdem Sie die Warteschlange erstellt haben.

4. Geben Sie einen Namen für die Warteschlange ein.
5. (Optional) Die Konsole legt Standardwerte für die [Konfigurationsparameter](#) der Warteschlange fest. Unter Konfiguration können Sie neue Werte für die folgenden Parameter festlegen:
 - a. Geben Sie für Sichtbarkeitszeitbeschränkung die Dauer und die Einheiten ein. Der Bereich liegt zwischen 0 und 12 Stunden. Der Standardwert ist 30 Sekunden.
 - b. Geben Sie unter Aufbewahrungszeitraum für Nachrichten die Dauer und die Einheiten ein. Der Bereich liegt zwischen 1 Minute und 14 Tagen. Der Standardwert ist 4 Tage.
 - c. Geben Sie für Zustellungsverzögerung die Dauer und die Einheiten ein. Der Bereich liegt zwischen 0 Sekunden und 15 Minuten. Der Standardwert ist 0 Sekunden.
 - d. Geben Sie für Maximale Nachrichtengröße einen Wert ein. Der Bereich reicht von 1 KiB bis 1024 KiB. Der Standardwert ist 1024 KiB.
 - e. Geben Sie für Wartezeit für den Empfang von Nachrichten einen Wert ein. Der Bereich liegt zwischen 0 und 20 Sekunden. Der Standardwert ist 0 Sekunden, der [kurze Abfragen](#) festlegt. Jeder Wert ungleich Null führt zu einer langen Abfrage.
6. (Optional) Definieren Sie eine Zugriffsrichtlinie. Die [Zugriffsrichtlinie](#) definiert die Konten, Benutzer und Rollen, die auf die Warteschlange zugreifen können. Die Zugriffsrichtlinie definiert auch die Aktionen (wie [SendMessage](#), [ReceiveMessage](#) oder [DeleteMessage](#)),

auf die die Benutzer zugreifen können. Die Standardrichtlinie erlaubt nur dem Eigentümer der Warteschlange, Nachrichten zu senden und zu empfangen.

Führen Sie zum Definieren der Zugriffsrichtlinie einen der folgenden Schritte aus:

- Wählen Sie Einfach, um zu konfigurieren, wer Nachrichten an die Warteschlange senden und wer Nachrichten aus der Warteschlange empfangen kann. Die Konsole erstellt die Richtlinie auf der Grundlage Ihrer Auswahl und zeigt die resultierende Zugriffsrichtlinie im schreibgeschützten JSON-Bereich an.
 - Wählen Sie Erweitert, um die JSON-Zugriffsrichtlinie direkt zu ändern. Auf diese Weise können Sie einen benutzerdefinierten Satz von Aktionen angeben, die jeder Prinzipal (Konto, Benutzer oder Rolle) ausführen kann.
7. Wählen Sie für die Redrive-Zulassungsrichtlinie die Option Aktiviert aus. Wählen Sie eine der folgenden Optionen aus: Alle zulassen, Nach Warteschlange oder Alle verweigern. Wenn Sie Nach Warteschlange wählen, geben Sie eine Liste mit bis zu 10 Quellwarteschlangen nach dem Amazon-Ressourcennamen (ARN) an.
 8. Amazon SQS bietet standardmäßig verwaltete serverseitige Verschlüsselung. Um einen Verschlüsselungsschlüsseltyp auszuwählen oder die von Amazon SQS verwaltete serverseitige Verschlüsselung zu deaktivieren, erweitern Sie Verschlüsselung. Weitere Informationen zu Verschlüsselungsschlüsseltypen finden Sie unter [Konfiguration der serverseitigen Verschlüsselung für eine Warteschlange mithilfe von SQS-verwalteten Verschlüsselungsschlüsseln](#) und [Konfiguration der serverseitigen Verschlüsselung für eine Warteschlange mithilfe der Amazon SQS SQS-Konsole](#).
-  **Note**

Wenn SSE aktiviert ist, werden anonyme SendMessage- und ReceiveMessage-Anfragen an die verschlüsselte Warteschlange abgewiesen. Die bewährten Sicherheitsmethoden von Amazon SQS raten davon ab, anonyme Anfragen zu verwenden. Wenn Sie anonyme Anfragen an eine Amazon-SQS-Warteschlange senden möchten, stellen Sie sicher, dass SSE deaktiviert ist.
9. (Optional) Um eine [Warteschlange für unzustellbare Nachrichten](#) für den Empfang von unzustellbaren Nachrichten zu konfigurieren, erweitern Sie Warteschlange für unzustellbare Nachrichten.
 10. (Optional) Erweitern Sie Tags, um der Warteschlange [Tags](#) hinzuzufügen.

11. Wählen Sie **Create queue** (Warteschlange erstellen) aus. Amazon SQS erstellt die Warteschlange und zeigt die Seite **Details** der Warteschlange an.

Amazon SQS verbreitet Informationen über die neue Warteschlange im gesamten System. Da es sich bei Amazon SQS um ein verteiltes System handelt, kann es zu einer leichten Verzögerung kommen, bevor die Konsole die Warteschlange auf der Warteschlangenseite anzeigt.

Senden einer Nachricht mithilfe einer Standardwarteschlange

Nachdem Ihre Warteschlange erstellt wurde, können Sie eine Nachricht an sie senden.

1. Wählen Sie im linken Navigationsbereich **Warteschlangen** aus. Wählen Sie in der Warteschlangenliste die Warteschlange aus, die Sie erstellt haben.
2. Wählen Sie unter **Aktionen** die Option **Nachrichten senden und empfangen**.

In der Konsole wird die Seite **Nachrichten senden und empfangen** angezeigt.

3. Geben Sie unter **Nachricht** den Nachrichtentext ein.
4. Für eine Standard-Warteschlange können Sie einen Wert für die Lieferversögerung eingeben und die Einheiten auswählen. Geben Sie beispielsweise **60** ein und wählen Sie **Sekunden**. Weitere Informationen finden Sie unter [Amazon-SQS-Nachrichten-Timer](#).
5. Klicken Sie auf **Send Message** (Nachricht senden).

Wenn Ihre Nachricht gesendet wurde, zeigt die Konsole eine Erfolgsmeldung an. Wählen Sie **Details anzeigen**, um Informationen zur gesendeten Nachricht anzuzeigen.

Erstellen einer Amazon-SQS-FIFO-Warteschlange und Senden einer Nachricht

Mit der Konsole können Sie eine Amazon SQS FIFO-Warteschlange erstellen und Nachrichten senden. In diesem Thema wird erklärt, wie Warteschlangenparameter, einschließlich Sichtbarkeits-Timeout, Nachrichtenspeicherung und Deduplizierung, eingerichtet und dabei bewährte Sicherheitsmethoden wie die Vermeidung vertraulicher Informationen in Warteschlangennamen und die Aktivierung der serverseitigen Verschlüsselung befolgt werden. Außerdem werden die Definition von Zugriffsrichtlinien, die Konfiguration von Warteschlangen für unberechtigte Nachrichten und das Senden von Nachrichten mit FIFO-spezifischen Attributen wie Nachrichtengruppen-ID und Deduplizierungs-ID behandelt.

Erstellen einer FIFO-Warteschlange mit der Amazon SQS SQS-Konsole

Sie können mit der Amazon-SQS-Konsole [FIFO-Warteschlangen](#) erstellen. Die Konsole bietet Standardwerte für alle Einstellungen mit Ausnahme des Warteschlangennamens.

Important

Am 17. August 2022 wurde die serverseitige Verschlüsselung (SSE) standardmäßig auf alle Amazon-SQS-Warteschlangen angewendet.

Fügen Sie keine persönlich identifizierbare Informationen (PII) oder andere vertrauliche oder sensible Informationen in Warteschlangennamen hinzu. Warteschlangennamen sind für viele Amazon Web Services zugänglich, einschließlich Abrechnung und CloudWatch Protokollen. Warteschlangennamen sind nicht für private oder sensible Daten gedacht.

So erstellen Sie eine Amazon-SQS-FIFO-Warteschlange

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie Create queue (Warteschlange erstellen) aus.
3. Für Typ ist standardmäßig der Standard-Warteschlangentyp festgelegt. Um eine FIFO-Warteschlange zu erstellen, wählen Sie FIFO.

Note

Sie können den Warteschlangentyp nicht ändern, nachdem Sie die Warteschlange erstellt haben.

4. Geben Sie einen Namen für die Warteschlange ein.

Der Name einer FIFO-Warteschlange muss mit dem Suffix `.fifo` enden. Das Suffix wird auf das Kontingent für Warteschlangennamen mit 80 Zeichen angerechnet. Um festzustellen, ob es sich bei einer Warteschlange um eine [FIFO-Warteschlange](#) handelt, können Sie überprüfen, ob der Warteschlangename mit dem Suffix endet.

5. (Optional) Die Konsole legt Standardwerte für die [Konfigurationsparameter](#) der Warteschlange fest. Unter Konfiguration können Sie neue Werte für die folgenden Parameter festlegen:
 - a. Geben Sie für Sichtbarkeitszeitbeschränkung die Dauer und die Einheiten ein. Der Bereich liegt zwischen 0 und 12 Stunden. Der Standardwert ist 30 Sekunden.

- b. Geben Sie unter Aufbewahrungszeitraum für Nachrichten die Dauer und die Einheiten ein. Der Bereich liegt zwischen 1 Minute und 14 Tagen. Der Standardwert ist 4 Tage.
- c. Geben Sie für Zustellungsverzögerung die Dauer und die Einheiten ein. Der Bereich liegt zwischen 0 Sekunden und 15 Minuten. Der Standardwert ist 0 Sekunden.
- d. Geben Sie für Maximale Nachrichtengröße einen Wert ein. Der Bereich reicht von 1 KiB bis 1024 KiB. Der Standardwert ist 1024 KiB.
- e. Geben Sie für Wartezeit für den Empfang von Nachrichten einen Wert ein. Der Bereich liegt zwischen 0 und 20 Sekunden. Der Standardwert ist 0 Sekunden, der [kurze Abfragen](#) festlegt. Jeder Wert ungleich Null führt zu einer langen Abfrage.
- f. Wählen Sie für eine FIFO-Warteschlange Inhaltsbasierte Deduplizierung, um die inhaltsbasierte Deduplizierung zu aktivieren. Die Standardeinstellung ist deaktiviert.
- g. (Optional) Damit eine FIFO-Warteschlange einen höheren Durchsatz für das Senden und Empfangen von Nachrichten in der Warteschlange ermöglicht, wählen Sie FIFO mit hohem Durchsatz aktivieren.

Wenn Sie diese Option wählen, werden die zugehörigen Optionen (Deduplizierungsbereich und FIFO-Durchsatz-Limit) auf die erforderlichen Einstellungen geändert, um einen hohen Durchsatz für FIFO-Warteschlangen zu aktivieren. Wenn Sie Einstellungen ändern, die für die Verwendung von FIFO mit hohem Durchsatz erforderlich sind, ist der normale Durchsatz für die Warteschlange wirksam und die Deduplizierung erfolgt wie angegeben. Weitere Informationen erhalten Sie unter [Hoher Durchsatz für FIFO-Warteschlangen in Amazon SQS](#) und [Amazon SQS SQS-Nachrichtenkontingente](#).

- 6. (Optional) Definieren Sie eine Zugriffsrichtlinie. Die [Zugriffsrichtlinie](#) definiert die Konten, Benutzer und Rollen, die auf die Warteschlange zugreifen können. Die Zugriffsrichtlinie definiert auch die Aktionen (wie [SendMessage](#), [ReceiveMessage](#) oder [DeleteMessage](#)), auf die die Benutzer zugreifen können. Die Standardrichtlinie erlaubt nur dem Eigentümer der Warteschlange, Nachrichten zu senden und zu empfangen.

Führen Sie zum Definieren der Zugriffsrichtlinie einen der folgenden Schritte aus:

- Wählen Sie Einfach, um zu konfigurieren, wer Nachrichten an die Warteschlange senden und wer Nachrichten aus der Warteschlange empfangen kann. Die Konsole erstellt die Richtlinie auf der Grundlage Ihrer Auswahl und zeigt die resultierende Zugriffsrichtlinie im schreibgeschützten JSON-Bereich an.

- Wählen Sie Erweitert, um die JSON-Zugriffsrichtlinie direkt zu ändern. Auf diese Weise können Sie einen benutzerdefinierten Satz von Aktionen angeben, die jeder Prinzipal (Konto, Benutzer oder Rolle) ausführen kann.
7. Wählen Sie für die Redrive-Zulassungsrichtlinie die Option Aktiviert aus. Wählen Sie eine der folgenden Optionen aus: Alle zulassen, Nach Warteschlange oder Alle verweigern. Wenn Sie Nach Warteschlange wählen, geben Sie eine Liste mit bis zu 10 Quellwarteschlangen nach dem Amazon-Ressourcennamen (ARN) an.
 8. Amazon SQS bietet standardmäßig verwaltete serverseitige Verschlüsselung. Um einen Verschlüsselungsschlüsseltyp auszuwählen oder die von Amazon SQS verwaltete serverseitige Verschlüsselung zu deaktivieren, erweitern Sie Verschlüsselung. Weitere Informationen zu Verschlüsselungsschlüsseltypen finden Sie unter [Konfiguration der serverseitigen Verschlüsselung für eine Warteschlange mithilfe von SQS-verwalteten Verschlüsselungsschlüsseln](#) und [Konfiguration der serverseitigen Verschlüsselung für eine Warteschlange mithilfe der Amazon SQS SQS-Konsole](#).
-  **Note**

Wenn SSE aktiviert ist, werden anonyme SendMessage- und ReceiveMessage-Anfragen an die verschlüsselte Warteschlange abgewiesen. Die bewährten Sicherheitsmethoden von Amazon SQS raten davon ab, anonyme Anfragen zu verwenden. Wenn Sie anonyme Anfragen an eine Amazon-SQS-Warteschlange senden möchten, stellen Sie sicher, dass SSE deaktiviert ist.
9. (Optional) Um eine [Warteschlange für unzustellbare Nachrichten](#) für den Empfang von unzustellbaren Nachrichten zu konfigurieren, erweitern Sie Warteschlange für unzustellbare Nachrichten.
 10. (Optional) Erweitern Sie Tags, um der Warteschlange [Tags](#) hinzuzufügen.
 11. Wählen Sie Create queue (Warteschlange erstellen) aus. Amazon SQS erstellt die Warteschlange und zeigt die Seite Details der Warteschlange an.

Amazon SQS verbreitet Informationen über die neue Warteschlange im gesamten System. Da es sich bei Amazon SQS um ein verteiltes System handelt, kann es zu einer leichten Verzögerung kommen, bevor die Konsole die Warteschlange auf der Warteschlangenseite anzeigt.

Nachdem Sie eine Warteschlange erstellt haben, können Sie [Nachrichten an sie senden](#) und Nachrichten [empfangen und löschen](#). Sie können auch alle Einstellungen der Warteschlangenkonfiguration [bearbeiten](#), mit Ausnahme des Warteschlangentyps.

Senden einer Nachricht mithilfe einer FIFO-Warteschlange

Nachdem Sie Ihre Warteschlange erstellt haben, können Sie eine Nachricht an sie senden.

1. Wählen Sie im linken Navigationsbereich Warteschlangen aus. Wählen Sie in der Warteschlangenliste die Warteschlange aus, die Sie erstellt haben.
2. Wählen Sie unter Aktionen die Option Nachrichten senden und empfangen.

In der Konsole wird die Seite Nachrichten senden und empfangen angezeigt.

3. Geben Sie unter Nachricht den Nachrichtentext ein.
4. Geben Sie für eine First-In-First-Out (FIFO-) Warteschlange eine Nachrichtengruppen-ID ein. Weitere Informationen finden Sie unter [Logik für die FIFO-Warteschlangenzustellung in Amazon SQS](#).
5. (Optional) Für eine FIFO-Warteschlange können Sie eine Nachrichteneduplizierungs-ID eingeben. Wenn Sie die inhaltsbasierte Deduplizierung für die Warteschlange aktiviert haben, ist die Nachrichteneduplizierungs-ID nicht erforderlich. Weitere Informationen finden Sie unter [Logik für die FIFO-Warteschlangenzustellung in Amazon SQS](#).
6. FIFO-Warteschlangen unterstützen keine Timer für einzelne Nachrichten. Weitere Informationen finden Sie unter [Amazon-SQS-Nachrichten-Timer](#).
7. Klicken Sie auf Send Message (Nachricht senden).

Wenn Ihre Nachricht gesendet wurde, zeigt die Konsole eine Erfolgsmeldung an. Wählen Sie Details anzeigen, um Informationen zur gesendeten Nachricht anzuzeigen.

Allgemeine Aufgaben für die ersten Schritte mit Amazon SQS

Nachdem Sie eine Warteschlange erstellt und gelernt haben, wie Sie Nachrichten senden, empfangen und löschen, möchten Sie vielleicht Folgendes ausprobieren:

- Lösen Sie eine [Lambda-Funktion](#) aus, um eingehende Nachrichten automatisch zu verarbeiten und so ereignisgesteuerte Workflows zu ermöglichen, ohne dass kontinuierliche Abfragen erforderlich sind.
- [Konfigurieren Sie Warteschlangen, einschließlich](#) SSE und anderer Funktionen.

- [Senden Sie eine Nachricht mit Attributen.](#)
- [Senden Sie eine Nachricht von einer VPC.](#)
- Entdecken Sie die [Funktionalität](#) und [Architektur](#) von Amazon SQS.
- Entdecken Sie [Richtlinien und Vorbehalte](#), die Ihnen helfen, Amazon SQS optimal zu nutzen.
- Erkunden Sie die Amazon SQS SQS-Beispiele für ein AWS SDK, z. B. den [AWS SDK for Java 2.x Developer Guide](#).
- Erfahren Sie mehr über [Amazon SQS AWS CLI SQS-Befehle](#).
- Erfahren Sie mehr über [Amazon SQS SQS-API-Aktionen](#).
- Erfahren Sie, wie Sie programmgesteuert mit Amazon SQS interagieren können. [Weitere Informationen finden Sie unter Arbeiten mit dem APIs Development Center und erkunden Sie es AWS :](#)
 - [Java](#)
 - [JavaScript](#)
 - [PHP](#)
 - [Python](#)
 - [Ruby](#)
 - [Windows und .NET](#)
- Erfahren Sie, wie Sie [Kosten und Ressourcen](#) überwachen können.
- Erfahren Sie, wie Sie [Ihre Daten schützen können](#).
- Erfahren Sie mehr über den [Amazon SQS SQS-Workflow](#).

Verwalten einer Amazon-SQS-Warteschlange

Erfahren Sie, wie Sie Amazon SQS SQS-Warteschlangen mithilfe der Konsole verwalten, einschließlich der Bearbeitung von Warteschlangeneinstellungen, Empfangen und Löschen von Nachrichten, Bestätigen, dass Warteschlangen leer sind und Warteschlangen löschen oder löschen. Machen Sie sich mit bewährten Methoden für eine effiziente Nachrichtenverarbeitung vertraut, z. B. die Verwendung langer Abfragen, die Verwaltung von Zeitüberschreitungen für die Sichtbarkeit und die Überprüfung von Messwerten mithilfe von Überwachungs-Dashboards oder dem AWS CLI. Folgen Sie den praktischen Schritten, um Warteschlangen zu verwalten und Nachrichten effektiv zu bearbeiten und gleichzeitig Unterbrechungen zu minimieren.

Bearbeiten einer Amazon SQS SQS-Warteschlange mit der Konsole

Sie können die Amazon SQS SQS-Konsole verwenden, um die Parameter der Warteschlangenkonfiguration (mit Ausnahme des Warteschlangentyps) zu bearbeiten und Funktionen nach Bedarf zu ändern oder zu entfernen.

So bearbeiten Sie eine Amazon-SQS-Warteschlange (Konsole)

1. Öffnen Sie die Seite [Warteschlangen](#) der Amazon-SQS-Konsole.
2. Wählen Sie eine Warteschlange und dann Bearbeiten aus.
3. (Optional) Aktualisieren Sie unter Konfiguration die [Konfigurationsparameter](#) der Warteschlange.
4. (Optional) Um die [Zugriffsrichtlinie](#) zu aktualisieren, ändern Sie unter Zugriffsrichtlinie die JSON-Richtlinie.
5. (Optional) Erweitern Sie zur Aktualisierung einer [Redrive-Zulassungsrichtlinie](#) für eine Warteschlange für unzustellbare Nachrichten die Option Redrive-Zulassungsrichtlinie.
6. (Optional) Erweitern Sie zum Aktualisieren oder Entfernen der [Verschlüsselung](#) die Option Verschlüsselung.
7. (Optional) Um eine [Warteschlange für unzustellbare Nachrichten](#) hinzuzufügen, zu aktualisieren oder zu entfernen (wodurch Sie unzustellbare Nachrichten empfangen können), erweitern Sie die Option Warteschlange für unzustellbare Nachrichten.
8. (Optional) Um die [Tags](#) für die Warteschlange hinzuzufügen, zu aktualisieren oder zu entfernen, erweitern Sie Tags.

9. Wählen Sie Speichern.

- In der Konsole wird die Seite Details für die Warteschlange angezeigt.

Empfangen und Löschen einer Nachricht in Amazon SQS

Nachdem Sie Nachrichten an eine Amazon SQS SQS-Warteschlange gesendet haben, können Sie sie abrufen und löschen, um Ihren Anwendungsworkflow zu verarbeiten. Dieser Prozess gewährleistet eine sichere und zuverlässige Nachrichtenverarbeitung. Dieses Thema führt Sie durch das Abrufen und Löschen von Nachrichten mit der Amazon SQS SQS-Konsole und erklärt die wichtigsten Einstellungen zur Optimierung dieses Vorgangs. Im Folgenden werden die wichtigsten Konzepte für das Empfangen und Löschen von Nachrichten beschrieben:

1. Empfangen von Nachrichten

- Wenn Sie Nachrichten aus einer Amazon SQS SQS-Warteschlange abrufen, können Sie nicht auf bestimmte Nachrichten abzielen. Geben Sie stattdessen die maximale Anzahl von Nachrichten an, die in einer einzelnen Anfrage abgerufen werden sollen (bis zu 10).
- Aufgrund des verteilten Charakters von Amazon SQS kann beim Abrufen aus einer Warteschlange mit wenigen Nachrichten eine leere Antwort zurückgegeben werden. Um dem entgegenzuwirken:
 - Verwenden Sie eine lange Umfrage, bei der gewartet wird, bis eine Nachricht verfügbar ist oder das Zeitlimit für die Umfrage überschritten wird. Dieser Ansatz reduziert unnötige Kosten für Umfragen und verbessert die Effizienz.
 - Stellen Sie die Anfrage bei Bedarf erneut aus.

2. Sichtbarkeit und Löschung von Nachrichten

- Nachrichten werden nach dem Abrufen nicht automatisch gelöscht. Diese Funktion stellt sicher, dass Sie Nachrichten bei Anwendungsfehlern oder Netzwerkunterbrechungen erneut verarbeiten können.
- Nach der Verarbeitung müssen Sie explizit eine Löschanforderung senden, um die Nachricht dauerhaft zu entfernen. Diese Aktion bestätigt die erfolgreiche Bearbeitung.
- Nachrichten, die mit der Amazon SQS SQS-Konsole abgerufen wurden, bleiben für den erneuten Abruf sichtbar. Passen Sie die Einstellung für das Sichtbarkeits-Timeout für automatisierte Umgebungen so an, dass Nachrichten während der Bearbeitung vorübergehend vor anderen Benutzern verborgen werden.

3. Timeout für die Sichtbarkeit

- Diese Einstellung bestimmt, wie lange eine Nachricht nach dem Abrufen verborgen bleibt. Legen Sie ein angemessenes Timeout fest, um sicherzustellen, dass Nachrichten nur einmal verarbeitet werden, und um Doppelarbeit bei der verteilten Verarbeitung zu vermeiden.

Um eine Nachricht über die Konsole zu empfangen und zu löschen

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie auf der Seite Warteschlangen die Warteschlange aus, aus der Sie Nachrichten empfangen möchten, und wählen Sie dann Nachrichten senden und empfangen aus.
4. Wählen Sie auf der Seite Nachrichten senden und empfangen die Option Umfrage für Nachrichten aus.

Amazon SQS zeigt einen Fortschrittsbalken an, der die Dauer der Abfrage angibt. Abgerufene Nachrichten werden im Bereich Nachrichten angezeigt und zeigen:

- Nachrichten-ID
 - Datum des Sendens
 - Größe
 - Anzahl empfangen
5. Um Nachrichten zu löschen, wählen Sie die Nachrichten aus, die Sie entfernen möchten, und wählen Sie Löschen.

Bestätigen Sie den Löschvorgang im Dialogfeld Nachrichten löschen, indem Sie Löschen auswählen.

Weitere Informationen zu erweiterten Vorgängen, einschließlich API-basiertem Abrufen und Löschen von Nachrichten, finden Sie im [Amazon SQS](#) API-Referenzhandbuch.

Bestätigen, dass eine Amazon SQS SQS-Warteschlange leer ist

In den meisten Fällen können Sie [lange Abfragen](#) verwenden, um festzustellen, ob eine Warteschlange leer ist. In seltenen Fällen erhalten Sie möglicherweise leere Antworten, auch wenn eine Warteschlange noch Nachrichten enthält. Dies gilt insbesondere, wenn Sie bei der Erstellung der Warteschlange einen niedrigen Wert für die Wartezeit für den Empfang von Nachrichten angegeben haben. In diesem Abschnitt wird beschrieben, wie Sie prüfen können, ob eine Warteschlange leer ist.

So stellen Sie fest, ob eine Warteschlange leer ist (Konsole)

1. Hindern Sie alle Produzenten daran, Nachrichten zu senden.
2. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
3. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
4. Wählen Sie auf der Seite Warteschlangen eine Warteschlange aus.
5. Wählen Sie die Registerkarte Überwachung.
6. Wählen Sie oben rechts in den Monitoring-Dashboards den Abwärtspfeil neben dem Aktualisierungssymbol aus. Wählen Sie im Dropdown-Menü Automatische Aktualisierung aus. Belassen Sie das Aktualisierungsintervall bei 1 Minute.
7. Beachten Sie die folgenden Dashboards:
 - Ungefähre Anzahl verzögerter Nachrichten
 - Ungefähre Anzahl nicht sichtbarer Nachrichten
 - Ungefähre Anzahl sichtbarer Nachrichten

Wenn alle für mehrere Minuten 0-Werte anzeigen, ist die Warteschlange leer.

Um zu bestätigen, dass eine Warteschlange leer ist (AWS CLI, AWS API)

1. Hindern Sie alle Produzenten daran, Nachrichten zu senden.
2. Führen Sie wiederholt einen der folgenden Befehle aus:
 - AWS CLI: [get-queue-attributes](#)
 - AWS API: [GetQueueAttributes](#)
3. Beachten Sie die Metriken für die folgenden Attribute:
 - `ApproximateNumberOfMessagesDelayed`
 - `ApproximateNumberOfMessagesNotVisible`
 - `ApproximateNumberOfMessagesVisible`

Wenn alle für mehrere Minuten 0 anzeigen, ist die Warteschlange leer.

Wenn Sie sich auf CloudWatch Amazon-Metriken verlassen, stellen Sie sicher, dass Sie mehrere aufeinanderfolgende Nulldatenpunkte sehen, bevor Sie die Warteschlange als leer betrachten. Weitere Informationen zu CloudWatch Metriken finden Sie unter [Verfügbare CloudWatch Metriken für Amazon SQS](#).

Löschen einer Amazon SQS SQS-Warteschlange

Wenn Sie eine Amazon SQS SQS-Warteschlange nicht mehr verwenden und nicht planen, sie in naher future zu verwenden, löschen Sie die Warteschlange.

Tip

Vergewissern Sie sich vor dem Löschen einer Warteschlange, dass sie leer ist, siehe hierzu [Bestätigen, dass eine Amazon SQS SQS-Warteschlange leer ist](#).

Sie können eine Warteschlange löschen, auch wenn sie nicht leer ist. Wenn Sie Nachrichten in einer Warteschlange löschen möchten, aber nicht die Warteschlange selbst, können Sie die [Warteschlange leeren](#).

So löschen Sie eine Warteschlange (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie auf der Seite Warteschlange die zu löschende Warteschlange aus.
4. Wählen Sie Löschen aus.
5. Bestätigen Sie im Dialogfeld Warteschlange löschen durch Eingabe von **delete**, dass die Warteschlange gelöscht werden soll.
6. Wählen Sie Löschen aus.

Um eine Warteschlange (AWS CLI und eine API) zu löschen

Wählen Sie je nach Bedarf die passende Methode zum Löschen Ihrer Warteschlange aus:

- AWS CLI: [aws sqs delete-queue](#)
- AWS API: [DeleteQueue](#)

Löschen von Nachrichten aus einer Warteschlange mithilfe der Amazon SQS SQS-Konsole

Um eine Amazon SQS SQS-Warteschlange beizubehalten, aber alle zugehörigen Nachrichten zu entfernen, können Sie die Warteschlange löschen. Dadurch werden alle Nachrichten gelöscht, auch die Nachrichten, die derzeit unsichtbar sind (während der Übertragung). Der Löschvorgang kann bis zu 60 Sekunden dauern. Warten Sie also unabhängig von der Größe der Warteschlange die vollen 60 Sekunden.

Important

Wenn Sie eine Warteschlange bereinigen, können die daraus gelöschten Nachrichten nicht mehr abgerufen werden.

So bereinigen Sie eine Warteschlange (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie auf der Seite Warteschlange, die zu bereinigende Warteschlange aus.
4. Wählen Sie unter Aktionen die Option Bereinigen aus.
5. Bestätigen Sie im Dialogfeld Warteschlange bereinigen die Bereinigung, indem Sie **purge** eingeben und Bereinigen auswählen.
 - Alle Nachrichten werden aus der Warteschlange entfernt. Die Konsole zeigt ein Bestätigungsbanner an.

Amazon-SQS-Standard-Warteschlangen

Amazon SQS bietet Standardwarteschlangen als Standard-Warteschlangentyp und unterstützt eine nahezu unbegrenzte Anzahl von API-Aufrufen pro Sekunde für Aktionen wie [SendMessageReceiveMessage](#), und. [DeleteMessage](#) Standardwarteschlangen stellen die at-least-once Nachrichtenzustellung sicher, aber aufgrund der stark verteilten Architektur kann es vorkommen, dass mehr als eine Kopie einer Nachricht zugestellt wird, und Nachrichten können gelegentlich nicht in der richtigen Reihenfolge ankommen. Trotzdem bemühen sich Standardwarteschlangen nach besten Kräften, die Reihenfolge beizubehalten, in der Nachrichten gesendet werden.

Wenn Sie eine Nachricht mit `sendMessage` senden, speichert Amazon SQS die Nachricht redundant in mehreren Verfügbarkeitszonen (AZs), bevor sie bestätigt wird. Diese Redundanz stellt sicher, dass kein einzelner Computer-, Netzwerk- oder AZ-Ausfall dazu führen kann, dass auf die Nachrichten zugegriffen werden kann.

Sie können Warteschlangen mit der Amazon SQS SQS-Konsole erstellen und konfigurieren. Detaillierte Anweisungen finden Sie unter [Erstellen einer Standardwarteschlange mit der Amazon SQS SQS-Konsole](#). Java-spezifische Beispiele finden Sie unter. [Beispiele für Amazon SQS Java SDK](#)

Anwendungsfälle für Standardwarteschlangen

Standard-Nachrichtenwarteschlangen eignen sich für verschiedene Szenarien, sofern Ihre Anwendung Nachrichten verarbeiten kann, die möglicherweise mehrmals oder nicht in der richtigen Reihenfolge ankommen. Zu den Beispielen gehören:

- Entkopplung von Live-Benutzeranfragen von intensiver Hintergrundarbeit — Benutzer können Medien hochladen, während das System sie im Hintergrund ändert oder codiert.
- Zuweisung von Aufgaben an mehrere Worker Nodes — zum Beispiel die Bearbeitung einer großen Anzahl von Anfragen zur Kreditkartenvalidierung.
- Batch-Meldungen für die future Verarbeitung — Planung mehrerer Einträge, die zu einem späteren Zeitpunkt zu einer Datenbank hinzugefügt werden sollen.

Informationen zu Kontingenten im Zusammenhang mit Standardwarteschlangen finden Sie unter. [Amazon SQS SQS-Standard-Warteschlangenkongimente](#)

Informationen zu bewährten Methoden für die Arbeit mit Standard-Warteschlangen finden Sie unter [Bewährte Methoden für Amazon SQS](#).

Amazon SQS SQS-Lieferung at-least-once

Amazon SQS speichert aus Gründen der Redundanz und Hochverfügbarkeit Kopien der Nachrichten auf mehreren Servern. In seltenen Fällen kann es vorkommen, dass einer der Server, auf dem eine Nachrichtenkopie gespeichert ist, nicht verfügbar ist, wenn Sie eine Nachricht erhalten oder löschen.

In diesem Fall wird die Kopie der Nachricht auf dem Server, der nicht verfügbar ist, nicht gelöscht, und Sie erhalten diese Nachrichtenkopie möglicherweise erneut, wenn Sie Nachrichten erhalten. Konzipieren Sie Ihre Anwendungen idempotent (d. h., die mehrmalige Verarbeitung derselben Nachricht darf die Anwendung nicht nachteilig beeinflussen).

IDs von Amazon-SQS-Warteschlangen und -Nachrichten

In diesem Thema werden die Identifikatoren von Standard- und FIFO-Warteschlangen beschrieben. Diese IDs helfen Ihnen beim Suchen und Bearbeiten spezifischer Warteschlangen und Nachrichten.

Identifikatoren für Amazon SQS SQS-Standardwarteschlangen

Weitere Informationen zu den folgenden IDs finden Sie in der [Amazon-Simple-Queue-Service-API-Referenz](#).

Warteschlangenname und URL

Wenn Sie eine neue Warteschlange erstellen, müssen Sie einen für Ihr AWS -Konto und Ihre Region eindeutigen Warteschlangennamen angeben. Amazon SQS weist jeder erstellten Warteschlange eine Kennung, eine sogenannte Warteschlangen-URL, zu, die den Warteschlangennamen und andere Amazon-SQS-Komponenten enthält. Wenn Sie eine Aktion für die Warteschlange ausführen möchten, müssen Sie deren Warteschlangen-URL angeben.

Nachfolgend finden Sie die Warteschlangen-URL für eine Warteschlange namens MyQueue, die einem Benutzer mit der AWS-Kontonummer 123456789012 gehört:

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

Sie können die URL einer Warteschlange programmgesteuert abrufen, indem Sie Ihre Warteschlangen auflisten und die Zeichenfolge analysieren, die der Kontonummer folgt. Weitere Informationen finden Sie unter [ListQueues](#).

Nachrichten-ID

Jeder Nachricht wird vom System eine Nachrichten-ID zugewiesen, die Amazon SQS in der [SendMessage](#)-Antwort zurückgibt. Diese ID dient der Identifizierung von Nachrichten. Die maximale Länge einer Nachrichten-ID beträgt 100 Zeichen.

Empfangs-Mitteilung

Beim Empfang einer Nachricht aus einer Warteschlange erhalten Sie jedes Mal eine Empfangs-Mitteilung für diese Nachricht. Diese Mitteilung ist der Aktion des Nachrichtenempfangs zugeordnet und nicht der Nachricht selbst. Um eine Nachricht zu löschen oder ihre Sichtbarkeit zu ändern, müssen Sie die Empfangs-Mitteilung (und nicht die Nachrichten-ID) angeben. Daher müssen Sie eine Nachricht immer zunächst empfangen, bevor Sie sie löschen können. Es ist nicht möglich, eine Nachricht in die Warteschlange zu setzen und dann zurückzurufen. Die maximale Länge einer Empfangs-Mitteilung beträgt 1.024 Zeichen.

Important

Wenn Sie eine Nachricht mehrmals empfangen, erhalten Sie jedes Mal eine unterschiedliche Empfangs-Mitteilung. Wenn Sie die Nachricht löschen möchten, müssen Sie die zuletzt empfangene Empfangs-Mitteilung angeben, da die Nachricht sonst möglicherweise nicht gelöscht wird.

Im Folgenden finden Sie ein Beispiel für eine Empfangsnummer, die sich über drei Zeilen erstreckt.

```
MbZj6wDWli+JvwWJaBV+3dcjk2YW2vA3+STFF1jTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq20KpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon-SQS-FIFO-Warteschlangen

FIFO-Warteschlangen (First-In-First-Out) verfügen über alle Funktionen der [Standardwarteschlangen](#), sind jedoch darauf ausgelegt, die Kommunikation zwischen Anwendungen zu verbessern, wenn die Reihenfolge der Vorgänge und Ereignisse kritisch ist oder wenn Duplikate nicht toleriert werden können.

Die wichtigsten Merkmale von FIFO-Warteschlangen sind die FIFO-Zustellung (First-In-First-Out) und die Exactly-Once-Verarbeitung:

- Die Reihenfolge, in der Nachrichten gesendet und empfangen werden, wird strikt beibehalten. Eine Nachricht wird einmal zugestellt und ist erst dann verfügbar, wenn ein Verbraucher sie verarbeitet und löscht.
- Duplikate werden nicht in die Warteschlange aufgenommen.

Darüber hinaus unterstützen FIFO-Warteschlangen Nachrichtengruppen, die mehrere geordnete Nachrichtengruppen innerhalb einer einzigen Warteschlange zulassen. Es gibt kein Kontingent für die Anzahl der Nachrichtengruppen in einer FIFO-Warteschlange.

In folgenden Situationen können beispielsweise FIFO-Warteschlangen verwendet werden:

1. E-Commerce- oder Managementsysteme, bei denen die Reihenfolge entscheidend ist
2. Integration mit Systemen von Drittanbietern, bei denen Ereignisse der Reihe nach verarbeitet werden müssen
3. Verarbeitung von Benutzereingaben in der angegebenen Reihenfolge
4. Kommunikation und Vernetzung – Senden und Empfangen von Daten und Informationen in derselben Reihenfolge
5. Computersysteme – Sicherstellen, dass vom Benutzer eingegebene Befehle in der richtigen Reihenfolge ausgeführt werden
6. Bildungseinrichtungen – Verhindern, dass sich Studierende vor dem Erstellen eines Benutzerkontos in einen Kurs einschreiben
7. Online-Ticketsysteme – Wo Tickets nach dem Prinzip „Wer zuerst kommt, mahlt zuerst“ verteilt werden

 Note

Darüber hinaus stellen FIFO-Warteschlangen auch eine garantierte einmalige Verarbeitung bereit, sind jedoch auf eine begrenzte Anzahl an Transaktionen pro Sekunde (TPS) beschränkt. Sie können den Amazon-SQS-Hochdurchsatzmodus mit Ihrer FIFO-Warteschlange verwenden, um Ihr Transaktions-Limit zu erhöhen. Einzelheiten zur Verwendung des Hochdurchsatzmodus finden Sie unter [Hoher Durchsatz für FIFO-Warteschlangen in Amazon SQS](#). Weitere Informationen zu Durchsatzkontingenten finden Sie unter [the section called “Nachrichtenkontingente”](#).

Amazon-SQS-FIFO-Warteschlangen sind in allen Regionen verfügbar, in denen Amazon SQS verfügbar ist.

Weitere Informationen zur Verwendung von FIFO-Warteschlangen bei komplexen Anordnungen finden Sie unter [Lösung komplexer Sortieraufgaben mit Amazon-SQS-FIFO-Warteschlangen](#).

Weitere Informationen zum Erstellen und Konfigurieren von Warteschlangen mithilfe der Amazon-SQS-Konsole finden Sie unter [Erstellen einer Standardwarteschlange mit der Amazon SQS SQS-Konsole](#). Java-Beispiele finden Sie unter [Beispiele für Amazon SQS Java SDK](#).

Bewährte Methoden für die Arbeit mit FIFO-Warteschlangen finden Sie unter [Bewährte Methoden für Amazon SQS](#)

Schlüsselbegriffe der Amazon SQS FIFO-Warteschlange

Die folgenden wichtigen Begriffe vermitteln Ihnen ein besseres Verständnis der Funktionalität von FIFO-Warteschlangen. Weitere Informationen finden Sie in der [API-Referenz zu Amazon Simple Queue Service](#).

Clients

Der Amazon SQS Buffered Asynchronous Client unterstützt derzeit keine FIFO-Warteschlangen.

Nachrichteneduplizierungs-ID

Ein Token, das in Amazon SQS FIFO-Warteschlangen verwendet wird, um Nachrichten eindeutig zu identifizieren und Duplikate zu verhindern. Wenn mehrere Nachrichten mit derselben Deduplizierungs-ID innerhalb eines Deduplizierungsintervalls von 5 Minuten gesendet werden, werden sie als Duplikate behandelt und es wird nur eine Kopie zugestellt. Wenn Sie keine

Deduplizierungs-ID angeben und die inhaltsbasierte Deduplizierung aktiviert ist, generiert Amazon SQS eine Deduplizierungs-ID, indem es den Nachrichtentext hasht. Dieser Mechanismus gewährleistet, dass Nachrichten exakt einmal zugestellt werden, indem doppelte Nachrichten innerhalb des angegebenen Zeitrahmens entfernt werden.

 Note

Amazon SQS verfolgt weiterhin die Deduplizierungs-ID, auch nachdem die Nachricht empfangen und gelöscht wurde.

Nachrichtengruppen-ID

In FIFO-Warteschlangen (First-In-First-Out) `MessageGroupId` ist dies ein Attribut, das Nachrichten in verschiedene Gruppen unterteilt. Nachrichten innerhalb derselben Nachrichtengruppe werden immer einzeln und in strikter Reihenfolge verarbeitet, sodass sichergestellt wird, dass keine zwei Nachrichten aus derselben Gruppe gleichzeitig verarbeitet werden. In Standardwarteschlangen `MessageGroupId` ermöglicht die Verwendung [fairer Warteschlangen](#). Wenn eine strikte Reihenfolge erforderlich ist, verwenden Sie eine FIFO-Warteschlange.

Empfangsanforderungsversuch-ID

Die ID des Versuchs einer Empfangsanforderung ist ein eindeutiges Token, das zur Deduplizierung von [ReceiveMessage](#)-Aufrufen in Amazon SQS verwendet wird.

Sequenznummer

Die große, nicht fortlaufende Zahl, die Amazon SQS jeder Nachricht zuweist.

Services

Wenn Ihre Anwendung mehrere AWS Dienste oder eine Mischung aus AWS externen Diensten verwendet, ist es wichtig zu verstehen, welche Servicefunktion FIFO-Warteschlangen nicht unterstützt.

Einige AWS oder externe Dienste, die Benachrichtigungen an Amazon SQS senden, sind möglicherweise nicht mit FIFO-Warteschlangen kompatibel, obwohl Sie eine FIFO-Warteschlange als Ziel festlegen können.

Die folgenden Funktionen von AWS Diensten sind derzeit nicht mit FIFO-Warteschlangen kompatibel:

- [Amazon-S3-Ereignis-Benachrichtigungen](#)
- [Auto Scaling Lifecycle Hooks](#)
- [AWS IoT Regelaktionen](#)
- [AWS Lambda Warteschlangen für unzustellbare Nachrichten](#)

Weitere Informationen zur Kompatibilität anderer Services mit FIFO-Warteschlangen finden Sie in Ihrer Service-Dokumentation.

Logik für die FIFO-Warteschlangenzustellung in Amazon SQS

Die folgenden Konzepte erläutern, wie Amazon SQS FIFO-Warteschlangen das Senden und Empfangen von Nachrichten handhaben, insbesondere bei der Nachrichtenreihenfolge und Nachrichtengruppe. IDs

Senden von Nachrichten

Amazon SQS FIFO-Warteschlangen behalten die Nachrichtenreihenfolge mithilfe einer eindeutigen Deduplizierung IDs und Nachrichtengruppe bei. IDs In diesem Thema wird die Bedeutung von Nachrichtengruppen IDs für die Einhaltung einer strikten Reihenfolge innerhalb von Gruppen hervorgehoben und bewährte Verfahren zur Sicherstellung einer zuverlässigen, geordneten Nachrichtenzustellung an mehrere Hersteller hervorgehoben.

1. Beibehaltung der Reihenfolge

- Wenn mehrere Nachrichten nacheinander an eine FIFO-Warteschlange mit eindeutiger Nachrichtendeduplizierung gesendet werden IDs, speichert Amazon SQS sie und bestätigt ihre Übertragung. Diese Nachrichten werden dann genau in der Reihenfolge empfangen und verarbeitet, in der sie übertragen wurden.

2. Nachrichtengruppen-ID

- In FIFO-Warteschlangen werden Nachrichten anhand ihrer Nachrichtengruppen-ID sortiert. Wenn mehrere Produzenten oder Threads Nachrichten mit derselben Nachrichtengruppen-ID senden, stellt Amazon SQS sicher, dass sie in der Reihenfolge gespeichert und verarbeitet werden, in der sie ankommen.
- Bewährtes Verfahren: Um eine strikte Nachrichtenreihenfolge für mehrere Produzenten zu gewährleisten, weisen Sie allen Nachrichten von jedem Produzenten eine eindeutige Nachrichtengruppen-ID zu.

3. Bestellung pro Gruppe

- Die FIFO-Warteschlangenlogik gilt für jede Nachrichtengruppen-ID:
 - Jede Nachrichtengruppen-ID steht für eine bestimmte, geordnete Gruppe von Nachrichten.
 - Innerhalb einer Nachrichtengruppen-ID werden alle Nachrichten in strenger Reihenfolge gesendet und empfangen.
 - Nachrichten mit unterschiedlichen Nachrichtengruppen IDs können in der falschen Reihenfolge eingehen oder verarbeitet werden.
- Anforderung — Sie müssen jeder Nachricht eine Nachrichtengruppen-ID zuordnen. Wenn eine Nachricht ohne Gruppen-ID gesendet wird, schlägt die Aktion fehl.
- Einzelgruppenszenario: Wenn Sie möchten, dass alle Nachrichten in strenger Reihenfolge verarbeitet werden, verwenden Sie für jede Nachricht dieselbe Nachrichtengruppen-ID.

Empfangen von Nachrichten

Amazon SQS FIFO-Warteschlangen kümmern sich um den Nachrichtenabruf, einschließlich Stapelverarbeitung, FIFO-Auftragsgarantien und Einschränkungen bei der Anforderung bestimmter Nachrichtengruppen. IDs In diesem Thema wird erklärt, wie Amazon SQS Nachrichten innerhalb und zwischen Nachrichtengruppen abrufen und IDs dabei strenge Reihenfolge- und Sichtbarkeitsregeln einhält.

1. Batch-Abruf

- Beim Empfang von Nachrichten aus einer FIFO-Warteschlange mit mehreren Nachrichtengruppen IDs geht Amazon SQS wie folgt vor:
 - Versucht, in einem einzigen Anruf so viele Nachrichten wie möglich mit derselben Nachrichtengruppen-ID zurückzugeben.
 - Ermöglicht es anderen Verbrauchern, Nachrichten aus verschiedenen Nachrichtengruppen IDs gleichzeitig zu verarbeiten.
- Wichtige Klarstellung
 - Möglicherweise erhalten Sie mehrere Nachrichten von derselben Nachrichtengruppen-ID in einem Batch (bis zu 10 Nachrichten in einem einzigen Anruf mithilfe des `MaxNumberOfMessages` Parameters).
 - Sie können jedoch in nachfolgenden Anfragen keine weiteren Nachrichten von derselben Nachrichtengruppen-ID empfangen, bis:
 - Die aktuell empfangenen Nachrichten werden gelöscht, oder
 - Sie werden wieder sichtbar (z. B. nach Ablauf des Sichtbarkeits-Timeouts).

2. FIFO-Bestellgarantie

- In einem Stapel abgerufene Nachrichten behalten ihre FIFO-Reihenfolge innerhalb der Gruppe bei.
- Wenn weniger als 10 Nachrichten für dieselbe Nachrichtengruppen-ID verfügbar sind, kann Amazon SQS Nachrichten aus anderen Nachrichtengruppen IDs in denselben Stapel aufnehmen, aber jede Gruppe behält die FIFO-Reihenfolge bei.

3. Einschränkungen für Verbraucher

- Sie können nicht explizit den Empfang von Nachrichten von einer bestimmten Nachrichtengruppen-ID anfordern.

Mehrere Versuche

Produzenten und Verbraucher können fehlgeschlagene Aktionen in Amazon SQS FIFO-Warteschlangen sicher wiederholen, ohne die Reihenfolge der Nachrichten zu unterbrechen oder Duplikate einzuführen. In diesem Thema wird erläutert, wie Deduplizierung IDs und Timeouts für Sichtbarkeit die Nachrichtenintegrität bei Wiederholungsversuchen sicherstellen.

1. Der Hersteller versucht es erneut

- Wenn eine [SendMessage](#)Aktion fehlschlägt, kann der Producer versuchen, die Nachricht mehrmals mit derselben Nachrichteneduplizierungs-ID zu senden.
- Solange der Producer vor Ablauf des Deduplizierungsintervalls mindestens eine Bestätigung erhält, versucht er es erneut:
 - Führen Sie keine doppelten Nachrichten ein.
 - Unterbrechen Sie nicht die Reihenfolge der Nachrichten.

2. Der Verbraucher versucht es erneut

- Schlägt eine [ReceiveMessage](#)Aktion fehl, kann der Verbraucher es mit derselben ID für den Versuch einer Empfangsanforderung so oft wie nötig wiederholen.
- Solange der Verbraucher mindestens eine Bestätigung erhält, bevor das Sichtbarkeits-Timeout abläuft, versucht er es erneut:
 - Unterbrechen Sie nicht die Reihenfolge der Nachrichten.

Zusätzliche Hinweise zum FIFO-Verhalten

Erfahren Sie mehr über den Umgang mit Sichtbarkeits-Timeouts, die Aktivierung der Parallelverarbeitung mit mehreren Nachrichtengruppen IDs und die Sicherstellung einer strikten sequentiellen Verarbeitung in Einzelgruppenszenarien.

1. Umgang mit Sichtbarkeits-Timeouts

- Wenn eine Nachricht abgerufen, aber nicht gelöscht wird, bleibt sie unsichtbar, bis das Sichtbarkeits-Timeout abgelaufen ist.
- Es werden keine weiteren Nachrichten mit derselben Nachrichtengruppen-ID zurückgegeben, bis die erste Nachricht gelöscht oder wieder sichtbar wird.

2. Parallelität und Parallelverarbeitung

- FIFO-Warteschlangen ermöglichen die parallel Verarbeitung von Nachrichten in verschiedenen Nachrichtengruppen. IDs
- Um die Parallelität zu maximieren, sollten Sie Ihr System mit mehreren Nachrichtengruppen IDs für unabhängige Workflows entwerfen.

3. Szenarien mit einer Gruppe

- Verwenden Sie für die strikte sequentielle Verarbeitung aller Nachrichten in einer FIFO-Warteschlange eine einzige Nachrichtengruppen-ID für alle Nachrichten in der Warteschlange.

Beispiele zum besseren Verständnis

Im Folgenden finden Sie praktische Szenarien, die das Verhalten von FIFO-Warteschlangen in Amazon SQS veranschaulichen.

1. Szenario 1: Einzelne Gruppen-ID

- Ein Producer sendet fünf Nachrichten mit derselben Nachrichtengruppen-ID Gruppe A.
- Ein Verbraucher erhält diese Nachrichten in FIFO-Reihenfolge. Solange der Verbraucher diese Nachrichten nicht löscht oder das Sichtbarkeits-Timeout abgelaufen ist, werden keine weiteren Nachrichten von Gruppe A empfangen.

2. Szenario 2: Mehrere Gruppen IDs

- Ein Produzent sendet fünf Nachrichten an Gruppe A und 5 an Gruppe B.
- Verbraucher 1 verarbeitet Nachrichten von Gruppe A, während Verbraucher 2 Nachrichten von Gruppe B verarbeitet. Dies ermöglicht eine parallel Verarbeitung mit strikter Reihenfolge innerhalb jeder Gruppe.

3. Szenario 3: Batch-Abruf

- Ein Produzent sendet sieben Nachrichten an Gruppe A und drei an Gruppe B.
- Ein einzelner Verbraucher ruft bis zu 10 Nachrichten ab. Wenn die Warteschlange dies zulässt, kann sie Folgendes zurückgeben:
 - Sieben Nachrichten aus Gruppe A und drei aus Gruppe B (oder weniger, wenn weniger Nachrichten von einer einzelnen Gruppe verfügbar sind).

Exactly-Once-Verarbeitung in Amazon SQS

Anders als bei Standard-Warteschlangen werden in FIFO-Warteschlangen keine Duplikate aufgenommen. Mit FIFO-Warteschlangen können Sie das Senden von Duplikaten an eine Warteschlange verhindern. Wenn Sie die Aktion `SendMessage` innerhalb des 5-minütigen Deduplizierungsintervalls erneut ausführen, stellt Amazon SQS keine Duplikate in die Warteschlange.

Um die Deduplizierung zu konfigurieren, müssen Sie eine der folgenden Aktionen ausführen:

- Aktivieren Sie inhaltsbasierte Deduplizierung. So wird Amazon SQS angewiesen, einen SHA-256-Hash zum Generieren der Nachrichteneduplizierungs-ID zu verwenden. Dabei wird der Inhalt der Nachricht, nicht jedoch die Attribute der Nachricht verwendet. Weitere Informationen finden Sie in der Dokumentation zu den Aktionen [CreateQueue](#), [GetQueueAttributes](#) und [SetQueueAttributes](#) in der Amazon-Simple-Queue-Service-API-Referenz.
- Stellen Sie die Nachrichteneduplizierungs-ID für die Nachricht explizit bereit (oder rufen Sie die Sequenznummer ab). Weitere Informationen finden Sie in der Dokumentation zu den Aktionen [SendMessage](#), [SendMessageBatch](#) und [ReceiveMessage](#) in der Amazon-Simple-Queue-Service-API-Referenz.

Von einer Standardwarteschlange zu einer FIFO-Warteschlange in Amazon SQS wechseln

Wenn Ihre bestehende Anwendung Standardwarteschlangen verwendet und Sie die Vorteile der FIFO-Warteschlangen für die Reihenfolge oder die einmalige Verarbeitung nutzen möchten, müssen Sie sowohl die Warteschlange als auch Ihre Anwendung korrekt konfigurieren.

Wesentliche Überlegungen

- Erstellen einer FIFO-Warteschlange: Sie können eine bestehende Standardwarteschlange nicht in eine FIFO-Warteschlange umwandeln. Sie müssen entweder eine neue FIFO-Warteschlange für Ihre Anwendung erstellen oder die vorhandene Standardwarteschlange löschen und sie als FIFO-Warteschlange neu erstellen.
- Verzögerungsparameter: FIFO-Warteschlangen unterstützen keine Verzögerungen pro Nachricht, sondern nur Verzögerungen pro Warteschlange. Wenn Ihre Anwendung den `DelaySeconds` Parameter für jede Nachricht festlegt, müssen Sie ihn so ändern, dass er stattdessen für die gesamte Warteschlange festgelegt `DelaySeconds` wird.
- Nachrichtengruppen-ID: Geben Sie für jede gesendete [Nachricht eine Nachrichtengruppen-ID](#) an. Diese ID ermöglicht die parallel Verarbeitung von Nachrichten unter Beibehaltung ihrer jeweiligen Reihenfolge. Verwenden Sie eine detaillierte Geschäftsdimension für die Nachrichtengruppen-ID, um FIFO-Warteschlangen besser skalieren zu können. Je mehr Nachrichtengruppen IDs Sie Nachrichten verteilen, desto mehr Nachrichten stehen zur Verfügung.
- Modus mit hohem Durchsatz: Verwenden Sie den empfohlenen [Modus mit hohem Durchsatz](#) für FIFO-Warteschlangen, um einen höheren Durchsatz zu erzielen. Weitere Informationen zu Nachrichtenkontingenten finden Sie unter [Amazon SQS SQS-Nachrichtenkontingente](#)

Checkliste für den Übergang zu FIFO-Warteschlangen

Überprüfen Sie vor dem Senden von Nachrichten an eine FIFO-Warteschlange Folgendes:

1. Konfigurieren Sie die Verzögerungseinstellungen

- Ändern Sie Ihre Anwendung, um Verzögerungen pro Nachricht zu entfernen.
- Stellen Sie den `DelaySeconds` Parameter für die gesamte Warteschlange ein.

2. Nachrichtengruppe festlegen IDs

- Organisieren Sie Nachrichten in Nachrichtengruppen, indem Sie eine Nachrichtengruppen-ID angeben, die auf einer Geschäftsdimension basiert.
- Verwenden Sie detailliertere Geschäftsdimensionen, um die Skalierbarkeit zu verbessern.

3. Behandeln Sie die Nachrichtendeduplizierung

- Wenn Ihre Anwendung keine Nachrichten mit identischen Nachrichtentexten senden kann, geben Sie für jede Nachricht eine eindeutige Deduplizierungs-ID an.
- Wenn Ihre Anwendung Nachrichten mit eindeutigen Nachrichtentexten sendet, aktivieren Sie die inhaltsbasierte Deduplizierung.

4. Konfigurieren Sie den Verbraucher

- Im Allgemeinen sind für den Verbraucher keine Codeänderungen erforderlich.
- Wenn die Verarbeitung von Nachrichten viel Zeit in Anspruch nimmt und das Sichtbarkeits-Timeout hoch eingestellt ist, sollten Sie erwägen, jeder `ReceiveMessage` Aktion eine ID für den Versuch einer Empfangsanforderung hinzuzufügen. Dies hilft bei erneuten Empfangsversuchen bei Netzwerkausfällen und verhindert, dass Warteschlangen aufgrund fehlgeschlagener Empfangsversuche unterbrochen werden.

Wenn Sie diese Schritte befolgen, können Sie sicherstellen, dass Ihre Anwendung mit FIFO-Warteschlangen ordnungsgemäß funktioniert und die Vorteile der Funktionen zur Reihenfolge und einmaligen Verarbeitung voll ausschöpfen. Weitere Informationen finden Sie in der [Amazon Simple Queue Service API-Referenz](#).

Amazon SQS FIFO-Warteschlange und Lambda-Parallelitätsverhalten

Durch die Verwendung einer FIFO-Warteschlange (First-In-First-Out) mit Lambda können Sie eine geordnete Verarbeitung von Nachrichten innerhalb jeder Nachrichtengruppe sicherstellen. Die Lambda-Funktion führt nicht mehrere Instanzen für dieselbe Nachrichtengruppe gleichzeitig aus, wodurch die Reihenfolge beibehalten wird. Es kann jedoch skaliert werden, um mehrere Nachrichtengruppen parallel zu verarbeiten, wodurch eine effiziente Verarbeitung der Arbeitslast Ihrer Warteschlange gewährleistet wird. Die folgenden Punkte beschreiben das Verhalten von Lambda-Funktionen bei der Verarbeitung von Nachrichten aus einer Amazon SQS SQS-FIFO-Warteschlange in Bezug auf die Nachrichtengruppe: IDs

- Einzelne Instanz pro Nachrichtengruppe: Zu jedem Zeitpunkt verarbeitet nur eine Lambda-Instanz Nachrichten von einer bestimmten Nachrichtengruppen-ID. Dadurch wird sichergestellt, dass Nachrichten innerhalb derselben Gruppe der Reihe nach verarbeitet werden, wobei die Integrität der FIFO-Sequenz gewahrt bleibt.
- Gleichzeitige Verarbeitung verschiedener Gruppen: Lambda kann Nachrichten aus verschiedenen Nachrichtengruppen IDs mithilfe mehrerer Instanzen gleichzeitig verarbeiten. Das bedeutet, dass eine Instanz der Lambda-Funktion Nachrichten von einer Nachrichtengruppen-ID verarbeitet, während andere Instanzen gleichzeitig Nachrichten aus einer anderen Nachrichtengruppe verarbeiten können IDs, wodurch die Parallelitätsfunktionen von Lambda genutzt werden, um mehrere Gruppen parallel zu verarbeiten.

Gruppierung von Nachrichten in der FIFO-Warteschlange

FIFO-Warteschlangen stellen sicher, dass Nachrichten in der exakten Reihenfolge verarbeitet werden, in der sie gesendet werden. Sie verwenden eine Nachrichtengruppen-ID, um Nachrichten zu gruppieren, die nacheinander verarbeitet werden sollen.

Nachrichten innerhalb derselben Nachrichtengruppe werden der Reihe nach verarbeitet, und es wird jeweils nur eine Nachricht aus jeder Gruppe verarbeitet, um diese Reihenfolge beizubehalten.

Lambda-Parallelität mit FIFO-Warteschlangen

Nachdem Sie Ihre Warteschlange erstellt haben, können Sie eine Nachricht an sie senden.

Wenn Sie eine Lambda-Funktion einrichten, um Nachrichten aus einer Amazon SQS SQS-FIFO-Warteschlange zu verarbeiten, respektiert Lambda die Bestellgarantien der FIFO-Warteschlange. Die folgenden Punkte beschreiben das Verhalten von Lambda-Funktionen in Bezug auf Parallelität und Skalierung bei der Verarbeitung von Nachrichten aus einer Amazon SQS SQS-FIFO-Warteschlange bei Verwendung einer Nachrichtengruppe. IDs

- **Parallelität innerhalb von Nachrichtengruppen:** Nur eine Lambda-Instanz verarbeitet Nachrichten für eine bestimmte Nachrichtengruppen-ID gleichzeitig. Dadurch wird sichergestellt, dass Nachrichten innerhalb einer Gruppe sequentiell behandelt werden.
- **Skalierung und mehrere Nachrichtengruppen:** Lambda kann zwar skaliert werden, um Nachrichten gleichzeitig zu verarbeiten, diese Skalierung erfolgt jedoch über verschiedene Nachrichtengruppen hinweg. Wenn Sie mehrere Nachrichtengruppen haben, kann Lambda mehrere Gruppen parallel verarbeiten, wobei jede Gruppe von einer separaten Lambda-Instanz verwaltet wird.

Weitere Informationen finden Sie unter [Skalierung und Parallelität in Lambda](#) im AWS Lambda Operator Guide.

Beispiel für einen Anwendungsfall

Angenommen, Ihre FIFO-Warteschlange empfängt Nachrichten mit derselben Nachrichtengruppen-ID und Ihre Lambda-Funktion hat ein hohes Parallelitätslimit (bis zu 1000).

Wenn eine Nachricht von der Gruppen-ID 'A' verarbeitet wird und eine weitere Nachricht von der Gruppen-ID 'A' eintrifft, löst die zweite Nachricht keine neue Lambda-Instanz aus, bis die erste Nachricht vollständig verarbeitet ist.

Wenn jedoch Nachrichten aus den Gruppen IDs 'A' und 'B' eintreffen, können beide Nachrichten gleichzeitig von separaten Lambda-Instanzen verarbeitet werden.

Hoher Durchsatz für FIFO-Warteschlangen in Amazon SQS

FIFO-Warteschlangen mit hohem Durchsatz in Amazon SQS verwalten effizient einen hohen Nachrichtendurchsatz bei gleichzeitiger Einhaltung einer strikten Nachrichtenreihenfolge und gewährleisten so Zuverlässigkeit und Skalierbarkeit für Anwendungen, die zahlreiche Nachrichten verarbeiten. Diese Lösung ist ideal für Szenarien, die sowohl einen hohen Durchsatz als auch eine geordnete Nachrichtenzustellung erfordern.

FIFO-Warteschlangen mit hohem Durchsatz in Amazon SQS sind in Szenarien nicht erforderlich, in denen eine strikte Reihenfolge der Nachrichten nicht entscheidend ist und in denen das Volumen eingehender Nachrichten relativ gering oder sporadisch ist. Wenn Sie beispielsweise über eine kleine Anwendung verfügen, die seltene oder nicht sequenzielle Nachrichten verarbeitet, sind die zusätzliche Komplexität und die Kosten, die mit FIFO-Warteschlangen mit hohem Durchsatz verbunden sind, möglicherweise nicht gerechtfertigt. Wenn Ihre Anwendung die verbesserten Durchsatzfunktionen von FIFO-Warteschlangen mit hohem Durchsatz nicht benötigt, ist die Entscheidung für eine standardmäßige Amazon SQS SQS-Warteschlange möglicherweise kostengünstiger und einfacher zu verwalten.

Um die Anforderungskapazität in FIFO-Warteschlangen mit hohem Durchsatz zu erhöhen, wird empfohlen, die Anzahl der Nachrichtengruppen zu erhöhen. Weitere Informationen zu Nachrichtenkontingenten mit hohem Durchsatz finden Sie unter [Amazon-SQS-Service-Quotas](#) im Allgemeine Amazon Web Services-Referenz.

Informationen zu Quotas pro Warteschlange und Strategien zur Datenverteilung finden Sie unter und [Amazon SQS SQS-Nachrichtenkontingente Partitionen und Datenverteilung für hohen Durchsatz für SQS-FIFO-Warteschlangen](#)

Anwendungsfälle für hohen Durchsatz für Amazon SQS FIFO-Warteschlangen

Die folgenden Anwendungsfälle beleuchten die vielfältigen Einsatzmöglichkeiten von FIFO-Warteschlangen mit hohem Durchsatz und verdeutlichen deren Wirksamkeit in verschiedenen Branchen und Szenarien:

1. Datenverarbeitung in Echtzeit: Anwendungen, die mit Echtzeit-Datenströmen zu tun haben, wie z. B. die Ereignisverarbeitung oder die Erfassung von Telemetriedaten, können von

- FIFO-Warteschlangen mit hohem Durchsatz profitieren, um den kontinuierlichen Zustrom von Nachrichten zu bewältigen und gleichzeitig deren Reihenfolge für eine genaue Analyse beizubehalten.
2. E-Commerce-Auftragsabwicklung: Auf E-Commerce-Plattformen, auf denen es entscheidend ist, die Reihenfolge der Kundentransaktionen aufrechtzuerhalten, stellen FIFO-Warteschlangen mit hohem Durchsatz sicher, dass Bestellungen auch während der Haupteinkaufssaison sequentiell und ohne Verzögerungen bearbeitet werden.
 3. Finanzdienstleistungen: Finanzinstitute, die hochfrequente Handels- oder Transaktionsdaten verarbeiten, verlassen sich auf FIFO-Warteschlangen mit hohem Durchsatz, um Marktdaten und Transaktionen mit minimaler Latenz zu verarbeiten und gleichzeitig die strengen regulatorischen Anforderungen für die Bestellung von Nachrichten einzuhalten.
 4. Medienstreaming: Streaming-Plattformen und Medienvertriebsdienste nutzen FIFO-Warteschlangen mit hohem Durchsatz, um die Bereitstellung von Mediendateien und Streaming-Inhalten zu verwalten und so ein reibungsloses Wiedergabeerlebnis für Benutzer zu gewährleisten und gleichzeitig die richtige Reihenfolge der Inhaltzustellung beizubehalten.

Partitionen und Datenverteilung für hohen Durchsatz für SQS-FIFO-Warteschlangen

Amazon SQS speichert FIFO-Warteschlangendaten in Partitionen. Eine Partition ist eine Speicherzuweisung für eine Warteschlange, die automatisch über mehrere Availability Zones innerhalb einer Region repliziert wird. AWS Sie verwalten keine Partitionen. Stattdessen kümmert sich Amazon SQS um die Partitionsverwaltung.

Für FIFO-Warteschlangen ändert Amazon SQS die Anzahl der Partitionen in einer Warteschlange in den folgenden Situationen:

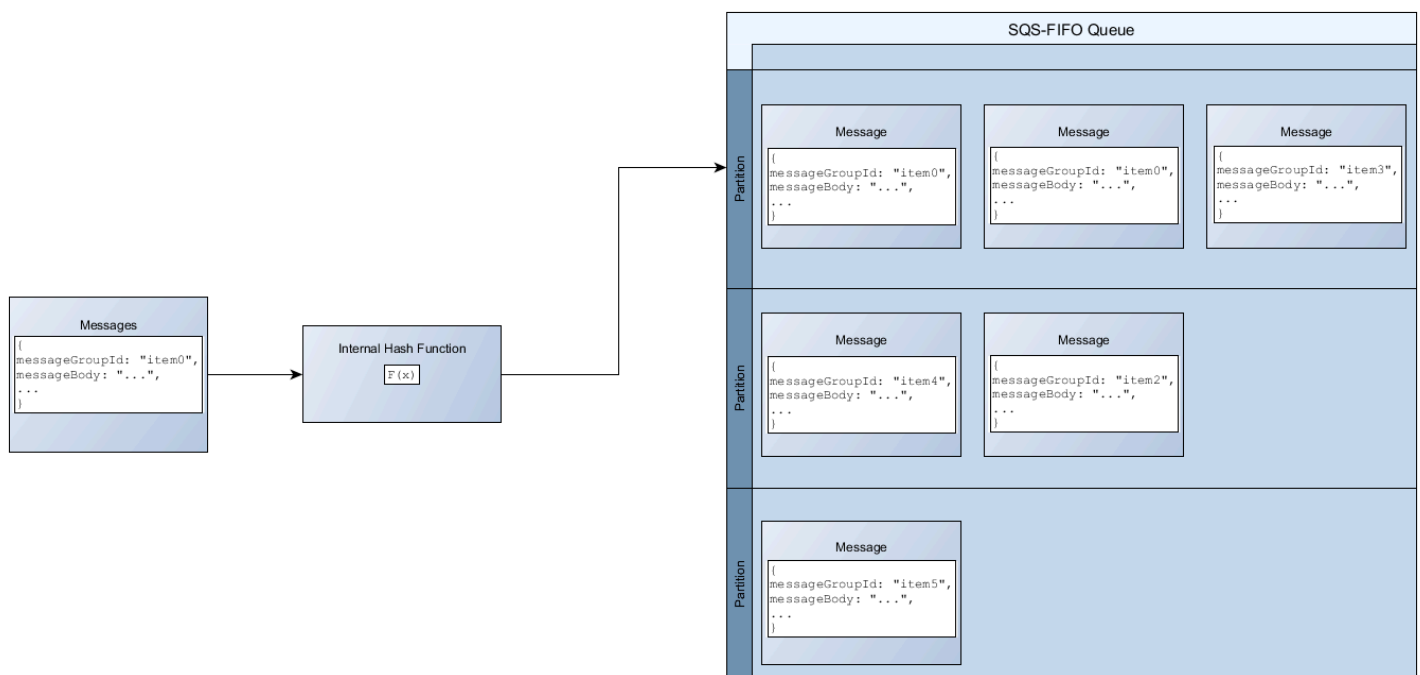
- Wenn sich die aktuelle Anforderungsrate dem, was die vorhandenen Partitionen unterstützen können, nähert oder dies übersteigt, werden zusätzliche Partitionen zugewiesen, bis die Warteschlange das regionale Kontingent erreicht hat. Informationen zu Kontingenten finden Sie unter [Amazon SQS SQS-Nachrichtenkontingente](#).
- Wenn die aktuellen Partitionen wenig ausgelastet sind, kann die Anzahl der Partitionen reduziert werden.

Die Partitionsverwaltung wird automatisch im Hintergrund ausgeführt und ist für Ihre Anwendungen transparent. Ihre Warteschlange und Ihre Nachrichten sind jederzeit verfügbar.

Verteilen von Daten nach Nachrichtengruppen IDs

Um eine Nachricht zu einer FIFO-Warteschlange hinzuzufügen, verwendet Amazon SQS den Wert der Nachrichtengruppen-ID jeder Nachricht als Eingabe für eine interne Hash-Funktion. Der Ausgabewert der Hash-Funktion bestimmt die Partition, in der die Nachricht gespeichert wird.

Das folgende Diagramm zeigt eine Warteschlange, die sich über mehrere Partitionen erstreckt. Die Nachrichtengruppen-ID der Warteschlange basiert auf der Elementnummer. Amazon SQS verwendet eine Hash-Funktion, um zu ermitteln, wo ein neues Element gespeichert werden soll, und zwar in diesem Fall basierend auf dem Hash-Wert der Zeichenfolge `item0`. Beachten Sie, dass die Elemente in derselben Reihenfolge gespeichert werden, in der sie der Warteschlange hinzugefügt werden. Die Position jedes einzelnen Elements wird durch den Hash-Wert seiner Nachrichtengruppen-ID bestimmt.



Note

Amazon SQS ist für die gleichmäßige Verteilung von Elementen auf die Partitionen einer FIFO-Warteschlange optimiert, unabhängig von der Anzahl der Partitionen. AWS empfiehlt die Verwendung einer Nachrichtengruppe IDs, die eine große Anzahl unterschiedlicher Werte haben kann.

Optimierung der Partitionsnutzung

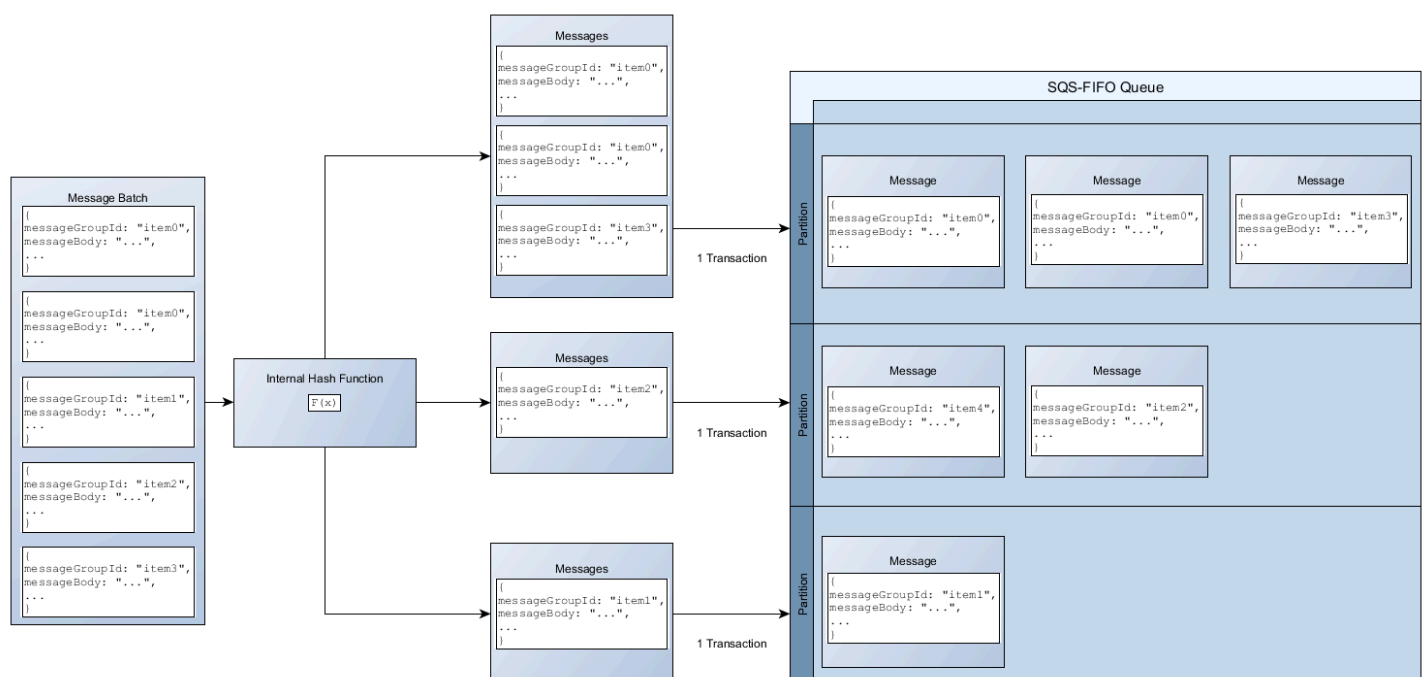
Jede Partition unterstützt bis zu 3 000 Nachrichten pro Sekunde bei Stapelverarbeitung oder bis zu 300 Nachrichten pro Sekunde bei Sende-, Empfangs- und Löschvorgängen in unterstützten Regionen. Weitere Informationen zu Nachrichtenkontingenten mit hohem Durchsatz finden Sie unter [Amazon-SQS-Service-Quotas](#) im Allgemeine Amazon Web Services-Referenz.

Bei der Verwendung von Batch APIs wird jede Nachricht auf der Grundlage des unter beschriebenen Prozesses weitergeleitet. [Verteilen von Daten nach Nachrichtengruppen IDs](#) Nachrichten, die an dieselbe Partition weitergeleitet werden, werden in einer einzigen Transaktion gruppiert und verarbeitet.

Um die Partitionsnutzung für die `SendMessageBatch` API zu optimieren, AWS empfiehlt es sich, Nachrichten nach Möglichkeit mit derselben Nachrichtengruppe zu stapeln IDs .

Um die Partitionsnutzung für `DeleteMessageBatch` und zu optimieren `ChangeMessageVisibilityBatch` APIs, AWS empfiehlt, `ReceiveMessage` Anfragen zu verwenden, bei denen der `MaxNumberOfMessages` Parameter auf 10 gesetzt ist, und die von einer einzelnen Anfrage zurückgegebenen Empfangs-Handles zu stapeln. `ReceiveMessage`

Im folgenden Beispiel wird ein Stapel von Nachrichten mit unterschiedlichen Nachrichtengruppen gesendet. IDs Der Stapel ist in drei Gruppen aufgeteilt, von denen jede auf das Kontingent für die Partition angerechnet wird.



 Note

Amazon SQS garantiert nur, dass Nachrichten mit der internen Hash-Funktion derselben Nachrichtengruppen-ID innerhalb einer Stapelanfrage gruppiert werden. Abhängig von der Ausgabe der internen Hash-Funktion und der Anzahl der Partitionen IDs können Nachrichten mit unterschiedlichen Nachrichtengruppen gruppiert werden. Da sich die Hash-Funktion oder die Anzahl der Partitionen jederzeit ändern kann, können Nachrichten, die an einem Punkt gruppiert sind, später möglicherweise nicht gruppiert werden.

Aktivierung eines hohen Durchsatzes für FIFO-Warteschlangen in Amazon SQS

Sie können den hohen Durchsatz für jede neue oder vorhandene FIFO-Warteschlange aktivieren. Das Feature umfasst drei neue Optionen beim Erstellen und Bearbeiten von FIFO-Warteschlangen:

- FIFO mit hohem Durchsatz aktivieren – Macht hohen Durchsatz für Nachrichten in der Warteschlange verfügbar.
- Deduplizierungsbereich – Gibt an, ob die Deduplizierung auf Warteschlangen- oder Nachrichtengruppenebene erfolgt.
- FIFO-Durchsatz-Limit – Gibt an, ob das Durchsatzkontingent für Nachrichten in der FIFO-Warteschlange auf Warteschlangen- oder Nachrichtengruppenebene festgelegt wird.

So aktivieren Sie den hohen Durchsatz für eine FIFO-Warteschlange (Konsole)

1. Beginnen Sie mit der [Erstellung](#) oder [Bearbeitung](#) einer FIFO-Warteschlange.
2. Wenn Sie Optionen für die Warteschlange angeben, wählen Sie FIFO mit hohem Durchsatz aktivieren.

Wenn Sie hohen Durchsatz für FIFO-Warteschlangen aktivieren, werden die entsprechenden Optionen wie folgt festgelegt:

- Der Deduplizierungsbereich ist auf Nachrichtengruppe festgelegt. Dies ist die erforderliche Einstellung für die Verwendung eines hohen Durchsatzes für FIFO-Warteschlangen.
- Das FIFO-Durchsatz-Limit ist auf Pro Nachrichtengruppen-ID festgelegt. Dies ist die erforderliche Einstellung für die Verwendung eines hohen Durchsatzes für FIFO-Warteschlangen.

Wenn Sie eine der Einstellungen ändern, die für die Verwendung eines hohen Durchsatzes für FIFO-Warteschlangen erforderlich sind, ist der normale Durchsatz für die Warteschlange wirksam und die Deduplizierung erfolgt wie angegeben.

3. Geben Sie weiter alle Optionen für die Warteschlange an. Wenn Sie damit fertig sind, wählen Sie Warteschlange erstellen oder Speichern.

Nachdem Sie die FIFO-Warteschlange erstellt oder bearbeitet haben, können Sie [Nachrichten an diese senden](#) sowie [Nachrichten empfangen und löschen](#), und dies alles mit einem höheren TPS-Wert. Informationen zu Kontingenten mit hohem Durchsatz finden Sie unter „Nachrichtendurchsatz“ in [Amazon SQS SQS-Nachrichtenkontingente](#).

FIFO-Warteschlangen- und Nachrichtenkennungen in Amazon SQS

In diesem Abschnitt werden die IDs von FIFO-Warteschlangen beschrieben. Diese IDs helfen Ihnen beim Suchen und Bearbeiten spezifischer Warteschlangen und Nachrichten.

Identifikatoren für FIFO-Warteschlangen in Amazon SQS

Weitere Informationen zu den folgenden IDs finden Sie in der [Amazon-Simple-Queue-Service-API-Referenz](#).

Warteschlangenname und URL

Wenn Sie eine neue Warteschlange erstellen, müssen Sie einen für Ihr AWS -Konto und Ihre Region eindeutigen Warteschlangennamen angeben. Amazon SQS weist jeder erstellten Warteschlange eine Kennung, eine sogenannte Warteschlangen-URL, zu, die den Warteschlangennamen und andere Amazon-SQS-Komponenten enthält. Wenn Sie eine Aktion für die Warteschlange ausführen möchten, müssen Sie deren Warteschlangen-URL angeben.

Der Name einer FIFO-Warteschlange muss mit dem Suffix `.fifo` enden. Das Suffix wird auf das Kontingent für Warteschlangennamen mit 80 Zeichen angerechnet. Um festzustellen, ob es sich bei einer Warteschlange um eine [FIFO-Warteschlange](#) handelt, können Sie überprüfen, ob der Warteschlangenname mit dem Suffix endet.

Im Folgenden finden Sie die Warteschlangen-URL für eine FIFO-Warteschlange, die einem Benutzer mit der AWS-Kontonummer MyQueue 123456789012 gehört.

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

Sie können die URL einer Warteschlange programmgesteuert abrufen, indem Sie Ihre Warteschlangen auflisten und die Zeichenfolge analysieren, die der Kontonummer folgt. Weitere Informationen finden Sie unter [ListQueues](#).

Nachrichten-ID

Jeder Nachricht wird vom System eine Nachrichten-ID zugewiesen, die Amazon SQS in der [SendMessage](#)-Antwort zurückgibt. Diese ID dient der Identifizierung von Nachrichten. Die maximale Länge einer Nachrichten-ID beträgt 100 Zeichen.

Empfangs-Mitteilung

Beim Empfang einer Nachricht aus einer Warteschlange erhalten Sie jedes Mal eine Empfangs-Mitteilung für diese Nachricht. Diese Mitteilung ist der Aktion des Nachrichtenempfangs zugeordnet und nicht der Nachricht selbst. Um eine Nachricht zu löschen oder ihre Sichtbarkeit zu ändern, müssen Sie die Empfangs-Mitteilung (und nicht die Nachrichten-ID) angeben. Daher müssen Sie eine Nachricht immer zunächst empfangen, bevor Sie sie löschen können. Es ist nicht möglich, eine Nachricht in die Warteschlange zu setzen und dann zurückzurufen. Die maximale Länge einer Empfangs-Mitteilung beträgt 1.024 Zeichen.

Important

Wenn Sie eine Nachricht mehrmals empfangen, erhalten Sie jedes Mal eine unterschiedliche Empfangs-Mitteilung. Wenn Sie die Nachricht löschen möchten, müssen Sie die zuletzt empfangene Empfangs-Mitteilung angeben, da die Nachricht sonst möglicherweise nicht gelöscht wird.

Es folgt ein Beispiel für eine Empfangs-Mitteilung (unterteilt in drei Zeilen).

```
MbZj6wDWli+JvwWJaBV+3dcjk2YW2vA3+STFF1jTM8tJJg6HRG6PYSasuWXPJB+Cw  
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq20KpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE  
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Zusätzliche Kennungen für Amazon-SQS-FIFO-Warteschlangen

Weitere Informationen zu den folgenden IDs finden Sie unter [Exactly-Once-Verarbeitung in Amazon SQS](#) und in der [Amazon-Simple-Queue-Service-API-Referenz](#).

Nachrichteneduplizierungs-ID

Ein Token, das in Amazon SQS FIFO-Warteschlangen verwendet wird, um Nachrichten eindeutig zu identifizieren und Duplikate zu verhindern. Wenn mehrere Nachrichten mit derselben Deduplizierungs-ID innerhalb eines Deduplizierungsintervalls von 5 Minuten gesendet werden, werden sie als Duplikate behandelt und es wird nur eine Kopie zugestellt. Wenn Sie keine Deduplizierungs-ID angeben und die inhaltsbasierte Deduplizierung aktiviert ist, generiert Amazon SQS eine Deduplizierungs-ID, indem es den Nachrichtentext hasht. Dieser Mechanismus gewährleistet, dass Nachrichten exakt einmal zugestellt werden, indem doppelte Nachrichten innerhalb des angegebenen Zeitrahmens entfernt werden.

Nachrichtengruppen-ID

Das `MessageGroupId` ist ein Attribut, das nur in Amazon SQS FIFO-Warteschlangen (First-In-First-Out) verwendet wird, um Nachrichten in verschiedene Gruppen zu organisieren. Nachrichten innerhalb derselben Nachrichtengruppe werden immer einzeln und in strikter Reihenfolge verarbeitet, sodass sichergestellt wird, dass keine zwei Nachrichten aus derselben Gruppe gleichzeitig verarbeitet werden. Standardwarteschlangen verwenden nicht `MessageGroupId` und bieten auch keine Garantie für die Reihenfolge. Wenn eine strikte Reihenfolge erforderlich ist, verwenden Sie stattdessen eine FIFO-Warteschlange.

Sequenznummer

Die große, nicht fortlaufende Zahl, die Amazon SQS jeder Nachricht zuweist.

Amazon-SQS-Kontingente

In diesem Thema werden die Kontingente und Einschränkungen für Amazon SQS FIFO- und Standardwarteschlangen erläutert und detailliert beschrieben, wie sie sich auf die Erstellung, Konfiguration und Nachrichtenverarbeitung von Warteschlangen auswirken. Erfahren Sie mehr über Einschränkungen wie Beschränkungen für die Aufbewahrung von Nachrichten, Obergrenzen für Nachrichten während der Übertragung und Durchsatzschwellenwerte sowie über Strategien zur Maximierung der Effizienz durch Batching, Optimierung von API-Aufrufen und lange Abfragen. In diesem Thema werden auch Benennungskonventionen, Kennzeichnungsregeln und Methoden zur Beantragung von Kontingenterhöhungen behandelt, um Workloads mit hohem Bedarf zu bewältigen und eine effektive Warteschlangenverwaltung und optimale Leistung sicherzustellen.

Amazon SQS FIFO-Warteschlangenkongingente

Amazon-SQS-Kontingente

In der folgenden Tabelle werden die Kontingente im Zusammenhang mit FIFO-Warteschlangen aufgeführt.

Kontingent	Beschreibung
Verzögerungwarteschlange	Die Standardverzögerung (Mindestverzögerung) für eine Warteschlange beträgt 0 Sekunden. Der Maximalwert beträgt 15 Minuten.
Aufgelistete Warteschlangen	1.000 Warteschlangen pro ListQueues -Anforderung.
Lange Wartezeit für Abfragen	Die maximale Wartezeit für lange Abfragen beträgt 20 Sekunden.
Mitteilungsgruppen	Es gibt kein Kontingent für die Anzahl der Nachricht engruppen in einer FIFO-Warteschlange.
Nachrichten pro Warteschlange (Rückstand)	Die Anzahl der Nachrichten, die eine Amazon-SQS-Warteschlange speichern kann, ist unbegrenzt.
Nachrichten pro Warteschlange (während der Übertragung)	FIFO-Warteschlangen unterstützen maximal 120.000 Bordnachrichten (Nachrichten, die von einem Verbrauch

Kontingent	Beschreibung
	er empfangen, aber noch nicht gelöscht wurden). Wenn dieses Limit erreicht wird, gibt Amazon SQS keinen Fehler zurück, aber die Verarbeitung kann beeinträchtigt werden. Sie können eine Erhöhung über dieses Limit hinaus beantragen, indem Sie sich an den AWS Support wenden.
Queue name (Name der Warteschlange)	Der Name einer FIFO-Warteschlange muss mit dem Suffix <code>.fifo</code> enden. Das Suffix wird auf das Kontingent für Warteschlangennamen mit 80 Zeichen angerechnet. Um festzustellen, ob es sich bei einer Warteschlange um eine FIFO-Warteschlange handelt, können Sie überprüfen, ob der Warteschlangenname mit dem Suffix endet.
Warteschlangen-Tag	Es wird nicht empfohlen, einer Warteschlange mehr als 50 Tags hinzuzufügen. Tagging unterstützt Unicode-Zeichen in UTF-8.
	Das Tag Key ist erforderlich, aber das Tag Value ist optional.
	Das Tag Key und das Tag Value unterscheiden zwischen Groß- und Kleinschreibung.
	Das Tag Key und das Tag Value können alphanumerischen Unicode-Zeichen in UTF-8 und Leerzeichen enthalten. Die folgenden Sonderzeichen sind zulässig: <code>_ . : / = + - @</code>
	Die Tags Key oder Value dürfen nicht das reservierte Präfix <code>aws:</code> enthalten (Sie können keine Tagschlüssel oder -werte mit diesem Präfix löschen).
	Die maximale Länge des Tags Key beträgt 128 Unicode-Zeichen in UTF-8. Das Tag Key darf nicht null oder leer sein.

Kontingent	Beschreibung
	Die maximale Länge des Tags Value beträgt 256 Unicode-Zeichen in UTF-8. Das Tag Value darf null oder leer sein. Tagging-Aktionen sind auf 30 TPS pro Tag begrenzt. AWS-Konto Wenn Ihre Anwendung einen höheren Durchsatz erfordert, reichen Sie eine Anfrage ein.

Amazon SQS SQS-Standard-Warteschlangenkontingente

In der folgenden Tabelle werden die Kontingente im Zusammenhang mit Standard-Warteschlangen aufgeführt.

Kontingent	Beschreibung
Verzögerungwarteschlange	Die Standardverzögerung (Mindestverzögerung) für eine Warteschlange beträgt 0 Sekunden. Der Maximalwert beträgt 15 Minuten.
Aufgelistete Warteschlangen	1.000 Warteschlangen pro ListQueues -Anforderung.
Lange Wartezeit für Abfragen	Die maximale Wartezeit für lange Abfragen beträgt 20 Sekunden.
Nachrichten pro Warteschlange (Rückstand)	Die Anzahl der Nachrichten, die eine Amazon-SQS-Warteschlange speichern kann, ist unbegrenzt.
Nachrichten pro Warteschlange (während der Übertragung)	Bei den meisten Standard-Warteschlangen (abhängig vom Warteschlangenverkehr und dem Nachricht enrückstand) kann es maximal etwa 120 000 In-Flight-Nachrichten geben (die von einem Verbraucher aus einer Warteschlange empfangen, aber noch nicht aus der Warteschlange gelöscht wurden). Wenn Sie dieses Kontingent während der Verwendung von Kurzabfragen erreichen, gibt Amazon SQS die Fehlermel

Kontingent	Beschreibung
	dung <code>OverLimit</code> zurück. Wenn Sie Langabfragen verwenden, gibt Amazon SQS keine Fehlermeldungen zurück. Um zu vermeiden, dass dieses Kontingent erreicht wird, sollten Sie Nachrichten aus der Warteschlange löschen, nachdem sie verarbeitet wurden. Sie können auch die Anzahl der Warteschlangen erhöhen, die Sie zur Verarbeitung Ihrer Nachrichten verwenden. Um eine Kontingenterhöhung anzufordern, übermitteln Sie eine Support-Anforderung.
Queue name (Name der Warteschlange)	Ein Warteschlangenname kann eine Länge von bis zu 80 Zeichen umfassen. Folgende Zeichen sind zulässig: alphanumerische Zeichen, Bindestriche (-) und Unterstriche (_).  Note Bei Warteschlangenamen ist die Groß- und Kleinschreibung zu beachten (<code>Test-queue</code> und <code>test-queue</code> sind z. B. zwei verschiedene Warteschlangen).
Warteschlangen-Tag	Es wird nicht empfohlen, einer Warteschlange mehr als 50 Tags hinzuzufügen. Tagging unterstützt Unicode-Zeichen in UTF-8. Das Tag Key ist erforderlich, aber das Tag Value ist optional. Das Tag Key und das Tag Value unterscheiden zwischen Groß- und Kleinschreibung. Das Tag Key und das Tag Value können alphanumerischen Unicode-Zeichen in UTF-8 und Leerzeichen enthalten. Die folgenden Sonderzeichen sind zulässig: <code>_ . : / = + - @</code>

Kontingent	Beschreibung
	Die Tags Key oder Value dürfen nicht das reservierte Präfix <code>aws:</code> enthalten (Sie können keine Tagschlüssel oder -werte mit diesem Präfix löschen).
	Die maximale Länge des Tags Key beträgt 128 Unicode-Zeichen in UTF-8. Das Tag Key darf nicht null oder leer sein.
	Die maximale Länge des Tags Value beträgt 256 Unicode-Zeichen in UTF-8. Das Tag Value darf null oder leer sein.
	Tagging-Aktionen sind auf 30 TPS pro Tag begrenzt. AWS-Konto Wenn Ihre Anwendung einen höheren Durchsatz erfordert, reichen Sie eine Anfrage ein .

Amazon SQS SQS-Nachrichtenkontingente

In der folgenden Tabelle werden die Kontingente im Zusammenhang mit Nachrichten aufgeführt.

Kontingent	Beschreibung
Mitteilungs-ID im Stapel	Eine Stapel-Nachrichten-ID kann bis zu 80 Zeichen lang sein. Folgende Zeichen sind zulässig: alphanumerische Zeichen, Bindestriche (-) und Unterstriche (_).
Nachrichtenattribute	Eine Nachricht kann bis zu 10 Metadatenattribute enthalten.
Nachrichtenstapel	Eine einzelne Nachrichtstapelanforderung kann maximal 10 Nachrichten umfassen. Weitere Informationen finden Sie unter Amazon konfigurieren SQSBuffered AsyncClient im Abschnitt Amazon-SQS-Stapelaktionen .
Nachrichteninhalt	Eine Nachricht kann nur XML, JSON und unformatierten Text enthalten. Die folgenden Unicode-Zeichen sind

Kontingent	Beschreibung
	zulässig: #x9 #xA #xD #x20 bis #xD7FF #xE000 bis #xFFFD #x10000 to #x10FFFF Alle Zeichen, die nicht in diese Liste enthalten sind, werden abgelehnt. Weitere Informationen finden Sie in der W3C-Spezifikation für Zeichen.
Nachrichtengruppen-ID	<code>MessageGroupId</code> ist für FIFO-Warteschlangen erforderlich. Wenn Sie <code>MessageGroupId</code> beim Senden einer Nachricht an eine FIFO-Warteschlange keine angeben, schlägt die Aktion fehl. In Standardwarteschlangen <code>MessageGroupId</code> ermöglicht die Verwendung fairer Warteschlangen. Wenn Sie faire Warteschlangen verwenden, empfehlen wir, <code>MessageGroupId</code> in allen Nachrichten ein zu verwenden. Die maximale Länge der <code>MessageGroupId</code> ist 128 Zeichen. Gültige Werte: alphanumerische Zeichen und Satzzeichen (! " # \$ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ` { } ~) .
Nachrichtenspeicherung	Standardmäßig wird eine Nachricht 4 Tage aufbewahrt. Die Mindestdauer 60 Sekunden (1 Minute). Die Höchstdauer ist 1 209 600 Sekunden (14 Tage).

Kontingent	Beschreibung
Nachrichtendurchsatz	Standardwarteschlangen Standardwarteschlangen unterstützen eine sehr hohe, fast unbegrenzte Anzahl von API-Aufrufen pro Sekunde und Aktion (SendMessage , ReceiveMessage , oder DeleteMessage). Aufgrund dieses hohen Durchsatzes eignen sie sich ideal für Anwendungsfälle, bei denen große Nachrichtenmengen schnell verarbeitet werden müssen, z. B. Datenstreaming in Echtzeit oder umfangreiche Anwendungen. Standardwarteschlangen werden zwar automatisch mit der Nachfrage skaliert, es ist jedoch wichtig, die Nutzungsmuster zu überwachen, um eine optimale Leistung zu gewährleisten, insbesondere in Regionen mit höherer Arbeitslast.

Kontingent	Beschreibung
	<u>FIFO-Warteschlangen</u> • Jede Partition in einer FIFO-Warteschlange ist auf 300 Transaktionen pro Sekunde pro API-Aktion (<code>SendMessage</code>, <code>ReceiveMessage</code>, und <code>DeleteMessage</code>) begrenzt. Dieses Limit gilt speziell für den Modus ohne hohen Durchsatz. Wenn Sie in den Modus mit hohem Durchsatz wechseln, können Sie dieses Standardlimit überschreiten. Informationen zum Aktivieren des Hochdurchsatzmodus finden Sie unter <u>Aktivierung eines hohen Durchsatzes für FIFO-Warteschlangen in Amazon SQS</u> • Wenn Sie Batching verwenden, unterstützen FIFO-Warteschlangen ohne hohen Durchsatz bis zu 3.000 Nachrichten pro Sekunde und API-Aktion (<code>SendMessage</code>, und <code>ReceiveMessage</code>). <code>DeleteMessage</code> Die 3.000 Transaktionen pro Sekunde repräsentieren 300 API-Aufrufe mit jeweils einem Stapel von 10 Nachrichten. <u>Hoher Durchsatz für FIFO-Warteschlangen</u> Die FIFO-Limits von Amazon SQS basieren auf der Anzahl der API-Anfragen, nicht auf Nachrichtenlimits. Für den Modus mit hohem Durchsatz lauten diese API-Anforderungslimits wie folgt: Grenzwerte für den Transaktionsdurchsatz (API-Aufrufe ohne Batchverarbeitung) Diese Grenzwerte definieren, wie oft jede API-Operation (wie <u>SendMessageReceiveMessage</u>, oder <u>DeleteMessage</u>) unabhängig voneinander ausgeführt werden kann, wodurch eine effiziente Systemleistung innerhalb der

Kontingent	Beschreibung
	zulässigen Transaktionen pro Sekunde (TPS) gewährleistet wird. Die folgenden Grenzwerte basieren auf API-Aufrufen, die nicht gebündelt wurden: • USA Ost (Nord-Virginia), USA West (Oregon) und Europa (Irland): Bis zu 70.000 Transaktionen pro Sekunde (TPS). • USA Ost (Ohio) und Europa (Frankfurt): Bis zu 19.000 TPS. • Asien-Pazifik (Mumbai), Asien-Pazifik (Singapur), Asien-Pazifik (Sydney) und Asien-Pazifik (Tokio): Bis zu 9.000 TPS. • Europa (London) und Südamerika (São Paulo): Bis zu 4.500 TPS. • Alle anderen AWS-Regionen: Standarddurchsatz von 2.400 TPS. Maximierung des Durchsatzes durch Batching Verarbeitet mehrere Nachrichten in einem einzigen API-Aufruf, was die Effizienz erheblich steigert. Anstatt jede Nachricht einzeln zu behandeln, können Sie mit Batching bis zu 10 Nachrichten in einer einzigen API-Anfrage senden, empfangen oder löschen. Dadurch wird die Gesamtzahl der API-Aufrufe reduziert, sodass Sie mehr Nachrichten pro Sekunde verarbeiten und gleichzeitig die Transaktionslimits (TPS) für die Region einhalten können, wodurch der Durchsatz und die Systemleistung maximiert werden. Weitere Informationen finden Sie unter Steigerung des Durchsatzes durch horizontale Skalierung und Action-Batching mit Amazon SQS.

Kontingent	Beschreibung
	Die folgenden Grenzwerte basieren auf gebündelten API-Aufrufen: • USA Ost (Nord-Virginia), USA West (Oregon) und Europa (Irland): Bis zu 700.000 Nachrichten pro Sekunde (das Zehnfache des Non-Batch-Limits von 70.000 TPS). • USA Ost (Ohio) und Europa (Frankfurt): Bis zu 190.000 Nachrichten pro Sekunde. • Asien-Pazifik (Mumbai), Asien-Pazifik (Singapur), Asien-Pazifik (Sydney) und Asien-Pazifik (Tokio): Bis zu 90.000 Nachrichten pro Sekunde. • Europa (London) und Südamerika (São Paulo): Bis zu 45.000 Nachrichten pro Sekunde. • Alle anderen AWS-Regionen: Bis zu 24.000 Nachrichten pro Sekunde. Optimierung des Durchsatzes über die Batchverarbeitung hinaus Durch Batching kann der Durchsatz zwar erheblich gesteigert werden, es ist jedoch wichtig, andere Strategien zur Optimierung der FIFO-Leistung in Betracht zu ziehen: • Nachrichten auf mehrere Nachrichtengruppen verteilen IDs — Da Nachrichten innerhalb einer einzelnen Gruppe sequentiell verarbeitet werden, ermöglicht die Verteilung Ihrer Arbeitslast auf mehrere Nachrichtengruppen eine bessere Parallelität und einen höheren Gesamtdurchsatz. Weitere Informationen finden Sie unter Partitionen und Datenverteilung für hohen Durchsatz für SQS-FIFO-Warteschlangen.

Kontingent	Beschreibung
	• Effizienter Einsatz von API-Aufrufen — Vermeiden Sie unnötige API-Aufrufe, wie z. B. häufige Änderungen der Sichtbarkeit oder wiederholte Löschungen von Nachrichten, um die Nutzung Ihres verfügbaren TPS zu optimieren und die Effizienz zu steigern. • Verwenden Sie lange Umfrageeingänge — Nutzen Sie lange Abfragen, indem Sie WaitTimeSeconds in Ihren Empfangsanfragen die Anzahl leerer Antworten, wenn keine Nachrichten verfügbar sind, reduzieren, unnötige API-Aufrufe reduzieren und Ihr TPS-Kontingent besser nutzen. • Durchsatzerhöhungen anfordern — Wenn Ihre Anwendung einen Durchsatz benötigt, der über den Standardgrenzwerten liegt, fordern Sie eine Erhöhung über die Service Quotas Quotas-Konsole an. Dies kann bei stark beanspruchten Workloads oder in Regionen mit niedrigeren Standardgrenzwerten erforderlich sein. Informationen zur Aktivierung des Hochdurchsatzmodus finden Sie unter Aktivierung eines hohen Durchsatzes für FIFO-Warteschlangen in Amazon SQS
Nachrichten-Timer	Die Standardverzögerung (Mindestverzögerung) für eine Nachricht beträgt 0 Sekunden. Der Maximalwert beträgt 15 Minuten.

Kontingent	Beschreibung
Nachrichtengröße	Die Mindestnachrichtengröße ist 1 Byte (1 Zeichen). Das Maximum ist 1.048.576 Byte (1 MiB). Um Nachrichten zu senden, die größer als 1 MiB sind, können Sie die Amazon SQS Extended Client Library für Java und die Amazon SQS Extended Client Library für Python verwenden. Diese Bibliothek erlaubt das Senden einer Amazon-SQS-Nachricht, die auf eine Nachricht ennutzlast in Amazon S3 verweist. Die maximale Nutzlastgröße beträgt 2 GB.  Note Diese erweiterte Bibliothek funktioniert nur für synchrone Clients.
Zeitbeschränkung für die Sichtbarkeit von Nachrichten	Die Standardzeitbeschränkung für die Sichtbarkeit einer Nachricht ist 30 Sekunden. Der Mindestwert beträgt 0 Sekunden. Der Höchstwert beträgt 12 Stunden.
Richtlinieninformationen	Das Höchstkontingent beträgt 8 192 Byte, 20 Anweisungen, 50 Prinzipale oder 10 Bedingungen. Weitere Informationen finden Sie unter Amazon SQS SQS-Richtlinienkontingente .

Amazon SQS SQS-Richtlinienkontingente

In der folgenden Tabelle werden die Kontingente im Zusammenhang mit Richtlinien aufgeführt.

Name	Maximum
Bytes	8,192
Bedingungen	10

Name	Maximum
Prinzipale	50
Anweisungen	20
Aktionen pro Anweisung	7

Features und Fähigkeiten von Amazon SQS

Dieses Thema enthält häufig verwendete Funktionen in Amazon SQS zur Verwaltung von Nachrichtenwarteschlangen, zur Optimierung der Leistung, zur Sicherstellung einer zuverlässigen Nachrichtenzustellung und zur effizienten Verarbeitung von Nachrichten.

Verwenden von Warteschlangen für unzustellbare Briefe in Amazon SQS

Amazon SQS unterstützt Warteschlangen (DLQs), auf die Quellwarteschlangen für Nachrichten abzielen können, die nicht erfolgreich verarbeitet wurden. DLQs sind nützlich für das Debuggen Ihrer Anwendung, da Sie nicht verbrauchte Nachrichten isolieren können, um festzustellen, warum die Verarbeitung nicht erfolgreich war. Um eine optimale Leistung zu erzielen, empfiehlt es sich, die Quellwarteschlange und die DLQ innerhalb derselben AWS-Konto Region zu halten. Sobald sich Nachrichten in einer Warteschlange für unzustellbare Nachrichten befinden, können Sie:

- Prüfen Sie die Protokolle auf Ausnahmen, die dazu geführt haben könnten, dass Nachrichten an eine Warteschlange für unzustellbare Nachrichten gesendet wurden.
- Analysieren Sie den Inhalt von Nachrichten, die in die Warteschlange für unzustellbare Nachrichten verschoben wurden, um Anwendungsprobleme zu diagnostizieren.
- Bestimmen, ob der Konsument ausreichend Zeit zum Verarbeiten der Nachrichten hatte
- [Verschieben Sie Nachrichten mithilfe von Redrive für unzustellbare Nachrichten aus der Warteschlange für unzustellbare Nachrichten.](#)

Sie müssen zuerst eine neue Warteschlange erstellen, bevor Sie sie als Warteschlange für unzustellbare Briefe konfigurieren können. Informationen zur Konfiguration einer Warteschlange für unzustellbare Nachrichten mithilfe der Amazon-SQS-Konsole finden Sie unter [Konfigurieren Sie eine Warteschlange für unzustellbare Briefe mit der Amazon SQS SQS-Konsole](#). Hilfe zu Warteschlangen für unzustellbare Nachrichten, z. B. zur Konfiguration eines Alarms für Nachrichten, die in eine Warteschleife mit unzustellbaren Nachrichten verschoben werden, finden Sie unter [Alarme für Warteschlangen mit unbestätigten Briefen mithilfe von Amazon erstellen CloudWatch](#)

 Note

Verwenden Sie keine Warteschlange für unzustellbare Nachrichten in Verbindung mit einer FIFO-Warteschlange, wenn Sie die Reihenfolge von Nachrichten oder Vorgängen nicht zerstören möchten. Verwenden Sie eine Warteschlange für unzustellbare Nachrichten beispielsweise nicht zusammen mit Anweisungen in einer Bearbeitungsentscheidungsliste (Edit Decision List, EDL) für eine Videobearbeitungs-Suite, wo sich Änderungen an der Reihenfolge von vorgenommenen Änderungen auf den Inhalt von Folgeänderungen auswirken können.

Verwenden von Richtlinien für Warteschlangen mit unzustellbaren Briefen

Verwenden Sie eine Redrive-Richtlinie, um die anzugeben. `maxReceiveCount` Dies gibt `maxReceiveCount` an, wie oft ein Verbraucher eine Nachricht aus einer Quellwarteschlange empfangen kann, bevor sie in eine Warteschlange für unzustellbare Nachrichten verschoben wird. Wenn für beispielsweise ein niedriger Wert wie 1 festgelegt `maxReceiveCount` ist, würde ein Fehler beim Empfang einer Nachricht dazu führen, dass die Nachricht in die Warteschlange für unzustellbare Briefe verschoben wird. Um sicherzustellen, dass Ihr System gegen Fehler resistent ist, legen Sie den Wert für `maxReceiveCount` hoch genug fest, um ausreichend Wiederholungsversuche zu ermöglichen.

Bei Standardwarteschlangen mit einer Redrive-Richtlinie, bei `maxReceiveCount` der ein Wert größer als 3 ist, verschiebt SQS sie an das Ende der Warteschlange, wenn sie dreimal oder öfter empfangen wird, ohne gelöscht zu werden. Die `ApproximateAgeOfOldestMessage` Metrik gibt dann das Alter der nächsten Nachricht wieder, die diesen Schwellenwert nicht überschritten hat.

Die Redrive-Zulassungsrichtlinie legt fest, welche Quellwarteschlange auf die Warteschlange für unzustellbare Nachrichten zugreifen kann. Sie können wählen, ob Sie alle Quellwarteschlangen, bestimmte Quellwarteschlangen zulassen oder allen Quellwarteschlangen die Verwendung der Warteschlange für unzustellbare Briefe verweigern möchten. Standardmäßig können alle Quellwarteschlangen die Warteschlange für unzustellbare Briefe verwenden. Wenn Sie mit dieser `byQueue` Option bestimmte Warteschlangen zulassen möchten, können Sie mit dem Amazon Resource Name (ARN) der Quellwarteschlange bis zu 10 Quellwarteschlangen angeben. Wenn Sie `denyAll` angeben, kann die Warteschlange nicht als Warteschlange für unzustellbare Nachrichten verwendet werden.

Grundlegendes zu den Aufbewahrungsfristen für Nachrichten in Warteschlangen mit unerlaubten Briefen

Bei Standard-Warteschlangen basiert der Ablauf einer Nachricht immer auf dem ursprünglichen Enqueue-Zeitstempel. Wenn eine Nachricht in eine Warteschlange für unzustellbare Nachrichten verschoben wird, bleibt der Enqueue-Zeitstempel unverändert. Die `ApproximateAgeOfOldestMessage` Metrik gibt an, wann die Nachricht in die Warteschlange für unzustellbare Briefe verschoben wurde, und nicht, wann die Nachricht ursprünglich gesendet wurde. Nehmen wir beispielsweise an, dass sich eine Nachricht einen Tag in der ursprünglichen Warteschlange befindet, bevor sie in eine Warteschlange für unzustellbare Nachrichten verschoben wird. Wenn die Aufbewahrungsfrist der Warteschlange für unzustellbare Nachrichten 4 Tage beträgt, wird die Nachricht nach 3 Tagen aus der Warteschlange für unzustellbare Nachrichten gelöscht und das `ApproximateAgeOfOldestMessage` ist 3 Tage. Es hat sich daher bewährt, die Aufbewahrungsdauer einer Warteschlange für unzustellbare Nachrichten immer so festzulegen, dass sie länger ist als die Aufbewahrungsdauer der ursprünglichen Warteschlange.

Bei FIFO-Warteschlangen wird der Enqueue-Zeitstempel zurückgesetzt, wenn die Nachricht in eine Warteschlange für unzustellbare Nachrichten verschoben wird. Die `ApproximateAgeOfOldestMessage`-Metrik gibt an, wann die Nachricht in die Warteschlange für unzustellbare Nachrichten verschoben wird. Im obigen Beispiel wird die Nachricht nach vier Tagen aus der Warteschlange für unzustellbare Briefe gelöscht, und das `ApproximateAgeOfOldestMessage` sind vier Tage.

Konfigurieren Sie eine Warteschlange für unzustellbare Briefe mit der Amazon SQS SQS-Konsole

Eine Warteschlange mit unerlaubten Briefen (DLQ) ist eine Warteschlange, die Nachrichten empfängt, die nicht erfolgreich verarbeitet wurden, aus einer anderen Warteschlange, der sogenannten Quellwarteschlange. Amazon SQS erstellt die Warteschlange für unzustellbare Nachrichten nicht automatisch. Sie müssen zuerst die Warteschlange erstellen, bevor Sie sie als Warteschlange für unzustellbare Nachrichten festlegen. Bei der Konfiguration einer DLQ muss der Queue-Typ mit dem Queue-Typ der Quell-Queue übereinstimmen — eine [FIFO-Warteschlange kann nur eine FIFO-DLQ](#) verwenden, und eine Standardwarteschlange kann nur eine [Standard-DLQ](#) verwenden. Sie können beim Erstellen oder Bearbeiten einer Warteschlange eine Warteschlange mit uneingeschränkten Briefen konfigurieren. Weitere Details finden Sie unter [Verwenden von Warteschlangen für unzustellbare Briefe in Amazon SQS](#).

Konfigurieren einer Warteschlange für unzustellbare Nachrichten für eine vorhandene Warteschlange (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie die Quellwarteschlange aus (die Warteschlange, die fehlgeschlagene Nachrichten an die Warteschlange mit unzustellbaren Nachrichten sendet) und wählen Sie dann Bearbeiten.
4. Scrollen Sie zum Abschnitt „Warteschlange für unzustellbare Briefe“ und schalten Sie die Option Aktiviert um.
5. Wählen Sie unter Einstellungen für Warteschlange für unzustellbare Briefe den Amazon-Ressourcennamen (ARN) einer vorhandenen Warteschlange aus, die Sie als Warteschlange für unzustellbare Briefe verwenden möchten.
6. Legen Sie den Wert für maximale Anzahl an Empfängen fest, der definiert, wie oft eine Nachricht empfangen werden kann, bevor sie an die Warteschlange für unzustellbare Briefe gesendet wird (gültiger Bereich: 1 bis 1.000).
7. Wählen Sie Speichern.

Erfahren Sie, wie Sie in Amazon SQS ein Redrive für Warteschlangen mit unerlaubten Briefen konfigurieren

Verwenden Sie Redrive für unzustellbare Nachrichten, um Nachrichten, die noch nicht verarbeitet wurden, aus einer Warteschlange für unzustellbare Nachrichten an ein anderes Ziel zu verschieben und dort zu verarbeiten. Standardmäßig verschiebt das Redrive für Warteschlangen für unzustellbare Nachrichten Nachrichten aus einer Warteschlange für unzustellbare Nachrichten in eine Quellwarteschlange. Sie können jedoch auch jede andere Warteschlange als Redrive-Ziel konfigurieren, wenn beide Warteschlangenarten vom gleichen Typ sind. Wenn es sich bei der Warteschlange für unzustellbare Nachrichten beispielsweise um eine FIFO-Warteschlange handelt, muss es sich bei der Redrive-Zielwarteschlange ebenfalls um eine FIFO-Warteschlange handeln. Darüber hinaus können Sie die Redrive-Geschwindigkeit konfigurieren, um die Geschwindigkeit festzulegen, mit der Amazon SQS Nachrichten verschiebt.

Note

Wird eine Nachricht von einer FIFO-Warteschlange in eine FIFO-DLQ verschoben, wird die Deduplizierungs-ID der ursprünglichen Nachricht durch die ID der ursprünglichen Nachricht ersetzt. Dadurch soll sichergestellt werden, dass die DLQ-Deduplizierung das Speichern von

zwei unabhängigen Nachrichten, die zufällig eine gemeinsame Deduplizierungs-ID haben, nicht verhindert.

In Warteschlangen für unzustellbare Nachrichten werden Nachrichten in der Reihenfolge ihres Eingangs erneut weitergeleitet, wobei mit der ältesten Nachricht begonnen wird. Die Zielwarteschlange nimmt jedoch die Nachrichten, die erneut gesendet wurden, sowie Nachrichten von anderen Produzenten auf, je nachdem, wie in welcher Reihenfolge sie empfangen wurden. Wenn ein Producer beispielsweise Nachrichten an eine FIFO-Quellwarteschlange sendet und gleichzeitig Nachrichten aus einer Warteschlange für unzustellbare Nachrichten empfängt, werden die weitergeleiteten Nachrichten mit den neuen Nachrichten des Produzenten verwoben.

 Note

Die Redrive-Aufgabe setzt die Aufbewahrungsfrist zurück. Alle neu zugestellten Nachrichten werden als neue Nachrichten betrachtet und `messageID` und `enqueueTime` den neu weitergeleiteten Nachrichten zugewiesen.

Konfiguration einer Redrive für Warteschlangen mit unerlaubten Briefen für eine bestehende Standardwarteschlange mithilfe der Amazon SQS SQS-API

Mit den API-Aktionen, und können Sie eine Warteschleife für unzustellbare Nachrichten `StartMessageMoveTask` erneut einrichten `ListMessageMoveTasks`: `CancelMessageMoveTask`

API-Aktion	Description
<u>StartMessageMoveTask</u>	Startet eine asynchrone Aufgabe, um Nachrichten von einer angegebenen Quellwarteschlange in eine angegebene Zielwarteschlange zu verschieben.
<u>ListMessageMoveTasks</u>	Ruft die neuesten Aufgaben zum Verschieben von Nachrichten (bis zu 10) in einer bestimmten Quellwarteschlange ab.
<u>CancelMessageMoveTask</u>	Bricht eine angegebene Aufgabe zum Verschieben von Nachrichten ab. Eine

API-Aktion	Description
	Nachrichtenbewegung kann nur abgebrochen werden, wenn der aktuelle Status RUNNING ist.

Mit der Amazon SQS SQS-Konsole ein Redrive für eine bestehende Standardwarteschlange konfigurieren

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie den Namen der Warteschlange, die Sie als [Warteschlange für unzustellbare Nachrichten](#) konfiguriert haben.
4. Wählen Sie DLQ-Redrive starten.
5. Führen Sie unter Redrive-Konfiguration für Nachrichtenziel einen der folgenden Schritte aus:
 - Um Nachrichten erneut in ihre Quellwarteschlange weiterzuleiten, wählen Sie Redrive zu Quell-Warteschlange(n).
 - Um Nachrichten erneut in eine andere Warteschlange weiterzuleiten, wählen Sie Redrive zu benutzerdefiniertem Ziel. Geben Sie dann den Amazon-Ressourcennamen (ARN) einer vorhandenen Zielwarteschlange an.
6. Wählen Sie unter Einstellungen für die Geschwindigkeitssteuerung eine der folgenden Optionen aus:
 - Systemoptimiert – Redrive von Nachrichten aus der Warteschlange für unzustellbare Nachrichten mit der maximalen Anzahl an Nachrichten pro Sekunde.
 - Benutzerdefinierte maximale Geschwindigkeit – Redrive von Nachrichten aus der Warteschlange für unzustellbare Nachrichten mit einer benutzerdefinierten Höchstzahl an Nachrichten pro Sekunde. Die maximal zulässige Rate beträgt 500 Nachrichten pro Sekunde.
 - Es wird empfohlen, mit einem kleinen Wert für die benutzerdefinierte maximale Geschwindigkeit zu beginnen und sicherzustellen, dass die Quellwarteschlange nicht mit Nachrichten überfüllt wird. Erhöhen Sie von dort aus schrittweise den Wert für die benutzerdefinierte maximale Geschwindigkeit und überwachen Sie weiterhin den Status der Quellwarteschlange.

7. Wenn Sie mit der Konfiguration des Redrives für die Warteschlange für unzustellbare Nachrichten fertig sind, wählen Sie Nachrichten-Redrive aus.

⚠ Important

Amazon SQS unterstützt das Filtern und Ändern von Nachrichten nicht, während sie aus der Warteschlange für unzustellbare Nachrichten zurückgeleitet werden.

Eine Redrive-Aufgabe für eine Warteschlange für unzustellbare Nachrichten kann maximal 36 Stunden lang ausgeführt werden. Amazon SQS unterstützt maximal 100 aktive Redrive-Aufgaben pro Konto.

8. Wenn Sie die Nachrichten-Redrive-Aufgabe abbrechen möchten, wählen Sie auf der Seite Details für Ihre Warteschlange die Option DLQ-Redrive abbrechen aus. Wenn Sie einen in Bearbeitung befindlichen Nachrichten-Redrive stornieren, verbleiben alle Nachrichten, die bereits erfolgreich in die Warteschlange für die Übertragung von Nachrichten verschoben wurden, in der Zielwarteschlange.

Konfigurieren von Warteschlangenberechtigungen für Redrives von Warteschlangen für unzustellbare Nachrichten

Sie können Benutzern Zugriff auf bestimmte Aktionen in der Warteschlange für unzustellbare Nachrichten gewähren, indem Sie Ihrer Richtlinie entsprechende Berechtigungen hinzufügen. Die Mindestberechtigungen für das Redrive einer Warteschlange für unzustellbare Nachrichten lauten wie folgt:

Mindestberechtigungen	Erforderliche API-Methoden
So starten Sie ein Nachrichten-Redrive	- Fügen Sie <code>sqs:StartMessageMoveTask</code> , <code>sqs:ReceiveMessage</code> , <code>sqs:DeleteMessage</code> und <code>sqs:GetQueueAttributes</code> der Warteschlange für unzustellbare Nachrichten hinzu. Wenn entweder die Warteschlange für unzustellbare Nachrichten oder die ursprüngliche Quellwarteschlange verschlüsselt sind (auch als SSE-Warteschlange bezeichnet), ist <code>kms:Decrypt</code> für jeden KMS-Schlüssel, der zum Verschlüsseln der Nachrichten verwendet wurde, ebenfalls erforderlich.

Mindestberechtigungen	Erforderliche API-Methoden
	Fügen Sie die <code>sqs:SendMessage</code> der Zielwarteschlange hinzu. Wenn die Zielwarteschlange verschlüsselt ist, sind <code>kms:GenerateDataKey</code> und <code>kms:Decrypt</code> ebenfalls erforderlich.
So stornieren Sie einen in Bearbeitung befindlichen Nachrichten-Redrive	Fügen Sie <code>sqs:CancelMessageMoveTask</code>, <code>sqs:ReceiveMessage</code>, <code>sqs>DeleteMessage</code> und <code>sqs:GetQueueAttributes</code> der Warteschlange für unzustellbare Nachrichten hinzu. Wenn die Warteschlange für unzustellbare Nachrichten verschlüsselt ist (auch als SSE-Warteschlange bezeichnet), ist <code>kms:Decrypt</code> ebenfalls erforderlich.
So zeigen Sie den Verschiebungsstatus einer Nachricht an	Fügen Sie <code>sqs:ListMessageMoveTasks</code> und <code>sqs:GetQueueAttributes</code> der Warteschlange für unzustellbare Nachrichten hinzu.

So konfigurieren Sie Berechtigungen für ein unverschlüsseltes Warteschlangenpaar (eine Quellwarteschlange mit einer Warteschlange für unzustellbare Nachrichten)

Gehen Sie wie folgt vor, um Mindestberechtigungen für eine Dead-Letter-Queue (DLQ) -Redrive zu konfigurieren:

- Öffnen Sie unter <https://console.aws.amazon.com/iam/> die IAM-Konsole.
- Klicken Sie im Navigationsbereich auf Policies (Richtlinien).
- Erstellen Sie eine neue [Richtlinie und fügen Sie die](#) folgenden Berechtigungen hinzu. Ordnen Sie die Richtlinie dem [IAM-Benutzer](#) oder der [IAM-Rolle](#) zu, die den Redrive-Vorgang ausführen wird.
 - Berechtigungen für die DLQ (Quellwarteschlange):
 - `sqs:StartMessageMoveTask`
 - `sqs:CancelMessageMoveTask`
 - `sqs:ListMessageMoveTasks`
 - `sqs:ReceiveMessage`

- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`
- `sqs:ListDeadLetterSourceQueues`
- Geben Sie den Ressourcen-ARN der DLQ (Quellwarteschlange) an (z. B. „arn:aws:sqs::: „).
`<DLQ_region> <DLQ_accountId> <DLQ_name>`
- Berechtigungen für die Zielwarteschlange:
 - `sqs:SendMessage`
 - Geben Sie die Resource ARN Zielwarteschlange an (z. B. „arn:aws:sqs: „).
`<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>`
- Berechtigungen für KMS-Schlüssel:
 - `kms:Decrypt`(Wird benötigt, um Nachrichten in der DLQ zu entschlüsseln.)
 - `kms:GenerateDataKey`(Wird benötigt, um Nachrichten in der Zielwarteschlange zu verschlüsseln.)
 - Resource ARNs:
 - Der ARN des KMS-Schlüssels, der zum Verschlüsseln von Nachrichten in der DLQ (Quellwarteschlange) verwendet wird (z. B. „arn:aws:kms: ::key/“). `<region> <accountId> <SourceQueueKeyId>`
 - Der ARN des KMS-Schlüssels, der zum Verschlüsseln von Nachrichten in der Zielwarteschlange verwendet wird (z. B. „arn:aws:kms: ::key/“). `<region> <accountId> <DestinationQueueKeyId>`

Ihre Zugriffsrichtlinie sollte wie folgt aussehen:

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:us-west-1:123456789012:<DLQ_name>",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/QueueRole": "source"
        }
    }
},
{
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-west-1:123456789012:<DestQueue_name>",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/QueueRole": "destination"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": [
        "arn:aws:kms:us-west-1:123456789012:key/<SourceQueueKeyId>",
        "arn:aws:kms:us-west-1:123456789012:key/<DestQueueKeyId>"
    ]
}
]
}

```

So konfigurieren Sie Berechtigungen mithilfe eines unverschlüsselten Warteschlangenpaars (einer Quellwarteschlange mit einer Warteschlange für unzustellbare Nachrichten)

Gehen Sie wie folgt vor, um die Mindestberechtigungen zu konfigurieren, die für den Umgang mit einer standardmäßigen, unverschlüsselten Warteschlange mit unerlaubten Briefen (DLQ) erforderlich sind. Erforderliche Mindestberechtigungen sind diejenigen zum Empfangen, Löschen und Abrufen

von Attributen aus der Warteschlange für unzustellbare Nachrichten sowie für das Senden von Attributen an die Quellwarteschlange.

1. Öffnen Sie unter <https://console.aws.amazon.com/iam/> die IAM-Konsole.
2. Klicken Sie im Navigationsbereich auf Policies (Richtlinien).
3. Erstellen Sie eine neue [Richtlinie und fügen Sie die](#) folgenden Berechtigungen hinzu. Ordnen Sie die Richtlinie dem [IAM-Benutzer](#) oder der [IAM-Rolle](#) zu, die den Redrive-Vorgang ausführen wird.
 - Berechtigungen für die DLQ (Quellwarteschlange):
 - `sqs:StartMessageMoveTask`
 - `sqs:CancelMessageMoveTask`
 - `sqs:ListMessageMoveTasks`
 - `sqs:ReceiveMessage`
 - `sqs>DeleteMessage`
 - `sqs:ListDeadLetterSourceQueues`
 - Geben Sie den Ressourcen-ARN der DLQ (Quellwarteschlange) an (z. B. „arn:aws:sqs::: „).
`<DLQ_region> <DLQ_accountId> <DLQ_name>`
 - Berechtigungen für die Zielwarteschlange:
 - `sqs:SendMessage`
 - Geben Sie die Resource ARN Zielwarteschlange an (z. B. „arn:aws:sqs: „).
`<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>`

Ihre Zugriffsrichtlinie sollte wie folgt aussehen:

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:us-west-1:111122223333:<DLQ_name>",
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/QueueRole": "source"
        }
    },
    {
        "Effect": "Allow",
        "Action": "sqs:SendMessage",
        "Resource": "arn:aws:sqs:us-west-1:111122223333:<DestQueue_name>",
        "Condition": {
            "StringEquals": {
                "aws:ResourceTag/QueueRole": "destination"
            }
        }
    }
]
}

```

Verwenden von Redrive für Warteschlangen mit VPC-Endpunktzugriffskontrolle

Wenn Sie den Zugriff auf eine bestimmte Warteschlange VPCs mithilfe der `aws:sourceVpc` Bedingung einschränken, müssen Sie für AWS Dienste eine Ausnahme einrichten, um die Funktion zur erneuten Übertragung von Warteschlangen für Dead-Letter (DLQ) zu aktivieren. Das liegt daran, dass der Amazon SQS-Service beim Verschieben von Nachrichten außerhalb Ihrer VPC arbeitet.

Um DLQ-Redrive-Operationen zuzulassen, fügen Sie die `aws:CalledViaLast` Bedingung zu Ihrer Warteschlangenrichtlinie hinzu. Auf diese Weise kann Amazon SQS API-Aufrufe in Ihrem Namen tätigen und gleichzeitig die VPC-Einschränkungen für den direkten Zugriff beibehalten.

Gehen Sie wie folgt vor, um sowohl VPC-eingeschränkten Zugriff als auch DLQ Redrive zuzulassen:

1. Verwenden Sie die `aws:CalledViaLast` Bedingung in Ihrer Warteschlangenrichtlinie.

2. Wenden Sie die Richtlinie sowohl auf die Quellwarteschlange als auch auf die DLQ an
3. Halten Sie VPC-Einschränkungen für den direkten Zugriff aus anderen Quellen aufrecht

Im Folgenden finden Sie ein Beispiel für eine Richtlinie, die diese Anforderungen implementiert:

JSON

```
{
  "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"
        }
      }
    }
  ]
}
```

```
}
```

- Ersetzen Sie die Platzhalterwerte durch Ihre tatsächlichen Werte
- Diese Richtlinie verwendet eine „Deny“-Anweisung mit Bedingungen, was sicherer ist als die Verwendung von „Allow“-Anweisungen
- Der `StringNotEqualsIfExists` Operator behandelt Fälle, in denen der Bedingungsschlüssel im Anforderungskontext möglicherweise nicht vorhanden ist.

Alternativ können Sie den `aws:ViaAWSService` Bedingungsschlüssel verwenden, um den dienstbasierten Zugriff zu ermöglichen und gleichzeitig die VPC-Einschränkungen beizubehalten. Dieser Bedingungsschlüssel gibt an, ob die Anfrage von einem AWS Dienst stammt. Hier ist ein Beispiel für eine Richtlinie, die `aws:ViaAWSService` anstelle von `aws:CalledViaLast` verwendet:

JSON

```
{
  "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"
  }
}
}
```

Der BoolIfExists Betreiber mit `aws:ViaAWSService` Bedingung stellt sicher, dass Anfragen zulässig sind, wenn sie von Diensten stammen, und behält gleichzeitig die VPC-Beschränkungen für den direkten Zugriff bei. Dies kann einfacher zu verstehen und zu verwalten sein, da es direkt prüft, ob die Anfrage von einem AWS Dienst gestellt wurde, anstatt zu überprüfen, welcher Dienst den letzten Anruf getätigt hat.

Weitere Informationen zu Bedingungsschlüsseln, die in IAM- und Ressourcenrichtlinien verwendet werden, finden Sie unter [IAM-JSON-Richtlinienelemente: Bedingung](#).

CloudTrail Aktualisierungs- und Genehmigungsanforderungen für Amazon SQS Dead-Letter Queue Redrive

Am 8. Juni 2023 führte Amazon SQS Dead-Letter Queue (DLQ) Redrive für AWS SDK und (CLI) ein. AWS Command Line Interface Diese Funktion ist eine Ergänzung zum bereits unterstützten DLQ-Redrive für die Konsole. AWS Wenn Sie die AWS Konsole bereits verwendet haben, um Nachrichten aus der Warteschleife unzustellbarer Nachrichten erneut zu versenden, sind Sie möglicherweise von den folgenden Änderungen betroffen:

CloudTrail Umbenennen von Ereignissen

Am 15. Oktober 2023 ändern sich die Namen der CloudTrail Ereignisse für das Redrive von Dead-Letter-Warteschlangen auf der Amazon SQS SQS-Konsole. Wenn Sie Alarmer für diese CloudTrail Ereignisse eingerichtet haben, müssen Sie sie jetzt aktualisieren. Im Folgenden sind die neuen CloudTrail Ereignisnamen für DLQ Redrive aufgeführt:

Früherer Ereignis-Name	Neuer Ereignis-Name
CreateMoveTask	StartMessageMoveTask
CancelMoveTask	CancelMessageMoveTask

Aktualisierte Berechtigungen

Amazon SQS ist in der SDK- und CLI-Version enthalten und hat außerdem die Warteschlangenberechtigungen für DLQ Redrive aktualisiert, um den bewährten Sicherheitsmethoden zu entsprechen. Verwenden Sie die folgenden Warteschlangenberechtigungsstypen, um Nachrichten von Ihrem DLQs erneut zu senden.

1. Aktionsbasierte Berechtigungen (Update für die DLQ-API-Aktionen)
2. Verwaltete Amazon-SQS-Richtlinienberechtigungen
3. Berechtigungsrichtlinie, die den sqs:*-Platzhalter verwendet

Important

Um DLQ Redrive für SDK oder CLI verwenden zu können, benötigen Sie eine DLQ-Redrive-Berechtigungsrichtlinie, die einer der oben genannten Optionen entspricht.

Wenn Ihre Warteschlangenberechtigungen für DLQ Redrive keiner der oben genannten Optionen entsprechen, müssen Sie Ihre Berechtigungen bis zum 31. August 2023 aktualisieren. Bis zum 31. August 2023 kann Ihr Konto Nachrichten mit den Berechtigungen, die Sie in der AWS -Konsole konfiguriert haben, nur in den Regionen erneut senden, in denen Sie DLQ Redrive zuvor verwendet haben. Nehmen wir zum Beispiel an, Sie hatten „Konto A“ sowohl in us-east-1 als auch eu-west-1. „Account A“ wurde vor dem 8. Juni 2023 verwendet, um Nachrichten auf der AWS Konsole in us-east-1 erneut zu senden, aber nicht in eu-west-1. Wenn die Richtlinienberechtigungen von „Konto A“ zwischen dem 8. Juni 2023 und dem 31. August 2023 keiner der oben genannten Optionen entsprechen, können sie nur verwendet werden, um Nachrichten auf der AWS Konsole in us-east-1 und nicht in eu-west-1 erneut zuzustellen.

⚠ Important

Wenn Ihre DLQ-Redrive-Berechtigungen nach dem 31. August 2023 mit keiner dieser Optionen übereinstimmen, kann Ihr Konto DLQ-Nachrichten nicht mehr über die AWS - Konsole erneut senden.

Wenn Sie jedoch im August 2023 die DLQ-Redrive-Funktion auf der AWS Konsole verwendet haben, haben Sie eine Verlängerung bis zum 15. Oktober 2023, um die neuen Berechtigungen gemäß einer dieser Optionen zu übernehmen.

Weitere Informationen finden Sie unter [the section called "Identifizierung der betroffenen Richtlinien"](#).

Im Folgenden finden Sie Beispiele für Warteschlangenberechtigungen für jede DLQ-Redrive-Option. Bei der Verwendung [serverseitiger verschlüsselter Warteschlangen \(SSE\)](#) ist die entsprechende AWS KMS Schlüsselberechtigung erforderlich.

Aktionsbasiert

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:us-west-1:123456789012:<DLQ_name>"
    },
    {
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-west-1:123456789012:<DestQueue_name>"
    }
  ]
}
```

```
    ]
}
```

Verwaltete Richtlinie

Die folgenden verwalteten Richtlinie enthalten die erforderlichen aktualisierten Berechtigungen:

- **Amazon SQSFull Access** — Beinhaltet die folgenden Aufgaben zur Neuaufstellung der Warteschlange mit unerlaubten Briefen: Starten, Stornieren und Auflisten.
- **Amazon SQSRead OnlyAccess** — Bietet Nur-Lese-Zugriff und beinhaltet die Redrive-Aufgabe für die Warteschlange mit versendeten Briefen.

Step 1

Add permissions

Step 2

Review

Add permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

☐ Add user to group

Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

☐ Copy permissions

Copy all group memberships, attached managed policies, inline policies, and any existing permissions boundaries from an existing user.

☒ Attach policies directly

Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1051)


[Create policy](#)

2 matches

<

1

>



☐	Policy name	Type	Attached entities
☒	AmazonSQSFullAccess	AWS managed	0
☐	AmazonSQSReadOnly...	AWS managed	0

Cancel

Next

Berechtigungsrichtlinie, die den sqs*-Platzhalter verwendet

JSON

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

```
    "Resource": "*"
  }
]
}
```

Identifizierung der betroffenen Richtlinien

Wenn Sie vom Kunden verwaltete Richtlinien (CMPs) verwenden, können Sie AWS CloudTrail und IAM verwenden, um die Richtlinien zu identifizieren, die von der Aktualisierung der Warteschlangenberechtigungen betroffen sind.

Note

Wenn Sie `AmazonSQSFullAccess` und `verwendenAmazonSQSReadOnlyAccess` verwenden, sind keine weiteren Maßnahmen erforderlich.

1. Melden Sie sich bei der Konsole an AWS CloudTrail .
2. Wählen Sie auf der Seite mit dem Ereignisverlauf unter **Attribute nachschlagen** im Dropdownmenü die Option **Ereignisname** aus. Suchen Sie dann nach `CreateMoveTask`.
3. Wählen Sie ein Ereignis, um die Seite **Details** zu öffnen. Rufen Sie im Abschnitt **Ereignisdatensätze** das `UserName` oder `RoleName` aus dem `userIdentity`-ARN ab.
4. Melden Sie sich an der IAM-Konsole an.
 - Wählen Sie für Benutzer „Benutzer“ aus. Wählen Sie den Benutzer mit dem im vorherigen Schritt identifizierten `UserName` aus.
 - Wählen Sie für Rollen „Rollen“ aus. Suchen Sie nach dem Benutzer mit dem im vorherigen Schritt angegebenen `RoleName`.
5. Überprüfen Sie auf der Seite **Details** im Abschnitt **Berechtigungen** alle Richtlinien mit dem `sqs:-`-Präfix in **Action** oder überprüfen Sie Richtlinien, in denen die Amazon-SQS-Warteschlange in **Resource** definiert ist.

Alarme für Warteschlangen mit unbestätigten Briefen mithilfe von Amazon erstellen CloudWatch

Richten Sie mithilfe der Metrik einen CloudWatch Alarm ein, um Nachrichten in einer Warteschlange für unzustellbare Nachrichten zu überwachen. [ApproximateNumberOfMessagesVisible](#)
Detaillierte Anweisungen finden Sie unter [CloudWatch Alarme für Amazon SQS-Metriken erstellen](#).
Wenn der Alarm ausgelöst wird und darauf hinweist, dass Nachrichten in die Warteschlange für unzustellbare Nachrichten verschoben wurden, können Sie die Warteschlange [abfragen](#), um sie zu überprüfen und abzurufen.

Nachrichten-Metadaten für Amazon SQS

Verwenden Sie Nachrichtenattribute, um benutzerdefinierte Metadaten zu Amazon SQS SQS-Nachrichten für Ihre Anwendungen hinzuzufügen. Verwenden Sie Nachrichtensystemattribute, um Metadaten für die Integration mit anderen zu speichern AWS-Services, z. AWS X-Ray

Amazon-SQS-Nachrichtenattribute

Amazon SQS ermöglicht es Ihnen, mithilfe von Nachrichtenattributen strukturierte Metadaten (wie Zeitstempel, Geodaten, Signaturen und Kennungen) in Nachrichten einzufügen. Jede Nachricht kann bis zu 10 Attribute haben. Nachrichtenattribute sind optional und separat vom Nachrichtentext (werden aber mit diesem versendet). Ihr Konsument kann Nachrichtenattribute für die Verarbeitung einer Nachricht auf eine bestimmte Weise verwenden, ohne erst den Nachrichtentext verarbeiten zu müssen. Weitere Informationen über das Senden von Nachrichten mit Attributen mithilfe der Amazon-SQS-Konsole finden Sie unter [Senden einer Nachricht mit Attributen mithilfe von Amazon SQS](#).

Note

Verwechseln Sie Nachrichtenattribute nicht mit Nachrichtensystemattributen: Während Sie Nachrichtenattribute verwenden können, um benutzerdefinierte Metadaten an Amazon SQS SQS-Nachrichten für Ihre Anwendungen anzuhängen, können Sie [Nachrichtensystemattribute](#) verwenden, um Metadaten für andere AWS Dienste zu speichern, z. AWS X-Ray

Themen

- [Nachrichtenattributkomponenten](#)

- [Datentypen für Nachrichtenattribute](#)
- [Berechnung des MD5 Message Digest für Nachrichtenattribute](#)

Nachrichtenattributkomponenten

Important

Alle Komponenten eines Nachrichtenattributs sind in der Beschränkung der Nachrichtengröße von 1 MiB enthalten.

Name, Type, Value und der Nachrichtentext dürfen nicht leer oder null sein.

Jedes Nachrichtenattribut besteht aus den folgenden Komponenten:

- Name – Der Name des Nachrichtenattributs kann die folgenden Zeichen enthalten: A-Z, a-z, 0-9, Unterstrich (`_`), Bindestrich (`-`) und Punkt (`.`). Beachten Sie die folgenden Einschränkungen:
 - Kann bis zu 256 Zeichen lang sein.
 - Darf nicht mit `AWS .` oder `Amazon .` (oder beliebige Varianten in Groß-/Kleinschreibung) beginnen
 - Berücksichtigt Groß- und Kleinschreibung
 - Muss unter allen Attributnamen für die Nachricht eindeutig sein
 - Darf nicht mit einem Punkt beginnen oder enden
 - Darf keine Punkte in einer Sequenz enthalten
- Typ – Der Datentyp des Nachrichtenattributs. Unterstützte Typen sind u. a.: `String`, `Number` und `Binary`. Sie können auch benutzerdefinierte Informationen für alle Datentypen hinzufügen. Der Datentyp weist dieselben Einschränkungen wie der Nachrichtentext auf (weitere Informationen finden Sie unter [SendMessage](#) in der Amazon-Simple-Queue-Service-API-Referenz). Darüber hinaus gelten die folgenden Einschränkungen:
 - Kann bis zu 256 Zeichen lang sein.
 - Berücksichtigt Groß- und Kleinschreibung
- Value – Der Wert des Nachrichtenattributs. Für den Datentyp `String` unterliegen die Attributwerte denselben Beschränkungen wie der Nachrichtentext.

Datentypen für Nachrichtenattribute

Über den Datentyp des Nachrichtenattributs wird Amazon SQS angewiesen, wie die entsprechenden Nachrichtenattributwerte verarbeitet werden. Beispiel: Wenn der Typ `Number` ist, überprüft Amazon SQS numerische Werte.

Amazon SQS unterstützt die logischen Datentypen `String`, `Number` und `Binary`, mit optionalen benutzerdefinierten Datentypenbezeichnungen im Format *`.custom-data-type`*.

- Zeichenfolge – `String`-Attribute können Unicode-Text unter Verwendung beliebiger gültiger XML-Zeichen speichern.
- Zahl – `Number`-Attribute können positive oder negative numerische Werte speichern. Zahlen können bis zu 38 Ziffern und Werte von 10^{-128} bis 10^{+126} umfassen.

Note

Amazon SQS entfernt voranstehende und nachgestellte Nullen.

- Binär – Binäre Attribute können beliebige binäre Daten speichern, wie etwa komprimierte Daten, verschlüsselte Daten oder Bilder.
- Benutzerdefiniert – Hängen Sie eine benutzerdefinierte Typenbezeichnung an jeden unterstützten Datentyp an, um einen benutzerdefinierten Datentyp zu erstellen. Beispiel:
 - `Number.byte`, `Number.short`, `Number.int` und `Number.float` können beim Unterscheiden zwischen den Zahlentypen helfen.
 - `Binary.gif` und `Binary.png` können beim Unterscheiden zwischen den Dateitypen helfen.

Note

Amazon SQS interpretiert, validiert oder verwendet die angefügten Daten nicht. Die benutzerdefinierte Typenbezeichnung unterliegt denselben Beschränkungen wie der Nachrichtentext.

Berechnung des MD5 Message Digest für Nachrichtenattribute

Wenn Sie den verwenden AWS SDK für Java, können Sie diesen Abschnitt überspringen. Die `MessageMD5ChecksumHandler` Klasse des SDK for Java unterstützt MD5 Message Digests für Amazon SQS SQS-Nachrichtenattribute.

Wenn Sie entweder die Query API oder eine der APIs verwenden AWS SDKs , die keine MD5 Message Digests für Amazon SQS SQS-Nachrichtenattribute unterstützt, müssen Sie die folgenden Richtlinien verwenden, um die MD5 Message Digest-Berechnung durchzuführen.

Note

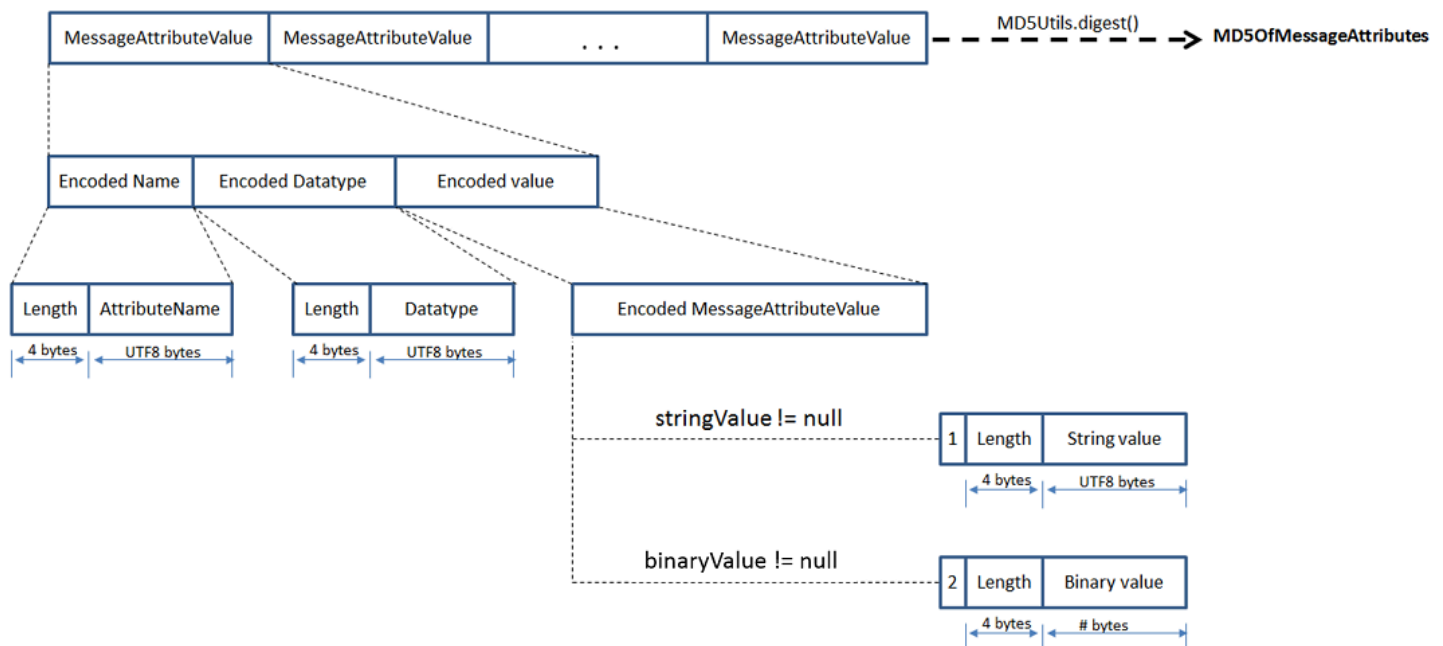
Schließen Sie immer benutzerdefinierte Datentyp-Suffixe in die Message-Digest-Berechnung ein. MD5

-Übersicht

Im Folgenden finden Sie eine Übersicht über den Algorithmus zur Berechnung des MD5 Message Digests:

1. Er sortiert alle Nachrichtenattribute in aufsteigender Reihenfolge nach ihrem Namen.
2. Er verschlüsselt die einzelnen Teile der Attribute (Name, Type und Value) im Puffer.
3. Er berechnet den Nachrichtendigest des gesamten Puffers.

Das folgende Diagramm zeigt die Kodierung des MD5 Message Digest für ein einzelnes Nachrichtenattribut:



So verschlüsseln Sie ein einzelnes SQS-Nachrichtenattribut

1. Verschlüsselung des Namens: Länge (4 Bytes) und UTF-8-Bytes des Namens.
2. Verschlüsselung des Datentyps: Länge (4 Bytes) und UTF-8-Bytes des Datentyps.
3. Verschlüsselung des Übertragungstyps (String oder Binary) des Werts (1 Byte).

Note

Für die logischen Datentypen `String` und `Number` wird der Übertragungstyp `String` verwendet.

Für den logischen Datentyp `Binary` wird der Übertragungstyp `Binary` verwendet.

- a. Verwenden Sie für den Transporttyp `String` die Verschlüsselung 1.
- b. Verwenden Sie für den Transporttyp `Binary` die Verschlüsselung 2.
4. Verschlüsselung des Attributwerts
 - a. Verschlüsselung des Attributwerts für den Übertragungstyp `String`: Länge (4 Bytes) und UTF-8 Bytes des Werts.
 - b. Verschlüsselung des Attributwerts für den Übertragungstyp `Binary`: Länge (4 Bytes) und Rohbytes des Werts.

Attribute des Amazon-SQS-Nachrichtensystems

Während Sie [Nachrichtenattribute](#) verwenden können, um Amazon-SQS-Nachrichten benutzerdefinierte Metadaten für Ihre Anwendungen anzufügen, können Sie Nachrichtensystemattribute verwenden, um Metadaten für andere AWS -Services wie AWS X-Ray zu speichern. Für weitere Informationen siehe den `MessageSystemAttribute`-Anforderungsparameter der API-Aktionen [SendMessage](#) und [SendMessageBatch](#), das `AWSTraceHeader`-Attribut der API-Aktion [ReceiveMessage](#) sowie den [MessageSystemAttributeValue](#)-Datentyp in der Amazon-Simple-Queue-Service-API-Referenz.

Nachrichtensystemattribute sind genau wie Nachrichtenattribute strukturiert, mit folgenden Ausnahmen:

- Derzeit wird `AWSTraceHeader` als einziges Nachrichtensystemattribut unterstützt. Sein Typ muss sein `String` und sein Wert muss eine korrekt formatierte AWS X-Ray Trace-Header-Zeichenfolge sein.
- Die Größe eines Nachrichtensystemattributs zählt nicht zur Gesamtgröße einer Nachricht.

Für die Verarbeitung von Amazon-SQS-Nachrichten erforderliche Ressourcen

Amazon SQS bietet Schätzungen der ungefähren Anzahl verzögerter, sichtbarer und nicht sichtbarer Nachrichten in einer Warteschlange, um Ihnen bei der Einschätzung der für die Verarbeitung benötigten Ressourcen zu helfen. Weitere Informationen zur Sichtbarkeit finden Sie unter [Amazon-SQS-Zeitbeschränkung für die Sichtbarkeit](#).

Note

Bei einigen Kennzahlen handelt es sich aufgrund der verteilten Architektur von Amazon SQS um ungefähre Ergebnisse. In den meisten Fällen sollte die Anzahl nahe an der tatsächlichen Anzahl von Nachrichten in der Warteschlange liegen.

Die folgende Tabelle enthält die Attributnamen, die mit der [GetQueueAttributes](#)-Aktion verwendet werden.

Aufgabe	Attributname
Abrufen der ungefähren Anzahl der Nachrichten, die von der Warteschlange abgerufen werden können.	<code>ApproximateNumberOfMessagesVisible</code>
Abrufen der ungefähren Anzahl der Nachrichten in der Warteschlange, die verzögert und zum Lesen nicht sofort verfügbar sind. Dies kann vorkommen, wenn die Warteschlange als Verzögerungswarteschlange konfiguriert ist oder eine Mitteilung mit einem Verzögerungsparameter gesendet worden ist.	<code>ApproximateNumberOfMessagesDelayed</code>

Aufgabe	Attributname
Abrufen der ungefähren Anzahl der in Übertragung befindlichen Mitteilungen. Die Mitteilungen befinden sich in Übertragung, wenn sie an einen Client gesendet, aber noch nicht gelöscht worden sind oder noch nicht das Ende ihres Sichtbarkeitsfensters erreicht haben.	<code>ApproximateNumberOfMessages</code> <code>NotVisible</code>

Paginierung der Amazon SQS SQS-Listenwarteschlangen

Die API-Methoden [listQueues](#) und [listDeadLetterQueues](#) unterstützen optionale Steuerelemente für die Paginierung. Standardmäßig geben diese API-Methoden bis zu 1 000 Warteschlangen in der Antwortnachricht zurück. Sie können den `MaxResults`-Parameter so einstellen, dass bei jeder Antwort weniger Ergebnisse zurückgegeben werden.

Setzen Sie den Parameter `MaxResults` in der Anforderung `listQueues` oder `listDeadLetterQueues`, um die maximale Anzahl von Ergebnissen anzugeben, die in der Antwort zurückgegeben werden sollen. Wenn Sie `MaxResults` nicht festlegen, enthält die Antwort maximal 1 000 Ergebnisse und der `NextToken`-Wert in der Antwort ist Null.

Wenn Sie `MaxResults` festlegen, enthält die Antwort einen Wert für `NextToken`, wenn weitere Ergebnisse zur Anzeige vorhanden sind. Verwenden Sie `NextToken` als Parameter in Ihrer nächsten Anforderung an `listQueues`, um die nächste Ergebnisseite zu erhalten. Wenn keine weiteren Ergebnisse zur Anzeige vorhanden sind, ist der `NextToken`-Wert in der Antwort Null.

Amazon-SQS-Kostenzuordnungs-Tags

Zum Strukturieren und Identifizieren Ihrer Amazon-SQS-Warteschlangen für die Kostenzuordnung können Sie Metadaten-Tags hinzufügen, die den Zweck, den Eigentümer oder die Umgebung einer Warteschlange identifizieren. Dies ist vor allem nützlich, wenn Sie viele Warteschlangen haben. Zur Konfiguration von Tags mit der Amazon-SQS-Konsole siehe [the section called “Konfigurieren von Tags für eine Warteschlange”](#)

Mithilfe von Tags zur Kostenzuweisung können Sie Ihre AWS Rechnung so organisieren, dass sie Ihrer eigenen Kostenstruktur entspricht. Melden Sie sich dazu an, damit Ihre AWS-Konto Rechnung

die Tagschlüssel und -werte enthält. Weitere Informationen finden Sie unter [Einrichten Ihres monatlichen Kostenzuordnungsberichts](#) im AWS Billing Benutzerhandbuch.

Jedes Tag besteht aus einem Schlüssel-Wert-Paar, das Sie definieren. Beispielsweise können Sie auf einfache Weise Ihre Produktions- und Test- Warteschlangen identifizieren, wenn Sie Ihre Warteschlangen wie folgt kennzeichnen:

Warteschlange	Key (Schlüssel)	Value (Wert)
MyQueueA	QueueType	Production
MyQueueB	QueueType	Testing

Note

Beachten Sie bei der Verwendung von Warteschlangen-Tags die folgenden Richtlinien:

- Es wird nicht empfohlen, einer Warteschlange mehr als 50 Tags hinzuzufügen. Tagging unterstützt Unicode-Zeichen in UTF-8.
- Tags haben keine semantische Bedeutung. Amazon SQS interpretiert Tags als Zeichenfolgen.
- Bei Tags muss die Groß- und Kleinschreibung beachtet werden.
- Ein neuer Tag mit einem Schlüssel, der mit dem eines vorhandenen Tags identisch ist, überschreibt den vorhandenen Tag.
- Tagging-Aktionen sind auf 30 TPS pro Tag begrenzt. AWS-Konto Wenn Ihre Anwendung einen höheren Durchsatz erfordert, [reichen Sie eine Anfrage ein](#).

Eine vollständige Liste von Tag-Einschränkungen finden Sie unter [Amazon SQS SQS-Standard-Warteschlangenkontingente](#).

Kurz- und Langabfragen in Amazon SQS

Amazon SQS bietet kurze und lange Abfrageoptionen für den Empfang von Nachrichten aus einer Warteschlange. Berücksichtigen Sie die Anforderungen Ihrer Anwendung an Reaktionsfähigkeit und Kosteneffizienz, wenn Sie zwischen diesen beiden Abfrageoptionen wählen:

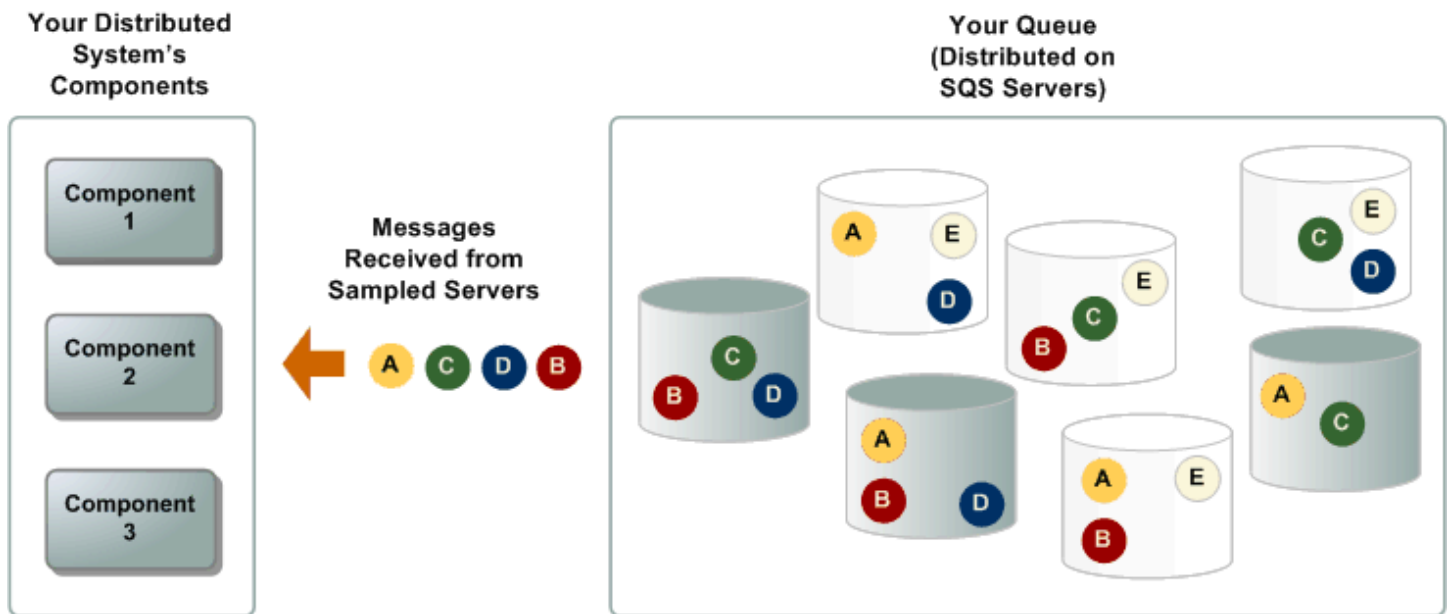
- Kurzes Polling (Standard) — Die [ReceiveMessage](#)Anfrage fragt eine Untergruppe von Servern ab (basierend auf einer gewichteten Zufallsverteilung), um verfügbare Nachrichten zu finden, und sendet eine sofortige Antwort, auch wenn keine Nachrichten gefunden wurden.
- Langes Polling — [ReceiveMessage](#)fragt alle Server nach Nachrichten ab und sendet bis zum angegebenen Höchstwert eine Antwort, sobald mindestens eine Nachricht verfügbar ist. Eine leere Antwort wird nur gesendet, wenn die Wartezeit für die Abfrage abgelaufen ist. Diese Option kann die Anzahl leerer Antworten reduzieren und möglicherweise die Kosten senken.

In den folgenden Abschnitten werden die Details von Kurz- und Langabfragen erläutert.

Abrufen von Nachrichten durch Kurzabfragen

Wenn Sie Nachrichten aus einer Warteschlange (FIFO oder Standard) mithilfe von Kurzabfragen verarbeiten, nimmt Amazon SQS eine Stichprobe von einer Teilmenge seiner Server (basierend auf einer gewichteten Zufallsverteilung) vor und gibt nur Nachrichten von diesen Servern zurück. Daher werden für eine bestimmte [ReceiveMessage](#)-Anforderung möglicherweise nicht alle Nachrichten zurückgegeben. Wenn Sie jedoch weniger als 1000 Nachrichten in Ihrer Warteschlange haben, gibt die nächste Anfrage Ihre Nachrichten zurück. Wenn Sie das Abrufen aus Ihren Warteschlangen fortsetzen, nimmt Amazon SQS Stichproben aller seiner Server und Sie erhalten alle Ihre Nachrichten.

Das folgende Diagramm zeigt das Verhalten von Nachrichten bei Kurzabfragen, die von einer Standard-Warteschlange zurückgegeben werden, nachdem eine Ihrer Systemkomponenten eine Empfangsanforderung stellt. Amazon SQS nimmt Stichproben von mehreren seiner Server (grau dargestellt) und gibt die Nachrichten A, C, D und B von diesen Servern zurück. Nachricht E wird nicht für diese Anforderung zurückgegeben, sondern für eine nachfolgende Anforderung.



Konsumieren von Nachrichten mithilfe von Langabfragen

Ist die Wartezeit für die [ReceiveMessage](#)-API-Aktion größer als 0, ist eine lange Abfrage wirksam. Die maximale Wartezeit für lange Abfragen beträgt 20 Sekunden. Lange Abfragen tragen dazu bei, die Kosten für die Nutzung von Amazon SQS zu senken, indem die Anzahl der leeren Antworten (wenn für eine `ReceiveMessage` Anfrage keine Nachrichten verfügbar sind) und der falschen leeren Antworten (wenn Nachrichten verfügbar sind, aber nicht in einer Antwort enthalten sind) reduziert wird. Informationen zum Aktivieren von Langabfragen für eine neue oder vorhandene Warteschlange mithilfe der Amazon-SQS-Konsole finden Sie unter [Konfiguration von Warteschlangenparametern mit der Amazon SQS SQS-Konsole](#). Bewährte Methoden finden Sie unter [Einrichtung von Long Polling in Amazon SQS](#).

Lange Abfragen bieten die folgenden Vorteile:

- Eliminieren leerer Antworten, da Amazon SQS wartet, bis eine Nachricht in der Warteschlange vorhanden ist, bevor eine Antwort gesendet wird. Sofern die Verbindung nicht abläuft, enthält die Antwort auf die `ReceiveMessage`-Anfrage mindestens eine der verfügbaren Nachrichten bis zur in der Aktion `ReceiveMessage` definierten maximalen Anzahl an Nachrichten. In seltenen Fällen erhalten Sie möglicherweise leere Antworten, auch wenn eine Warteschlange noch Nachrichten enthält, insbesondere wenn Sie einen niedrigen Wert für den [ReceiveMessageWaitTimeSeconds](#)-Parameter angeben.
- Reduzieren Sie falsche Leerantworten, indem Sie alle – nicht nur eine Teilmenge – von Amazon-SQS-Servern abfragen.

- Zurückgeben von Nachrichten, sobald sie verfügbar werden.

Informationen darüber, wie Sie überprüfen, ob eine Warteschlange leer ist, finden Sie unter [Bestätigen, dass eine Amazon SQS SQS-Warteschlange leer ist](#).

Unterschiede zwischen Lang- und Kurzabfragen

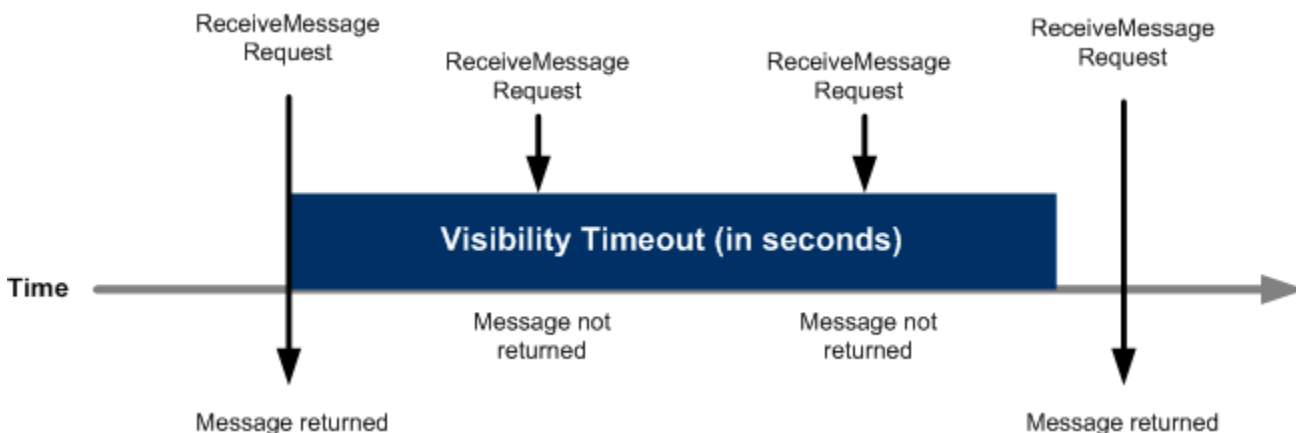
Kurzabfragen werden ausgeführt, wenn der Parameter [WaitTimeSeconds](#) einer [ReceiveMessage](#)-Anfrage mit einer der beiden folgenden Methoden auf den Wert 0 festgelegt wurde:

- Der `ReceiveMessage`-Aufruf legt `WaitTimeSeconds` auf 0 fest.
- Der `ReceiveMessage`-Aufruf legt `WaitTimeSeconds` nicht fest, aber das Warteschlangenattribut [ReceiveMessageWaitTimeSeconds](#) ist auf 0 festgelegt.

Amazon-SQS-Zeitbeschränkung für die Sichtbarkeit

Wenn Sie eine Nachricht aus einer Amazon SQS SQS-Warteschlange erhalten, verbleibt sie in der Warteschlange, wird aber vorübergehend für andere Verbraucher unsichtbar. Diese Unsichtbarkeit wird durch das Sichtbarkeits-Timeout gesteuert, das sicherstellt, dass andere Verbraucher dieselbe Nachricht nicht verarbeiten können, während Sie daran arbeiten. Amazon SQS bietet zwei Optionen zum Löschen von Nachrichten nach der Verarbeitung:

- Manuelles Löschen — Sie löschen Nachrichten explizit mithilfe der [DeleteMessage](#)-Aktion.
- Automatisches Löschen — In bestimmten AWS SDKs Fällen werden Nachrichten bei erfolgreicher Verarbeitung automatisch gelöscht, wodurch Arbeitsabläufe vereinfacht werden.



Anwendungsfälle mit Timeout bei der Sichtbarkeit

Lang andauernde Aufgaben verwalten — Verwenden Sie das Sichtbarkeits-Timeout, um Aufgaben zu bearbeiten, die längere Verarbeitungszeiten erfordern. Legen Sie ein angemessenes Sichtbarkeits-Timeout für Nachrichten fest, die eine längere Verarbeitungszeit erfordern. Dadurch wird sichergestellt, dass andere Benutzer dieselbe Nachricht während der Bearbeitung nicht aufnehmen, wodurch Doppelarbeit vermieden und die Systemeffizienz erhalten bleibt.

Implementieren Sie Wiederholungsmechanismen — Verlängern Sie das Sichtbarkeits-Timeout programmgesteuert für Aufgaben, die nicht innerhalb des ursprünglichen Timeouts abgeschlossen werden können. Wenn eine Aufgabe nicht innerhalb des anfänglichen Sichtbarkeits-Timeouts abgeschlossen werden kann, können Sie das Timeout programmgesteuert verlängern. Auf diese Weise kann Ihr System erneut versuchen, die Nachricht zu verarbeiten, ohne dass sie für andere Benutzer sichtbar wird, was die Fehlertoleranz und Zuverlässigkeit verbessert. Kombinieren Sie es mit Dead-Letter-Warteschlangen (DLQs), um anhaltende Fehler zu beheben.

Verteilte Systeme koordinieren — Verwenden Sie das SQS-Sichtbarkeits-Timeout, um Aufgaben in verteilten Systemen zu koordinieren. Legen Sie Zeitlimits für die Sichtbarkeit fest, die Ihren erwarteten Verarbeitungszeiten für verschiedene Komponenten entsprechen. Dies trägt zur Aufrechterhaltung der Konsistenz bei und verhindert Konkurrenzprobleme in komplexen, verteilten Architekturen.

Optimieren Sie die Ressourcennutzung — Passen Sie die SQS-Sichtbarkeits-Timeouts an, um die Ressourcennutzung in Ihrer Anwendung zu optimieren. Durch die Festlegung geeigneter Timeouts können Sie sicherstellen, dass Nachrichten effizient verarbeitet werden, ohne unnötig Ressourcen zu beanspruchen. Dies führt zu einer besseren Gesamtleistung und Kosteneffektivität des Systems.

Einstellung und Anpassung des Sichtbarkeits-Timeouts

Das Sichtbarkeits-Timeout beginnt, sobald Ihnen eine Nachricht zugestellt wird. Während dieses Zeitraums wird von Ihnen erwartet, dass Sie die Nachricht verarbeiten und löschen. Wenn Sie sie nicht vor Ablauf des Timeouts löschen, wird die Nachricht wieder in der Warteschlange sichtbar und kann von einem anderen Verbraucher abgerufen werden. Das standardmäßige Sichtbarkeits-Timeout für eine Warteschlange beträgt 30 Sekunden. Sie können dieses Zeitlimit jedoch an die Zeit anpassen, die Ihre Anwendung zum Verarbeiten und Löschen einer Nachricht benötigt. Sie können auch ein bestimmtes Sichtbarkeits-Timeout für einzelne Nachrichten festlegen, ohne die Gesamteinstellung der Warteschlange zu ändern. Verwenden Sie die [ChangeMessageVisibility](#)Aktion, um das Timeout nach Bedarf programmgesteuert zu verlängern oder zu verkürzen.

In Flugnachrichten und Kontingenten

In Amazon SQS sind Bordnachrichten Nachrichten, die von einem Verbraucher empfangen, aber noch nicht gelöscht wurden. Für Standardwarteschlangen gibt es ein Limit von etwa 120.000 Nachrichten während der Übertragung, abhängig vom Warteschlangenverkehr und dem Nachrichtenrückstand. Wenn Sie dieses Limit erreichen, gibt Amazon SQS eine `OverLimit` Fehlermeldung zurück, die darauf hinweist, dass keine weiteren Nachrichten empfangen werden können, bis einige Bordnachrichten gelöscht wurden. Bei FIFO-Warteschlangen hängen die Grenzwerte von den aktiven Nachrichtengruppen ab.

- Bei Verwendung von Kurzabfragen — Wenn dieses Limit bei der Verwendung von Kurzabfragen erreicht wird, gibt Amazon SQS einen `OverLimit` Fehler zurück, der darauf hinweist, dass keine weiteren Nachrichten empfangen werden können, bis einige Bordnachrichten gelöscht wurden.
- Wenn Sie Long Polling verwenden — Wenn Sie Long Polling verwenden, gibt Amazon SQS keinen Fehler zurück, wenn das Limit für eingehende Nachrichten erreicht ist. Stattdessen werden keine neuen Nachrichten zurückgegeben, bis die Anzahl der Nachrichten während der Übertragung unter das Limit fällt.

Gehen Sie wie folgt vor, um Bordnachrichten effektiv zu verwalten:

1. Sofortiges Löschen — Löschen Sie Nachrichten (manuell oder automatisch) nach der Verarbeitung, um die Anzahl während des Fluges zu reduzieren.
2. Überwachen mit CloudWatch — Stellen Sie Alarme für hohe Flugzahlen ein, um zu verhindern, dass das Limit erreicht wird.
3. Last verteilen — Wenn Sie eine große Menge an Nachrichten verarbeiten, sollten Sie zusätzliche Warteschlangen oder Verbraucher verwenden, um die Auslastung auszugleichen und Engpässe zu vermeiden.
4. Eine Erhöhung des Kontingents beantragen — Senden Sie eine Anfrage an den [AWS Support](#), wenn höhere Limits erforderlich sind.

Grundlegendes zum Sichtbarkeits-Timeout in Standard- und FIFO-Warteschlangen

Sowohl in Standard- als auch in FIFO-Warteschlangen (First-In-First-Out) verhindert das Sichtbarkeits-Timeout, dass mehrere Benutzer dieselbe Nachricht gleichzeitig verarbeiten. Aufgrund

des at-least-once Zustellungsmodells von Amazon SQS gibt es jedoch keine absolute Garantie dafür, dass eine Nachricht während des Sichtbarkeits-Timeouts nicht mehr als einmal zugestellt wird.

- **Standardwarteschlangen** — Das Sichtbarkeits-Timeout in Standardwarteschlangen verhindert, dass mehrere Verbraucher dieselbe Nachricht gleichzeitig verarbeiten. Aufgrund des at-least-once Zustellungsmodells garantiert Amazon SQS jedoch nicht, dass eine Nachricht innerhalb des Sichtbarkeits-Timeouts nicht mehr als einmal zugestellt wird.
- **FIFO-Warteschlangen** — Bei FIFO-Warteschlangen werden Nachrichten mit derselben Nachrichtengruppen-ID in einer strikten Reihenfolge verarbeitet. Wenn sich eine Nachricht mit einer Nachrichtengruppen-ID in Bearbeitung befindet, werden nachfolgende Nachrichten in dieser Gruppe erst verfügbar gemacht, wenn die aktuelle Nachricht entweder gelöscht wird oder das Sichtbarkeits-Timeout abgelaufen ist. Dadurch wird die Gruppe jedoch nicht auf unbestimmte Zeit „gesperrt“ — jede Nachricht wird nacheinander verarbeitet, und nur wenn jede Nachricht gelöscht wird oder wieder sichtbar wird, ist die nächste Nachricht in dieser Gruppe für Verbraucher verfügbar. Dieser Ansatz gewährleistet eine geordnete Verarbeitung innerhalb der Gruppe, ohne die Gruppe unnötig an der Zustellung von Nachrichten zu hindern.

Umgang mit Fehlern

Wenn Sie eine Nachricht nicht verarbeiten und löschen, bevor das Sichtbarkeits-Timeout abgelaufen ist — aufgrund von Anwendungsfehlern, Abstürzen oder Verbindungsproblemen —, wird die Nachricht wieder in der Warteschlange sichtbar. Die Nachricht kann dann von demselben oder einem anderen Benutzer für einen weiteren Verarbeitungsversuch abgerufen werden. Dadurch wird sichergestellt, dass Nachrichten auch dann nicht verloren gehen, wenn die erste Verarbeitung fehlschlägt. Ein zu hohes Sichtbarkeits-Timeout kann jedoch dazu führen, dass unverarbeitete Nachrichten erneut angezeigt werden, was wiederum zu einer Verlangsamung der Wiederholungsversuche führen kann. Es ist wichtig, ein angemessenes Sichtbarkeits-Timeout festzulegen, das auf der erwarteten Verarbeitungszeit basiert, damit Nachrichten rechtzeitig bearbeitet werden können.

Änderung und Beendigung des Sichtbarkeits-Timeouts

Sie können das Sichtbarkeits-Timeout mit der folgenden `ChangeMessageVisibility` Aktion ändern oder beenden:

- **Das Timeout ändern** — Passen Sie das Sichtbarkeits-Timeout dynamisch an, indem Sie [ChangeMessageVisibility](#). Auf diese Weise können Sie die Timeout-Dauern verlängern oder reduzieren, um sie an die Verarbeitungsanforderungen anzupassen.
- **Beenden des Timeouts** — Wenn Sie sich entscheiden, eine empfangene Nachricht nicht zu verarbeiten, beenden Sie das Sichtbarkeits-Timeout, indem Sie den Wert während der Aktion `VisibilityTimeout` auf 0 Sekunden setzen. `ChangeMessageVisibility`. Dadurch steht die Nachricht sofort anderen Benutzern zur Verarbeitung zur Verfügung.

Best Practices

Verwenden Sie die folgenden bewährten Methoden zur Verwaltung von Sichtbarkeits-Timeouts in Amazon SQS, einschließlich der Festlegung, Anpassung und Verlängerung von Timeouts sowie der Behandlung unbearbeiteter Nachrichten mithilfe von Dead-Letter-Warteschlangen (DLQs).

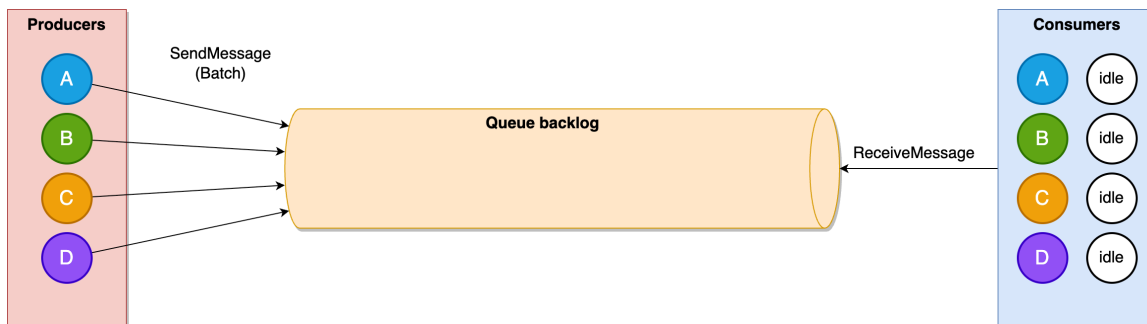
- **Einstellung und Anpassung des Timeouts.** Stellen Sie zunächst das Sichtbarkeits-Timeout so ein, dass es der maximalen Zeit entspricht, die Ihre Anwendung normalerweise zum Verarbeiten und Löschen einer Nachricht benötigt. Wenn Sie sich über die genaue Bearbeitungszeit nicht sicher sind, beginnen Sie mit einem kürzeren Timeout (z. B. 2 Minuten) und verlängern Sie es nach Bedarf. Implementieren Sie einen Heartbeat-Mechanismus, um das Sichtbarkeits-Timeout regelmäßig zu verlängern und so sicherzustellen, dass die Nachricht unsichtbar bleibt, bis die Verarbeitung abgeschlossen ist. Dadurch werden Verzögerungen bei der erneuten Verarbeitung unbearbeiteter Nachrichten minimiert und eine vorzeitige Sichtbarkeit verhindert.
- **Verlängerung des Timeouts und Handhabung des 12-Stunden-Limits.** Wenn Ihre Verarbeitungszeit variiert oder das ursprünglich festgelegte Timeout überschreiten könnte, verwenden Sie die `ChangeMessageVisibility` Aktion, um das Sichtbarkeits-Timeout während der Verarbeitung der Nachricht zu verlängern. Beachten Sie, dass das Sichtbarkeits-Timeout eine maximale Grenze von 12 Stunden ab dem ersten Empfang der Nachricht hat. Durch eine Verlängerung des Timeouts wird dieses 12-Stunden-Limit nicht zurückgesetzt. Wenn Ihre Bearbeitung mehr Zeit als dieses Limit in Anspruch nimmt, sollten Sie erwägen, die Aufgabe zu verwenden AWS Step Functions oder in kleinere Schritte zu unterteilen.
- **Umgang mit unverarbeiteten Nachrichten.** Um Nachrichten zu verwalten, bei denen mehrere Verarbeitungsversuche fehlschlagen, konfigurieren Sie eine Dead-Letter-Warteschlange (DLQ). Auf diese Weise wird sichergestellt, dass Nachrichten, die nach mehreren Wiederholungen nicht verarbeitet werden können, zur weiteren Analyse oder Bearbeitung getrennt erfasst werden. Dadurch wird verhindert, dass sie wiederholt in der Hauptwarteschlange zirkulieren.

Amazon SQS: Faire Warteschlangen

Amazon SQS Fair Queues mindert automatisch die Auswirkungen von Noisy-Neighbor-Störungen in Multi-Tenant-Warteschlangen, die Nachrichten von mehreren logischen Entitäten wie Kunden, Client-Anwendungen oder Nachrichtentypen enthalten. In diesen Umgebungen mit gemeinsam genutzten Warteschlangen ist die Verweilzeit eine wichtige Leistungskennzahl. Dabei wird die Gesamtzeit gemessen, die Nachrichten von der Ankunft bis zur Verarbeitung in einer Warteschlange verbringen. Wenn ein Mandant einen Rückstand in der Warteschlange erzeugt, indem er mehr Nachrichten veröffentlicht, als das System verarbeiten kann, minimieren faire Warteschlangen die Auswirkungen auf die Verweildauer anderer Mandanten.

Stetiger Zustand

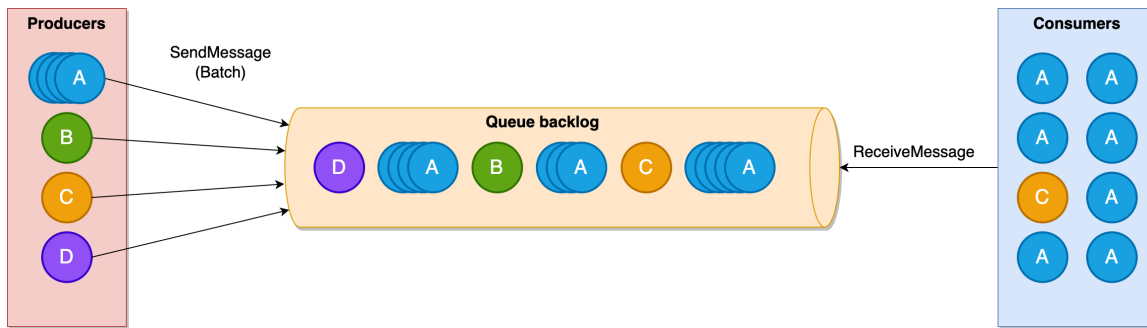
Das folgende Diagramm zeigt eine Warteschlange mit mehreren Mandanten, die Nachrichten von vier verschiedenen Mandanten (mit A, B, C und D bezeichnet) enthält. Die Warteschlange arbeitet in einem stabilen Zustand, und es gibt keinen Nachrichtenrückstand, da Verbraucher Nachrichten erhalten, sobald sie in der Warteschlange erscheinen. Alle Mieter haben geringe Verweilzeiten. In diesem stabilen Zustand sind nicht alle Verbraucherkapazitäten voll ausgelastet.



Lärmbelästigung durch die Nachbarn

Störende Nebengeräusche treten auf, wenn ein Mandant in einer Warteschlange mit mehreren Mandanten einen Rückstand erzeugt, wodurch die Verweildauer der Nachrichten für alle anderen Mandanten verlängert wird. Ein Mandant kann zu einem lauten Nachbarn werden, wenn er eine größere Menge an Nachrichten sendet als andere Mieter oder wenn Verbraucher länger brauchen, um Nachrichten von diesem bestimmten Mandanten zu verarbeiten.

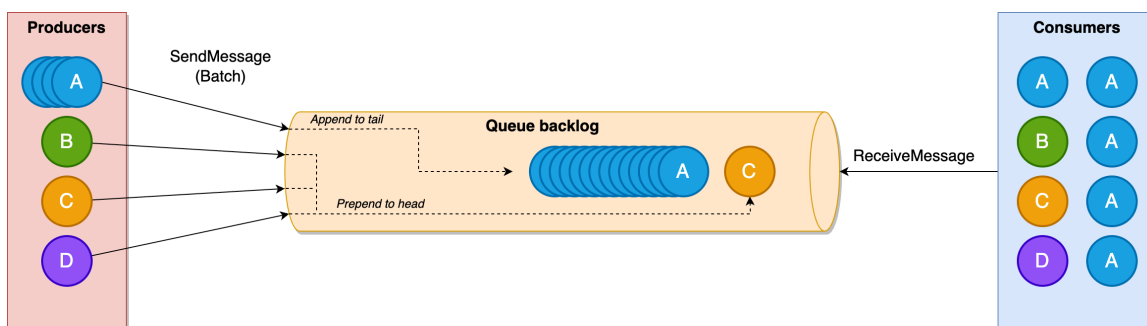
Dieses Diagramm zeigt, wie der erhöhte Verkehr von Mandant A zu einem Rückstau in der Warteschlange führt. Die Verbraucher sind damit beschäftigt, nur die Nachrichten von Mandant A zu verarbeiten, während Nachrichten von anderen Mandanten im Backlog warten, was zu längeren Verweilzeiten für alle Mandanten führt.



Schadensbegrenzung durch faire Warteschlangen

Amazon SQS erkennt störende Nachbarn, indem es die Nachrichtenverteilung zwischen den Mandanten während der Verarbeitung überwacht (Status „während der Übertragung“). Wenn ein Mieter im Vergleich zu anderen eine unverhältnismäßig große Anzahl an Nachrichten während des Fluges hat, identifiziert Amazon SQS diesen Mandanten als lauten Nachbarn und priorisiert die Nachrichtenzustellung für andere Mieter. Dieser Ansatz reduziert die Auswirkungen auf die Verweildauer der anderen Mieter.

Dieses Diagramm zeigt, wie Amazon SQS Fair Queues das Problem Noisy Neighbor löst. Wenn ein Mandant (Mandant A) laut wird, priorisiert Amazon SQS die Rückgabe von Nachrichten von anderen Mandanten (B, C und D). Diese Priorisierung trägt dazu bei, niedrige Verweilzeiten für stille Mieter (Mandanten B, C und D) aufrechtzuerhalten, während die Verweildauer für die Nachrichten von Mandant A erhöht wird, bis der Warteschlangenrückstand aufgebraucht ist, ohne dass sich dies auf andere Mieter auswirkt.



Note

Amazon SQS begrenzt die Nutzungsrate pro Mandant nicht. Es ermöglicht Verbrauchern, Nachrichten von lauten Nachbarn zu empfangen, wenn genügend Kapazitäten vorhanden sind und in der Warteschlange keine weiteren Nachrichten zurückgesendet werden können. Wie Amazon SQS SQS-Standardwarteschlangen ermöglichen faire

Warteschlangen einen praktisch unbegrenzten Durchsatz, und die Anzahl der Mandanten, die Sie in Ihrer Warteschlange haben könnten, ist unbegrenzt.

Einzelheiten dazu, wie Amazon SQS störende Nachbarn erkennt und die Reihenfolge der Nachrichtenzustellung verwaltet, finden Sie unter [So funktionieren faire Warteschlangen in Amazon SQS](#).

Wie verwendet man faire Warteschlangen

Um faire Warteschlangen zu ermöglichen, sollten Nachrichtenproduzenten eine Mandanten-ID hinzufügen, indem sie bei ausgehenden Nachrichten einen `MessageGroupId` Wert setzen:

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

Die Fairness-Funktion wird automatisch in allen Amazon SQS SQS-Standardwarteschlangen für Nachrichten mit der Eigenschaft angewendet. `MessageGroupId` Sie erfordert keine Änderung des Verbraucher-codes, sie hat keine Auswirkungen auf die API-Latenz und ist nicht mit Durchsatzbeschränkungen verbunden.

Note

`MessageGroupId` hat bei Standardwarteschlangen mit fairen Warteschlangen nicht das gleiche Verhalten wie `MessageGroupId` bei FIFO-Warteschlangen. `MessageGroupId` wird in Standardwarteschlangen nur als Mandanten-ID für faire Warteschlangen verwendet und erzwingt keine Nachrichtenreihenfolge. Einzelheiten zu `MessageGroupId` FIFO-Warteschlangen finden Sie unter [Verwenden der Nachrichtengruppen-ID mit Amazon SQS FIFO-Warteschlangen](#).

Wann sollten faire Warteschlangen verwendet werden

Erwägen Sie, faire Warteschlangen zu verwenden, wenn alle der folgenden Punkte auf Ihre Warteschlange zutreffen:

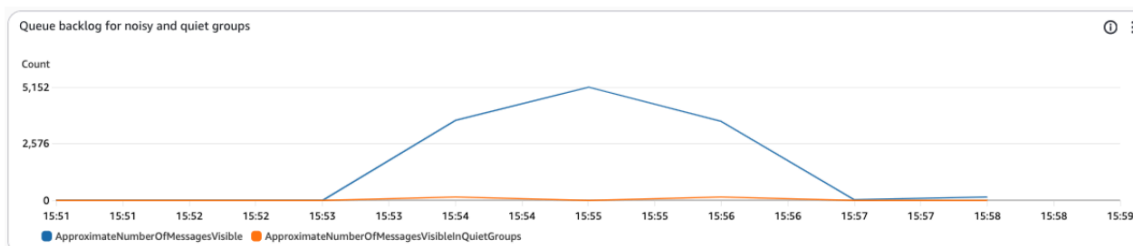
- Die Warteschlange ist mandantenfähig. Die Warteschlange enthält Nachrichten, die zu mehreren logischen Entitäten gehören, z. B. Kunden, Client-Anwendungen oder Anforderungstypen, und Sie können jede Entität mit einem `MessageGroupId` identifizieren.
- Die Warteschlange hat einen hohen Durchsatz. Bei geringem Durchsatz führt der Burst eines Mandanten selten zu einem Backlog, der sich auf andere Mandanten auswirkt. Bei hohem Durchsatz ist es nicht praktikabel, die Kundenflotte zu stark zu skalieren, um jeden Anstieg zu absorbieren, und selbst bei auto-scaling kann die Verzögerung, bis neue Verbraucher online gehen, zu Rückstaus führen.
- Die Verweildauer ist Teil der Servicequalität Ihrer Anwendung. Faire Warteschlangen schützen ruhige Mieter vor längeren Verweilzeiten, die durch einen lauten Nachbarn verursacht werden. Wenn Ihre Anwendung nicht empfindlich auf die Verweildauer reagiert, ist der von Fair Queues bereitgestellte Schutz vor störenden Nachbarn möglicherweise nicht erforderlich.

Messwerte für faire Warteschlangen CloudWatch

Amazon SQS bietet zusätzliche CloudWatch Metriken, mit denen Sie die Minderung der Auswirkungen von Störgeräuschen auf Ihre Nachbarn überwachen können. Beispielsweise können Sie `ApproximateNumberOfMessagesVisibleInQuietGroups` Metriken mit Standardmetriken auf Warteschlangenebene vergleichen. Bei einem Anstieg des Datenverkehrs für einen bestimmten Mandanten können die allgemeinen Kennzahlen auf Warteschlangenebene auf steigende Rückstände oder ein älteres Nachrichtenalter hinweisen. Wenn Sie sich die stillen Gruppen jedoch isoliert ansehen, können Sie feststellen, dass die meisten nicht verrauschten Nachrichtengruppen oder Mandanten nicht betroffen sind.

Im Folgenden finden Sie ein Beispiel, bei dem die standardmäßige Queue-Backlog-Metrik (`ApproximateNumberOfMessagesVisible`) aufgrund eines verrauschten Mandanten zunimmt, während der Backlog für nicht laute Mandanten () niedrig bleibt.

`ApproximateNumberOfMessagesVisibleInQuietGroups`



Eine vollständige Liste der Amazon CloudWatch SQS-Metriken und deren Beschreibungen finden Sie unter [CloudWatch Metriken für Amazon SQS](#).

So funktionieren Amazon SQS Fair Queues

Amazon SQS Fair Queues mindern automatisch die Auswirkungen von Noisy-Neighbor-Störungen in Warteschlangen mit mehreren Mandanten, indem verhindert wird, dass ein einzelner Mandant die Nachrichtenverarbeitung für andere Mandanten verlangsamt. Wenn Fair Queues feststellt, dass ein Mandant einen unverhältnismäßigen Teil der Nutzerkapazität der Warteschlange nutzt, priorisieren sie die Zustellung von Nachrichten von stillen Mietern.

Identifizierung des Mieters

In fairen Warteschlangen werden die Mieter bei jeder Nachricht anhand der `MessageGroupId` Immobilie identifiziert. Nachrichten, die dasselbe verwenden, `MessageGroupId` gehören demselben Mandanten. Eine faire Warteschlange kann eine Mischung aus Nachrichten mit und ohne `MessageGroupId` enthalten. Jede Nachricht ohne `MessageGroupId` wird so behandelt, als ob sie ihrem eigenen Mandanten gehört.

Wir empfehlen, für jede Nachricht, die Sie an eine faire Warteschlange senden, ein `MessageGroupId` zu setzen und dabei eine Kennung zu verwenden, die einer echten Entität in Ihrem System zugeordnet ist, z. B. einer Kunden-ID, einer Client-Anwendungs-ID oder einem Anforderungstyp.

Wie funktioniert die Erkennung

Amazon SQS erkennt laute Nachbarn mithilfe von zwei Maßnahmen:

- Anteil an Parallelität — die Bordinformationen des Mieters als Bruchteil aller Bordinformationen in der Warteschlange. Dadurch werden Mandanten erkannt, bei denen viele Nachrichten gleichzeitig verarbeitet werden, sei es aufgrund eines hohen Sendevolumens, einer langsamen Verarbeitung oder beidem.
- Anteil an der Bearbeitungszeit — der aktuelle Anteil des Mandanten an der gesamten Bearbeitungszeit des Verbrauchers. Dadurch werden Mandanten erkannt, deren Nachrichten zwar nur wenige, aber langsam zu verarbeiten sind, sodass der Mieter viel Zeit in Anspruch nimmt, ohne viele Nachrichten gleichzeitig im Umlauf zu haben.

Ein Mieter wird als lauter Nachbar markiert, wenn eine der folgenden Bedingungen zutrifft:

- Anteil an Parallelität: Auf den Mandanten entfallen mehr als 10% der laufenden Nachrichten in der Warteschlange und er hat mindestens 30 seiner eigenen Nachrichten im Umlauf.
- Anteil der Bearbeitungszeit: Der Anteil des Mieters an der gesamten Bearbeitungszeit der Verbraucher liegt in letzter Zeit bei über 10%.

Da es sich bei Amazon SQS um ein verteiltes System handelt, handelt es sich bei diesen Schwellenwerten um ungefähre Werte, sodass die Erkennung möglicherweise nicht genau bei den beschriebenen Werten aktiviert wird.

Wie ändert sich die Reihenfolge der Nachrichtenzustellung

Sobald ein Mandant als lauter Nachbar erkannt wird, priorisiert Amazon SQS die Zustellung von Nachrichten von leisen Mietern. Verbraucherinstanzen, die Nachrichten abfragen, empfangen Nachrichten von stillen Mandanten, wann immer Nachrichten von leisen Mandanten verfügbar sind. Diese Priorisierung wird so lange fortgesetzt, bis der Parallelitätsanteil und der Anteil an der Bearbeitungszeit des Nachbarn auf Werte gesunken sind, die mit denen von leisen Mandanten vergleichbar sind.

Nachrichten, bei denen es zu Störungen kommt, werden nicht gelöscht oder gedrosselt. Ihre Verweildauer nimmt zu, während Nachrichten, die sich an unruhige Mandanten richten, priorisiert werden, und kehrt zur Normalität zurück, sobald der Rückstand behoben ist. Wenn keine Quiet-Tenant-Nachrichten warten, empfangen Consumer-Instances wie gewohnt Noisy Neighbor-Nachrichten.

Wenn mehrere Mandanten gleichzeitig als Noisy Neighbours erkannt werden, erhalten Consumer-Instances zuerst Nachrichten von dem Mandanten mit den wenigsten Inflight-Nachrichten. Dies trägt dazu bei, die Verarbeitungszeit zwischen den lauten Mietern auszugleichen.

Wenn ein lauter Nachbar leise wird

Ein Mieter wird nicht mehr wie ein lauter Nachbar behandelt, wenn eine der folgenden Bedingungen zutrifft:

- Der Rückstand des Mieters wurde vollständig aufgebraucht.
- Über einen ununterbrochenen Zeitraum von 5 Minuten waren keine Nachrichten des Mieters unterwegs.

Danach werden die Nachrichten des Mieters nicht mehr depriorisiert und genauso behandelt wie Nachrichten von anderen unauffälligen Mietern.

Überlegungen zur Kapazität der Verbraucher

Damit die Maßnahme zur Erhöhung des Gleichzeitigkeitsanteils effektiv funktioniert, müssen Ihre Kunden genügend Nachrichten gleichzeitig verarbeiten, sodass der Anteil eines Mandanten an

den Nachrichten während der Übertragung herausstechen kann. Berücksichtigen Sie dies bei der Dimensionierung Ihrer Kundenflotte.

Wenn Lambda über die Zuordnung von Ereignisquellen als Verbraucher verwendet wird, hängt die Anzahl der laufenden Nachrichten sowohl von der Parallelität der Lambda-Funktionen als auch von der Batchgröße ab. Bewerten Sie diese Einstellungen zusammen, wenn Sie die Größe Ihrer Verbraucher festlegen.

Wenn die Anzahl der Nachrichten während der Übertragung zu gering ist, als dass der Schwellenwert für die gleichzeitige Nutzung ausgelöst werden könnte, kann das Maß für den Anteil der Verarbeitungszeit dennoch störende Nachbarn erkennen. Faire Warteschlangen funktionieren jedoch am besten, wenn die Verbraucher genügend Nachrichten gleichzeitig verarbeiten, sodass beide Maßnahmen bewertet werden können.

Amazon-SQS-Verzögerungswarteschlangen

Mit Verzögerungswarteschlangen können Sie die Zustellung neuer Nachrichten an Verbraucher um einige Sekunden verschieben, z. B. wenn Ihre Verbraucheranwendung zusätzliche Zeit für die Verarbeitung von Nachrichten benötigt. Wenn Sie eine Verzögerungswarteschlange erstellen, bleiben an diese Warteschlange gesendete Nachrichten für Konsumenten für die Dauer des Verzögerungszeitraums unsichtbar. Die Standardverzögerung (Mindestverzögerung) für eine Warteschlange beträgt 0 Sekunden. Der Maximalwert beträgt 15 Minuten. Weitere Informationen zur Konfiguration von Verzögerungswarteschlangen mit der Konsole finden Sie unter [Konfiguration von Warteschlangenparametern mit der Amazon SQS SQS-Konsole](#).

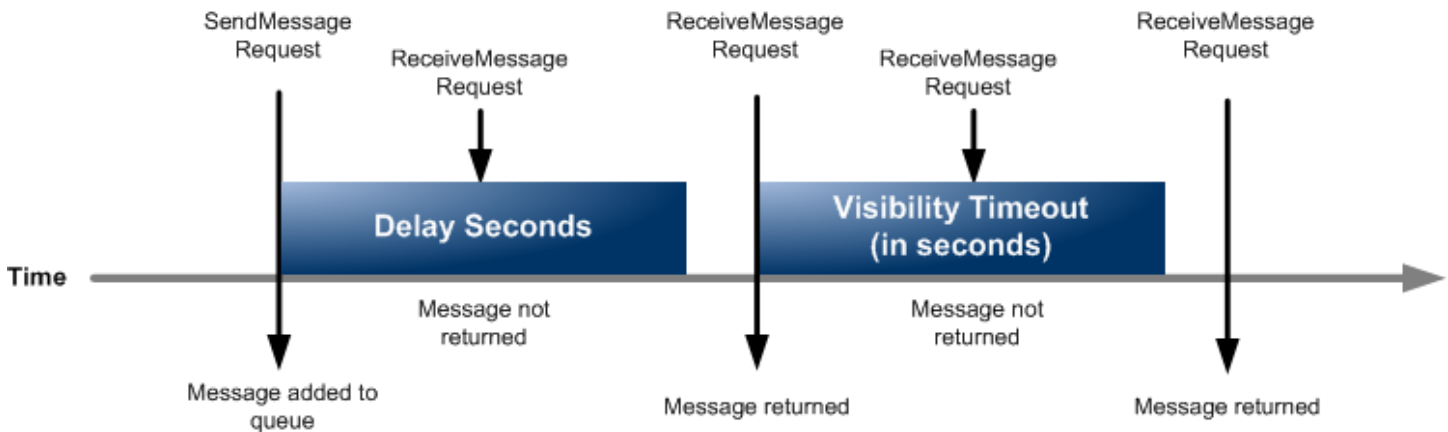
Note

Für Standard-Warteschlangen ist die Einstellung für die Verzögerung pro Warteschlange nicht retroaktiv – wenn Sie die Einstellung ändern, wirkt sich dies nicht auf die Verzögerung von Nachrichten aus, die sich bereits in der Warteschlange befinden.

Für FIFO-Warteschlangen ist die Einstellung für die Verzögerung pro Warteschlange retroaktiv – wenn Sie die Einstellung ändern, wirkt sich dies auf die Verzögerung von Nachrichten aus, die sich bereits in der Warteschlange befinden.

Verzögerungswarteschlangen funktionieren vergleichbar mit [Zeitbeschränkungen für die Sichtbarkeit](#), da durch beide Funktionen Nachrichten eine bestimmte Zeit lang für Konsumenten nicht verfügbar gemacht werden. Der Unterschied zwischen beidem besteht darin, dass in

Verzögerungswarteschlangen die Nachricht direkt nach dem Hinzufügen zur Warteschlange verborgen wird, bei Zeitbeschränkungen für die Sichtbarkeit hingegen erst, nachdem die Nachricht aus der Warteschlange abgerufen wurde. Das folgende Diagramm verdeutlicht die Beziehung zwischen Verzögerungswarteschlangen und Zeitbeschränkungen für die Sichtbarkeit.



Erweiterte Planungsoptionen

Die Verzögerungswarteschlangen und Nachrichtentimer von Amazon SQS ermöglichen zwar die Planung der Nachrichtenzustellung bis zu 15 Minuten in der future, Sie benötigen jedoch möglicherweise flexiblere Planungsfunktionen. In solchen Fällen sollten Sie die Verwendung von [EventBridge Scheduler](#) in Betracht ziehen, mit dem Sie Milliarden einmaliger oder wiederkehrender API-Aktionen ohne zeitliche Einschränkungen planen können. EventBridge Scheduler ist die empfohlene Lösung für erweiterte Anwendungsfälle zur Nachrichtenplanung.

Um Verzögerungssekunden für einzelne Nachrichten und nicht für eine gesamte Warteschlange festzulegen, verwenden Sie [Nachrichten-Timer, damit Amazon SQS den Wert des Nachrichten-Timers](#) anstelle des DelaySeconds Werts der Verzögerungswarteschlange verwenden kann. DelaySeconds [EventBridge Scheduler](#) unterstützt auch die Planung einzelner Nachrichten.

Temporäre Amazon-SQS-Warteschlangen

Temporäre Warteschlangen helfen Ihnen, Entwicklungszeit und Bereitstellungskosten zu sparen, wenn Sie gängige Nachrichtenmuster wie Anfrage-Antwort verwenden. Sie können den [temporären Warteschlangen-Client](#) verwenden, um kostengünstige, anwendungsverwaltete temporäre Warteschlangen mit hohem Durchsatz zu erstellen.

Der Client ordnet mehrere temporäre Warteschlangen – anwendungsverwaltete Warteschlangen, die bei Bedarf für einen bestimmten Prozess erstellt werden – automatisch einer einzigen Amazon-SQS-Warteschlange zu. Auf diese Weise kann Ihre Anwendung weniger API-Aufrufe durchführen und

einen höheren Durchsatz verzeichnen, wenn der Datenverkehr zu jeder temporären Warteschlange gering ist. Wenn eine temporäre Warteschlange nicht mehr verwendet wird, bereinigt der Client die temporäre Warteschlange automatisch. Dies gilt auch dann, wenn einige Prozesse, die den Client nutzen, nicht ordnungsgemäß heruntergefahren werden.

Im Folgenden werden die Vorteile von temporären Warteschlangen beschrieben:

- Sie dienen als einfache Kommunikationskanäle für bestimmte Threads oder Prozesse.
- Sie können erstellt und gelöscht werden, ohne dass zusätzliche Kosten anfallen.
- Diese sind API-kompatibel mit statischen (normalen) Amazon-SQS-Warteschlangen. Dies bedeutet, dass vorhandener Code, mit dem Nachrichten gesendet und empfangen werden, Nachrichten an virtuelle Warteschlangen senden und von ihnen empfangen kann.

Virtuelle Warteschlangen

Virtuelle Warteschlangen sind lokale Datenstrukturen, die vom Client temporärer Warteschlangen erstellt werden. Mit virtuellen Warteschlangen können Sie mehrere Ziele mit geringem Datenverkehr in einer einzigen Amazon-SQS-Warteschlange zusammenfassen. Bewährte Methoden finden Sie unter Vermeiden Sie die Wiederverwendung derselben Nachrichtengruppen-ID in virtuellen Warteschlangen.

Note

- Beim Erstellen einer virtuellen Warteschlange werden nur temporäre Datenstrukturen erstellt, in denen Konsumenten Nachrichten empfangen können. Da eine virtuelle Warteschlange keine API-Aufrufe an Amazon SQS sendet, entstehen für virtuelle Warteschlangen keine Kosten.
- TPS-Kontingente gelten für alle virtuellen Warteschlangen in einer einzelnen Hostwarteschlange. Weitere Informationen finden Sie unter [Amazon SQS SQS-Nachrichtenkontingente](#).

Die `AmazonSQSVirtualQueuesClient`-Wrapper-Klasse wurde um neue Unterstützung für Attribute im Zusammenhang mit virtuellen Warteschlangen erweitert. Um eine virtuelle Warteschlange erstellen zu können, müssen Sie die `CreateQueue`-API-Aktion mit dem Attribut `HostQueueURL` aufrufen. Dieses Attribut gibt die vorhandene Warteschlange an, in der die virtuellen Warteschlangen gehostet werden.

Die URL einer virtuellen Warteschlange hat das folgende Format.

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

Wenn ein Produzent die API-Aktion `SendMessage` oder `SendMessageBatch` für die URL einer virtuellen Warteschlange aufruft, verfährt der Client temporärer Warteschlangen wie folgt:

1. Er extrahiert den Namen der virtuellen Warteschlange.
2. Er fügt den Namen der virtuellen Warteschlange als zusätzliches Nachrichtenattribut an.
3. Er sendet die Nachricht an die Host-Warteschlange.

Während der Produzent Nachrichten sendet, fragt ein Hintergrund-Thread die Host-Warteschlange ab und sendet die empfangenen Nachrichten gemäß den entsprechenden Nachrichtenattributen an virtuelle Warteschlangen.

Während der Konsument die `ReceiveMessage`-API-Aktion für die URL einer virtuellen Warteschlange aufruft, blockiert der Client temporärer Warteschlangen den Aufruf lokal, bis der Hintergrund-Thread eine Nachricht in die virtuelle Warteschlange sendet. (Dieser Vorgang ist mit dem Vorabrufen von Nachrichten im [gepufferten asynchronen Client](#) vergleichbar: eine einzelne API-Aktion kann für bis zu 10 virtuelle Warteschlangen Nachrichten bereitstellen.) Wenn eine virtuelle Warteschlange gelöscht wird, werden alle Client-seitigen Ressourcen entfernt, ohne Amazon SQS selbst aufzurufen.

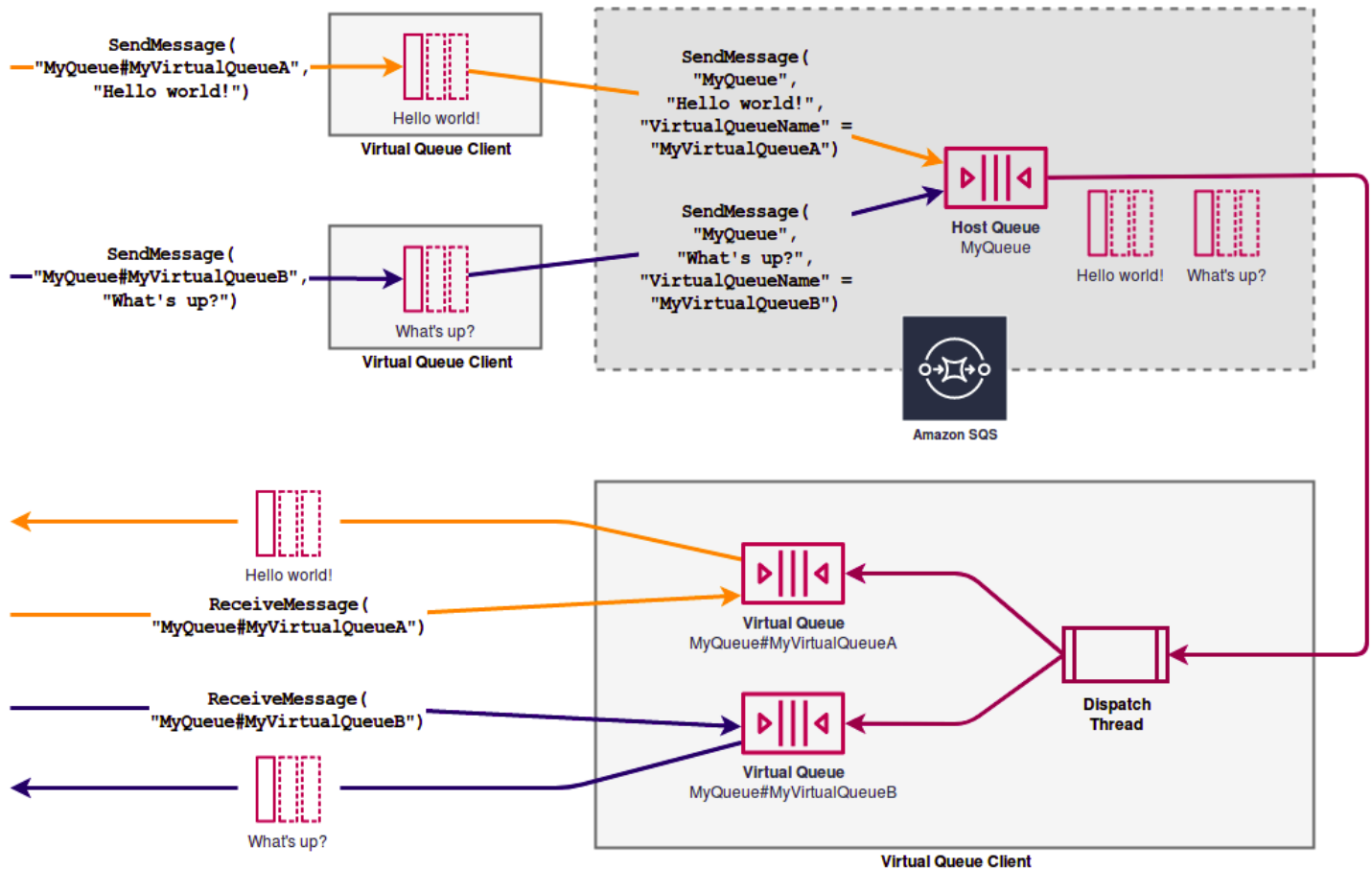
Die `AmazonSQSTemporaryQueuesClient`-Klasse wandelt alle von ihr erstellten Warteschlangen automatisch in temporäre Warteschlangen um. Sie erstellt außerdem Host-Warteschlangen mit denselben Warteschlangenattributen automatisch nach Bedarf. Den Namen dieser Warteschlangen ist ein konfigurierbares Präfix (standardmäßig `__RequesterClientQueues__`) gemeinsam, durch das sie als temporäre Warteschlangen identifiziert werden. Auf diese Weise kann der Client als Drop-In-Ersatz fungieren, durch den vorhandener Code optimiert wird, mit dem Warteschlangen erstellt und gelöscht werden. Der Client umfasst auch die `AmazonSQSRequester`- und `AmazonSQSResponder`-Schnittstellen, die eine bidirektionale Kommunikation zwischen Warteschlangen ermöglichen.

Request-Response-Messaging-Muster (virtuelle Warteschlangen)

Der häufigste Anwendungsfall für temporäre Warteschlangen ist das request-response-Messaging-Muster, bei dem ein Anforderer eine temporäre Warteschlange für den Empfang der einzelnen Antwortnachrichten erstellt. Um zu vermeiden, dass für jede Antwortnachricht eine Amazon-SQS-Warteschlange erstellt wird, können Sie mit dem Client für temporäre Warteschlangen mehrere

temporäre Warteschlangen ohne Amazon-SQS-API-Aufrufe erstellen. Weitere Informationen finden Sie unter Implementieren von Anfrage-Antwort-Systemen.

Das folgende Diagramm zeigt eine geläufige Konfiguration mit diesem Muster.



Beispielszenario: Verarbeiten einer Anmeldeanforderung

Das folgende Beispielszenario veranschaulicht, wie die `AmazonSQSRequester`- und `AmazonSQSResponder`-Schnittstellen zur Verarbeitung der Anmeldeanforderung eines Benutzers verwendet werden.

Auf der Client-Seite

```
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();
}
}
```

Das Senden einer Anmeldeanforderung bewirkt Folgendes:

1. Erstellt eine temporäre Warteschlange.
2. Es fügt die URL der temporären Warteschlange als Attribut an die Nachricht an.
3. Es sendet die Nachricht.
4. Es empfängt eine Antwort von der temporären Warteschlange.
5. Es löscht die temporäre Warteschlange.
6. Es gibt die Antwort zurück.

Auf der Server-Seite

Im folgenden Beispiel wird davon ausgegangen, dass bei der Erstellung ein Thread zur Abfrage der Warteschlange und zum Aufruf der Methode `handleLoginRequest()` für jede Nachricht erstellt wird. Darüber hinaus handelt es sich bei `doLogin()` um eine angenommene Methode.

```
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));
}
}
```

Bereinigen von Warteschlangen

Um sicherzustellen, dass Amazon SQS alle von virtuellen Warteschlangen beanspruchten Ressourcen im Arbeitsspeicher freigibt, wenn die Anwendung den Client für temporäre Warteschlangen nicht mehr benötigt, sollte die Methode `shutdown()` aufgerufen werden. Sie können auch die Methode `shutdown()` der `AmazonSQSRequester`-Schnittstelle verwenden.

Der Client temporärer Warteschlangen bietet auch eine Option, mit der sich verwaiste Host-Warteschlangen vermeiden lassen. Alle Warteschlangen, die innerhalb eines Zeitraums (standardmäßig fünf Minuten) einen API-Aufruf empfangen, werden vom Client mit der API-Aktion `TagQueue` als weiterhin genutzte Warteschlangen markiert.

Note

API-Aktionen, die für eine Warteschlange ausgeführt werden, markieren diese als nicht im Leerlauf befindlich. Dazu gehört auch eine `ReceiveMessage`-Aktion, die keine Nachrichten zurückgibt.

Der Hintergrund-Thread verwendet die API-Aktionen `ListQueues` und `ListTags`, um alle Warteschlange mit dem konfigurierten Präfix zu überprüfen und alle Warteschlangen zu entfernen, die nicht schon mindestens fünf Minuten lang gekennzeichnet sind. Sollte ein Client nicht ordnungsgemäß heruntergefahren werden, können die anderen aktiven Clients ihn auf diese Weise bereinigen. Um einer Verdopplung des Arbeitsaufwands entgegenzuwirken, kommunizieren alle Clients mit demselben Präfix über eine freigegebene, interne Arbeitswarteschlange, die nach dem Präfix benannt ist, miteinander.

Amazon-SQS-Nachrichten-Timer

Nachrichtentimer ermöglichen es Ihnen, einen anfänglichen Unsichtbarkeitszeitraum für eine Nachricht festzulegen, wenn sie zu einer Warteschlange hinzugefügt wird. Wenn Sie beispielsweise eine Nachricht mit einem 45-Sekunden-Timer senden, bleibt sie den Verbrauchern in den ersten 45 Sekunden verborgen. Die Standardverzögerung (Mindestverzögerung) für eine Nachricht beträgt 0 Sekunden. Der Maximalwert beträgt 15 Minuten. Informationen zum Senden von Nachrichten mit Timern über die Konsole finden Sie unter [Senden einer Nachricht mithilfe einer Standardwarteschlange](#).

Note

FIFO-Warteschlangen unterstützen keine Timer für einzelne Nachrichten.

Um eine Verzögerungszeit für eine gesamte Warteschlange und nicht für einzelne Nachrichten festzulegen, verwenden Sie [Verzögerungswarteschlangen](#). Eine Nachrichten-Timer-Einstellung für eine einzelne Nachricht überschreibt jeden `DelaySeconds`-Wert auf einer Amazon-SQS-Verzögerungswarteschlange.

Erweiterte Planungsoptionen

Die Verzögerungswarteschlangen und Nachrichtentimer von Amazon SQS ermöglichen zwar die Planung der Nachrichtenzustellung bis zu 15 Minuten in der future, Sie benötigen jedoch möglicherweise flexiblere Planungsfunktionen. In solchen Fällen sollten Sie die Verwendung von [EventBridge Scheduler](#) in Betracht ziehen, mit dem Sie Milliarden einmaliger oder wiederkehrender API-Aktionen ohne zeitliche Einschränkungen planen können. EventBridge Scheduler ist die empfohlene Lösung für erweiterte Anwendungsfälle zur Nachrichtenplanung.

Zugriff auf Amazon EventBridge Pipes über die Amazon SQS SQS-Konsole

Amazon EventBridge Pipes verbindet Quellen mit Zielen. Pipes sind für point-to-point Integrationen zwischen unterstützten Quellen und Zielen vorgesehen und unterstützen erweiterte Transformationen und Anreicherungen. EventBridge Pipes bieten eine hoch skalierbare Möglichkeit, Ihre Amazon SQS-Warteschlange mit AWS Services wie Step Functions, Amazon SQS und API Gateway sowie Software-as-a-Service (SaaS) -Anwendungen von Drittanbietern wie Salesforce zu verbinden.

Zum Einrichten einer Pipe wählen Sie die Quelle aus, fügen Sie optionale Filterung hinzu, definieren Sie die optionale Anreicherung und wählen Sie das Ziel für die Ereignisdaten.

Auf der Detailseite für eine Amazon-SQS-Warteschlange können Sie sich die Pipes ansehen, die diese Warteschlange als Quelle verwenden. Von dort aus können Sie auch Folgendes tun:

- Starten Sie die EventBridge Konsole, um die Pipe-Details anzuzeigen.
- Starten Sie die EventBridge Konsole, um eine neue Pipe mit der Warteschlange als Quelle zu erstellen.

Weitere Informationen zur Konfiguration einer Amazon SQS SQS-Warteschlange als Pipe-Quelle finden Sie unter [Amazon SQS SQS-Warteschlange als Quelle](#) im EventBridge Amazon-Benutzerhandbuch. [Weitere Informationen zu EventBridge Pipes im Allgemeinen finden Sie unter EventBridge Pipes.](#)

So greifen Sie auf EventBridge Pipes für eine bestimmte Amazon SQS SQS-Warteschlange zu

1. Öffnen Sie die Seite [Warteschlangen](#) der Amazon-SQS-Konsole.
2. Wählen Sie eine Warteschlange aus.
3. Wählen Sie auf der Detailseite der Warteschlange die Registerkarte EventBridge Pipes.

Die Registerkarte EventBridge Pipes enthält eine Liste aller Pipes, die derzeit so konfiguriert sind, dass sie die ausgewählte Warteschlange als Quelle verwenden, darunter:

- Name der Pipe
- aktueller Status
- Pipe-Ziel
- wann die Pipe zuletzt geändert wurde

4. Falls gewünscht, können Sie weitere Pipe-Details anzeigen oder eine neue Pipe erstellen:

- So greifen Sie auf weitere Details zu einer Pipe zu:

Wählen Sie den Namen der Pipe aus.

Dadurch wird die Seite mit den Pipe-Details der EventBridge Konsole geöffnet.

- So erstellen Sie eine neue Pipe:

Wählen Sie Amazon-SQS-Warteschlange mit Pipe verbinden.

Dadurch wird die Seite „Pipe erstellen“ der EventBridge Konsole geöffnet, auf der die Amazon SQS SQS-Warteschlange als Pipe-Quelle angegeben ist. Weitere Informationen finden Sie unter [Erstellen einer EventBridge Pipe](#) im EventBridge Amazon-Benutzerhandbuch.

Important

Eine Nachricht in einer Amazon-SQS-Warteschlange wird von einer einzelnen Pipe gelesen und dann nach der Verarbeitung aus der Warteschlange gelöscht, unabhängig davon, ob die Nachricht dem Filter entspricht, den Sie für diese Pipe konfiguriert haben. Gehen Sie vorsichtig vor, wenn Sie mehrere Pipes so konfigurieren, dass sie dieselbe Warteschlange als Quelle verwenden.

Verwaltung großer Amazon SQS SQS-Nachrichten mit Extended Client Library und Amazon Simple Storage Service

Verwenden Sie die [Amazon SQS Extended Client Library für Java](#) und die [Amazon SQS Extended Client Library für Python](#), um große Nachrichten zu senden, insbesondere für Nutzlasten zwischen 256 KB und 2 GB. Diese Bibliotheken speichern die Nachrichtennutzlast in einem Amazon S3 S3-Bucket und senden einen Verweis auf das gespeicherte Objekt in der Amazon SQS SQS-Warteschlange.

Note

Die Amazon SQS Extended Client Libraries sind sowohl mit Standard- als auch mit FIFO-Warteschlangen kompatibel.

Verwaltung großer Amazon SQS SQS-Nachrichten mit Java und Amazon S3

Verwenden Sie die [Amazon SQS Extended Client Library für Java](#) mit Amazon S3, um große Amazon SQS SQS-Nachrichten zu verwalten, insbesondere für Nutzlasten im Bereich von 256 KB bis 2 GB. Die Bibliothek speichert die Nachrichtennutzlast in einem Amazon S3 S3-Bucket und sendet eine Nachricht mit einem Verweis auf das gespeicherte Objekt in der Amazon SQS SQS-Warteschlange.

Mit der Amazon SQS Extended Client Library für Java können Sie:

- Festlegen, ob Nachrichten grundsätzlich oder ab einer Nachrichtengröße über 256 KB in Amazon S3 gespeichert werden
- Senden einer Nachricht, die auf ein einzelnes in einem S3-Bucket gespeichertes Nachrichtenobjekt verweist
- Rufen Sie das Nachrichtenobjekt aus einem Amazon S3 S3-Bucket ab
- Löschen Sie das Nachrichtenobjekt aus einem Amazon S3 S3-Bucket

Voraussetzungen

Das folgende Beispiel verwendet das AWS Java-SDK. Informationen zur Installation und Einrichtung des SDK finden Sie unter [Setup the AWS SDK for Java](#) im AWS SDK für Java Developer Guide.

Bevor Sie den Beispielcode ausführen, konfigurieren Sie Ihre AWS Anmeldeinformationen. Weitere Informationen finden Sie unter [Einrichten von AWS Anmeldeinformationen und Region für die Entwicklung](#) im AWS SDK für Java Entwicklerhandbuch.

Für das [SDK für Java](#) und die Amazon SQS Extended Client Library für Java ist das J2SE Development Kit 8.0 oder eine neuere Version erforderlich.

Note

Sie können die Amazon SQS Extended Client Library für Java verwenden, um Amazon-SQS-Nachrichten mithilfe von Amazon S3 nur mit der AWS SDK für Java zu verwalten. Sie können dies nicht mit der AWS CLI Amazon SQS SQS-Konsole, der Amazon SQS SQS-HTTP-API oder einer der anderen tun. AWS SDKs

AWS SDK for Java 2.x Beispiel: Verwendung von Amazon S3 zur Verwaltung großer Amazon SQS SQS-Nachrichten

Das folgende Beispiel für ein SDK für SDK for Java 2.x verwendet die Extended Client Library für Java, um mit großen Nachrichten zu arbeiten. Im Konstruktor der folgende Code:

- Erstellt einen Amazon S3 S3-Bucket mit einem zufälligen Namen
- Erstellt eine SQS-Warteschlange, die beginnt mit MyQueue
- Schließt einen standardmäßigen Java-SDK-Amazon S3-Client in eine Instanz von `AmazonSQSExtendedClient`

In der `sendAnReceiveMessage` Methode sendet das Beispiel eine zufällige Nachricht, die in einem Amazon S3 S3-Bucket gespeichert ist, weil sie mehr als 256 KB groß ist (die standardmäßige maximale Nachrichtengröße). Schließlich ruft die Methode die Nachricht ab und zeigt Informationen darüber auf der Konsole an.

Das vollständige Beispiel finden Sie unter https://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());
    }
}

```

Sie können [Apache Maven verwenden](#), um Amazon SQS Extended Client für Ihr Java-Projekt zu konfigurieren und zu erstellen oder um das SDK selbst zu erstellen. Geben Sie einzelne Module aus dem SDK an, die Sie in Ihrer Anwendung verwenden.

```

<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>
</dependencies>

```

```
</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>
```

Verwaltung großer Amazon SQS SQS-Nachrichten mit Python und Amazon S3

Verwenden Sie die Amazon SQS [Amazon SQS Extended Client Library for Python](#) mit Amazon S3, um große Amazon SQS SQS-Nachrichten zu verwalten, insbesondere für Nutzlasten zwischen 256 KB und 2 GB. Die Bibliothek speichert die Nachrichtennutzlast in einem Amazon S3 S3-Bucket und sendet eine Nachricht mit einem Verweis auf das gespeicherte Objekt in der Amazon SQS SQS-Warteschlange.

Mit der Amazon SQS Extended Client Library für Python können Sie:

- Geben Sie an, ob Payloads immer in Amazon S3 oder nur in Amazon S3 gespeichert werden, wenn die Payloadgröße 256 KB überschreitet
- Senden Sie eine Nachricht, die auf ein einzelnes Nachrichtenobjekt verweist, das in einem Amazon S3 S3-Bucket gespeichert ist
- Rufen Sie das entsprechende Payload-Objekt aus einem Amazon S3 S3-Bucket ab
- Löschen Sie das entsprechende Payload-Objekt aus einem Amazon S3 S3-Bucket

Voraussetzungen

Im Folgenden sind die Voraussetzungen für die Verwendung der Amazon SQS Extended Client Library für Python aufgeführt:

- Ein AWS Konto mit den erforderlichen Anmeldeinformationen. Um ein AWS Konto zu erstellen, navigieren Sie zur [AWS Startseite](#) und wählen Sie dann AWS Konto erstellen. Folgen Sie den Anweisungen. Informationen zu Anmeldeinformationen finden Sie unter [Anmeldeinformationen](#).
- Ein AWS SDK: Das Beispiel auf dieser Seite verwendet das AWS Python-SDK Boto3. Informationen zur Installation und Einrichtung des SDK finden Sie in der Dokumentation [zum AWS SDK für Python](#) im AWS SDK for Python Developer Guide
- Python 3.x (oder höher) und `pip`.
- [Die Amazon SQS Extended Client Library für Python, erhältlich bei PyPI](#)

Note

Sie können die Amazon SQS Extended Client Library for Python verwenden, um Amazon SQS SQS-Nachrichten mithilfe von Amazon S3 nur mit dem AWS SDK für Python zu verwalten. Sie können dies nicht mit der AWS CLI, der Amazon SQS SQS-Konsole, der Amazon SQS SQS-HTTP-API oder einer der anderen tun. AWS SDKs

Konfigurieren der Nachrichtenspeicherung

Der Amazon SQS Extended Client verwendet die folgenden Nachrichtenattribute, um die Amazon S3 S3-Nachrichtenspeicheroptionen zu konfigurieren:

- `large_payload_support`: Der Amazon S3 S3-Bucket-Name zum Speichern großer Nachrichten.
- `always_through_s3`: Wenn `True`, dann werden alle Nachrichten in Amazon S3 gespeichert. Falls `False`, werden Nachrichten, die kleiner als 256 KB sind, nicht in den S3-Bucket serialisiert. Der Standardwert ist `False`.
- `use_legacy_attribute`: Falls `True` alle veröffentlichten Nachrichten das reservierte Nachrichtenattribut Legacy (SQSLargePayloadSize) anstelle des aktuellen reservierten Nachrichtenattributs (ExtendedPayloadSize) verwenden.

Verwaltung großer Amazon SQS SQS-Nachrichten mit der Extended Client Library für Python

Das folgende Beispiel erstellt einen Amazon S3 S3-Bucket mit einem zufälligen Namen. Anschließend erstellt es eine Amazon SQS SQS-Warteschlange mit dem Namen `MyQueue` und

sendet eine Nachricht, die in einem S3-Bucket gespeichert ist und mehr als 256 KB groß ist, an die Warteschlange. Schließlich ruft der Code die Nachricht ab, gibt Informationen über die Nachricht zurück und löscht die Nachricht, die Warteschlange und den Bucket.

```
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
```

Konfiguration von Amazon SQS SQS-Warteschlangen mit der Amazon SQS SQS-Konsole

Verwenden Sie die Amazon SQS SQS-Konsole, um Amazon SQS SQS-Warteschlangen und -Funktionen zu konfigurieren und zu verwalten. Sie können auch:

- Aktivieren Sie die serverseitige Verschlüsselung für mehr Sicherheit.
- Ordnen Sie der Bearbeitung unverarbeiteter Nachrichten eine Warteschlange für unzustellbare Nachrichten zu.
- Richten Sie einen Trigger ein, um eine Lambda-Funktion für die ereignisgesteuerte Verarbeitung aufzurufen.

Attributbasierte Zugriffskontrolle für Amazon SQS

Was ist ABAC?

Die attributbasierte Zugriffskontrolle (ABAC) ist ein Autorisierungsprozess, bei dem Berechtigungen auf der Grundlage von Tags definiert werden, die Benutzern und Ressourcen zugewiesen sind. AWS ABAC bietet eine detaillierte und flexible Zugriffskontrolle auf der Grundlage von Attributen und Werten, reduziert das Sicherheitsrisiko im Zusammenhang mit neu konfigurierten rollenbasierten Richtlinien und zentralisiert die Prüfung und Verwaltung von Zugriffsrichtlinien. Weitere Details zu ABAC finden Sie unter [Was ist ABAC für AWS?](#) im IAM-Benutzerhandbuch.

Amazon SQS unterstützt ABAC, indem es Ihnen ermöglicht, den Zugriff auf Ihre Amazon-SQS-Warteschlangen anhand der Tags und Aliase zu kontrollieren, die mit einer Amazon-SQS-Warteschlange verknüpft sind. Die Tag- und Alias-Bedingungsschlüssel, die ABAC in Amazon SQS aktivieren, autorisieren IAM-Prinzipale, Amazon-SQS-Warteschlangen zu verwenden, ohne Richtlinien zu bearbeiten oder Erteilungen zu verwalten. Weitere Informationen zu ABAC-Bedingungsschlüsseln finden Sie unter [Bedingungsschlüssel für Amazon SQS](#) in der Service Authorization Reference.

Mit ABAC können Sie Tags verwenden, um IAM-Zugriffsberechtigungen und Richtlinien für Ihre Amazon-SQS-Warteschlangen zu konfigurieren, was Ihnen hilft, Ihr Berechtigungsmanagement zu skalieren. Sie können in IAM eine einzige Berechtigungsrichtlinie erstellen, indem Sie Tags verwenden, die Sie jeder Geschäftsrolle hinzufügen – ohne die Richtlinie jedes Mal aktualisieren zu müssen, wenn Sie eine neue Ressource hinzufügen. Sie können IAM-Prinzipalen auch Tags

zuordnen, um eine ABAC-Richtlinie zu erstellen. Sie können ABAC-Richtlinien entwerfen, um Amazon-SQS-Operationen zuzulassen, wenn das Tag in der IAM-Benutzerrolle, die den Aufruf tätigt, mit dem Amazon-SQS-Warteschlangen-Tag übereinstimmt. Weitere Informationen zum Tagging finden Sie unter [AWS Tagging-Strategien](#) und [Amazon-SQS-Kostenzuordnungs-Tags](#).

Note

ABAC für Amazon SQS ist derzeit in allen AWS Handelsregionen verfügbar, in denen Amazon SQS verfügbar ist, mit den folgenden Ausnahmen:

- Asien-Pazifik (Hyderabad)
- Asien-Pazifik (Melbourne)
- Europa (Spain)
- Europa (Zürich)

Weshalb sollte ich ABAC in Amazon SQS verwenden?

Hier sind einige Vorteile der Verwendung von ABAC in Amazon SQS:

- ABAC für Amazon SQS erfordert weniger Berechtigungsrichtlinien. Sie müssen keine verschiedenen Richtlinien für verschiedene Job-Funktionen erstellen müssen. Sie können Ressourcen- und Anforderungs-Tags verwenden, die für mehr als eine Warteschlange gelten, wodurch der Betriebsaufwand reduziert wird.
- Verwenden Sie ABAC, um Teams schnell zu skalieren. Berechtigungen für neue Ressourcen werden automatisch basierend auf Tags erteilt, wenn Ressourcen bei ihrer Erstellung entsprechend gekennzeichnet werden.
- Verwenden Sie Berechtigungen für den IAM-Prinzipal, um den Ressourcenzugriff einzuschränken. Sie können Tags für den IAM-Prinzipal erstellen und diese verwenden, um den Zugriff auf bestimmte Aktionen einzuschränken, die den Tags auf dem IAM-Prinzipal entsprechen. Auf diese Weise können Sie den Prozess der Erteilung von Anforderungsberechtigungen automatisieren.
- Verfolgen Sie, wer auf Ihre Ressourcen zugreift. Sie können die Identität einer Sitzung anhand der Benutzerattribute unter AWS CloudTrail ermitteln.

Themen

- [Tagging für die Zugriffskontrolle in Amazon SQS](#)

- [Erstellen von IAM-Benutzern und Amazon-SQS-Warteschlangen](#)
- [Testen der attributbasierten Zugriffskontrolle in Amazon SQS](#)

Tagging für die Zugriffskontrolle in Amazon SQS

Im Folgenden finden Sie ein Beispiel für die Verwendung von Tags für die Zugriffskontrolle in Amazon SQS. Die IAM-Richtlinie beschränkt einen IAM-Benutzer auf alle Amazon-SQS-Aktionen für alle Warteschlangen, die ein Ressourcen-Tag mit der Schlüsselumgebung und der Wertproduktion enthalten. Weitere Informationen finden Sie unter [Attributbasierte Zugriffskontrolle mit Stichwörtern](#) und Organizations. AWS

JSON

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

Erstellen von IAM-Benutzern und Amazon-SQS-Warteschlangen

In den folgenden Beispielen wird erklärt, wie eine ABAC-Richtlinie zur Steuerung des Zugriffs auf Amazon SQS mithilfe von und erstellt wird. AWS-Managementkonsole CloudFormation

Mit dem AWS-Managementkonsole

Erstellen eines IAM-Benutzers

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter <https://console.aws.amazon.com/iam/>
2. Wählen Sie im linken Navigationsbereich Benutzer aus.
3. Wählen Sie Benutzer hinzufügen und geben Sie einen Namen in das Textfeld Benutzername ein.
4. Wählen Sie das Feld Zugriffsschlüssel – Programmatischer Zugriff und Weiter: Berechtigungen aus.
5. Wählen Sie Weiter: Tags aus.
6. Fügen Sie den Tag-Schlüssel als `environment` und den Tag-Wert als `beta` hinzu.
7. Wählen Sie Weiter:Prüfung und dann Benutzer erstellen aus.
8. Speichern Sie Ihre Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel an einem sicheren Ort.

IAM-Benutzerberechtigungen hinzufügen

1. Wählen Sie den IAM-Benutzer aus, den Sie erstellt haben.
2. Wählen Sie Inline-Richtlinie hinzufügen.
3. Fügen Sie auf der Registerkarte JSON die folgende Richtlinie ein.
4. Wählen Sie Richtlinie prüfen.
5. Wählen Sie Richtlinie erstellen aus.

Verwenden AWS CloudFormation

Verwenden Sie die folgende CloudFormation Beispieldvorlage, um einen IAM-Benutzer mit einer angehängten Inline-Richtlinie und einer Amazon SQS SQS-Warteschlange zu erstellen:

```
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"
```

Testen der attributbasierten Zugriffskontrolle in Amazon SQS

Die folgenden Beispiele zeigen, wie Sie die attributbasierte Zugriffssteuerung in Amazon SQS testen können.

Erstellen einer Warteschlange, bei der der Tag-Schlüssel auf „environment“ und der Tag-Wert auf „prod“ gesetzt ist

Führen Sie diesen AWS CLI-Befehl aus, um zu testen, wie die Warteschlange erstellt wird, wobei der Tag-Schlüssel auf environment und der Tag-Wert auf prod gesetzt ist. Wenn Sie nicht über AWS CLI verfügen, können Sie es [herunterladen und für Ihren Computer konfigurieren](#).

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

Sie erhalten eine AccessDenied-Fehlermeldung vom Amazon-SQS-Endpunkt:

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

Dies liegt daran, dass der Tag-Wert des IAM-Benutzers nicht mit dem im [CreateQueue](#)-API-Aufruf übergebenen Tag übereinstimmt. Denken Sie daran, dass wir dem IAM-Benutzer ein Tag zugewiesen haben, bei dem der Schlüssel auf environment und der Wert auf beta gesetzt ist.

Erstellen Sie eine Warteschlange, bei der der Tag-Schlüssel auf „Umgebung“ und der Tag-Wert auf „Beta“ gesetzt ist

Führen Sie diesen CLI-Befehl aus, um zu testen, wie eine Warteschlange erstellt wird, bei der der Tag-Schlüssel auf `environment` und der Tag-Wert auf `beta` gesetzt sind.

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

Sie erhalten eine Meldung, die die erfolgreiche Erstellung der Warteschlange bestätigt, ähnlich der folgenden.

```
{
  "QueueUrl": "<queueUrl>"
}
```

Senden einer Mitteilung an eine Warteschlange

Führen Sie diesen CLI-Befehl aus, um das Senden einer Nachricht an eine Warteschlange zu testen.

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

Die Antwort zeigt eine erfolgreiche Nachrichtenzustellung an die Amazon-SQS-Warteschlange. Die IAM-Benutzerberechtigung ermöglicht Ihnen, eine Nachricht an eine Warteschlange zu senden, die über ein `beta`-Tag verfügt. Die Antwort beinhaltet `MD5ofMessageBody` und `MessageId` mit der Nachricht.

```
{
  "MD5ofMessageBody": "<MD5ofMessageBody>",
  "MessageId": "<MessageId>"
}
```

Konfiguration von Warteschlangenparametern mit der Amazon SQS SQS-Konsole

Beim [Erstellen](#) oder [Bearbeiten](#) einer Warteschlange können Sie die folgenden Parameter konfigurieren:

- **Sichtbarkeitszeitbeschränkung** – Der Zeitraum, für den eine Nachricht, die aus einer Warteschlange (von einem Verbraucher) empfangen wurde, für die anderen Nachrichtenkonsumenten nicht sichtbar ist. Weitere Informationen finden Sie unter [Sichtbarkeitszeitbeschränkung](#).

 Note

Wenn Sie die Sichtbarkeitszeitbeschränkung mithilfe der Konsole konfigurieren, wird der Timeout-Wert für alle Nachrichten in der Warteschlange konfiguriert. Um das Timeout für einzelne oder mehrere Nachrichten zu konfigurieren, müssen Sie einen der AWS SDKs verwenden.

- **Aufbewahrungszeitraum für Nachrichten** – Der Zeitraum, für den Amazon SQS Nachrichten aufbewahrt, die in der Warteschlange verbleiben. Standardmäßig werden Nachrichten in einer Warteschlange vier Tage lang beibehalten. Sie können eine Warteschlange so konfigurieren, dass sie Nachrichten bis zu 14 Tage aufbewahrt. Weitere Informationen finden Sie unter [Aufbewahrungszeitraum für Nachrichten](#).
- **Zustellungsverzögerung** – Der Zeitraum, um den Amazon SQS die Zustellung der Nachricht verzögert, die der Warteschlange hinzugefügt wird. Weitere Informationen finden Sie unter [Zustellungsverzögerung](#).
- **Maximale Nachrichtengröße** – Die maximale Nachrichtengröße für diese Warteschlange. Weitere Informationen finden Sie unter [Maximale Nachrichtengröße](#).
- **Wartezeit für den Empfang von Nachrichten** – Die maximale Zeit, die Amazon SQS darauf wartet, dass Nachrichten verfügbar werden, nachdem die Warteschlange eine Empfangsanforderung erhalten hat. Weitere Informationen finden Sie unter [Kurz- und Langabfragen in Amazon SQS](#).
- **Inhaltsbasierte Deduplizierung aktivieren** — Amazon SQS kann automatisch eine Deduplizierung IDs basierend auf dem Nachrichtentext erstellen. Weitere Informationen finden Sie unter [Amazon-SQS-FIFO-Warteschlangen](#).
- **FIFO mit hohem Durchsatz aktivieren** – Wird verwendet, um einen hohen Durchsatz für Nachrichten in der Warteschlange zu aktivieren. Wenn Sie diese Option wählen, werden die zugehörigen Optionen ([Deduplizierungsbereich](#) und [FIFO-Durchsatz-Limit](#)) auf die erforderlichen Einstellungen geändert, um einen hohen Durchsatz für FIFO-Warteschlangen zu aktivieren. Weitere Informationen erhalten Sie unter [Hoher Durchsatz für FIFO-Warteschlangen in Amazon SQS](#) und [Amazon SQS SQS-Nachrichtenkontingente](#).
- **Redrive-Richtlinie**: definiert, welche Quellwarteschlangen diese Warteschlange als Warteschlange für unzustellbare Nachrichten verwenden können. Weitere Informationen finden Sie unter [Verwenden von Warteschlangen für unzustellbare Briefe in Amazon SQS](#).

So konfigurieren Sie Warteschlangenparameter für eine vorhandene Warteschlange (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus. Wählen Sie eine Warteschlange aus und klicken Sie auf Bearbeiten.
3. Scrollen Sie zum Abschnitt Konfiguration.
4. Geben Sie für Sichtbarkeitszeitbeschränkung die Dauer und die Einheiten ein. Der Bereich liegt zwischen 0 Sekunden und 12 Stunden. Der Standardwert ist 30 Sekunden.
5. Geben Sie unter Aufbewahrungszeitraum für Nachrichten die Dauer und die Einheiten ein. Der gültige Bereich beträgt 1 Minute bis 14 Tage. Der Standardwert ist 4 Tage.
6. Geben Sie für eine Standard-Warteschlange einen Wert für die Wartezeit für den Empfang von Nachrichten ein. Der Bereich liegt zwischen 0 und 20 Sekunden. Der Standardwert ist 0 Sekunden, der [kurze Abfragen](#) festlegt. Jeder Wert ungleich Null führt zu einer langen Abfrage.
7. Geben Sie für Zustellungsverzögerung die Dauer und die Einheiten ein. Der Bereich liegt zwischen 0 Sekunden und 15 Minuten. Der Standardwert ist 0 Sekunden.
8. Geben Sie für Maximale Nachrichtengröße einen Wert ein. Der Bereich reicht von 1 KiB bis 1024 KiB. Der Standardwert ist 1024 KiB.
9. Wählen Sie für eine FIFO-Warteschlange Aktivieren der inhaltsbasierten Deduplizierung, um die inhaltsbasierte Deduplizierung zu aktivieren. Die Standardeinstellung ist deaktiviert.
10. (Optional) Damit eine FIFO-Warteschlange einen höheren Durchsatz für das Senden und Empfangen von Nachrichten in der Warteschlange ermöglicht, wählen Sie FIFO mit hohem Durchsatz aktivieren.

Wenn Sie diese Option wählen, werden die zugehörigen Optionen (Deduplizierungsbereich und FIFO-Durchsatz-Limit) auf die erforderlichen Einstellungen geändert, um einen hohen Durchsatz für FIFO-Warteschlangen zu aktivieren. Wenn Sie Einstellungen ändern, die für die Verwendung von FIFO mit hohem Durchsatz erforderlich sind, ist der normale Durchsatz für die Warteschlange wirksam und die Deduplizierung erfolgt wie angegeben. Weitere Informationen erhalten Sie unter [Hoher Durchsatz für FIFO-Warteschlangen in Amazon SQS](#) und [Amazon SQS SQS-Nachrichtenkontingente](#).

11. Wählen Sie für die Redrive-Zulassungsrichtlinie die Option Aktiviert aus. Wählen Sie eine der folgenden Optionen aus: Alle zulassen (Standard), Nach Warteschlange oder Alle verweigern. Wenn Sie Nach Warteschlange wählen, geben Sie eine Liste mit bis zu 10 Quellwarteschlangen nach dem Amazon-Ressourcennamen (ARN) an.

12. Wenn Sie mit der Konfiguration der Warteschlangenparameter fertig sind, wählen Sie Speichern.

Konfiguration einer Zugriffsrichtlinie in Amazon SQS

Wenn Sie eine Warteschlange [bearbeiten](#), können Sie deren Zugriffsrichtlinie so konfigurieren, dass gesteuert wird, wer mit der Warteschlange interagieren kann.

- Die Zugriffsrichtlinie definiert, welche Konten, Benutzer und Rollen berechtigt sind, auf die Warteschlange zuzugreifen.
- Sie gibt die zulässigen Aktionen an, z. B. [SendMessageReceiveMessage](#), oder [DeleteMessage](#).
- Standardmäßig ist nur der Besitzer der Warteschlange berechtigt, Nachrichten zu senden und zu empfangen.

Um die Zugriffsrichtlinie für eine bestehende Warteschlange (Konsole) zu konfigurieren

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie eine Warteschlange aus und klicken Sie auf Bearbeiten.
4. Scrollen Sie zum Abschnitt Zugriffsrichtlinie.
5. Bearbeiten Sie die Anweisungen zur Zugriffsrichtlinie im Eingabefeld. Weitere Informationen zu den Anweisungen zur Zugriffsrichtlinie finden Sie unter [Identity and Access Management in Amazon SQS](#).
6. Wenn Sie mit der Konfiguration der Zugriffsrichtlinie fertig sind, wählen Sie Speichern.

Konfiguration der serverseitigen Verschlüsselung für eine Warteschlange mithilfe von SQS-verwalteten Verschlüsselungsschlüsseln

Zusätzlich zur [Standardoption](#) für verwaltete serverseitige Verschlüsselung (SSE) von Amazon SQS können Sie mit Amazon SQS Managed SSE (SSE-SQS) eine benutzerdefinierte verwaltete serverseitige Verschlüsselung erstellen, die von SQS verwaltete Verschlüsselungsschlüssel verwendet, um sensible Daten zu schützen, die über Nachrichtenwarteschlangen gesendet werden.

Mit SSE-SQS müssen Sie keine Verschlüsselungsschlüssel erstellen und verwalten oder Ihren Code ändern, um Ihre Daten zu verschlüsseln. SSE-SQS ermöglicht Ihnen die sichere Übertragung von Daten und hilft Ihnen dabei, strenge Verschlüsselungsvorschriften und gesetzliche Anforderungen ohne zusätzliche Kosten zu erfüllen.

SSE-SQS schützt Daten im Ruhezustand mit 256-Bit Advanced Encryption Standard (AES-256). SSE verschlüsselt Nachrichten, sobald sie bei Amazon SQS eingehen. Amazon SQS speichert Nachrichten in verschlüsselter Form und entschlüsselt sie nur, wenn sie an einen autorisierten Verbraucher gesendet werden.

 Note

- Die SSE-Standardoption ist nur wirksam, wenn Sie eine Warteschlange ohne Angabe von Verschlüsselungsattributen erstellen.
- Mit Amazon SQS können Sie die gesamte Warteschlangenverschlüsselung ausschalten. Wenn Sie KMS-SSE ausschalten, wird SQS-SSE daher nicht automatisch aktiviert. Wenn Sie SQS-SSE nach dem Ausschalten von KMS-SSE aktivieren möchten, müssen Sie der Anfrage eine Attributänderung hinzufügen.

So konfigurieren Sie die SSE-SQS-Verschlüsselung für eine Warteschlange (Konsole)

 Note

Jede neue Warteschlange, die mit dem HTTP-Endpunkt (ohne TLS) erstellt wurde, aktiviert standardmäßig keine SSE-SQS-Verschlüsselung. Es ist eine bewährte Sicherheitsmethode, Amazon-SQS-Warteschlangen mit HTTPS- oder [Signature Version 4](#)-Endpunkten zu erstellen.

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie eine Warteschlange und anschließend Bearbeiten aus.
4. Erweitern Sie Verschlüsselung.
5. Wählen Sie unter Serverseitige Verschlüsselung Aktivieren aus.

Note

Wenn SSE aktiviert ist, werden anonyme SendMessage- und ReceiveMessage-Anfragen an die verschlüsselte Warteschlange abgewiesen. Die bewährten Sicherheitsmethoden von Amazon SQS raten davon ab, anonyme Anfragen zu verwenden. Wenn Sie anonyme Anfragen an eine Amazon-SQS-Warteschlange senden möchten, stellen Sie sicher, dass SSE deaktiviert ist.

6. Wählen Sie Amazon-SQS-Schlüssel (SSE-SQS) aus. Für die Nutzung dieser Option fallen keine zusätzlichen Kosten an.
7. Wählen Sie Speichern.

Konfiguration der serverseitigen Verschlüsselung für eine Warteschlange mithilfe der Amazon SQS SQS-Konsole

Um die Daten in den Nachrichten einer Warteschlange zu schützen, hat Amazon SQS die serverseitige Verschlüsselung (SSE) standardmäßig für alle neu erstellten Warteschlangen aktiviert. Amazon SQS ist in den Amazon Web Services Key Management Service (Amazon Web Services KMS) integriert, um [KMS-Schlüssel](#) für die serverseitige Verschlüsselung (SSE) zu verwalten. Für weitere Informationen zur Nutzung von SSE siehe [Verschlüsselung im Ruhezustand in Amazon SQS](#).

Für den KMS-Schlüssel, den Sie Ihrer Warteschlange zuweisen, muss eine Schlüsselrichtlinie gelten, die Berechtigungen für alle Prinzipale beinhaltet, die zur Nutzung der Warteschlange berechtigt sind. Informationen finden Sie unter [Schlüsselverwaltung](#).

Wenn Sie nicht der Besitzer des KMS-Schlüssels sind oder wenn Sie sich mit einem Konto anmelden, das über keine `kms:ListAliases`- und `kms:DescribeKey`-Berechtigungen verfügt, können Sie auf der Amazon-SQS-Konsole keine Informationen über den KMS aufrufen. Bitten Sie den Inhaber des KMS, Ihnen diese Berechtigungen zu erteilen. Weitere Informationen finden Sie unter [Schlüsselverwaltung](#).

Wenn Sie eine Warteschlange [erstellen](#) oder [bearbeiten](#), können Sie SSE-KMS konfigurieren.

So konfigurieren Sie SSE-KMS für eine vorhandene Warteschlange (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.

2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie eine Warteschlange und anschließend Bearbeiten aus.
4. Erweitern Sie Verschlüsselung.
5. Wählen Sie unter Serverseitige Verschlüsselung Aktivieren aus.

 Note

Wenn SSE aktiviert ist, werden anonyme SendMessage- und ReceiveMessage-Anfragen an die verschlüsselte Warteschlange abgewiesen. Die bewährten Sicherheitsmethoden von Amazon SQS raten davon ab, anonyme Anfragen zu verwenden. Wenn Sie anonyme Anfragen an eine Amazon-SQS-Warteschlange senden möchten, stellen Sie sicher, dass SSE deaktiviert ist.

6. Wählen Sie AWS Key Management Service-Schlüssel (SSE-KMS) aus.

In der Konsole werden die Beschreibung, das Konto und der KMS-Schlüssel-ARN des KMS-Schlüssels angezeigt.

7. Geben Sie die KMS-Schlüssel-ID für die Warteschlange an. Weitere Informationen finden Sie unter [Wichtige Begriffe](#).
 - a. Wählen Sie die Option KMS-Schlüsselalias auswählen.
 - b. Der Standardschlüssel ist der von Amazon Web Services verwaltete KMS-Schlüssel für Amazon SQS. Um diesen Schlüssel zu verwenden, wählen Sie ihn aus der KMS-Schlüsselliste aus.
 - c. Um einen benutzerdefinierten KMS-Schlüssel aus Ihrem Amazon-Web-Services-Konto zu verwenden, wählen Sie ihn aus der KMS-Schlüsselliste aus. Anweisungen zum Erstellen benutzerdefinierter KMS-Schlüssel finden Sie unter [Erstellen von Schlüsseln](#) im Amazon-Web-Services-Key-Management-Service-Entwicklerhandbuch.
 - d. Um einen benutzerdefinierten KMS-Schlüssel, der nicht in der Liste enthalten ist, oder einen benutzerdefinierten KMS-Schlüssel von einem anderen Amazon-Web-Services-Konto zu verwenden, wählen Sie KMS-Schlüsselalias eingeben aus und geben Sie den Amazon-Ressourcennamen (ARN) des KMS-Schlüssels ein.
8. (Optional) Geben Sie für den Zeitraum der Wiederverwendung von Datenschlüsseln einen Wert zwischen 1 Minute und 24 Stunden an. Der Standardwert ist 5 Minuten. Weitere Informationen finden Sie unter [Grundlegendes zum Wiederverwendungszeitraum für den Datenschlüssel](#).
9. Wenn Sie mit der Konfiguration von SSE-KMS fertig sind, wählen Sie Speichern.

Konfiguration von Kostenzuweisungs-Tags für eine Warteschlange mithilfe der Amazon SQS SQS-Konsole

Um Ihre Amazon SQS SQS-Warteschlangen zu organisieren und zu identifizieren, können Sie Kostenzuweisungs-Tags hinzufügen. Weitere Informationen finden Sie unter [Amazon-SQS-Kostenzuordnungs-Tags](#).

- Auf der Registerkarte Tagging auf der Detailseite werden die Tags der Warteschlange angezeigt.
- Sie können Tags hinzufügen oder ändern, wenn Sie eine Warteschlange [erstellen](#) oder [bearbeiten](#).

So konfigurieren Sie Tags für eine vorhandene Warteschlange (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie eine Warteschlange aus und klicken Sie auf Bearbeiten.
4. Scrollen Sie zum Abschnitt Tags.
5. Fügen Sie die Warteschlangen-Tags hinzu, ändern oder entfernen Sie sie:
 - a. Um einen Tag hinzuzufügen, wählen Sie Neuen Tag hinzufügen, geben Sie einen Schlüssel und einen Wert ein und wählen Sie dann Änderungen anwenden.
 - b. Um einen Tag zu aktualisieren, ändern Sie dessen Schlüssel und Wert.
 - c. Wählen Sie zum Entfernen eines Tags neben einem Schlüssel-Wert-Paar Tag entfernen aus.
6. Wenn Sie mit der Konfiguration der Tags fertig sind, wählen Sie Speichern.

Abonnieren einer Warteschlange für ein Amazon SNS SNS-Thema mithilfe der Amazon SQS SQS-Konsole

Sie können eine oder mehrere Amazon SQS SQS-Warteschlangen für ein Amazon SNS SNS-Thema abonnieren. Wenn Sie eine Nachricht zu einem Thema veröffentlichen, sendet Amazon SNS die Nachricht an jede abonnierte Warteschlange. Amazon SQS verwaltet das Abonnement und kümmert sich um die erforderlichen Berechtigungen. Weitere Informationen finden Sie unter [Was ist Amazon SNS?](#) im Entwicklerhandbuch für Amazon Simple Notification Service.

Wenn Sie eine Amazon SQS-Warteschlange für ein Amazon SNS-Thema abonnieren, verwendet Amazon SNS HTTPS, um Nachrichten an Amazon SQS weiterzuleiten. Weitere Informationen zur Verwendung von Amazon SNS mit verschlüsselten Amazon-SQS-Warteschlangen finden Sie unter [Konfigurieren Sie KMS-Berechtigungen für AWS service](#).

 Important

Amazon SQS unterstützt maximal 20 Anweisungen für jede Zugriffsrichtlinie. Das Abonnieren eines Amazon-SNS-Themas fügt eine solche Anweisung hinzu. Eine Überschreitung dieser Zahl führt dazu, dass das Abonnement für das Thema nicht zugestellt werden kann.

Um eine Warteschlange für ein Amazon SNS SNS-Thema zu abonnieren (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie in der Liste der Warteschlangen die Warteschlange aus, für die das Amazon-SNS-Thema abonniert werden soll.
4. Wählen Sie im Menü Actions (Aktionen) die Option Subscribe to Amazon SNS topic (Amazon-SNS-Thema abonnieren) aus.
5. Wählen Sie im Menü Geben Sie ein Amazon SNS SNS-Thema an, das für diese Warteschlange verfügbar ist, das Amazon SNS SNS-Thema für Ihre Warteschlange aus.

Wenn das SNS-Thema nicht aufgeführt ist, wählen Sie Enter Amazon SNS topic ARN und geben Sie dann den Amazon-Ressourcennamen (ARN) des Themas ein.

6. Wählen Sie Speichern.
7. Um das Abonnement zu verifizieren, veröffentlichen Sie eine Nachricht zum Thema und sehen Sie sich die Nachricht in der Warteschlange an. Weitere Informationen finden Sie unter [Veröffentlichen von Amazon-SNS-Nachrichten](#) im Amazon-Simple-Notification-Service-Entwicklerhandbuch.

Kontoübergreifende Abonnements

Wenn sich Ihre Amazon SQS SQS-Warteschlange und Ihr Amazon SNS SNS-Thema unterscheiden AWS-Konten, sind zusätzliche Berechtigungen erforderlich.

Eigentümer des Themas (Konto A)

Ändern Sie die Zugriffsrichtlinie des Amazon SNS SNS-Themas, sodass die Amazon SQS SQS-Warteschlangen AWS-Konto abonnieren können. Beispiel für eine Richtlinienerklärung:

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

Diese Richtlinie ermöglicht 111122223333 das Abonnieren eines KontosMyTopic.

Besitzer der Warteschlange (Konto B)

Ändern Sie die Zugriffsrichtlinie der Amazon SQS SQS-Warteschlange, sodass das Amazon SNS SNS-Thema Nachrichten senden kann. Beispiel für eine Richtlinienerklärung:

```
{
  "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" }
  }
}
```

Diese Richtlinie ermöglicht MyTopic das Senden von Nachrichten anMyQueue.

Regionsübergreifende Abonnements

Um ein Amazon SNS SNS-Thema in einem anderen Format zu abonnieren AWS-Region, stellen Sie sicher, dass:

- Die Zugriffsrichtlinie des Amazon SNS SNS-Themas ermöglicht regionsübergreifende Abonnements.
- Die Zugriffsrichtlinie der Amazon SQS SQS-Warteschlange ermöglicht es dem Amazon SNS SNS-Thema, Nachrichten zwischen Regionen zu senden.

Weitere Informationen finden Sie unter [Senden von Amazon SNS SNS-Nachrichten an eine Amazon SQS SQS-Warteschlange oder AWS Lambda Funktion in einer anderen Region](#) im Amazon Simple Notification Service Developer Guide.

Konfiguration einer Amazon SQS SQS-Warteschlange zum Auslösen einer Funktion AWS Lambda

Sie können eine Lambda-Funktion verwenden, um Nachrichten aus einer Amazon SQS SQS-Warteschlange zu verarbeiten. Lambda fragt die Warteschlange ab und ruft Ihre Funktion synchron auf, wobei ein Stapel von Nachrichten als Ereignis übergeben wird.

Timeout für Sichtbarkeit konfigurieren

Stellen Sie das Sichtbarkeits-Timeout der Warteschlange auf mindestens das Sechsfache des [Funktions-Timeouts](#) ein. Dadurch wird sichergestellt, dass Lambda genügend Zeit hat, um es erneut zu versuchen, falls eine Funktion während der Verarbeitung eines vorherigen Batches gedrosselt wird.

Verwenden einer Warteschlange mit unerlaubten Briefen (DLQ)

Geben Sie eine Warteschlange für unzustellbare Briefe an, um Nachrichten zu erfassen, die von der Lambda-Funktion nicht verarbeitet werden können.

Umgang mit mehreren Warteschlangen und Funktionen

Eine Lambda-Funktion kann mehrere Warteschlangen verarbeiten, indem sie für jede Warteschlange eine separate Ereignisquelle erstellt. Sie können derselben Warteschlange auch mehrere Lambda-Funktionen zuordnen.

Berechtigungen für verschlüsselte Warteschlangen

Wenn Sie eine verschlüsselte Warteschlange mit einer Lambda-Funktion verknüpfen, Lambda aber keine Nachrichten abfragt, fügen Sie die `kms:Decrypt`-Berechtigung zu Ihrer Lambda-Ausführungsrolle hinzu.

Einschränkungen

Die Warteschlange und die Lambda-Funktion müssen identisch AWS-Region sein.

Eine [verschlüsselte Warteschlange](#), die den Standardschlüssel (AWS verwalteter KMS-Schlüssel für Amazon SQS) verwendet, kann keine Lambda-Funktion in einer anderen aufrufen. AWS-Konto

Einzelheiten zur Implementierung finden Sie unter [Using AWS Lambda with Amazon SQS](#) im AWS Lambda Developer Guide.

Voraussetzungen

Um Lambda-Funktions-Auslöser zu konfigurieren, müssen Sie die folgenden Anforderungen erfüllen:

- Wenn Sie einen Benutzer verwenden, muss Ihre Amazon-SQS-Rolle die folgenden Berechtigungen einschließen:
 - `lambda:CreateEventSourceMapping`
 - `lambda:ListEventSourceMappings`
 - `lambda:ListFunctions`
- Die Lambda-Ausführungsrolle muss die folgenden Berechtigungen enthalten:
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `sqs:ReceiveMessage`
- Wenn Sie eine verschlüsselte Warteschlange mit einer Lambda-Funktion verknüpfen, fügen Sie die `kms:Decrypt`-Berechtigung zur Lambda-Ausführungsrolle hinzu.

Weitere Informationen finden Sie unter [Übersicht über die Zugriffsverwaltung in Amazon SQS](#).

So konfigurieren Sie eine Warteschlange, um eine Lambda-Funktion auszulösen (Konsole)

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie auf der Seite Warteschlange die zu konfigurierende Warteschlange aus.
4. Wählen Sie auf der Seite der Warteschlange die Registerkarte Lambda-Auslöser aus.
5. Wählen Sie auf der Seite Lambda-Auslöser einen Lambda-Auslöser aus.

Wenn die Liste den benötigten Lambda-Auslöser nicht enthält, wählen Sie Lambda-Funktions-Auslöser konfigurieren. Geben Sie den Amazon-Ressourcennamen (ARN) der Lambda-Funktion ein oder wählen Sie eine vorhandene Ressource aus. Wählen Sie dann Speichern.

6. Wählen Sie Speichern. In der Konsole wird Konfiguration gespeichert und die Seite Details für die Warteschlange angezeigt.

Auf der Seite Details werden auf der Registerkarte Lambda-Auslöser die Lambda-Funktion und ihr Status angezeigt. Es dauert etwa 1 Minute, bis die Lambda-Funktion der Warteschlange zugeordnet wird.

7. Zum Überprüfen der Ergebnisse der Konfiguration können Sie [eine Nachricht an Ihre Warteschlange senden](#) und dann die ausgelöste Lambda-Funktion in der Lambda-Konsole anzeigen.

Automatisieren von Benachrichtigungen von AWS Services an Amazon SQS mithilfe von Amazon EventBridge

EventBridge Mit Amazon können Sie Ereignisse wie Anwendungsprobleme oder Ressourcenänderungen nahezu in Echtzeit automatisieren AWS-Services und darauf reagieren.

- Sie können Regeln erstellen, um bestimmte Ereignisse zu filtern und automatisierte Aktionen zu definieren, wenn eine Regel einem Ereignis entspricht.
- EventBridge unterstützt mehrere Ziele, einschließlich Amazon SQS SQS-Standard- und FIFO-Warteschlangen, die Ereignisse im JSON-Format empfangen.

Weitere Informationen finden Sie unter [EventBridge Amazon-Ziele](#) im [EventBridge Amazon-Benutzerhandbuch](#).

Senden einer Nachricht mit Attributen mithilfe von Amazon SQS

Für Standard- und FIFO-Warteschlangen können Sie strukturierte Metadaten zu Nachrichten hinzufügen, darunter Zeitstempel, Geodaten, Signaturen und Kennungen. Weitere Informationen finden Sie unter [Amazon-SQS-Nachrichtenattribute](#).

Um mit der Amazon SQS SQS-Konsole eine Nachricht mit Attributen an eine Warteschlange zu senden

1. Öffnen Sie die Amazon SQS SQS-Konsole unter <https://console.aws.amazon.com/sqs/>.
2. Wählen Sie im Navigationsbereich Queues (Warteschlangen) aus.
3. Wählen Sie auf der Seite Warteschlangen eine Warteschlange aus.
4. Wählen Sie Nachrichten senden und empfangen.

5. Geben Sie die Nachrichtenattributparameter ein.
 - a. Geben Sie im Textfeld „Name“ einen eindeutigen Namen mit bis zu 256 Zeichen ein.
 - b. Wählen Sie für den Attributtyp Zeichenfolge, Zahl oder Binär.
 - c. (Optional) Geben Sie einen benutzerdefinierten Datentyp ein. Sie könnten beispielsweise **byte**, **int** oder **float** als benutzerdefinierte Datentypen für Zahl hinzufügen.
 - d. Geben Sie den Wert des Nachrichtenattributs in das Textfeld ein.

▼ Message attributes - Optional [Info](#)

▼

6. Klicken Sie auf Attribut hinzufügen, um ein weiteres Nachrichtenattribut hinzuzufügen.

▼ Message attributes - Optional [Info](#)

▼

▼

7. Sie können die Attributwerte jederzeit ändern, bevor die Nachricht gesendet wird.
8. Um ein Attribut zu löschen, wählen Sie Entfernen. Um das erste Attribut zu löschen, schließen Sie Nachrichtenattribute.
9. Wenn Sie mit dem Hinzufügen von Attributen zur Nachricht fertig sind, wählen Sie Nachricht senden. Wenn Ihre Nachricht gesendet wurde, zeigt die Konsole eine Erfolgsmeldung an. Um Informationen zu den Nachrichtenattributen der gesendeten Nachricht anzuzeigen, wählen Sie Details anzeigen. Wählen Sie Fertig, um das Dialogfeld mit den Nachrichtendetails zu schließen.

Bewährte Methoden für Amazon SQS

Amazon SQS verwaltet und verarbeitet Nachrichtenwarteschlangen, sodass verschiedene Teile einer Anwendung Nachrichten zuverlässig und in großem Umfang austauschen können. In diesem Thema werden die wichtigsten bewährten Methoden für den Betrieb behandelt, darunter die Verwendung langer Abfragen zur Reduzierung leerer Antworten, die Implementierung von Warteschlangen für unberechtigte Nachrichten zur Behandlung von Verarbeitungsfehlern und die Optimierung von Warteschlangenberechtigungen aus Sicherheitsgründen.

Topics

- [Amazon SQS SQS-Fehlerbehandlung und problematische Meldungen](#)
- [Deduplizierung und Gruppierung von Amazon SQS SQS-Nachrichten](#)
- [Amazon SQS SQS-Nachrichtenverarbeitung und Timing](#)

Amazon SQS SQS-Fehlerbehandlung und problematische Meldungen

Dieses Thema enthält detaillierte Anweisungen zur Verwaltung und Minimierung von Fehlern in Amazon SQS, einschließlich Techniken zur Behandlung von Anforderungsfehlern, zur Erfassung problematischer Nachrichten und zur Konfiguration der Aufbewahrung von Warteschlangen für unzustellbare Nachrichten, um die Nachrichtenzuverlässigkeit zu gewährleisten.

Topics

- [Behandlung von Anforderungsfehlern in Amazon SQS](#)
- [Erfassung problematischer Nachrichten in Amazon SQS](#)
- [Einrichtung der Aufbewahrung von Warteschlangen für unzustellbare Briefe in Amazon SQS](#)

Behandlung von Anforderungsfehlern in Amazon SQS

Für den Umgang mit Anforderungsfehlern nutzen Sie eine der folgenden Strategien:

- Wenn Sie ein AWS SDK verwenden, steht Ihnen bereits eine automatische Wiederholungs- und Backoff-Logik zur Verfügung. Weitere Informationen finden Sie unter [Wiederholversuche bei Fehlern und exponentielles Backoff in AWS](#) im Allgemeine Amazon Web Services-Referenz.

- Wenn Sie die AWS SDK-Funktionen für Wiederholungen und Backoffs nicht verwenden, warten Sie eine Pause (z. B. 200 ms), bevor Sie die [ReceiveMessage](#)Aktion erneut versuchen, nachdem Sie keine Nachrichten, eine Zeitüberschreitung oder eine Fehlermeldung von Amazon SQS erhalten haben. Für die nachfolgende Verwendung der Aktion `ReceiveMessage`, mit der dieselben Ergebnisse erzielt werden, lassen Sie eine längere Pause (z. B. 400 ms) zu.

Erfassung problematischer Nachrichten in Amazon SQS

Um alle Nachrichten zu erfassen, die nicht verarbeitet werden können, und um genaue CloudWatch Messwerte zu sammeln, konfigurieren Sie eine Warteschlange für [unzustellbare Nachrichten](#).

- Die Redrive-Richtlinie leitet Nachrichten an eine Warteschlange für unzustellbare Nachrichten weiter, nachdem die Quell-Warteschlange eine Nachricht mehrfach nicht verarbeiten kann.
- Durch Verwenden einer Warteschlange für unzustellbare Nachrichten wird die Anzahl der Nachrichten reduziert und das Risiko von Poison-Pill-Nachrichten gesenkt, d. h. Nachrichten, die empfangen, aber nicht verarbeitet werden können.
- Die Aufnahme einer Giftpillen-Nachricht in eine Warteschlange kann die [ApproximateAgeOfOldestMessage](#) CloudWatch Metrik verfälschen, da das falsche Alter der Giftpillen-Nachricht angegeben wird. Das Konfigurieren einer Warteschlange für unzustellbare Nachrichten hilft, Fehlalarme bei der Verwendung dieser Metrik zu vermeiden.

Einrichtung der Aufbewahrung von Warteschlangen für unzustellbare Briefe in Amazon SQS

Bei Standard-Warteschlangen basiert der Ablauf einer Nachricht immer auf dem ursprünglichen Enqueue-Zeitstempel. Wenn eine Nachricht in eine Warteschlange für unzustellbare Nachrichten verschoben wird, bleibt der Enqueue-Zeitstempel unverändert. Die `ApproximateAgeOfOldestMessage`-Metrik gibt an, wann die Nachricht in die Warteschlange für unzustellbare Nachrichten verschoben wurde, und nicht, wann die Nachricht ursprünglich gesendet wurde. Nehmen wir beispielsweise an, dass sich eine Nachricht einen Tag in der ursprünglichen Warteschlange befindet, bevor sie in eine Warteschlange für unzustellbare Nachrichten verschoben wird. Wenn die Aufbewahrungsfrist der Warteschlange für unzustellbare Nachrichten 4 Tage beträgt, wird die Nachricht nach 3 Tagen aus der Warteschlange für unzustellbare Nachrichten gelöscht und das `ApproximateAgeOfOldestMessage` ist 3 Tage. Es hat sich daher bewährt, die Aufbewahrungsdauer einer Warteschlange für unzustellbare Nachrichten immer so festzulegen, dass sie länger ist als die Aufbewahrungsdauer der ursprünglichen Warteschlange.

Bei FIFO-Warteschlangen wird der Enqueue-Zeitstempel zurückgesetzt, wenn die Nachricht in eine Warteschlange für unzustellbare Nachrichten verschoben wird. Die `ApproximateAgeOfOldestMessage`-Metrik gibt an, wann die Nachricht in die Warteschlange für unzustellbare Nachrichten verschoben wird. Im obigen Beispiel wird die Nachricht nach 4 Tagen aus der Warteschlange für unzustellbare Nachrichten gelöscht und das `ApproximateAgeOfOldestMessage` ist 4 Tage.

Deduplizierung und Gruppierung von Amazon SQS SQS-Nachrichten

Dieses Thema enthält bewährte Methoden zur Sicherstellung einer konsistenten Nachrichtenverarbeitung in Amazon SQS. Es erklärt, wie man Folgendes verwendet:

- [MessageDeduplicationId](#) um doppelte Nachrichten in FIFO-Warteschlangen zu verhindern.
- [MessageGroupId](#) um die Reihenfolge der Nachrichten innerhalb verschiedener Nachrichtengruppen zu verwalten.

Topics

- [Vermeidung inkonsistenter Nachrichtenverarbeitung in Amazon SQS](#)
- [Verwenden der Nachrichtendeduplizierungs-ID in Amazon SQS](#)
- [Verwenden der Nachrichtengruppen-ID mit Amazon SQS FIFO Queues](#)
- [Verwenden der Amazon-SQS-Empfangsanforderungsversuch-ID](#)

Vermeidung inkonsistenter Nachrichtenverarbeitung in Amazon SQS

Da es sich bei Amazon SQS um ein verteiltes System handelt, ist es möglich, dass ein Konsument selbst dann keine Nachricht empfängt, wenn Amazon SQS die Nachricht als übermittelt markiert, während sie erfolgreich von einem `ReceiveMessage`-API-Methodenaufruf zurückgegeben wird. In diesem Fall vermerkt Amazon SQS die Nachricht mindestens einmal als zugestellt, obwohl der Konsument sie nie erhalten hat. Da unter diesen Bedingungen keine zusätzlichen Versuche zur Zustellung von Nachrichten unternommen werden, empfehlen wir nicht, als Anzahl der maximalen Eingänge für eine [Warteschlange für unzustellbare Nachrichten](#) „1“ einzustellen.

Verwenden der Nachrichteneduplizierungs-ID in Amazon SQS

[MessageDeduplicationId](#) ist ein Token, das nur in Amazon SQS FIFO-Warteschlangen verwendet wird, um eine doppelte Nachrichtenzustellung zu verhindern. Es stellt sicher, dass innerhalb eines 5-minütigen Deduplizierungsfensters nur eine Instanz einer Nachricht mit derselben Deduplizierungs-ID verarbeitet und zugestellt wird.

Wenn Amazon SQS bereits eine Nachricht mit einer bestimmten Deduplizierungs-ID akzeptiert hat, werden alle nachfolgenden Nachrichten mit derselben ID bestätigt, aber nicht an Verbraucher zugestellt.

Note

Amazon SQS verfolgt weiterhin die Deduplizierungs-ID, auch nachdem die Nachricht empfangen und gelöscht wurde.

Themen

- [Wann muss eine Nachrichteneduplizierungs-ID in Amazon SQS angegeben werden?](#)
- [Aktivierung der Deduplizierung für ein Einzelprodukt-/Verbrauchersystem in Amazon SQS](#)
- [Wiederherstellungsszenarien bei Ausfällen in Amazon SQS](#)
- [Konfiguration von Sichtbarkeits-Timeouts in Amazon SQS](#)

Wann muss eine Nachrichteneduplizierungs-ID in Amazon SQS angegeben werden?

In den folgenden Szenarien sollte ein Hersteller eine ID für die Nachrichteneduplizierung angeben:

- Beim Senden identischer Nachrichtentexte müssen diese als einzigartig behandelt werden.
- Stellen Sie beim Senden von Nachrichten mit demselben Inhalt, aber unterschiedlichen Nachrichtenattributen sicher, dass jede Nachricht separat verarbeitet wird.
- Wenn Nachrichten mit unterschiedlichem Inhalt gesendet werden (z. B. ein Wiederholungszähler im Nachrichtentext), Amazon SQS sie aber als Duplikate erkennen muss.

Aktivierung der Deduplizierung für ein Einzelprodukt-/Verbrauchersystem in Amazon SQS

Wenn Sie nur einen Produzenten und einen einzigen Verbraucher haben und Nachrichten einzigartig sind, weil sie eine anwendungsspezifische Nachrichten-ID im Hauptteil enthalten, gehen Sie wie folgt vor:

- Aktivieren Sie die inhaltsbasierte Deduplizierung für die Warteschlange (jede Ihrer Nachrichten hat einen eindeutigen Text). Der Produzent kann die Nachrichteneduplizierungs-ID weglassen.
- Wenn die inhaltsbasierte Deduplizierung für eine Amazon SQS FIFO-Warteschlange aktiviert ist und eine Nachricht mit einer Deduplizierungs-ID gesendet wird, überschreibt die Deduplizierungs-ID die generierte inhaltsbasierte [SendMessage](#)Deduplizierungs-ID.
- Obwohl der Konsument keine Empfangsanforderungsversuch-ID für jede Anforderung angeben muss, hat sich dies als Methode bewährt, da Wiederholungen nach Fehlschlägen schneller durchgeführt werden können.
- Sie können erneut versuchen, Anforderungen zu senden oder zu empfangen, da diese die Reihenfolge der Nachrichten in FIFO-Warteschlangen nicht stören.

Wiederherstellungsszenarien bei Ausfällen in Amazon SQS

Der Deduplizierungsprozess in FIFO-Warteschlangen ist zeitkritisch. Achten Sie beim Entwerfen Ihrer Anwendung darauf, dass sowohl der Hersteller als auch der Endverbraucher nach Client- oder Netzwerkausfällen wieder fit sind, ohne dass es zu Duplikaten oder Verarbeitungsfehlern kommt.

Überlegungen des Herstellers

- Amazon SQS erzwingt ein Deduplizierungsfenster von 5 Minuten.
- Wenn ein Producer eine [SendMessage](#)Anfrage nach 5 Minuten erneut versucht, behandelt Amazon SQS sie als neue Nachricht, wodurch möglicherweise Duplikate erstellt werden.

Überlegungen für Verbraucher

- Wenn ein Verbraucher eine Nachricht nicht verarbeitet, bevor das Sichtbarkeits-Timeout abgelaufen ist, kann es sein, dass ein anderer Verbraucher sie empfängt und verarbeitet, was zu einer doppelten Verarbeitung führt.
- Passen Sie das Sichtbarkeits-Timeout an die Bearbeitungszeit Ihrer Anwendung an.

- Verwenden Sie die [ChangeMessageVisibility](#)API, um das Timeout zu verlängern, solange eine Nachricht noch verarbeitet wird.
- Wenn eine Nachricht wiederholt nicht verarbeitet werden kann, leiten Sie sie an eine [Warteschlange mit unerlaubten Briefen \(DLQ\)](#) weiter, anstatt sie auf unbestimmte Zeit erneut verarbeiten zu lassen.
- Der Produzent muss das Deduplizierungsintervall der Warteschlange kennen. Amazon SQS verfügt über ein Deduplizierungsintervall von 5 Minuten. Wiederholungsversuche von SendMessage-Anforderungen nach Ablauf des Deduplizierungsintervalls können zu doppelten Nachrichten in der Warteschlange führen. Beispiel: Ein mobiles Gerät in einem Auto sendet Nachrichten, deren Reihenfolge wichtig ist. Wenn das Auto für einen bestimmten Zeitraum die Funkverbindung verliert, bevor eine Bestätigung eingeht, kann ein Wiederholungsversuch der Anforderungen nach Wiederherstellung der Funkverbindung zu einem Duplikat führen.
- Der Konsument muss über eine Zeitbeschränkung für die Sichtbarkeit verfügen, die das Risiko minimiert, dass Nachrichten nicht verarbeitet werden können, bevor die Zeitbeschränkung für die Sichtbarkeit abgelaufen ist. Sie können die Zeitbeschränkung für die Sichtbarkeit erweitern, während die Nachrichten verarbeitet werden, indem Sie die Aktion [ChangeMessageVisibility](#) aufrufen. Wenn die Zeitbeschränkung für die Sichtbarkeit jedoch abgelaufen ist, kann ein anderer Konsument sofort mit der Nachrichtenverarbeitung beginnen, was dazu führt, dass eine Nachricht mehrfach verarbeitet wird. Um dieses Szenario zu vermeiden, konfigurieren Sie eine [Warteschlange für unzustellbare Nachrichten](#).

Konfiguration von Sichtbarkeits-Timeouts in Amazon SQS

Um eine zuverlässige Nachrichtenverarbeitung zu gewährleisten, legen Sie das Sichtbarkeits-Timeout so fest, dass es länger ist als das AWS SDK-Lese-Timeout. Dies gilt, wenn die [ReceiveMessage](#)API sowohl für kurze als auch für lange Abfragen verwendet wird. Ein längeres Sichtbarkeits-Timeout verhindert, dass Nachrichten anderen Benutzern zur Verfügung stehen, bevor die ursprüngliche Anfrage abgeschlossen ist, wodurch das Risiko einer doppelten Verarbeitung verringert wird.

Verwenden der Nachrichtengruppen-ID mit Amazon SQS FIFO Queues

In FIFO-Warteschlangen (First-In-First-Out) [MessageGroupId](#) ist dies ein Attribut, das Nachrichten in verschiedene Gruppen unterteilt. Nachrichten innerhalb derselben Nachrichtengruppe werden immer einzeln und in strikter Reihenfolge verarbeitet, sodass sichergestellt wird, dass keine zwei

Nachrichten aus derselben Gruppe gleichzeitig verarbeitet werden. In Standardwarteschlangen `MessageGroupId` ermöglicht die Verwendung [fairer Warteschlangen](#). Wenn eine strikte Reihenfolge erforderlich ist, verwenden Sie eine FIFO-Warteschlange.

Themen

- [Verschachteln mehrerer bestellter Nachrichtengruppen in Amazon SQS](#)
- [Vermeidung doppelter Verarbeitung in einem System mit mehreren Herstellern/Verbrauchern in Amazon SQS](#)
- [Vermeiden Sie große Nachrichtenrückstände mit derselben Nachrichtengruppen-ID in Amazon SQS](#)
- [Vermeiden Sie die Wiederverwendung derselben Nachrichtengruppen-ID mit virtuellen Warteschlangen in Amazon SQS](#)

Verschachteln mehrerer bestellter Nachrichtengruppen in Amazon SQS

Um mehrere geordnete Nachrichtengruppen innerhalb einer einzigen FIFO-Warteschlange zu verschachteln, weisen Sie jeder Gruppe eine [MessageGroupId](#) zu (z. B. Sitzungsdaten für verschiedene Benutzer). Auf diese Weise können mehrere Benutzer gleichzeitig aus der Warteschlange lesen und gleichzeitig sicherstellen, dass Nachrichten innerhalb derselben Gruppe der Reihe nach verarbeitet werden.

Wenn eine Nachricht mit einer bestimmten `MessageGroupId` Nachricht verarbeitet wird und unsichtbar ist, kann kein anderer Verbraucher Nachrichten aus derselben Gruppe verarbeiten, bis das Sichtbarkeits-Timeout abgelaufen ist oder die Nachricht gelöscht wird.

Vermeidung doppelter Verarbeitung in einem System mit mehreren Herstellern/Verbrauchern in Amazon SQS

In einem System mit hohem Durchsatz und niedriger Latenz, in dem die Reihenfolge der Nachrichten keine Priorität hat, können Produzenten jeder Nachricht eine eindeutige [MessageGroupId](#) Nummer zuweisen. Dadurch wird sichergestellt, dass Amazon SQS FIFO-Warteschlangen Duplikate eliminieren, selbst in einer Konfiguration mit mehreren Produzenten/mehreren Verbrauchern. Dieser Ansatz verhindert zwar doppelte Nachrichten, garantiert jedoch nicht die Reihenfolge der Nachrichten, da jede Nachricht als eigene unabhängige Gruppe behandelt wird.

In jedem System mit mehreren Herstellern und Verbrauchern besteht immer die Gefahr einer doppelten Zustellung. Wenn ein Verbraucher eine Nachricht nicht verarbeitet, bevor das

Sichtbarkeits-Timeout abgelaufen ist, stellt Amazon SQS die Nachricht wieder zur Verfügung, sodass sie möglicherweise von einem anderen Verbraucher abgeholt werden kann. Um dies zu vermeiden, stellen Sie sicher, dass die Einstellungen für die Nachrichtenbestätigung und das Sichtbarkeits-Timeout auf der Grundlage der Verarbeitungszeit korrekt sind.

Vermeiden Sie große Nachrichtenrückstände mit derselben Nachrichtengruppen-ID in Amazon SQS

FIFO-Warteschlangen unterstützen maximal 120.000 Nachrichten während der Übertragung (Nachrichten, die von einem Verbraucher empfangen, aber noch nicht gelöscht wurden). Wenn dieses Limit erreicht wird, gibt Amazon SQS keinen Fehler zurück, aber die Verarbeitung kann beeinträchtigt werden. Sie können eine Erhöhung über dieses Limit hinaus beantragen, indem Sie sich an den [AWS Support](#) wenden.

FIFO-Warteschlangen scannen die ersten 120.000 Nachrichten, um die verfügbaren Nachrichtengruppen zu ermitteln. Wenn sich in einer einzelnen Nachrichtengruppe ein großer Rückstand ansammelt, bleiben Nachrichten von anderen Gruppen, die später gesendet werden, blockiert, bis der Rückstand verarbeitet ist.

Note

Ein Nachrichtenrückstand kann entstehen, wenn ein Verbraucher eine Nachricht wiederholt nicht verarbeiten kann. Dies kann auf Probleme mit dem Nachrichteninhalt oder auf Fehler auf Kundenseite zurückzuführen sein. Um Verzögerungen bei der Nachrichtenverarbeitung zu vermeiden, konfigurieren Sie eine [Warteschlange für unzustellbare Nachrichten, sodass](#) unverarbeitete Nachrichten nach mehreren fehlgeschlagenen Versuchen verschoben werden. Dadurch wird sichergestellt, dass andere Nachrichten in derselben Nachrichtengruppe verarbeitet werden können, wodurch Systemengpässe vermieden werden.

Vermeiden Sie die Wiederverwendung derselben Nachrichtengruppen-ID mit virtuellen Warteschlangen in Amazon SQS

Wenn Sie virtuelle Warteschlangen mit einer gemeinsam genutzten Host-Warteschlange verwenden, vermeiden Sie es, dieselben [MessageGroupId](#) in verschiedenen virtuellen Warteschlangen wiederzuverwenden. Wenn sich mehrere virtuelle Warteschlangen dieselbe Host-Warteschlange teilen und Nachrichten derselben enthalten, können sich diese Nachrichten gegenseitig blockieren [MessageGroupId](#), wodurch eine effiziente Verarbeitung verhindert wird. Um eine

reibungslose Nachrichtenverarbeitung zu gewährleisten, weisen Sie Nachrichten in verschiedenen virtuellen Warteschlangen eindeutige `MessageGroupId` Werte zu.

Verwenden der Amazon-SQS-Empfangsanforderungsversuch-ID

Die ID des Versuchs einer Empfangsanforderung ist ein eindeutiges Token, das zur Deduplizierung von [ReceiveMessage](#)-Aufrufen in Amazon SQS verwendet wird. Bei einem Netzwerkausfall oder Verbindungsproblem zwischen Ihrer Anwendung und Amazon SQS empfiehlt es sich, Folgendes zu tun:

- Geben Sie bei einem Aufruf eine ID für den Versuch, eine Empfangsanfrage zu erhalten, `anReceiveMessage`.
- Versuchen Sie es erneut mit derselben ID für den Versuch, eine Empfangsanfrage zu empfangen, falls der Vorgang fehlschlägt.

Amazon SQS SQS-Nachrichtenverarbeitung und Timing

Dieses Thema bietet umfassende Anleitungen zur Optimierung der Geschwindigkeit und Effizienz der Nachrichtenverarbeitung in Amazon SQS, einschließlich Strategien für die zeitnahe Nachrichtenverarbeitung, die Auswahl des besten Abfragemodus und die Konfiguration langer Abfragen zur Verbesserung der Leistung.

Topics

- [Rechtzeitige Verarbeitung von Nachrichten in Amazon SQS](#)
- [Einrichtung von Long Polling in Amazon SQS](#)
- [Verwenden des entsprechenden Abfragemodus in Amazon SQS](#)

Rechtzeitige Verarbeitung von Nachrichten in Amazon SQS

Die Einstellung der Zeitbeschränkung für die Sichtbarkeit hängt davon ab, wie lange Ihre Anwendung braucht, um eine Nachricht zu verarbeiten und zu löschen. Benötigt Ihre Anwendung beispielsweise 10 Sekunden für die Verarbeitung einer Nachricht und Sie setzen die Zeitbeschränkung für die Sichtbarkeit auf 15 Minuten, müssen Sie relativ lange warten, bis Sie erneut versuchen können, die Nachricht zu verarbeiten, wenn der vorherige Versuch einer Verarbeitung fehlgeschlagen ist. Benötigt Ihre Anwendung alternativ 10 Sekunden für die Verarbeitung einer Nachricht, Sie setzen die Zeitbeschränkung für die Sichtbarkeit jedoch auf nur 2 Sekunden, wird eine duplizierte Nachricht von

einem anderen Verbraucher empfangen, während der ursprüngliche Verbraucher die Nachricht noch verarbeitet.

Um sicherzustellen, dass ausreichend Zeit für die Verarbeitung von Nachrichten vorhanden ist, nutzen Sie eine der folgenden Strategien:

- Wenn Sie wissen (oder annähernd schätzen können), wie lange die Verarbeitung einer Nachricht dauert, erweitern Sie die Zeitbeschränkung für die Sichtbarkeit auf die maximale Zeit, die erforderlich ist, um die Nachricht zu verarbeiten und zu löschen. Weitere Informationen finden Sie unter [Konfiguration der Sichtbarkeitszeitbeschränkung](#).
- Wenn Sie nicht wissen, wie lange die Bearbeitung einer Nachricht dauert, erstellen Sie einen Heartbeat für Ihren Kundenprozess: Geben Sie das anfängliche Timeout für die Sichtbarkeit an (z. B. 2 Minuten) und verlängern Sie dann, solange Ihr Kunde noch an der Nachricht arbeitet, die Sichtbarkeitszeitbeschränkung jede Minute um 2 Minuten.

Important

Die maximale Zeitbeschränkung für die Sichtbarkeit beträgt 12 Stunden ab dem Zeitpunkt, an dem Amazon SQS die `ReceiveMessage`-Anforderung empfängt. Durch die Verlängerung der Sichtbarkeitszeitbeschränkung wird das Maximum von 12 Stunden nicht zurückgesetzt.

Darüber hinaus können Sie das Timeout für eine einzelne Nachricht möglicherweise nicht auf die vollen 12 Stunden (z. B. 43 200 Sekunden) festlegen, seit die `ReceiveMessage`-Anfrage den Timer initiiert hat. Wenn Sie beispielsweise eine Nachricht erhalten und sofort das Maximum von 12 Stunden festlegen, indem Sie einen `ChangeMessageVisibility`-Aufruf mit einem `VisibilityTimeout` von 43 200 Sekunden senden, schlägt der Vorgang wahrscheinlich fehl. Die Verwendung eines Werts von 43 195 Sekunden funktioniert jedoch, sofern es nicht zu einer erheblichen Verzögerung zwischen dem Anfordern der Nachricht über `ReceiveMessage` und der Aktualisierung der Sichtbarkeitszeitbeschränkung kommt. Wenn Ihr Verbraucher länger als 12 Stunden benötigt, sollten Sie die Verwendung von Step Functions in Betracht ziehen.

Einrichtung von Long Polling in Amazon SQS

Ist die Wartezeit für die [ReceiveMessage](#)-API-Aktion größer als 0, ist eine lange Abfrage wirksam. Die maximale Wartezeit für lange Abfragen beträgt 20 Sekunden. Lange Abfragen tragen dazu bei,

die Kosten für die Nutzung von Amazon SQS zu senken, indem die Anzahl der leeren Antworten (wenn für eine `ReceiveMessage` Anfrage keine Nachrichten verfügbar sind) und der falschen leeren Antworten (wenn Nachrichten verfügbar sind, aber nicht in einer Antwort enthalten sind) reduziert wird. Weitere Informationen finden Sie unter [Kurz- und Langabfragen in Amazon SQS](#).

Um eine optimale Nachrichtenverarbeitung sicherzustellen, wenden Sie die folgenden Strategien an:

- In den meisten Fällen können Sie die `ReceiveMessage`-Wartezeit auf 20 Sekunden setzen. Wenn 20 Sekunden für Ihre Anwendung zu lang ist, legen Sie eine kürzere `ReceiveMessage`-Wartezeit fest (mindestens 1 Sekunde). Wenn Sie kein AWS SDK für den Zugriff auf Amazon SQS verwenden oder wenn Sie ein AWS SDK für eine kürzere Wartezeit konfigurieren, müssen Sie Ihren Amazon SQS SQS-Client möglicherweise ändern, um entweder längere Anfragen zuzulassen oder eine kürzere Wartezeit für lange Abfragen zu verwenden.
- Wenn Sie Langabfragen für mehrere Warteschlangen implementieren, verwenden Sie einen Thread für jede Warteschlange statt eines einzelnen Threads für alle Warteschlangen. Durch die Verwendung eines einzelnen Threads für jede Warteschlange kann Ihre Anwendung die Nachrichten in jeder der Warteschlangen verarbeiten, sobald diese verfügbar sind, während bei Verwendung eines einzelnen Threads für die Abfrage mehrerer Warteschlangen dazu führen könnte, dass Ihre Anwendung die Nachrichten nicht verarbeiten kann, die in anderen Warteschlangen zur Verfügung stehen, während die Anwendung auf die Warteschlange wartet (bis zu 20 Sekunden), die keine verfügbaren Nachrichten enthält.

 Important

Um HTTP-Fehler zu vermeiden, stellen Sie sicher, dass das HTTP-Reaktions-Timeout für `ReceiveMessage`-Anforderungen länger ist als der `WaitTimeSeconds`-Parameter. Weitere Informationen finden Sie unter [ReceiveMessage](#).

Verwenden des entsprechenden Abfragemodus in Amazon SQS

- Mit einer Langabfrage können Sie Nachrichten aus Ihrer Amazon-SQS-Warteschlange nutzen, sobald diese verfügbar sind.
 - Um die Verwendungskosten für Amazon SQS zu senken und die Anzahl der leeren Empfangsvorgänge in einer leeren Warteschlange zu reduzieren, (Antworten auf die `ReceiveMessage`-Aktion, mit der keine Nachrichten zurückgegeben werden), aktivieren Sie die Langabfrage. Weitere Informationen finden Sie unter [Amazon-SQS-Langabfragen](#).

- Um die Effizienz beim Abfragen für mehrere Threads mit mehreren Empfangsvorgängen zu steigern, verringern Sie die Anzahl der Threads.
- Langabfragen sind Kurzabfragen in den meisten Fällen vorzuziehen.
- Eine Kurzabfrage gibt Antworten sofort zurück, auch wenn die abgefragte Amazon-SQS-Warteschlange leer ist.
- Um die Anforderungen einer Anwendung zu erfüllen, die sofortige Antworten auf die Abfrage `ReceiveMessage` erwartet, verwenden Sie die Kurzabfrage.
- Die Kurzabfrage wird mit denselben Kosten in Rechnung gestellt wie die Langabfrage.

Beispiele für Amazon SQS Java SDK

AWS SDK für Java Damit können Sie Java-Anwendungen erstellen, die mit Amazon SQS und anderen AWS-Services interagieren.

- Zur Installation und Einrichtung des SDK siehe [Erste Schritte](#) im AWS SDK for Java 2.x - Entwicklerleitfaden.
- Informationen zu grundlegenden Warteschlangenvorgängen — wie das Erstellen einer Warteschlange oder das Senden einer Nachricht — finden Sie unter [Arbeiten mit Amazon SQS](#) Message Queues im Entwicklerhandbuch.AWS SDK for Java 2.x
- Dieses Handbuch enthält auch Beispiele für zusätzliche Amazon SQS SQS-Funktionen, wie z. B.:
 - [Serverseitige Verschlüsselung mit Amazon SQS SQS-Warteschlangen verwenden](#)
 - [Konfiguration von Tags für eine Amazon SQS SQS-Warteschlange](#)
 - [Nachrichtenattribute an eine Amazon SQS SQS-Warteschlange senden](#)

Serverseitige Verschlüsselung mit Amazon SQS SQS-Warteschlangen verwenden

Verwenden Sie die AWS SDK für Java , um serverseitige Verschlüsselung (SSE) zu einer Amazon SQS SQS-Warteschlange hinzuzufügen. Jede Warteschlange verwendet einen AWS Key Management Service (AWS KMS)-KMS-Schlüssel, um die Datenverschlüsselungsschlüssel zu generieren. In diesem Beispiel wird der von AWS verwaltete KMS-Schlüssel für Amazon SQS verwendet.

Weitere Informationen zum Verwenden von SSE und zur Rolle des KMS-Schlüssels finden Sie unter [Verschlüsselung im Ruhezustand in Amazon SQS](#).

Hinzufügen von SSE zu einer vorhandenen Warteschlange

Um die serverseitige Verschlüsselung für eine vorhandene Warteschlange zu aktivieren, verwenden Sie die [SetQueueAttributes](#)-Methode, um das KmsMasterKeyId-Attribut festzulegen.

Im folgenden Codebeispiel wird der AWS KMS key als AWS verwalteter KMS-Schlüssel für Amazon SQS festgelegt. Das Beispiel legt außerdem den [Zeitraum für die AWS KMS key -Wiederverwendung](#) auf 140 Sekunden fest.

Bevor Sie den Beispielcode ausführen, stellen Sie sicher, dass Sie Ihre AWS Anmeldeinformationen festgelegt haben. Weitere Informationen finden Sie unter [Einrichten von AWS Anmeldeinformationen und Region für die Entwicklung](#) im AWS SDK for Java 2.x Entwicklerhandbuch.

```
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();
    }
}
```


Deaktivieren von SSE für eine Warteschlange

Um die serverseitige Verschlüsselung für eine vorhandene Warteschlange zu deaktivieren, setzen Sie das Attribut `KmsMasterKeyId` mit der Aktion `SetQueueAttributes` auf eine leere Zeichenfolge.

Important

`null` ist kein gültiger Wert für `KmsMasterKeyId`.

Erstellen einer Warteschlange mit SSE

Um SSE beim Erstellen der Warteschlange zu aktivieren, fügen Sie das `KmsMasterKeyId`-Attribut der [CreateQueue](#)-API-Methode hinzu.

Das folgende Beispiel erstellt eine neue Warteschlange mit aktivierter SSE. Die Warteschlange verwendet den von AWS verwalteten KMS-Schlüssel für Amazon SQS. Das Beispiel legt außerdem den [Zeitraum für die AWS KMS key -Wiederverwendung](#) auf 160 Sekunden fest.

Bevor Sie den Beispielcode ausführen, stellen Sie sicher, dass Sie Ihre AWS Anmeldeinformationen festgelegt haben. Weitere Informationen finden Sie unter [Einrichten von AWS Anmeldeinformationen und Region für die Entwicklung](#) im AWS SDK for Java 2.x Entwicklerhandbuch.

```
// 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);
```

Abrufen der SSE-Attribute

Informationen zum Abrufen von Warteschlangenattributen finden Sie unter [Beispiele](#) in der Amazon-Simple-Queue-Service-API-Referenz.

Um die KMS-Schlüssel-ID oder den Zeitraum für die Wiederverwendung des Datenschlüssels für eine bestimmte Warteschlange abzurufen, führen Sie die [GetQueueAttributes](#)-Methode aus und rufen Sie die Werte `KmsMasterKeyId` und `KmsDataKeyReusePeriodSeconds` ab.

Konfiguration von Tags für eine Amazon SQS SQS-Warteschlange

Sie können Ihren Amazon-SQS-Warteschlangen Kostenzuordnungs-Tags hinzufügen, um sie zu organisieren und zu identifizieren. Die folgenden Beispiele veranschaulichen, wie Tags mithilfe von AWS SDK für Java konfiguriert werden. Weitere Informationen finden Sie unter [Amazon-SQS-Kostenzuordnungs-Tags](#).

Bevor Sie den Beispielcode ausführen, stellen Sie sicher, dass Sie Ihre AWS Anmeldeinformationen festgelegt haben. Weitere Informationen finden Sie unter [Einrichten von AWS Anmeldeinformationen und Region für die Entwicklung](#) im AWS SDK for Java 2.x Entwicklerhandbuch.

Auflisten von Tags

Verwenden Sie die `ListQueueTags`-Methode, um die Tags für eine Warteschlange aufzulisten.

```
// 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() ));
```

Hinzufügen oder Aktualisieren von Tags

Verwenden Sie die `TagQueue`-Methode, um Tag-Werte für eine Warteschlange hinzuzufügen oder zu aktualisieren.

```
// 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);
```

Entfernen von Tags

Verwenden Sie die `UntagQueue`-Methode, um ein oder mehrere Tags aus der Warteschlange zu entfernen. Im folgenden Beispiel wird das `Accounting ID`-Tag entfernt.

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

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

Nachrichtenattribute an eine Amazon SQS SQS-Warteschlange senden

Sie können strukturierte Metadaten (wie etwa Zeitstempel, geospatiale Daten, Signaturen und Kennungen) in Nachrichten einschließen, indem Sie Nachrichtenattribute verwenden. Weitere Informationen finden Sie unter [Amazon-SQS-Nachrichtenattribute](#).

Bevor Sie den Beispielcode ausführen, stellen Sie sicher, dass Sie Ihre AWS Anmeldeinformationen festgelegt haben. Weitere Informationen finden Sie unter [Einrichten von AWS Anmeldeinformationen und Region für die Entwicklung](#) im AWS SDK for Java 2.x Entwicklerhandbuch.

Definieren von Attributen

Fügen Sie zum Definieren eines Attributs für eine Nachricht den folgenden Code hinzu, der den `MessageAttributeValue`-Datentyp verwendet. Weitere Informationen erhalten Sie unter [Nachrichtenattributkomponenten](#) und [Datentypen für Nachrichtenattribute](#).

Der berechnet AWS SDK für Java automatisch die Prüfsummen für den Nachrichtentext und die Nachrichtenattribute und vergleicht sie mit den Daten, die Amazon SQS zurückgibt. Weitere Informationen finden Sie im [AWS SDK for Java 2.x -Entwicklerhandbuch](#) und in [Berechnung des MD5 Message Digest für Nachrichtenattribute](#) für andere Programmiersprachen.

String

Dieses Beispiel definiert ein String-Attribut mit der Bezeichnung Name und dem Wert Jane.

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

Number

Dieses Beispiel definiert ein Number-Attribut mit der Bezeichnung AccurateWeight und dem Wert 230.000000000000000001.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccurateWeight", new MessageAttributeValue()
    .withDataType("Number")
    .withStringValue("230.000000000000000001"));
```

Binary

Dieses Beispiel definiert ein Binary-Attribut mit dem Namen ByteArray und dem Wert eines nicht initialisierten 10-Byte-Arrays.

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

String (custom)

Dieses Beispiel definiert das benutzerdefinierte Attribut String.EmployeeId mit der Bezeichnung EmployeeId und dem Wert ABC123456.

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

Number (custom)

Dieses Beispiel definiert das benutzerdefinierte Attribut `Number.AccountId` mit der Bezeichnung `AccountId` und dem Wert `000123456`.

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

Note

Da der Basisdatentyp `Number` ist, gibt die [ReceiveMessage](#)-Aktion `123456` zurück.

Binary (custom)

Dieses Beispiel definiert das benutzerdefinierte Attribut `Binary.JPEG` mit dem Namen `ApplicationIcon` und dem Wert eines nicht initialisierten 10-Byte-Arrays.

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

Senden einer Nachricht mit Attributen

In diesem Beispiel werden die Attribute der `SendMessageRequest` hinzugefügt, bevor die Nachricht gesendet wird.

```
// 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

Wenn Sie eine Nachricht an eine First-In-First-Out (FIFO-) Warteschlange senden, stellen Sie sicher, dass die `sendMessage` Methode ausgeführt wird, nachdem Sie die Nachrichtengruppen-ID angegeben haben.

Wenn Sie die [SendMessageBatch](#)-Aktion anstelle von [SendMessage](#) verwenden, müssen Sie Nachrichtenattribute für jede Nachricht in dem Stapel angeben.

Verwendung APIs mit Amazon SQS

Dieses Thema enthält Informationen zur Konstruktion von Amazon SQS SQS-Endpunkten, zur Erstellung von API-Abfragen mit den Methoden GET und POST und zur Verwendung von Batch-API-Aktionen. Detaillierte Informationen zu Amazon-SQS-[Aktionen](#) – einschließlich Parametern, Fehlern, Beispielen und [Datentypen](#) – finden Sie in der [Amazon-Simple-Queue-Service-API-Referenz](#).

Um mit einer Vielzahl von Programmiersprachen auf Amazon SQS zuzugreifen, können Sie auch verwenden [AWS SDKs](#), die die folgenden automatischen Funktionen enthalten:

- Kryptographisches Signieren Ihrer Serviceanfragen
- Wiederholen von Anfragen
- Umgang mit Fehlerreaktionen

Weitere Informationen finden Sie unter [the section called “Arbeitet mit AWS SDKs”](#).

Informationen zum Befehlszeilen-Tool finden Sie in den Amazon-SQS-Abschnitten in der [AWS CLI - Befehlsreferenz](#) und der [AWS -Tools für PowerShell -Cmdlet-Referenz](#).

Amazon SQS APIs mit AWS JSON-Protokoll

Amazon SQS verwendet das AWS JSON-Protokoll als Transportmechanismus für alle Amazon SQS APIs auf den angegebenen [AWS SDK-Versionen](#). AWS Das JSON-Protokoll bietet einen höheren Durchsatz, eine geringere Latenz und eine schnellere application-to-application Kommunikation. AWS Das JSON-Protokoll ist im Vergleich zum AWS Abfrageprotokoll effizienter bei serialization/deserialization der Bearbeitung von Anfragen und Antworten. Wenn Sie das AWS Abfrageprotokoll immer noch lieber mit SQS verwenden möchten APIs, finden Sie [Welche Sprachen werden für das in Amazon SQS APIs verwendete AWS JSON-Protokoll unterstützt?](#) die AWS SDK-Versionen, die das Amazon SQS AWS SQS-Abfrageprotokoll unterstützen.

Amazon SQS verwendet das AWS JSON-Protokoll für die Kommunikation zwischen AWS SDK-Clients (z. B. Java, Python, Golang JavaScript) und dem Amazon SQS-Server. Eine HTTP-Anfrage eines Amazon-SQS-API-Vorgangs akzeptiert Eingaben im JSON-Format. Der Amazon-SQS-Vorgang wird ausgeführt und die Ausführungsantwort im JSON-Format an den SDK-Client zurückgesendet. Im Vergleich zu AWS Query ist AWS JSON einfacher, schneller und effizienter, wenn es darum geht, Daten zwischen Client und Server zu transportieren.

- AWS Das JSON-Protokoll fungiert als Vermittler zwischen dem Amazon SQS SQS-Client und dem Server.
- Der Server versteht die Programmiersprache, in der der Amazon SQS SQS-Vorgang erstellt wurde, nicht, aber er versteht das AWS JSON-Protokoll.
- Das AWS JSON-Protokoll verwendet die Serialisierung (Objekt in das JSON-Format konvertieren) und die Deserialisierung (JSON-Format in Objekt konvertieren) zwischen Amazon SQS SQS-Client und -Server.

Weitere Informationen zum AWS JSON-Protokoll mit Amazon SQS finden Sie unter [Amazon SQS AWS JSON-Protokoll FAQs](#).

AWS Das JSON-Protokoll ist in der angegebenen [AWS SDK-Version](#) verfügbar. Informationen zur SDK-Version und zu den Veröffentlichungsdaten der verschiedenen Sprachvarianten finden Sie in der [AWS SDKs Matrix zur Unterstützung von Versionen AWS](#) SDKs und Tools im Referenzhandbuch zu Tools

Abfrage-API-Anfragen mithilfe des AWS JSON-Protokolls in Amazon SQS stellen

In diesem Thema wird erklärt, wie Sie einen Amazon SQS SQS-Endpoint erstellen, POST-Anfragen stellen und Antworten interpretieren.

Note

AWS Das JSON-Protokoll wird für die meisten Sprachvarianten unterstützt. Eine Liste der unterstützten Sprachvarianten finden Sie unter [Welche Sprachen werden für das in Amazon SQS APIs verwendete AWS JSON-Protokoll unterstützt?](#).

Erstellen eines Endpunkts

Für die Arbeit mit Amazon-SQS-Warteschlangen müssen Sie einen Endpunkt erstellen. Informationen zu Amazon-SQS-Endpunkten finden Sie auf den folgenden Seiten im Allgemeine Amazon Web Services-Referenz:

- [Regionale Endpunkte](#)
- [Endpunkte und Kontingente von Amazon Simple Queue Service](#)

Jeder Amazon-SQS-Endpunkt ist unabhängig. Wenn z. B. bei zwei Warteschlangen mit dem identischen Namen MyQueue eine über Endpunkt `sqs.us-east-2.amazonaws.com` und die andere über Endpunkt `sqs.eu-west-2.amazonaws.com` verfügt, nutzen die beiden Warteschlangen keine Daten gemeinsam.

Das folgende Beispiel zeigt einen Endpunkt, der eine Anforderung zum Erstellen einer Warteschlange stellt.

```
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

Bei Warteschlangennamen und Warteschlangen wird zwischen Groß- und Kleinschreibung URLs unterschieden.

Die Struktur von **AUTHPARAMS** hängt davon ab, wie Sie Ihre API-Anforderung signieren. Weitere Informationen finden Sie unter [AWS API-Anfragen signieren](#) in der Allgemeinen Referenz zu Amazon Web Services.

Durchführen einer POST-Anforderung

Eine Amazon-SQS-POST-Anforderung sendet Abfrageparameter als Formular im Text einer HTTP-Anforderung.

Im Folgenden finden Sie ein Beispiel für einen HTTP-Header mit der X-Amz-Target-Einstellung auf AmazonSQS.<operationName> und einen HTTP-Header mit der Content-Type-Einstellung auf application/x-amz-json-1.0.

```
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"
}
```

Diese HTTP-POST-Anforderung sendet eine Nachricht an eine Amazon-SQS-Warteschlange.

Note

Beide HTTP-Header, X-Amz-Target und Content-Type, sind erforderlich.
Der HTTP-Client fügt abhängig von der HTTP-Version des Clients möglicherweise weitere Elemente zur HTTP-Anforderung hinzu.

Interpretieren von Amazon-SQS-JSON-API-Antworten

Wenn Sie eine Anfrage an Amazon SQS senden, wird eine JSON-Antwort mit den Ergebnissen zurückgegeben. Die Antwortstruktur hängt von der API-Aktion ab, die Sie verwendet haben.

Weitere Informationen zu diesen Antworten finden Sie unter:

- Die spezifische API-Aktion in den [API-Aktionen](#) in der Amazon Simple Queue Service API-Referenz
- Die [Amazon SQS AWS JSON-Protokoll FAQs](#)

Struktur einer JSON-Antwort bei Erfolg

Ist die Anfrage erfolgreich, ist das Hauptantwortelement `x-amzn-RequestId`, das den Universal Unique Identifier (UUID) der Anfrage sowie weitere angehängte Antwortfelder enthält. Beispielsweise enthält die folgende [CreateQueue](#)-Antwort das Feld `QueueUrl`, das wiederum die URL der erstellten Warteschlange enthält.

```
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"
}
```

Struktur einer JSON-Antwort bei Fehlschlagen

Wenn eine Anfrage nicht erfolgreich ist, gibt Amazon SQS die Hauptantwort zurück, einschließlich des HTTP-Headers und des Texts.

`x-amzn-RequestId` enthält im HTTP-Header die UUID der Anfrage. `x-amzn-query-error` enthält zwei Informationen: die Art des Fehlers, und ob es sich bei dem Fehler um einen Produzenten- oder einen Konsumentenfehler handelt.

`"__type"` gibt im Antworttext weitere Fehlerdetails an und `Message` zeigt die Fehlerbedingung in lesbarem Format an.

Im Folgenden finden Sie eine Beispielfehlerantwort im JSON-Format:

```
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-Protokoll FAQs

Dieses Thema behandelt häufig gestellte Fragen zur Verwendung des AWS JSON-Protokolls mit Amazon SQS.

Was ist das AWS JSON-Protokoll und wie unterscheidet es sich von bestehenden Amazon SQS SQS-API-Anfragen und -Antworten?

JSON ist eine der am weitesten verbreiteten und akzeptierten Verbindungsmethoden für die Kommunikation zwischen heterogenen Systemen. Amazon SQS verwendet JSON als Medium für die Kommunikation zwischen einem AWS SDK-Client (z. B. Java, Python, Golang JavaScript) und einem Amazon SQS-Server. Eine HTTP-Anfrage eines Amazon-SQS-API-Vorgangs akzeptiert Eingaben in Form von JSON. Der Amazon-SQS-Vorgang wird ausgeführt und die Ausführungsantwort wird im JSON-Format an den SDK-Client zurückgesendet. Im Vergleich zu AWS -Abfragen ist JSON effizienter, wenn es darum geht, Daten zwischen Client und Server zu transportieren.

- Das Amazon SQS AWS JSON-Protokoll fungiert als Vermittler zwischen dem Amazon SQS SQS-Client und dem Server.
- Der Server versteht die Programmiersprache, in der der Amazon SQS SQS-Vorgang erstellt wurde, nicht, aber er versteht das AWS JSON-Protokoll.
- Das Amazon SQS AWS JSON-Protokoll verwendet die Serialisierung (Objekt in das JSON-Format konvertieren) und Deserialisierung (JSON-Format in Objekt konvertieren) zwischen dem Amazon SQS SQS-Client und -Server.

Wie fange ich mit AWS JSON-Protokollen für Amazon SQS an?

Um mit der neuesten AWS SDK-Version zu beginnen und schnelleres Messaging für Amazon SQS zu erreichen, aktualisieren Sie Ihr AWS SDK auf die angegebene Version oder eine nachfolgende Version. Weitere Informationen zu SDK-Clients finden Sie in der Spalte „Leitfaden“ in der Tabelle unten.

Im Folgenden finden Sie eine Liste der SDK-Versionen in allen Sprachvarianten für das AWS JSON-Protokoll zur Verwendung mit Amazon SQS: APIs

Sprache	SDK-Client-Reposit ory	Erforderliche SDK- Clientversion	Richtlinie

Sprache	SDK-Client-Repository	Erforderliche SDK-Clientversion	Richtlinie
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/2 aws-sdk-go-v	v1.28.0	AWS SDK for Go V2
Java 1.x	aws/ aws-sdk-java	1.12.585	AWS SDK für Java
Java 2.x	Ahnen/ 2 aws-sdk-java-v	2.21,19	AWS SDK für Java
JavaScript v2.x	aws/ aws-sdk-js	JavaScript nein AWS	
JavaScript v3.x	aws/3 aws-sdk-js-v	v3.447.0	JavaScript nein 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)
Python-botocore	boto/botocore	1.31.82	AWS SDK for Python (Boto3)
awscli	AWS -CLI	1.29.82	AWS-Befehlszeilenchnittstelle
Ruby	sägen/ aws-sdk-ruby	1,67,0	AWS SDK for Ruby

Was sind die Risiken, wenn ich das JSON-Protokoll für meine Amazon-SQS-Workloads aktiviere?

Wenn Sie eine benutzerdefinierte Implementierung von AWS SDK oder eine Kombination aus benutzerdefinierten Clients und AWS SDK für die Interaktion mit Amazon SQS verwenden, das AWS abfragebasierte (auch bekannt als XML-basierte) Antworten generiert, ist dies möglicherweise nicht mit dem AWS JSON-Protokoll kompatibel. Wenn Sie auf Probleme stoßen, wenden Sie sich an den AWS Support.

Was ist, wenn ich bereits die neueste AWS SDK-Version verwende, aber meine Open-Source-Lösung JSON nicht unterstützt?

Sie müssen Ihre SDK-Version auf die Version ändern, die vor der von Ihnen verwendeten Version liegt. Weitere Informationen [Wie fange ich mit AWS JSON-Protokollen für Amazon SQS an?](#) finden Sie unter. AWS Die unter aufgeführten SDK-Versionen [Wie fange ich mit AWS JSON-Protokollen für Amazon SQS an?](#) verwenden das JSON-Wire-Protokoll für Amazon SQS. APIs Wenn Sie Ihr AWS SDK auf die vorherige Version ändern, verwendet Ihr Amazon SQS APIs die AWS Abfrage.

Welche Sprachen werden für das in Amazon SQS APIs verwendete AWS JSON-Protokoll unterstützt?

Amazon SQS unterstützt alle Sprachvarianten, sofern sie allgemein verfügbar AWS SDKs sind (GA). Derzeit unterstützen wir Kotlin, Rust oder Swift nicht. Weitere Informationen zu anderen Sprachvarianten finden Sie unter [Tools, auf der Grundlage von AWS](#).

Welche Regionen werden für das in Amazon SQS verwendete AWS JSON-Protokoll unterstützt APIs

Amazon SQS unterstützt das AWS JSON-Protokoll in allen [AWS Regionen](#), in denen Amazon SQS verfügbar ist.

Welche Latenzverbesserungen kann ich erwarten, wenn ich ein Upgrade auf die angegebenen AWS SDK-Versionen für Amazon SQS mithilfe des AWS JSON-Protokolls durchführe?

AWS Das JSON-Protokoll ist im Vergleich zum Abfrageprotokoll effizienter bei der Serialisierung und Deserialisierung von Anfragen und Antworten. AWS Basierend auf AWS Leistungstests für eine Nachrichtennutzlast von 5 KB reduziert das JSON-Protokoll für Amazon SQS die Latenz bei

der end-to-end Nachrichtenverarbeitung um bis zu 23% und reduziert die clientseitige CPU- und Speicherauslastung der Anwendung.

Wird das AWS Abfrageprotokoll veraltet sein?

AWS Das Abfrageprotokoll wird weiterhin unterstützt. Sie können das AWS Abfrageprotokoll weiterhin verwenden, solange für Ihre AWS SDK-Version eine andere Version festgelegt ist als die, die unter [Wie fange ich mit AWS JSON-Protokollen für Amazon SQS an?](#) aufgeführt ist.

Wo finde ich weitere Informationen zum AWS JSON-Protokoll?

Weitere Informationen zum JSON-Protokoll finden Sie unter [AWS -JSON-1.0-Protokoll](#) in der Smithy-Dokumentation. Weitere Informationen zu Amazon-SQS-API-Anfragen mit dem AWS -JSON-Protokoll finden Sie unter [Abfrage-API-Anfragen mithilfe des AWS JSON-Protokolls in Amazon SQS stellen](#).

Abfrage-API-Anfragen mithilfe des AWS Abfrageprotokolls in Amazon SQS stellen

In diesem Thema wird erklärt, wie Sie einen Amazon SQS SQS-Endpoint erstellen, GET- und POST-Anfragen stellen und Antworten interpretieren.

Erstellen eines Endpunkts

Für die Arbeit mit Amazon-SQS-Warteschlangen müssen Sie einen Endpunkt erstellen. Informationen zu Amazon-SQS-Endpunkten finden Sie auf den folgenden Seiten im Allgemeine Amazon Web Services-Referenz:

- [Regionale Endpunkte](#)
- [Endpunkte und Kontingente von Amazon Simple Queue Service](#)

Jeder Amazon-SQS-Endpunkt ist unabhängig. Wenn z. B. bei zwei Warteschlangen mit dem identischen Namen MyQueue eine über Endpunkt `sqs.us-east-2.amazonaws.com` und die andere über Endpunkt `sqs.eu-west-2.amazonaws.com` verfügt, nutzen die beiden Warteschlangen keine Daten gemeinsam.

Das folgende Beispiel zeigt einen Endpunkt, der eine Anforderung zum Erstellen einer Warteschlange stellt.


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

Note

Bei Warteschlangennamen und Warteschlangen wird zwischen Groß- und URLs Kleinschreibung unterschieden.

Die Struktur von **AUTHPARAMS** hängt davon ab, wie Sie Ihre API-Anforderung signieren. Weitere Informationen finden Sie unter [AWS API-Anfragen signieren](#) in der Allgemeinen Referenz zu Amazon Web Services.

Durchführen einer GET-Anforderung

Eine Amazon-SQS-GET-Anforderung ist als URL mit den folgenden Komponenten aufgebaut:

- Endpunkt – Die Ressource, für die die Anforderung ausgeführt wird (der [Warteschlangenname und die URL](#)), z. B.: `https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`
- Aktion – Die [Aktion](#), die Sie für den Endpunkt ausführen möchten. Ein Fragezeichen (?) trennt den Endpunkt von der Aktion, z. B.: `?Action=SendMessage&MessageBody=Your%20Message%20Text`
- Parameter – Beliebige Anforderungsparameter. Jeder Parameter ist durch ein kaufmännisches Und (&) vom Text getrennt, zum Beispiel: `&Version=2012-11-05&AUTHPARAMS`

Im Folgenden finden Sie ein Beispiel für eine GET-Anforderung, die eine Nachricht an eine Amazon-SQS-Warteschlange sendet.

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

Note

Bei Warteschlangennamen und Warteschlangen wird zwischen Groß- und Kleinschreibung URLs unterschieden.

Da dies bei GET-Anfragen der Fall ist URLs, müssen Sie alle Parameterwerte URL-kodieren.

Da Leerzeichen nicht erlaubt sind URLs, ist jedes Leerzeichen URL-kodiert als. %20 Für den Rest des Beispiels wurde zum Zweck der besseren Lesbarkeit keine URL-Codierung durchgeführt.

Durchführen einer POST-Anforderung

Eine Amazon-SQS-POST-Anforderung sendet Abfrageparameter als Formular im Text einer HTTP-Anforderung.

Es folgt ein Beispiel eines HTTP-Headers mit der Einstellung `application/x-www-form-urlencoded` für Content-Type.

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

Auf den Header folgt eine [form-urlencoded](#)-GET-Anforderung, die eine Nachricht an eine Amazon-SQS-Warteschlange sendet. Jeder Parameter ist durch ein kaufmännisches Und (&) vom Text getrennt.

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

Note

Nur der Content-Type HTTP-Header ist erforderlich. Der `AUTHPARAMS` ist für die GET-Anforderung derselbe.

Der HTTP-Client fügt abhängig von der HTTP-Version des Clients möglicherweise weitere Elemente zur HTTP-Anforderung hinzu.

Interpretieren von Amazon-SQS-XML-API-Antworten

Wenn Sie eine Anfrage an Amazon SQS senden, wird eine XML-Antwort mit den Ergebnissen der Anfrage zurückgegeben. Informationen zur Struktur und zu den Einzelheiten dieser Antworten finden Sie in den spezifischen [API-Aktionen](#) in der Amazon Simple Queue Service API-Referenz.

Struktur einer XML-Antwort bei Erfolg

Wenn die Anforderung erfolgreich ist, wird das Hauptantwortelement nach der Aktion mit dem Zusatz Response (zum Beispiel *ActionNameResponse*) benannt.

Dieses Element enthält die folgenden untergeordneten Elemente:

- **ActionNameResult** – Enthält ein aktionsspezifisches Element. Das Element `CreateQueueResult` enthält das Element `QueueUrl`, das wiederum die URL der erstellten Warteschlange enthält.
- **ResponseMetadata** – Enthält die `RequestId`, die wiederum die UUID (Universal Unique Identifier) der Anforderung enthält.

Nachfolgend finden Sie ein Beispiel für eine Antwort bei Erfolg im XML-Format:

```
<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-Fehler-Antwortstruktur

Wenn eine Anforderung fehlschlägt, gibt Amazon SQS immer das Haupt-Antwort-Element `ErrorResponse` zurück. Dieses Element enthält ein `Error`- und ein `RequestId`-Element.

Das Element `Error` enthält die folgenden untergeordneten Elemente:

- **Type** – Gibt an, ob es sich bei dem Fehler um einen Produzenten- oder einen Konsumentenfehler handelt.
- **Code** – Gibt den Typ des Fehlers an.
- **Message** – Gibt die Fehlerbedingung in einem lesbaren Format an.
- **Detail** – (Optional) Gibt zusätzliche Details zu dem Fehler an.

Das Element `RequestId` enthält die UUID der Anforderung.

Nachfolgend finden Sie ein Beispiel für eine Antwort bei Fehlschlagen im XML-Format:

```
<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>
```

Authentifizieren von Anfragen für Amazon SQS

Authentifizierung bezeichnet den Prozess der Identifizierung und Überprüfung des Absenders einer Anforderung. Während der ersten Phase der Authentifizierung überprüft AWS die Identität des Produzenten, und ob der Produzent [für die Verwendung von AWS registriert ist](#) (weitere Informationen finden Sie unter [Schritt 1: Erstellen Sie ein AWS-Konto und IAM-Benutzer](#)). Halten Sie sich AWS als Nächstes an das folgende Verfahren:

1. Der Produzent (Absender) erhält die notwendigen Anmeldeinformationen.
2. Der Produzent sendet eine Anforderung mit den Anmeldeinformationen an den Konsumenten (Empfänger).
3. Der Konsument überprüft anhand der Anmeldeinformationen, ob der Produzent die Anforderung gesendet hat.
4. Es tritt einer der beiden folgenden Fälle ein:

- Wenn die Authentifizierung erfolgreich durchgeführt wurde, wird die Anforderung vom Konsumenten verarbeitet.
- Wenn die Authentifizierung fehlschlägt, wird die Anforderung vom Verbraucher abgelehnt und es wird eine Fehlermeldung zurückgegeben.

Grundlegender Authentifizierungsprozess mit HMAC-SHA

Beim Zugriff auf Amazon SQS mit der Abfrage-API müssen Sie die folgenden Elemente angeben, um die Anforderung zu authentifizieren:

- Die AWS Zugangsschlüssel-ID, mit der Sie AWS-Konto identifiziert werden und AWS anhand derer Ihr geheimer Zugriffsschlüssel nachgeschlagen wird.
 - Die HMAC-SHA-Anforderungssignatur, berechnet anhand Ihres geheimen Zugangsschlüssels (ein gemeinsam genutzter geheimer Schlüssel, der nur Ihnen bekannt ist und AWS— weitere Informationen finden Sie unter). [RFC2104](#) Das [AWS -SDK](#) übernimmt die Signatur. Wenn Sie jedoch eine Abfrageanforderung über HTTP oder HTTPS senden, müssen Sie in jede Abfrageanforderung eine Signatur einschließen.
1. Ableiten eines Signature Version 4-Signaturschlüssels. Weitere Informationen finden Sie unter [Ableiten des Signaturschlüssels mit Java](#).

Note

Amazon SQS unterstützt Signature Version 4, die im Vergleich zu früheren Versionen eine verbesserte SHA256 Sicherheit und Leistung bietet. Wenn Sie neue Anwendungen erstellen, die Amazon SQS nutzen, verwenden Sie Signature Version 4.

2. Codieren Sie die Anforderungssignatur mit Base64. Das folgende Java-Codebeispiel führt folgende Schritte aus:

```
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);  
}  
}
```

- Der Zeitstempel (oder die Ablaufzeit) der Anforderung. Der Zeitstempel, den Sie in der Anforderung verwenden, muss ein `dateTime`-Objekt mit [dem vollständigen Datum, einschließlich Stunden, Minuten und Sekunden, sein](#). Beispiel: 2007-01-31T23:59:59Z Obwohl dies nicht erforderlich ist, empfohlen wird, die Angabe des Objekts in der UTC-Zeitzone (Greenwich Mean Time).

Note

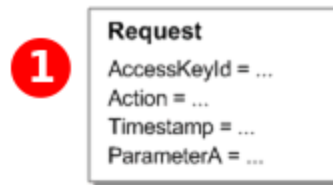
Stellen Sie sicher, dass Ihre Serverzeit richtig eingestellt ist. Wenn Sie einen Zeitstempel (statt eines Ablaufs) angeben, läuft die Anfrage automatisch 15 Minuten nach der angegebenen Zeit ab (Anfragen, deren Zeitstempel mehr als 15 Minuten vor der aktuellen Uhrzeit auf AWS Servern liegen, werden AWS nicht verarbeitet).

Falls Sie .NET verwenden, dürfen Sie keine extrem spezifischen Zeitstempel senden (da unterschiedlich interpretiert wird, nach welchem Zeitraum die Anforderung verworfen werden sollte). In diesem Fall sollten Sie manuell `dateTime`-Objekte mit einer Genauigkeit von nicht mehr als einer Millisekunde erstellen.

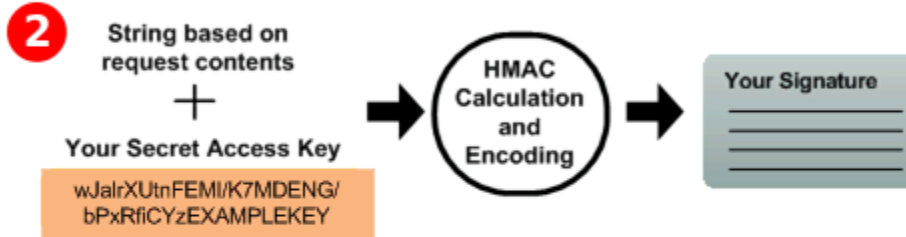
Teil 1: Die Anforderung von dem Benutzer

Gehen Sie wie folgt vor, um AWS Anfragen mithilfe einer HMAC-SHA-Anforderungssignatur zu authentifizieren.

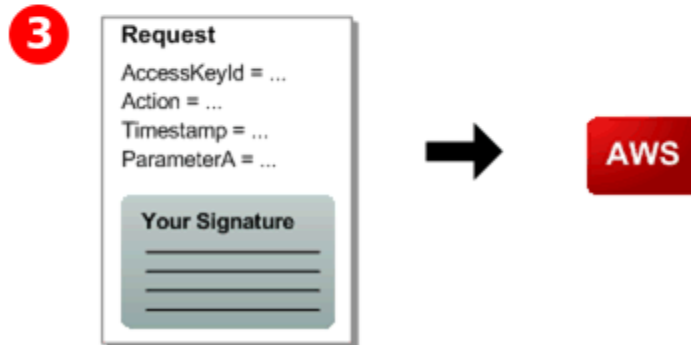
Create a request:



Create an
HMAC-SHA
signature:



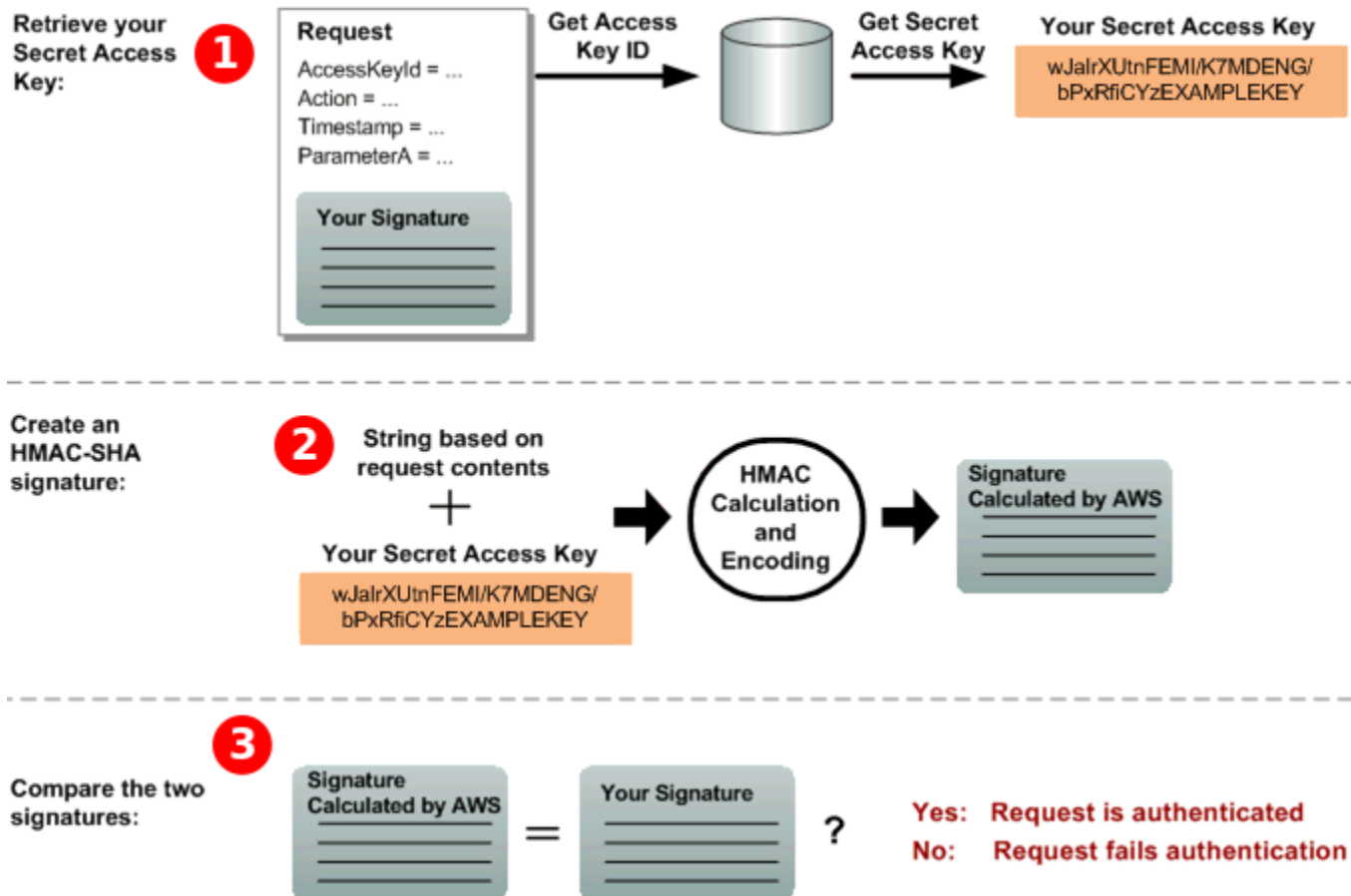
Send the request
and signature to
AWS:



1. Konstruieren Sie eine Anfrage an AWS.
2. Berechnen Sie eine verschlüsselte Hash-Signatur (HMAC-SHA) für die Nachrichtenauthentifizierung anhand des geheimen Zugriffsschlüssels.
3. Fügen Sie die Signatur und Ihre Zugriffsschlüssel-ID in die Anfrage ein und senden Sie die Anfrage dann an AWS.

Teil 2: Die Antwort von AWS

AWS startet als Antwort den folgenden Prozess.



1. AWS verwendet die Zugriffsschlüssel-ID, um nach Ihrem geheimen Zugriffsschlüssel zu suchen.
2. AWS generiert eine Signatur aus den Anforderungsdaten und dem Secret Access Key und verwendet dabei denselben Algorithmus, den Sie zur Berechnung der Signatur verwendet haben, die Sie in der Anfrage gesendet haben.
3. Es tritt einer der beiden folgenden Fälle ein:
 - Wenn die AWS generierte Signatur mit der Signatur übereinstimmt, die Sie in der Anfrage gesendet haben, wird die Anfrage als authentisch AWS betrachtet.
 - Schlägt der Vergleich fehl, wird die Anfrage verworfen und es wird ein Fehler AWS zurückgegeben.

Amazon-SQS-Stapelaktionen

Amazon SQS bietet Batch-Aktionen, mit denen Sie Kosten senken und bis zu 10 Nachrichten mit einer einzigen Aktion bearbeiten können. Zu diesen Batch-Aktionen gehören:

- [SendMessageBatch](#)
- [DeleteMessageBatch](#)
- [ChangeMessageVisibilityBatch](#)

Mithilfe von Batch-Aktionen können Sie mehrere Operationen in einem einzigen API-Aufruf ausführen, wodurch die Leistung optimiert und die Kosten gesenkt werden können. Sie können die Batch-Funktionalität mithilfe der Abfrage-API oder eines beliebigen AWS SDK nutzen, das Amazon SQS SQS-Batch-Aktionen unterstützt.

Wichtige Details

- Größenbeschränkung für Nachrichten: Die Gesamtgröße aller Nachrichten, die in einem einzigen SendMessageBatch Aufruf gesendet werden, darf 1.048.576 Byte (1 MiB) nicht überschreiten
- Berechtigungen: Sie können Berechtigungen für, oder nicht explizit festlegen. SendMessageBatch DeleteMessageBatch ChangeMessageVisibilityBatch Stattdessen werden durch das Festlegen von Berechtigungen für SendMessageDeleteMessage, oder Berechtigungen für die entsprechenden Batchversionen der Aktionen ChangeMessageVisibility festgelegt.
- Konsolenunterstützung: Die Amazon SQS SQS-Konsole unterstützt keine Batch-Aktionen. Sie müssen die Abfrage-API oder ein AWS SDK verwenden, um Batch-Operationen durchzuführen.

Stapelverarbeitungsaktionen für Nachrichten

Um Kosten und Effizienz weiter zu optimieren, sollten Sie die folgenden bewährten Methoden für die Batchverarbeitung von Nachrichtenaktionen berücksichtigen:

- Batch-API-Aktionen: Verwenden Sie die [Amazon SQS SQS-Batch-API-Aktionen](#), um Nachrichten zu senden, zu empfangen und zu löschen und um das Timeout für die Nachrichtensichtbarkeit mehrerer Nachrichten mit einer einzigen Aktion zu ändern. Dies reduziert die Anzahl der API-Aufrufe und die damit verbundenen Kosten.
- Clientseitige Pufferung und lange Abfragen: Kombinieren Sie die clientseitige Pufferung [mit der Batchverarbeitung von Anfragen, indem Sie lange Abfragen zusammen mit dem gepufferten asynchronen Client verwenden, der im Lieferumfang von enthalten ist](#). AWS SDK für Java Dieser Ansatz trägt dazu bei, die Anzahl der Anfragen zu minimieren und die Verarbeitung großer Nachrichtenmengen zu optimieren.

 Note

Der Amazon SQS Buffered Asynchronous Client unterstützt derzeit keine FIFO-Warteschlangen.

Aktivierung der clientseitigen Pufferung und der Batchverarbeitung von Anfragen mit Amazon SQS

[AWS SDK für Java](#) enthält den `AmazonSQSBufferedAsyncClient`, der auf Amazon SQS zugreift. Dieser Client ermöglicht eine einfache Batchverarbeitung von Anfragen mithilfe der clientseitigen Pufferung. Aufrufe vom Client werden zuerst gepuffert und dann als Batch-Anfrage an Amazon SQS gesendet.

Mit der clientseitigen Pufferung können bis zu 10 Anforderungen zwischengespeichert und als Stapelanforderung gesendet werden. Dadurch werden Ihre Kosten für die Verwendung von Amazon SQS gesenkt und die Anzahl der gesendeten Anforderungen verringert.

`AmazonSQSBufferedAsyncClient` puffert synchrone und asynchrone Aufrufe. Als Stapel verarbeitete Anforderungen und die Unterstützung für [langes Abrufen](#) können ebenfalls zu einem erhöhten Durchsatz beitragen. Weitere Informationen finden Sie unter [Steigerung des Durchsatzes durch horizontale Skalierung und Action-Batching mit Amazon SQS](#).

Da `AmazonSQSBufferedAsyncClient` die gleiche Schnittstelle implementiert wie `AmazonSQSAsyncClient`, sollte die Migration von `AmazonSQSAsyncClient` zu `AmazonSQSBufferedAsyncClient` in der Regel lediglich minimale Änderungen an Ihrem vorhandenen Code erfordern.

 Note

Der Amazon SQS Buffered Asynchronous Client unterstützt derzeit keine FIFO-Warteschlangen.

Amazon verwenden SQSBuffered AsyncClient

Bevor Sie beginnen, führen Sie die Schritte in [Einrichten von Amazon SQS](#) aus.

AWS SDK for Java 1.x

Für AWS SDK for Java 1.x können Sie ein neues erstellen, das auf dem folgenden Beispiel `AmazonSQSBufferedAsyncClient` basiert:

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

Nachdem Sie den neuen `AmazonSQSBufferedAsyncClient` erstellt haben, können Sie ihn verwenden, um mehrere Anforderungen an Amazon SQS zu senden (wie mit dem `AmazonSQSAsyncClient`), beispielsweise:

```
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.setRequestBody( 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);
```

Amazon konfigurieren SQSBuffered AsyncClient

`AmazonSQSBufferedAsyncClient` ist mit Einstellungen vorkonfiguriert, die für die meisten Anwendungsfälle verwendet werden können. Sie können `AmazonSQSBufferedAsyncClient` weiter konfigurieren, z. B.:

1. Erstellen Sie eine Instance der Klasse `QueueBufferConfig` mit den erforderlichen Konfigurationsparametern.
2. Stellen Sie die Instance dem `AmazonSQSBufferedAsyncClient`-Konstruktor zur Verfügung.

```
// 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 Konfigurationsparameter

Parameter	Standardwert	Description
longPoll	true	Wenn longPoll auf true festgelegt ist, versucht AmazonSQSBufferedAsyncClient, Nachrichten durch langes Abrufen abzurufen.
longPollWaitTimeoutSeconds	20 s	Die maximale Dauer (in Sekunden), die ein ReceiveMessage - Aufruf beim Warten auf das Auftauchen von Nachrichten in der Warteschlange auf dem Server blockiert wird, bevor er mit einem leeren Empfangsergebnis zurückkehrt.  Note Wenn langes Abrufen deaktiviert ist, hat diese Einstellung keine Auswirkungen.

Parameter	Standardwert	Description
maxBatchOpenMs	200 ms	Die maximale Dauer (in Millisekunden), die ein ausgehender Aufruf auf andere Aufrufe desselben Typs wartet, mit denen er Nachrichten desselben Typs zu einem Stapel zusammenfasst. Je höher die Einstellung, desto weniger Stapel sind erforderlich, um dieselbe Anzahl an Aufgaben auszuführen (der erste Aufruf eines Stapels muss jedoch länger in der Warteschlange warten). Wenn Sie diesen Parameter auf 0 einstellen, warten gesendete Anforderungen nicht auf andere Anforderungen, wodurch die Stapelverarbeitung effektiv deaktiviert wird.

Parameter	Standardwert	Description
<code>maxBatchSize</code>	10 Anforderungen pro Stapel	Die maximale Anzahl von Nachrichten, die in einer einzelnen Stapelanforderung zusammengefasst werden. Je höher die Einstellung, desto weniger Stapel müssen die gleiche Anzahl von Anforderungen ausführen.  Note 10 Anforderungen pro Stapel ist der maximal zulässige Wert für Amazon SQS.
<code>maxBatchSizeBytes</code>	1 MiB	Die maximale Größe eines Nachrichtenstapels in Bytes, die der Client versucht, an Amazon SQS zu senden.  Note 1 MiB ist der maximal zulässige Wert für Amazon SQS.

Parameter	Standardwert	Description
<code>maxDoneReceiveBatches</code>	10 Stapel	Die maximale Anzahl von Empfangsstapeln, die <code>AmazonSQSBufferedAsyncClient</code> vorab abruft und clientseitig speichert. Je höher die Einstellung, desto mehr Empfangsanforderungen können erfüllt werden, ohne dass ein Aufruf an Amazon SQS gestartet werden muss (je mehr Nachrichten jedoch vorab abgerufen werden, desto länger bleiben sie im Puffer, was bedeutet, dass deren Zeitbeschränkung für die Sichtbarkeit abläuft).  Note 0 gibt an, dass der gesamte Vorabruf von Nachrichten deaktiviert ist und Nachrichten nur bei Bedarf abgerufen werden.

Parameter	Standardwert	Description
<code>maxInflightOutboundBatches</code>	5 Stapel	Die maximale Anzahl der aktiven ausgehenden Stapel, die gleichzeitig verarbeitet werden können. Je höher die Einstellung, desto schneller können ausgehende Stapel gesendet werden (in Abhängigkeit von anderen Kontingenten, z. B. durch CPU oder Bandbreite) und desto mehr Threads werden von <code>AmazonSQSBufferedAsyncClient</code> verbraucht.

Parameter	Standardwert	Description
<code>maxInflightReceiveBatches</code>	10 Stapel	Die maximale Anzahl der aktiven Empfangsstapel, die gleichzeitig verarbeitet werden können. Je höher die Einstellung, desto mehr Nachrichten können empfangen werden (in Abhängigkeit von anderen Kontingenten, z. B. durch CPU oder Bandbreite) und desto mehr Threads werden von <code>AmazonSQSBufferedAsyncClient</code> verbraucht.  Note 0 gibt an, dass der gesamte Vorabruf von Nachrichten deaktiviert ist und Nachrichten nur bei Bedarf abgerufen werden.

Parameter	Standardwert	Description
<code>visibilityTimeoutSeconds</code>	-1	Wenn dieser Parameter auf einen positiven Wert ungleich null festgelegt ist, überschreibt die hier festgelegte Zeitbeschränkung für die Sichtbarkeit diejenige, die für die Warteschlange festgelegt ist, über die Nachrichten abgerufen werden.  Note -1 gibt an, dass die Standardeinstellung für die Warteschlange ausgewählt ist. Sie können als Zeitbeschränkung für die Sichtbarkeit nicht 0 festlegen.

AWS SDK for Java 2.x

Für AWS SDK for Java 2.x können Sie ein neues erstellen, das auf dem folgenden Beispiel `SqsAsyncBatchManager` basiert:

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

Nachdem Sie den neuen `SqsAsyncBatchManager` erstellt haben, können Sie ihn verwenden, um mehrere Anforderungen an Amazon SQS zu senden (wie mit dem `SqsAsyncClient`), beispielsweise:

```
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);
```

Konfiguration `SqsAsyncBatchManager`

`SqsAsyncBatchManager` ist mit Einstellungen vorkonfiguriert, die für die meisten Anwendungsfälle verwendet werden können. Sie können `SqsAsyncBatchManager` weiter konfigurieren, z. B.:

Erstellen einer benutzerdefinierten Konfiguration über `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-Parameter

Parameter	Standardwert	Description
maxBatchSize	10 Anforderungen pro Stapel	Die maximale Anzahl von Nachrichten, die in einer einzelnen Stapelanforderung zusammengefasst werden. Je höher die Einstellung, desto weniger Stapel müssen die gleiche Anzahl von Anforderungen ausführen.
 Note Der zulässige Höchstwert für Amazon SQS beträgt 10 Anfragen pro Stapel.		
sendRequestFrequency	200 ms	Die maximale Dauer (in Millisekunden), die ein ausgehender Aufruf auf

Parameter	Standardwert	Description
		andere Aufrufe desselben Typs wartet, mit denen er Nachrichten desselben Typs zu einem Stapel zusammenfasst. Je höher die Einstellung, desto weniger Stapel sind erforderlich, um dieselbe Anzahl an Aufgaben auszuführen (der erste Aufruf eines Stapels muss jedoch länger in der Warteschlange warten). Wenn Sie diesen Parameter auf 0 einstellen, warten gesendete Anforderungen nicht auf andere Anforderungen, wodurch die Stapelverarbeitung effektiv deaktiviert wird.

Parameter	Standardwert	Description
<code>receiveMessageVisibilityTimeout</code>	-1	Wenn dieser Parameter auf einen positiven Wert ungleich null festgelegt ist, überschreibt die hier festgelegte Zeitbeschränkung für die Sichtbarkeit diejenige, die für die Warteschlange festgelegt ist, über die Nachrichten abgerufen werden.  Note 1 gibt an, dass die Standardeinstellung für die Warteschlange ausgewählt ist. Sie können als Zeitbeschränkung für die Sichtbarkeit nicht 0 festlegen.
<code>receiveMessageMinWaitDuration</code>	50 ms	Die Mindestzeit (in Millisekunden), die ein <code>receiveMessage</code> Anruf darauf wartet, dass verfügbare Nachrichten abgerufen werden. Je höher die Einstellung, desto weniger Batches sind erforderlich, um dieselbe Anzahl von Anfragen auszuführen.

Steigerung des Durchsatzes durch horizontale Skalierung und Action-Batching mit Amazon SQS

Amazon SQS unterstützt Nachrichtenübermittlung mit hohem Durchsatz. Einzelheiten zu Durchsatzgrenzen finden Sie unter: [Amazon SQS SQS-Nachrichtenkontingente](#)

So maximieren Sie den Durchsatz:

- [Skalieren](#) Sie Hersteller und Verbraucher horizontal, indem Sie jeweils weitere Instanzen hinzufügen.
- Verwenden Sie [Action-Batching](#), um mehrere Nachrichten in einer einzigen Anfrage zu senden oder zu empfangen und so den Aufwand für API-Aufrufe zu reduzieren.

Horizontale Skalierung

Da Sie über ein HTTP-Anfrage-Antwort-Protokoll auf Amazon SQS zugreifen, beschränkt die Anforderungslatenz (der Zeitraum zwischen dem Initiieren einer Anforderung und dem Empfangen einer Antwort) den Durchsatz, den Sie über einen einzelnen Thread über eine einzelne Verbindung erzielen können. Wenn beispielsweise der Mittelwert der Latenz eines auf EC2 basierenden Clients zu Amazon SQS in derselben Region ca. 20 ms beträgt, liegt der Mittelwert des maximalen Durchsatzes eines einzelnen Threads über eine einzelne Verbindung bei 50 TPS.

Horizontale Skalierung bedeutet, dass die Anzahl Ihrer Nachrichtenproduzenten (die [SendMessage](#)-Anforderungen erstellen) und Konsumenten (die [ReceiveMessage](#)- und [DeleteMessage](#)-Anforderungen erstellen) erhöht wird, um den Gesamtdurchsatz der Warteschlange zu steigern. Sie können auf drei Arten horizontal skalieren:

- Erhöhen der Anzahl der Threads pro Client
- Hinzufügen weiterer Clients
- Erhöhen der Anzahl der Threads pro Client und Hinzufügen weiterer Clients

Wenn Sie weitere Clients hinzufügen, erzielen Sie eine wesentliche lineare Steigerung des Durchsatzes der Warteschlange. Wenn Sie die Anzahl der Clients beispielsweise verdoppeln, verdoppeln Sie auch den Durchsatz.

Stapelverarbeitung von Aktionen

Mit der Stapelverarbeitung wird in den einzelnen Roundtrips an den Service mehr Arbeit ausgeführt (z. B. das Senden mehrerer Nachrichten mit einer einzelnen `SendMessageBatch`-Anforderung). Die Amazon-SQS-Stapelaktionen sind [SendMessageBatch](#), [DeleteMessageBatch](#) und [ChangeMessageVisibilityBatch](#). Um die Stapelverarbeitung zu nutzen, ohne Ihre Produzenten und Konsumenten zu ändern, können Sie den [Amazon SQS Buffered Asynchronous Client](#) verwenden.

Note

Da mit [ReceiveMessage](#) 10 Nachrichten gleichzeitig verarbeitet werden können, wird keine `ReceiveMessageBatch`-Aktion ausgeführt.

Die Stapelverarbeitung verteilt die Latenz der Stapelaktion über mehrere Nachrichten in einer Stapelanforderung, anstatt die gesamte Latenz für eine einzelne Nachricht (z. B. eine [SendMessage](#)-Anforderung) zu akzeptieren. Da jeder Roundtrip mehr Arbeit verrichtet, nutzen Stapelanforderungen Threads und Verbindungen effektiver und verbessern somit den Durchsatz.

Sie können die Stapelverarbeitung mit der horizontalen Skalierung kombinieren, um einen Durchsatz mit weniger Threads, Verbindungen und Anforderungen zu bieten, als dies bei einzelnen Nachrichtenforderungen der Fall ist. Mit Amazon-SQS-Aktionen im Stapel können Sie bis zu 10 Nachrichten gleichzeitig senden, empfangen oder löschen. Da in Amazon SQS Gebühren nach Anforderung berechnet werden, kann die Stapelverarbeitung wesentlich zur Verringerung der Kosten beitragen.

Mit der Stapelverarbeitung kann eine gewisse Komplexität für Ihre Anwendung einhergehen (z. B. muss die Anwendung Nachrichten vor der Übermittlung sammeln oder gelegentlich länger auf eine Antwort warten). Die Stapelverarbeitung kann in folgenden Fällen jedoch weiterhin effektiv sein:

- Ihre Anwendung erstellt in kurzer Zeit viele Nachrichten, sodass es niemals zu einer sehr langen Verzögerung kommt.
- Anders als typische Nachrichtenproduzenten, die Nachrichten als Antwort auf Ereignisse senden müssen, die sie nicht steuern können, ruft ein Nachrichtenkonsument Nachrichten nach eigenem Ermessen aus einer Warteschlange ab.

⚠ Important

Eine Stapelanforderung kann erfolgreich ausgeführt werden, auch wenn einzelne Nachrichten im Stapel fehlgeschlagen sind. Überprüfen Sie nach einer Stapelanforderung stets, ob einzelne Nachrichten nicht zugestellt werden konnten, und führen Sie diese bei Bedarf erneut aus.

Funktionierendes Java-Beispiel für einzelne Operationsanforderungen und Stapelanforderungen

Voraussetzungen

Fügen Sie dem Pfad für Ihre Java-Build-Klasse die Pakete `aws-java-sdk-sqs.jar`, `aws-java-sdk-ec2.jar` und `commons-logging.jar` hinzu. Das folgende Beispiel zeigt diese Abhängigkeiten in der `pom.xml`-Datei eines Maven-Projekts.

```
<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

Das nachstehende Java-Code-Beispiel veranschaulicht ein einfaches Produzent-Konsument-Muster. Der Haupt-Thread ruft eine Reihe von Produzenten- und Konsumenten-Threads auf, die 1-KB-Nachrichten für eine bestimmte Zeit verarbeiten. Dieses Beispiel enthält Produzenten und

Konsumenten, die einzelne Operationsanforderungen senden, und solche, die Stapelanforderungen senden.

```
/*
 * 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);
                }
            }
        } catch (Exception e) {
            // Handle exception
        }
    }
}
```

```

        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.
    }
}
}
```

Überwachen von Volume-Metriken aus der Beispielausführung

Amazon SQS erstellt automatisch Volumen-Metriken für gesendete, empfangene und gelöschte Nachrichten. Auf diese und andere Metriken können Sie über die Registerkarte Monitoring (Überwachung) für Ihre Warteschlange oder in der [CloudWatch -Konsole](#) zugreifen.

Note

Nach dem Starten der Warteschlange kann es bis zu 15 Minuten dauern, bis die Metriken verfügbar sind.

Amazon SQS mit einem AWS SDK verwenden

AWS Software Development Kits (SDKs) sind für viele gängige Programmiersprachen verfügbar. Jedes SDK bietet eine API, Codebeispiele und Dokumentation, die es Entwicklern erleichtern, Anwendungen in ihrer bevorzugten Sprache zu erstellen.

SDK-Dokumentation	Codebeispiele
AWS SDK für C++	AWS SDK für C++ Codebeispiele
AWS CLI	AWS CLI Code-Beispiele
AWS SDK für Go	AWS SDK für Go Code-Beispiele
AWS SDK für Java	AWS SDK für Java Code-Beispiele
AWS SDK für JavaScript	AWS SDK für JavaScript Code-Beispiele
AWS SDK für Kotlin	AWS SDK für Kotlin Code-Beispiele
AWS SDK für .NET	AWS SDK für .NET Code-Beispiele
AWS SDK für PHP	AWS SDK für PHP Code-Beispiele
AWS -Tools für PowerShell	AWS -Tools für PowerShell Code-Beispiele
AWS SDK für Python (Boto3)	AWS SDK für Python (Boto3) Code-Beispiele
AWS SDK für Ruby	AWS SDK für Ruby Code-Beispiele
AWS SDK für Rust	AWS SDK für Rust Code-Beispiele
AWS SDK für SAP ABAP	AWS SDK für SAP ABAP Code-Beispiele
AWS SDK für Swift	AWS SDK für Swift Code-Beispiele

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie ein Codebeispiel an, indem Sie unten den Link [Provide feedback \(Feedback geben\)](#) auswählen.

Verwenden von JMS mit Amazon SQS

Die Amazon SQS Java Messaging Library ist eine Java-Message-Service (JMS)-Schnittstelle für Amazon SQS, mit der Sie Amazon SQS in Anwendungen nutzen können, die bereits JMS verwenden. Mithilfe der Schnittstelle können Sie mit nur wenigen Codeänderungen Amazon SQS als JMS-Anbieter verwenden. In Verbindung mit der AWS SDK für Java können Sie mithilfe der Amazon SQS Java Messaging Library JMS-Verbindungen und -Sitzungen sowie Produzenten und Konsumenten erstellen, die Nachrichten an und von Amazon-SQS-Warteschlangen senden und empfangen.

Die Bibliothek unterstützt das Senden und Empfangen von Nachrichten an eine Warteschlange (das point-to-point JMS-Modell) gemäß der [JMS 1.1-Spezifikation](#). Die Bibliothek unterstützt das synchrone Senden von Text-, Byte- und Objektnachrichten an Amazon-SQS-Warteschlangen. Außerdem wird das synchrone und asynchrone Empfangen von Objekten unterstützt.

Informationen zu Funktionen der Amazon SQS Java Messaging Library, die die JMS 1.1-Spezifikation unterstützen, finden Sie unter [Von Amazon SQS unterstützte JMS 1.1-Implementierungen](#) und [Amazon](#) SQS. FAQs

Voraussetzungen für die Arbeit mit JMS und Amazon SQS

Folgende Voraussetzungen müssen erfüllt sein, bevor Sie den Vorgang starten:

- SDK für Java

Es gibt zwei Möglichkeiten, das SDK für Java in Ihr Projekt einzubinden:

- Laden Sie das SDK für Java herunter und installieren Sie es.
- Verwenden Sie Maven, um die Amazon SQS Java Messaging Library zu erhalten.

Note

Das SDK für Java ist als Abhängigkeit enthalten.

Für das [SDK für Java](#) und die Amazon SQS Extended Client Library für Java ist das J2SE Development Kit 8.0 oder eine neuere Version erforderlich.

Weitere Informationen zum Herunterladen des SDK für Java finden Sie unter [SDK für Java](#).

- Amazon SQS Java Messaging Library

Wenn Sie Maven nicht verwenden, müssen Sie das Paket `amazon-sqs-java-messaging-lib.jar` zum Java-Erstellungspfad hinzufügen. Weitere Informationen zum Herunterladen der Bibliothek finden Sie unter [Amazon SQS Java Messaging Library](#).

 Note

Die Amazon SQS Java Messaging Library bietet Unterstützung für [Maven](#) und das [Spring Framework](#).

Codebeispiele, in denen Maven, das Spring Framework und die Amazon SQS Java Messaging Library verwendet werden, finden Sie unter [Funktionierende Java-Beispiele für die Verwendung von JMS mit Amazon SQS SQS-Standardwarteschlangen](#).

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
  <version>1.0.4</version>
  <type>jar</type>
</dependency>
```

- Amazon-SQS-Warteschlange

Erstellen Sie eine Warteschlange mit der AWS-Managementkonsole für Amazon SQS, der `CreateQueue` API oder dem verpackten Amazon SQS-Client, der in der Amazon SQS Java Messaging Library enthalten ist.

- Weitere Informationen zum Erstellen einer Warteschlange mit Amazon SQS mithilfe der AWS-Managementkonsole oder der `CreateQueue`-API finden Sie unter [Erstellen einer Warteschlange](#).
- Weitere Informationen zur Verwendung der Amazon SQS Java Messaging Library finden Sie unter [Verwenden der Amazon SQS Java Messaging-Bibliothek](#).

Verwenden der Amazon SQS Java Messaging-Bibliothek

Verwenden Sie die Codebeispiele in diesem Abschnitt, um mit der Verwendung des Java Message Service (JMS) mit Amazon SQS zu beginnen. In den folgenden Abschnitten erfahren Sie, wie Sie eine JMS-Verbindung und eine Sitzung erstellen sowie eine Nachricht senden und empfangen.

Das integrierte Amazon-SQS-Clientobjekt der Amazon SQS Java Messaging Library prüft, ob bereits eine Amazon-SQS-Warteschlange vorhanden ist. Ist dies nicht der Fall, wird sie vom Client erstellt.

Erstellen einer JMS-Verbindung

Bevor Sie beginnen, informieren Sie sich über die Voraussetzungen unter [Voraussetzungen für die Arbeit mit JMS und Amazon SQS](#).

1. Erstellen Sie eine Verbindungs-Factory und rufen Sie die Methode `createConnection` für die Factory auf.

```
// 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();
```

Die Klasse `SQSConnection` erweitert `javax.jms.Connection`. In Verbindung mit den JMS-Standardverbindungsmethoden bietet `SQSConnection` zusätzliche Methoden wie `getAmazonSQSClient` und `getWrappedAmazonSQSClient`. Mit beiden Methoden lassen sich administrative Aufgaben wie das Erstellen neuer Warteschlangen ausführen, die in der JMS-Spezifikation nicht enthalten sind. Die `getWrappedAmazonSQSClient`-Methode stellt jedoch auch eine integrierte Version des Amazon-SQS-Client bereit, die von der aktuellen Verbindung verwendet wird. Der Wrapper wandelt alle Ausnahmen des Clients in eine `JMSEException` um. So können diese von bestehendem Code, in dem Vorkommen von `JMSEException` erwartet werden, einfacher verwendet werden.

2. Sie können die von `getAmazonSQSClient` und `getWrappedAmazonSQSClient` zurückgegebenen Clientobjekte für administrative Aufgaben verwenden, die in der JMS-Spezifikation nicht enthalten sind (z. B. Erstellen einer neuen Amazon-SQS-Warteschlange).

Wenn Ihr bestehender Code JMS-Ausnahmen erwartet, müssen Sie `getWrappedAmazonSQSClient` verwenden:

- Wenn Sie `getWrappedAmazonSQSClient` verwenden, wandelt das zurückgegebene Clientobjekt alle Ausnahmen in JMS-Ausnahmen um.

- Wenn Sie `getAmazonSQSClient` verwenden, sind alle Ausnahmen Amazon-SQS-Ausnahmen.

Erstellen einer Amazon-SQS-Warteschlange

Das integrierte Clientobjekt prüft, ob eine Amazon-SQS-Warteschlange vorhanden ist.

Ist dies nicht der Fall, wird sie vom Client erstellt. Ist bereits eine Warteschlange vorhanden, gibt die Funktion nichts zurück. Weitere Informationen finden Sie im Abschnitt "Create the queue if needed" im Beispiel [TextMessageSender.java](#).

So erstellen Sie eine Standardwarteschlange

```
// 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");
}
```

So erstellen Sie eine FIFO-Warteschlange

```
// 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

Der Name einer FIFO-Warteschlange muss mit dem Suffix `.fifo` enden.

Weitere Informationen zum `ContentBasedDeduplication`-Attribut finden Sie unter [Exactly-Once-Verarbeitung in Amazon SQS](#).

Synchrones Senden von Nachrichten

1. Wenn die Verbindung und die zugrundeliegende Amazon-SQS-Warteschlange bereit sind, erstellen Sie eine nicht transaktionsorientierte JMS-Sitzung mit dem Modus `AUTO_ACKNOWLEDGE`.

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. Erstellen Sie zum Senden einer Textnachricht an die Warteschlange eine JMS-Warteschlangenidentität und einen Nachrichtenproduzenten.

```
// 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. Erstellen Sie eine Textnachricht und senden Sie sie an die Warteschlange.

- Um eine Nachricht an eine Standard-Warteschlange zu senden, müssen Sie keine weiteren Parameter festlegen.

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

- Um eine Nachricht an eine FIFO-Warteschlange zu senden, müssen Sie die Nachrichtengruppen-ID festlegen. Darüber hinaus können Sie eine Nachrichteneduplizierungs-ID festlegen. Weitere Informationen finden Sie unter [Schlüsselbegriffe der Amazon SQS FIFO-Warteschlange](#).

```
// 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"));
```

Synchrones Empfangen von Nachrichten

1. Erstellen Sie zum Empfangen von Nachrichten einen Konsumenten für dieselbe Warteschlange und rufen Sie die Methode `start` auf.

Sie können die Methode `start` für die Verbindung jederzeit aufrufen. Der Konsument beginnt jedoch erst mit dem Abrufen von Nachrichten, wenn Sie die Methode aufrufen.

```
// Create a consumer for the 'MyQueue'
MessageConsumer consumer = session.createConsumer(queue);
// Start receiving incoming messages
connection.start();
```

2. Rufen Sie die Methode `receive` mit einer Zeitbeschränkung von 1 Sekunde auf dem Konsumenten auf und geben Sie den Inhalt der empfangenen Nachricht aus.
- Nachdem Sie eine Nachricht aus einer Standard-Warteschlange abgerufen haben, können Sie auf den Inhalt der Nachricht zugreifen.

```
// 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());
}
```

```
}
```

- Nachdem Sie eine Nachricht aus einer FIFO-Warteschlange abgerufen haben, können Sie auf den Inhalt der Nachricht sowie auf andere FIFO-spezifische Nachrichtenattribute wie die Nachrichtengruppen-ID, die Nachrichteneduplizierungs-ID und die Sequenznummer zugreifen. Weitere Informationen finden Sie unter [Schlüsselbegriffe der Amazon SQS FIFO-Warteschlange](#).

```
// 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. Schließen Sie die Verbindung und die Sitzung.

```
// Close the connection (and the session).
connection.close();
```

Die Ausgabe sieht folgendermaßen oder ähnlich aus:

```
JMS Message ID:8example-588b-44e5-bbcb-d816example2
Received: Hello World!
```

Note

Sie können das Spring Framework zum Initialisieren dieser Objekte verwenden. Weitere Informationen finden Sie unter `SpringExampleConfiguration.xml`, `SpringExample.java`. Informationen zu den anderen unterstützenden Klassen finden Sie unter `ExampleConfiguration.java` und `ExampleCommon.java` im Abschnitt

Funktionierende Java-Beispiele für die Verwendung von JMS mit Amazon SQS SQS-Standardwarteschlangen.

Umfassende Beispiele für das Senden und Empfangen von Objekten finden Sie unter [TextMessageSender.java](#) und [SyncMessageReceiver.java](#).

Asynchrones Empfangen von Nachrichten

Im Beispiel unter [Verwenden der Amazon SQS Java Messaging-Bibliothek](#) wird eine Nachricht an MyQueue gesendet und synchron empfangen.

Im folgenden Beispiel wird gezeigt, wie Nachrichten mithilfe eines Listeners asynchron empfangen werden können.

1. Implementieren Sie die `MessageListener`-Schnittstelle.

```
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 (JMSEException e) {
            e.printStackTrace();
        }
    }
}
```

Beim Empfang einer Nachricht wird die Methode `onMessage` der `MessageListener`-Schnittstelle aufgerufen. In dieser Implementierung des Listeners wird der in der Nachricht gespeicherte Text ausgegeben.

2. Statt die Methode `receive` auf dem Konsumenten explizit aufzurufen, richten Sie den Nachrichten-Listener des Konsumenten auf einer Instance der `MyListener`-Implementierung ein. Der Haupt-Thread wartet eine Sekunde lang.

```
// 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);
```

Die restlichen Schritte sind identisch wie im Beispiel [Verwenden der Amazon SQS Java Messaging-Bibliothek](#). Ein umfassendes Beispiel für einen asynchronen Konsumenten finden Sie unter `AsyncMessageReceiver.java` in den [Funktionierende Java-Beispiele für die Verwendung von JMS mit Amazon SQS SQS-Standardwarteschlangen](#).

Die Ausgabe für dieses Beispiel sieht in etwa wie folgt aus:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Verwenden des Client-Bestätigungsmodus

In dem Beispiel in [Verwenden der Amazon SQS Java Messaging-Bibliothek](#) wird der `AUTO_ACKNOWLEDGE`-Modus verwendet, um jede empfangene Nachricht automatisch zu bestätigen (und somit aus der zugrundeliegenden Amazon-SQS-Warteschlange zu löschen).

1. Um die Nachrichten nach der Verarbeitung explizit zu bestätigen, müssen Sie die Sitzung im `CLIENT_ACKNOWLEDGE`-Modus erstellen.

```
// Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

2. Zeigen Sie die Nachricht nach dem Empfangen an und bestätigen Sie sie explizit.

```
// 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

Wenn eine Nachricht in diesem Modus bestätigt wird, werden alle Nachrichten, die vor dieser Nachricht empfangen wurden, ebenfalls implizit bestätigt. Wenn Sie beispielsweise 10 Nachrichten empfangen haben und nur die 10. Nachricht bestätigen (in der Reihenfolge, in der die Nachrichten empfangen wurden) werden die vorherigen 9 Nachrichten ebenfalls bestätigt.

Die restlichen Schritte sind identisch wie im Beispiel [Verwenden der Amazon SQS Java Messaging-Bibliothek](#). Ein umfassendes Beispiel für einen synchronen Konsumenten im Client-Bestätigungsmodus finden Sie unter `SyncMessageReceiverClientAcknowledge.java` in den [Funktionierende Java-Beispiele für die Verwendung von JMS mit Amazon SQS SQS-Standardwarteschlangen](#).

Die Ausgabe für dieses Beispiel sieht in etwa wie folgt aus:

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5  
Received: Hello World!  
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

Verwenden des ungeordneten Bestätigungsmodus

Wenn Sie den `CLIENT_ACKNOWLEDGE`-Modus verwenden, werden alle Nachrichten, die vor einer explizit bestätigten Nachricht empfangen wurden, automatisch bestätigt. Weitere Informationen finden Sie unter [Verwenden des Client-Bestätigungsmodus](#).

Die Amazon SQS Java Messaging Library stellt einen weiteren Bestätigungsmodus bereit. Wenn Sie den `UNORDERED_ACKNOWLEDGE`-Modus verwenden, müssen alle empfangenen Nachrichten unabhängig von der Empfangsreihenfolge einzeln und explizit vom Client bestätigt werden. Erstellen Sie hierzu eine Sitzung im `UNORDERED_ACKNOWLEDGE`-Modus.

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.  
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

Die restlichen Schritte sind identisch wie im Beispiel [Verwenden des Client-Bestätigungsmodus](#). Ein umfassendes Beispiel für einen synchronen Konsumenten im UNORDERED_ACKNOWLEDGE-Modus finden Sie unter `SyncMessageReceiverUnorderedAcknowledge.java`.

In diesem Beispiel sieht die Ausgabe in etwa wie folgt aus:

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Verwenden des Java Message Service mit anderen Amazon SQS SQS-Clients

Wenn Sie den Amazon SQS Java Message Service (JMS) -Client mit dem AWS SDK verwenden, wird die Amazon SQS SQS-Nachrichtengröße auf 256 KB begrenzt. Sie können jedoch mit einem beliebigen Amazon-SQS-Client einen JMS-Anbieter erstellen. Verwenden Sie beispielsweise den JMS-Client mit der Amazon SQS Extended Client Library für Java, um eine Amazon-SQS-Nachricht zu senden, die einen Verweis auf eine Nachrichtennutzlast (bis zu 2 GB) in Amazon S3 enthält. Weitere Informationen finden Sie unter [Verwaltung großer Amazon SQS SQS-Nachrichten mit Java und Amazon S3](#).

Das folgende Java-Codebeispiel erstellt den JMS-Provider für die Extended Client Library.

Sehen Sie sich die Voraussetzungen unter an, [Voraussetzungen für die Arbeit mit JMS und Amazon SQS](#) bevor Sie dieses Beispiel testen.

```
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);
```

Im folgenden Java-Codebeispiel wird die Verbindungs-Factory erstellt:

```
// 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();
```

Funktionierende Java-Beispiele für die Verwendung von JMS mit Amazon SQS SQS-Standardwarteschlangen

Das folgende Codebeispiel veranschaulicht die Verwendung des Java Message Service (JMS) mit Amazon-SQS-Standard-Warteschlangen. Weitere Informationen zur Arbeit mit FIFO-Warteschlangen finden Sie unter [So erstellen Sie eine FIFO-Warteschlange](#), [Synchrones Senden von Nachrichten](#) und [Synchrones Empfangen von Nachrichten](#). (Der synchrone Empfang von Nachrichten ist für Standard- und FIFO-Warteschlangen identisch. Nachrichten in FIFO-Warteschlangen enthalten jedoch mehr Attribute.)

Sehen Sie sich die Voraussetzungen unter an, [Voraussetzungen für die Arbeit mit JMS und Amazon SQS](#) bevor Sie die folgenden Beispiele testen.

ExampleConfiguration.java

Das folgende Codebeispiel für Java SDK v 1.x legt den Standard-Warteschlangennamen, die Region und die Anmeldeinformationen fest, die mit den anderen Java-Beispielen verwendet werden sollen.

```
/*
 * 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

Im folgenden Java-Codebeispiel wird ein Textnachrichtenproduzent erstellt.

```
/*
 * 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 JMSEException {
        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

Im folgenden Java-Codebeispiel wird ein synchroner Nachrichtenkonsument erstellt.

```

/*
 * 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 JMSEException {
    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 (JMSEException e) {
    System.err.println( "Error receiving from SQS: " + e.getMessage() );
    e.printStackTrace();
}
}
}

```

AsyncMessageReceiver.java

Im folgenden Java-Codebeispiel wird ein asynchroner Nachrichtenkonsument erstellt.

```

/*
 * 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 JMSEException, 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.getMessageID() );
        timeOfLastMessage = System.nanoTime();
    } catch (JMSEException e) {
        System.err.println( "Error processing message: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
}
}

```

SyncMessageReceiverClientAcknowledge.java

Im folgenden Java-Codebeispiel wird ein synchroner Konsument im Client-Bestätigungsmodus erstellt.

```

/*
 * 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 JMSEException, 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 JMSEException
 */
private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSEException {
    // 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 JMSEException
 */
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSEException {
    // 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

Im folgenden Java-Codebeispiel wird ein synchroner Konsument im ungeordneten Bestätigungsmodus erstellt.

```
/*
 * 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 JMSEException, 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 JMSEException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSEException {
        // 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 JMSEException
*/
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSEException {
    // 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

Das folgende XML-Codebeispiel ist eine Bean-Konfigurationsdatei für [SpringExample.java](#).

```

<!--
    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

Im folgenden Java-Codebeispiel wird die Bean-Konfigurationsdatei verwendet, um Ihre Objekte zu initialisieren.

```
/*
 * 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 JMSEException {
        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 (JMSEException e) {
    System.err.println( "Error receiving from SQS: " + e.getMessage() );
    e.printStackTrace();
}
}
}

```

ExampleCommon.java

Im folgenden Java-Codebeispiel wird geprüft, ob bereits eine Amazon-SQS-Warteschlange vorhanden ist, und gegebenenfalls eine neue Warteschlange erstellt. Außerdem ist Beispiel-Code für die Protokollierung enthalten.

```

/*
 * 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 JMSEException {
        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 JMSEException {
        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() );
        }
    }
}

```

```
}
```

Von Amazon SQS unterstützte JMS 1.1-Implementierungen

Die Amazon SQS Java Messaging Library unterstützt die folgenden [JMS-1.1-Implementierungen](#). Weitere Informationen zu den unterstützten Features und Fähigkeiten der Amazon SQS Java Messaging Library finden Sie in den [Häufig gestellten Fragen zu Amazon SQS](#).

Unterstützte gängige Schnittstellen

- `Connection`
- `ConnectionFactory`
- `Destination`
- `Session`
- `MessageConsumer`
- `MessageProducer`

Unterstützte Nachrichtentypen

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

Unterstützte Nachrichtenbestätigungsmodi

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

 Note

Der `UNORDERED_ACKNOWLEDGE`-Modus ist nicht Bestandteil der JMS 1.1-Spezifikation. Mithilfe dieses Modus kann Amazon SQS einem JMS-Client explizit die Bestätigung einer Nachricht erlauben.

JMS-definierte Kopfzeilen und reservierte Eigenschaften

Für den Nachrichtenversand

Beim Senden von Nachrichten können Sie die folgenden Kopfzeilen und Eigenschaften für einzelne Nachrichten festlegen:

- `JMSXGroupID` (erforderlich für FIFO-Warteschlangen, nicht zulässig für Standard-Warteschlangen)
- `JMS_SQS_DeduplicationId` (optional für FIFO-Warteschlangen, nicht zulässig für Standard-Warteschlangen)

Nach dem Senden von Nachrichten legt Amazon SQS die folgenden Kopfzeilen und Eigenschaften für einzelne Nachrichten fest:

- `JMSMessageID`
- `JMS_SQS_SequenceNumber` (nur für FIFO-Warteschlangen)

Für den Nachrichtenempfang

Beim Empfang von Nachrichten legt Amazon SQS die folgenden Kopfzeilen und Eigenschaften für einzelne Nachrichten fest:

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID` (nur für FIFO-Warteschlangen)
- `JMS_SQS_DeduplicationId` (nur für FIFO-Warteschlangen)
- `JMS_SQS_SequenceNumber` (nur für FIFO-Warteschlangen)

Tutorials zu Amazon SQS

In diesem Thema finden Sie Tutorials, mit denen Sie sich mit den Funktionen und Funktionen von Amazon SQS vertraut machen können.

Tutorials

- [Erstellen einer Amazon SQS SQS-Warteschlange mit CloudFormation](#)
- [Tutorial: Senden einer Nachricht an eine Amazon-SQS-Warteschlange aus Amazon Virtual Private Cloud](#)

Erstellen einer Amazon SQS SQS-Warteschlange mit CloudFormation

Verwenden Sie die CloudFormation Konsole zusammen mit einer JSON- oder YAML-Vorlage, um eine Amazon SQS SQS-Warteschlange zu erstellen. Weitere Informationen finden Sie unter [Arbeiten mit CloudFormation Vorlagen](#) und der [AWS::SQS::QueueResource](#) im AWS CloudFormation Benutzerhandbuch.

Wird verwendet, CloudFormation um eine Amazon SQS SQS-Warteschlange zu erstellen.

1. Kopieren Sie den JSON-Code in eine Datei mit dem Namen `MyQueue.json`. Lassen Sie zum Erstellen einer Standard-Warteschlange die Eigenschaften `FifoQueue` und `ContentBasedDeduplication` weg. Weitere Informationen zur inhaltsbasierten Deduplizierung finden Sie unter [Exactly-Once-Verarbeitung in Amazon SQS](#).

Note

Der Name einer FIFO-Warteschlange muss mit dem Suffix `.fifo` enden.

```
{
  "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. Melden Sie sich bei der [CloudFormation -Konsole](#) an und wählen Sie dann Create Stack (Stack erstellen) aus.
3. Wählen Sie im Bereich Specify Template (Vorlage angeben) die Option Upload a template file (Vorlagendatei hochladen) aus, wählen Sie Ihre MyQueue.json-Datei aus und klicken Sie dann auf Next (Weiter).
4. Geben Sie auf der Seite Specify Details MyQueue unter Stack Name ein und wählen Sie Next aus.
5. Wählen Sie auf der Seite Optionen Weiter aus.

6. Klicken Sie auf der Seite Review auf Create.

CloudFormation beginnt mit der Erstellung des MyQueue Stacks und zeigt den Status `CREATE_IN_PROGRESS` an. Wenn der Prozess abgeschlossen ist, zeigt CloudFormation den Status `CREATE_COMPLETE` an.

Filter: Active ▾		By Stack Name		Showing 1 stack	
	Stack Name	Created Time	Status	Description	
☒	MyQueue	2017-02-20 11:39:47 UTC-0800	CREATE_COMPLETE		

7. (Optional) Um den Namen, die URL und den ARN der Warteschlange anzuzeigen, wählen Sie den Namen des Stacks aus und erweitern Sie auf der nächsten Seite den Abschnitt Outputs.

Tutorial: Senden einer Nachricht an eine Amazon-SQS-Warteschlange aus Amazon Virtual Private Cloud

Dieses Tutorial zeigt Ihnen, wie Sie Nachrichten über ein sicheres, privates Netzwerk an eine Amazon SQS SQS-Warteschlange senden. Das Netzwerk umfasst:

- Eine VPC, die eine Amazon EC2 EC2-Instance enthält.
- Ein VPC-Schnittstellen-Endpunkt, der es der Amazon EC2 EC2-Instance ermöglicht, sich mit Amazon SQS zu verbinden, ohne das öffentliche Internet zu nutzen.

Selbst in einem vollständig privaten Netzwerk können Sie eine Verbindung zur Amazon EC2 EC2-Instance herstellen und Nachrichten an die Amazon SQS SQS-Warteschlange senden. Weitere Informationen finden Sie unter [Endpunkte von Amazon Virtual Private Cloud für Amazon SQS](#).

Important

- Sie können Amazon Virtual Private Cloud nur mit HTTPS-Amazon-SQS-Endpunkten verwenden.
- Wenn Sie Amazon SQS für das Senden von Nachrichten von Amazon VPC konfigurieren, müssen Sie privates DNS aktivieren und Endpunkte im Format `sqs.us-east-2.amazonaws.com` oder `sqs.us-east-2.api.aws` für den Dual-Stack-Endpunkt angeben.

- Amazon SQS unterstützt auch FIPS-Endpunkte mithilfe des `com.amazonaws.region.sqs-fips` Endpoint Service. Sie können im folgenden Format eine Verbindung zu FIPS-Endpunkten herstellen: `sqs-fips.region.amazonaws.com`
- Bei Verwendung des Dual-Stack-Endpunkts in Amazon Virtual Private Cloud werden Anfragen mit IPv4 und IPv6 gesendet.
- Ein privates DNS unterstützt keine Legacy-Endpunkte wie z. B. `queue.amazonaws.com` oder `us-east-2.queue.amazonaws.com`.

Schritt 1: Erstellen eines Amazon EC2-Schlüsselpaares

Ein Schlüsselpaar ermöglicht Ihnen die Verbindung mit einer EC2-Instance. Es besteht aus einem öffentlichen Schlüssel, mit dem Ihre Anmeldeinformationen verschlüsselt werden, und einem privaten Schlüssel, mit dem sie entschlüsselt werden.

1. Melden Sie sich bei der [Amazon-EC2-Konsole](#) an.
2. Klicken Sie im Navigationsmenü unter Network & Security (Netzwerk und Sicherheit) auf Key Pairs (Schlüsselpaare).
3. Wählen Sie Create Key Pair aus.
4. Geben Sie im Dialogfeld Create Key Pair (Schlüsselpaar erstellen) im Feld Key pair name (Schlüsselpaarname) als Namen `SQS-VPCE-Tutorial-Key-Pair` ein und klicken Sie auf Create (Erstellen).
5. Ihr Browser lädt die private Schlüsseldatei `SQS-VPCE-Tutorial-Key-Pair.pem` automatisch herunter.

Important

Speichern Sie diese Datei an einem sicheren Ort. EC2 generiert eine `.pem`-Datei für dasselbe Schlüsselpaar kein zweites Mal.

6. Um einem SSH-Client das Herstellen einer Verbindung mit Ihrer EC2-Instance zu ermöglichen, legen Sie die Berechtigungen für Ihre private Schlüsseldatei so fest, dass nur Ihrem Benutzer Leseberechtigungen für sie gewährt werden, zum Beispiel:

```
chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
```

Schritt 2: Ressourcen erstellen AWS

Um die erforderliche Infrastruktur einzurichten, müssen Sie eine CloudFormation Vorlage verwenden, bei der es sich um einen Plan für die Erstellung eines Stacks handelt, der aus AWS Ressourcen wie Amazon EC2 EC2-Instances und Amazon SQS SQS-Warteschlangen besteht.

Der Stack für dieses Tutorial enthält die folgenden Ressourcen:

- Eine VPC und die zugehörigen Netzwerkressourcen, einschließlich eines Subnetzes, einer Sicherheitsgruppe, eines Internet-Gateways und einer Routing-Tabelle.
- Eine im VPC-Subnetz gestartete Amazon-EC2-Instance
- Eine Amazon-SQS-Warteschlange

1. Laden Sie die Vorlage mit dem CloudFormation Namen von herunter. [SQS-VPCE-Tutorial-CloudFormation.yaml](#) GitHub
2. Melden Sie sich bei der [CloudFormation -Konsole](#) an.
3. Wählen Sie Stack erstellen aus.
4. Wählen Sie auf der Seite Select Template (Vorlage auswählen) die Option Upload a template to Amazon S3 (Eine Vorlage zu Amazon S3 hochladen). Wählen Sie dann die Datei SQS-VPCE-SQS-Tutorial-CloudFormation.yaml aus und klicken Sie auf Next (Weiter).
5. Führen Sie auf der Seite Specify DB Details (Festlegen von DB-Detail) die folgenden Schritte aus:
 - a. Geben Sie unter Stack name (Stack-Name) SQS-VPCE-Tutorial-Stack ein.
 - b. Wählen Sie für KeyNameSQS-VPCE-Tutorial-Key-Pair.
 - c. Wählen Sie Weiter aus.
6. Wählen Sie auf der Seite Optionen Weiter aus.
7. Wählen Sie auf der Seite Überprüfen im Abschnitt Funktionen die Option Ich bestätige, dass möglicherweise IAM-Ressourcen mit benutzerdefinierten Namen erstellt werden. AWS CloudFormation , und wählen Sie dann Erstellen aus.

CloudFormation beginnt mit der Erstellung des Stacks und zeigt den Status CREATE_IN_PROGRESS an. Wenn der Prozess abgeschlossen ist, zeigt CloudFormation den Status CREATE_COMPLETE an.

Schritt 3: Bestätigen, dass Ihre EC2-Instance nicht öffentlich zugänglich ist

Ihre CloudFormation Vorlage startet eine EC2-Instance, die SQS-VPCE-Tutorial-EC2-Instance in Ihre VPC benannt ist. Diese EC2-Instance lässt keinen ausgehenden Datenverkehr zu und kann keine Nachrichten an Amazon SQS senden. Um dies zu überprüfen, müssen Sie eine Verbindung mit der Instance herstellen. Versuchen Sie dann, eine Verbindung mit einem öffentlichen Endpunkt herzustellen und eine Nachricht an Amazon SQS zu senden.

1. Melden Sie sich bei der [Amazon-EC2-Konsole](#) an.
2. Wählen Sie im Navigationsmenü unter Instances die Option Instances.
3. Wählen Sie SQS-VPCE - aus. Tutorial-EC2Instance
4. Kopieren Sie den Hostnamen unter Public DNS, z. B. `ec2-203-0-113-0.us-west-2.compute.amazonaws.com`.
5. Stellen Sie über das Verzeichnis, in dem sich [das Schlüsselpaar befindet, das Sie zuvor erstellt haben](#), mit dem folgenden Befehl eine Verbindung mit der Instance her, zum Beispiel:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. Versuchen Sie, eine Verbindung mit einem öffentlichen Endpunkt herzustellen, zum Beispiel:

```
ping amazon.com
```

Der Verbindungsversuch schlägt erwartungsgemäß fehl.

7. Melden Sie sich bei der [Amazon-SQS-Konsole](#) an.
8. Wählen Sie aus der Liste der Warteschlangen die Warteschlange aus, die mit Ihrer Vorlage erstellt wurde, z. B. VPCE-SQS-Tutorial-Stack- -1 IJK. CloudFormation CFQueue ABCDEFGH2
9. Kopieren Sie in der Tabelle Details die URL, z. B. `https://sqs.us-east-2.amazonaws.com/123456789012/`
10. Versuchen Sie, über Ihre EC2-Instance mit dem folgenden Befehl eine Nachricht an die Warteschlange zu veröffentlichen, zum Beispiel:

```
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."
```

Der Sendeversuch schlägt erwartungsgemäß fehl.

 Important

Später, wenn Sie einen VPC-Endpunkt für Amazon SQS erstellen, wird Ihr Sendeversuch erfolgreich sein.

Schritt 4: Erstellen eines Amazon-VPC-Endpunkts für Amazon SQS

Zum Herstellen einer Verbindung Ihrer VPC mit Amazon SQS müssen Sie einen Schnittstellen-VPC-Endpunkt definieren. Nachdem Sie den Endpunkt hinzugefügt haben, können Sie die Amazon-SQS-API von der EC2-Instance in Ihrer VPC verwenden. Auf diese Weise können Sie Nachrichten an eine Warteschlange innerhalb des AWS Netzwerks senden, ohne das öffentliche Internet zu überqueren.

 Note

Die EC2-Instance hat immer noch keinen Zugriff auf andere AWS Dienste und Endpunkte im Internet.

1. Melden Sie sich bei der [Amazon VPC-Konsole](#) an.
2. Wählen Sie im Navigationsmenü Endpoints (Endpunkte).
3. Klicken Sie auf Endpunkt erstellen.
4. Wählen Sie auf der Seite Endpunkt erstellen für Servicename den Servicenamen für Amazon SQS aus.

 Note

Die Dienstnamen variieren je nach aktueller AWS Region. Wenn Sie sich beispielsweise in USA Ost (Ohio) befinden, lautet der Servicename `com.amazonaws.us-east-2.sqs`.

5. Wählen Sie für VPC die Option SQS-VPCE-Tutorial-VPC.
6. Wählen Sie für Subnets (Subnetze) das Subnetz aus, dessen Subnet ID (Subnetz-ID) den Wert SQS-VPCE-Tutorial-Subnet enthält.

7. Wählen Sie für Security group (Sicherheitsgruppe) die Option Select security groups (Sicherheitsgruppen auswählen). Wählen Sie dann die Sicherheitsgruppe aus, deren Group Name (Gruppenname) den Wert SQS VPCE Tutorial Security Group enthält.
8. Wählen Sie Endpunkt erstellen aus.

Der Schnittstellen-VPCE-Endpunkt wird erstellt und seine ID wird angezeigt, z. B. `vpce-0ab1cdef2ghi3j456k`.

9. Klicken Sie auf Schließen.

Die Amazon-VPC-Konsole öffnet die Seite Endpunkte.

Amazon VPC beginnt mit der Erstellung des Endpunkts und zeigt als Status ausstehend an. Wenn der Prozess abgeschlossen ist, zeigt Amazon VPC als Status verfügbar an.

Schritt 5: Senden einer Nachricht an Ihre Amazon-SQS-Warteschlange

Da Ihre VPC jetzt einen Endpunkt für Amazon SQS erhält, können Sie sich mit Ihrer EC2-Instance verbinden und Nachrichten zu Ihrer Warteschlange senden.

1. Verbinden Sie sich erneut mit Ihrer EC2-Instance, z. B.:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

2. Versuchen Sie erneut, mit dem folgenden Befehl eine Nachricht in der Warteschlange zu veröffentlichen, zum Beispiel:

```
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."
```

Der Sendeversuch ist erfolgreich und die MD5 Zusammenfassung des Nachrichtentexts und der Nachrichten-ID werden angezeigt, zum Beispiel:

```
{
  "MD5ofMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
  "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

Hinweise zum Empfangen und Löschen der Nachricht aus der Warteschlange, die mit Ihrer CloudFormation Vorlage erstellt wurde (z. B. VPCE-SQS-Tutorial-Stack- -1 IJK), finden Sie unter. CFQueue ABCDEFGH2 [Empfangen und Löschen einer Nachricht in Amazon SQS](#)

Weitere Informationen zum Löschen Ihrer Ressourcen finden Sie unter:

- [Löschen eines VPC-Endpunkts](#) im Amazon-VPC-Benutzerhandbuch
- [Löschen einer Amazon SQS SQS-Warteschlange](#)
- [Beenden Sie Ihre Instance](#) im Amazon EC2 EC2-Benutzerhandbuch
- [Löschen Ihrer VPC](#) im Amazon-VPC-Benutzerhandbuch
- [Löschen eines Stacks auf der CloudFormation Konsole](#) im AWS CloudFormation Benutzerhandbuch
- [Löschen Ihres Schlüsselpaars](#) im Amazon EC2 EC2-Benutzerhandbuch

Codebeispiele für Amazon SQS mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie Amazon SQS mit einem AWS Software Development Kit (SDK) verwendet wird.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Codebeispiele

- [Grundlegende Beispiele für die Verwendung von Amazon SQS AWS SDKs](#)
 - [Hello Amazon SQS](#)
 - [Aktionen für Amazon SQS mit AWS SDKs](#)
 - [Verwendung von AddPermission mit einer CLI](#)
 - [Verwendung ChangeMessageVisibility mit einem AWS SDK oder CLI](#)
 - [Verwendung von ChangeMessageVisibilityBatch mit einer CLI](#)
 - [Verwendung CreateQueue mit einem AWS SDK oder CLI](#)
 - [Verwendung DeleteMessage mit einem AWS SDK oder CLI](#)
 - [Verwendung DeleteMessageBatch mit einem AWS SDK oder CLI](#)
 - [Verwendung DeleteQueue mit einem AWS SDK oder CLI](#)
 - [Verwendung GetQueueAttributes mit einem AWS SDK oder CLI](#)
 - [Verwendung GetQueueUrl mit einem AWS SDK oder CLI](#)
 - [Verwendung von ListDeadLetterSourceQueues mit einer CLI](#)
 - [Verwendung ListQueues mit einem AWS SDK oder CLI](#)
 - [Verwendung von PurgeQueue mit einer CLI](#)
 - [Verwendung ReceiveMessage mit einem AWS SDK oder CLI](#)
 - [Verwendung von RemovePermission mit einer CLI](#)

- [Verwendung SendMessage mit einem AWS SDK oder CLI](#)
- [Verwendung SendMessageBatch mit einem AWS SDK oder CLI](#)
- [Verwendung SetQueueAttributes mit einem AWS SDK oder CLI](#)
- [Szenarien für die Verwendung von Amazon SQS AWS SDKs](#)
 - [Entwickeln einer Webanwendung, die Nachrichten mithilfe von Amazon SQS sendet und abrufen](#)
 - [Erstellen einer Messenger-Anwendung mit Step Functions](#)
 - [Erstellen Sie eine Amazon-Text-Explorer-Anwendung](#)
 - [Mit einem SDK erstellen und in einem FIFO-Amazon-SNS-Thema veröffentlichen AWS](#)
 - [Erkennen Sie Personen und Objekte in einem Video mit Amazon Rekognition mithilfe eines SDK AWS](#)
 - [Verwalten Sie große Amazon SQS SQS-Nachrichten mithilfe von Amazon S3 mit einem SDK AWS](#)
 - [Empfangen und Verarbeiten Amazon S3 S3-Ereignisbenachrichtigungen mithilfe eines AWS SDK](#)
 - [Veröffentlichen Sie Amazon SNS SNS-Nachrichten mithilfe eines SDK in Amazon SQS SQS-Warteschlangen AWS](#)
 - [Senden und Empfangen von Nachricht stapeln mit Amazon SQS mithilfe eines SDK AWS](#)
 - [Verwenden Sie das AWS Message Processing Framework for .NET, um Amazon SQS SQS-Nachrichten zu veröffentlichen und zu empfangen](#)
 - [Verwenden der Amazon SQS Java Messaging Library, um mit der Java-Message-Service\(JMS\)-Schnittstelle zu arbeiten](#)
 - [Arbeiten Sie mit Warteschlangen-Tags und Amazon SQS mithilfe eines SDK AWS](#)
- [Serverless-Beispiele für Amazon SQS](#)
 - [Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser](#)
 - [Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser](#)

Grundlegende Beispiele für die Verwendung von Amazon SQS AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie die Grundlagen von Amazon Simple Queue Service mit verwenden können AWS SDKs.

Beispiele

- [Hello Amazon SQS](#)
- [Aktionen für Amazon SQS mit AWS SDKs](#)
 - [Verwendung von AddPermission mit einer CLI](#)
 - [Verwendung ChangeMessageVisibility mit einem AWS SDK oder CLI](#)
 - [Verwendung von ChangeMessageVisibilityBatch mit einer CLI](#)
 - [Verwendung CreateQueue mit einem AWS SDK oder CLI](#)
 - [Verwendung DeleteMessage mit einem AWS SDK oder CLI](#)
 - [Verwendung DeleteMessageBatch mit einem AWS SDK oder CLI](#)
 - [Verwendung DeleteQueue mit einem AWS SDK oder CLI](#)
 - [Verwendung GetQueueAttributes mit einem AWS SDK oder CLI](#)
 - [Verwendung GetQueueUrl mit einem AWS SDK oder CLI](#)
 - [Verwendung von ListDeadLetterSourceQueues mit einer CLI](#)
 - [Verwendung ListQueues mit einem AWS SDK oder CLI](#)
 - [Verwendung von PurgeQueue mit einer CLI](#)
 - [Verwendung ReceiveMessage mit einem AWS SDK oder CLI](#)
 - [Verwendung von RemovePermission mit einer CLI](#)
 - [Verwendung SendMessage mit einem AWS SDK oder CLI](#)
 - [Verwendung SendMessageBatch mit einem AWS SDK oder CLI](#)
 - [Verwendung SetQueueAttributes mit einem AWS SDK oder CLI](#)

Hello Amazon SQS

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung von Amazon SQS beginnen.

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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();
        }
    }
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Code für die CMake Datei CMake Lists.txt.

```
# 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})

```

Code für die Quelldatei „hello_sqs.cpp“.

```

#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;
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) unter AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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\t%v\n", queueUrl)
    }
}
}

```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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);
        }
    }
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Initialisieren Sie einen Amazon-SQS-Client und listen Sie Warteschlangen auf.

```
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"));
};
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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")
            }
    }
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der API-Referenz zum AWS SDK für Kotlin.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Die Datei Package.swift

```
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/aws-labs/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")

    ]
}
```

Der Swift-Quellcode, `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)
        }
    }
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktionen für Amazon SQS mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie einzelne Amazon SQS SQS-Aktionen mit AWS SDKs ausführen. Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes finden.

Diese Auszüge rufen die Amazon-SQS-API auf und sind Codeauszüge aus größeren Programmen, die im Kontext ausgeführt werden müssen. Sie können Aktionen im Kontext unter [Szenarien für die Verwendung von Amazon SQS AWS SDKs](#) anzeigen.

Die folgenden Beispiele enthalten nur die am häufigsten verwendeten Aktionen. Eine vollständige Liste finden Sie in der [API-Referenz für Amazon Simple Email Service](#).

Beispiele

- [Verwendung von AddPermission mit einer CLI](#)
- [Verwendung ChangeMessageVisibility mit einem AWS SDK oder CLI](#)
- [Verwendung von ChangeMessageVisibilityBatch mit einer CLI](#)
- [Verwendung CreateQueue mit einem AWS SDK oder CLI](#)
- [Verwendung DeleteMessage mit einem AWS SDK oder CLI](#)
- [Verwendung DeleteMessageBatch mit einem AWS SDK oder CLI](#)
- [Verwendung DeleteQueue mit einem AWS SDK oder CLI](#)
- [Verwendung GetQueueAttributes mit einem AWS SDK oder CLI](#)
- [Verwendung GetQueueUrl mit einem AWS SDK oder CLI](#)
- [Verwendung von ListDeadLetterSourceQueues mit einer CLI](#)
- [Verwendung ListQueues mit einem AWS SDK oder CLI](#)
- [Verwendung von PurgeQueue mit einer CLI](#)
- [Verwendung ReceiveMessage mit einem AWS SDK oder CLI](#)
- [Verwendung von RemovePermission mit einer CLI](#)
- [Verwendung SendMessage mit einem AWS SDK oder CLI](#)
- [Verwendung SendMessageBatch mit einem AWS SDK oder CLI](#)
- [Verwendung SetQueueAttributes mit einem AWS SDK oder CLI](#)

Verwendung von **AddPermission** mit einer CLI

Die folgenden Code-Beispiele zeigen, wie `AddPermission` verwendet wird.

CLI

AWS CLI

So fügen Sie einer Warteschlange ein Thema hinzu

In diesem Beispiel kann das angegebene AWS Konto Nachrichten an die angegebene Warteschlange senden.

Befehl:

```
aws sqs add-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessageFromMyQueue --aws-account-ids 12345EXAMPLE --actions SendMessage
```

Ausgabe:

```
None.
```

- Einzelheiten zur API finden Sie [AddPermission](#) in der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel kann der angegebene AWS-Konto Benutzer Nachrichten aus der angegebenen Warteschlange senden.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE  
-Label SendMessageFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [AddPermission AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel können die angegebenen Benutzer AWS-Konto Nachrichten aus der angegebenen Warteschlange senden.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE  
-Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-  
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [AddPermission AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **ChangeMessageVisibility** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ChangeMessageVisibility` verwendet wird.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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();
}

```

- Einzelheiten zur API finden Sie [ChangeMessageVisibility](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So ändern Sie die Timeout-Sichtbarkeit einer Nachricht

In diesem Beispiel wird die Timeout-Sichtbarkeit der angegebenen Nachricht auf 10 Stunden (10 Stunden x 60 Minuten x 60 Sekunden) geändert.

Befehl:

```
aws sqs change-message-visibility --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --visibility-timeout 36000
```

Ausgabe:

None.

- Einzelheiten zur API finden Sie [ChangeMessageVisibility](#) in der AWS CLI Befehlsreferenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie eine Amazon-SQS-Nachricht und ändern Sie deren Timeout-Sichtbarkeit.

```
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;
};
```

- Einzelheiten zur API finden Sie [ChangeMessageVisibility](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie eine Amazon-SQS-Nachricht und ändern Sie deren Timeout-Sichtbarkeit.

```
// 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");
    }
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ChangeMessageVisibility](#) in der AWS SDK für JavaScript API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel wird das Sichtbarkeits-Timeout für die Nachricht mit dem angegebenen Empfangs-Handle in der angegebenen Warteschlange auf 10 Stunden geändert (10 Stunden * 60 Minuten * 60 Sekunden = 36 000 Sekunden).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -VisibilityTimeout 36000
```

- Einzelheiten zur API finden Sie unter [ChangeMessageVisibility AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird das Sichtbarkeits-Timeout für die Nachricht mit dem angegebenen Empfangs-Handle in der angegebenen Warteschlange auf 10 Stunden geändert (10 Stunden * 60 Minuten * 60 Sekunden = 36 000 Sekunden).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -VisibilityTimeout 36000
```

- Einzelheiten zur API finden Sie unter [ChangeMessageVisibility AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Ruby

SDK für Ruby

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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

```

- Einzelheiten zur API finden Sie [ChangeMessageVisibility](#) in der AWS SDK für Ruby API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung von **ChangeMessageVisibilityBatch** mit einer CLI

Die folgenden Code-Beispiele zeigen, wie ChangeMessageVisibilityBatch verwendet wird.

CLI

AWS CLI

So ändern Sie die Timeout-Sichtbarkeit mehrerer Nachrichten als Batch

In diesem Beispiel wird die Timeout-Sichtbarkeit der beiden angegebenen Nachrichten auf 10 Stunden (10 Stunden x 60 Minuten x 60 Sekunden) geändert.

Befehl:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

Eingabedatei (change-message-visibility-batch.json):

```
[
  {
    "Id": "FirstMessage",
    "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
    "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
    "ReceiptHandle": "AQEBkTUH...HifSnw==",
    "VisibilityTimeout": 36000
  }
]
```

Ausgabe:

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

- Einzelheiten zur API finden Sie [ChangeMessageVisibilityBatch](#) in der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel wird das Sichtbarkeits-Timeout für zwei Nachrichten mit den angegebenen Empfangs-Handles in der angegebenen Warteschlange geändert. Das Sichtbarkeits-Timeout der ersten Nachricht wird auf 10 Stunden geändert (10 Stunden * 60 Minuten * 60 Sekunden = 36 000 Sekunden). Das Sichtbarkeits-Timeout der zweiten Nachricht wird auf 5 Stunden geändert (5 Stunden * 60 Minuten * 60 Sekunden = 18 000 Sekunden).

```
$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
```

Ausgabe:

Failed	Successful
-----	-----
{}	{Request2, Request1}

- Einzelheiten zur API finden Sie unter [ChangeMessageVisibilityBatch AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird das Sichtbarkeits-Timeout für zwei Nachrichten mit den angegebenen Empfangs-Handles in der angegebenen Warteschlange geändert. Das Sichtbarkeits-Timeout der ersten Nachricht wird auf 10 Stunden geändert (10 Stunden * 60 Minuten * 60 Sekunden = 36 000 Sekunden). Das Sichtbarkeits-Timeout der zweiten Nachricht wird auf 5 Stunden geändert (5 Stunden * 60 Minuten * 60 Sekunden = 18 000 Sekunden).

```
$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
```

Ausgabe:

Failed	Successful
-----	-----
{}	{Request2, Request1}

- Einzelheiten zur API finden Sie unter [ChangeMessageVisibilityBatch AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **CreateQueue** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CreateQueue` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Veröffentlichen Sie Nachrichten in Warteschlangen](#)
- [Senden und Empfangen von Stapeln von Nachrichten](#)
- [Verwenden der Amazon SQS Java Messaging Library, um mit der JMS-Schnittstelle zu arbeiten](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Eine Warteschlange mit einem bestimmten Namen erstellen

```
/// <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;
}
```

Erstellen Sie eine Amazon-SQS-Warteschlange und senden Sie eine Nachricht an diese.

```
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;
}

```

```
}
```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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();
}

```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So erstellen Sie eine Warteschlange

In diesem Beispiel wird eine Warteschlange mit dem angegebenen Namen erstellt, die Aufbewahrungsdauer für Nachrichten auf 3 Tage (3 Tage * 24 Stunden * 60 Minuten * 60 Sekunden) festgelegt, und die Warteschlange für unzustellbare Nachrichten der Warteschlange wird auf die angegebene Warteschlange mit einer maximalen Empfangszahl von 1 000 Nachrichten festgelegt.

Befehl:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

Eingabedatei (create-queue.json):

```

{
  "RedrivePolicy": "{\n\"deadLetterTargetArn\": \"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\", \"maxReceiveCount\": \"1000\"}",
  "MessageRetentionPeriod": "259200"
}

```

Ausgabe:

```

{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}

```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
}

```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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)
```

```
        .numberOfMessages(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);
}
}
}

```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie eine Standard-Warteschlange von Amazon SQS.

```

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;
};

```


Erstellen einer Amazon-SQS-Warteschlange mit Langabfrage

```
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;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie eine Standard-Warteschlange von Amazon SQS.

```
// 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);
  }
});
```

Erstellen Sie eine Amazon-SQS-Warteschlange, die auf eine Nachricht wartet.

```
// 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);
}
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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()
    }
}
```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der API-Referenz zum AWS SDK für Kotlin.

PowerShell

Tools für V4 PowerShell

Beispiel 1: In diesem Beispiel wird eine Warteschlange mit dem angegebenen Namen erstellt.

```
New-SQSQueue -QueueName MyQueue
```

Ausgabe:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [CreateQueue AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird eine Warteschlange mit dem angegebenen Namen erstellt.

```
New-SQSQueue -QueueName MyQueue
```

Ausgabe:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [CreateQueue AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
```

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.
```

```

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)

def create_queue(self, queue_name: str, is_fifo: bool = False) -> str:
    """
    Create an SQS queue.

    :param queue_name: The name of the queue to create.
    :param is_fifo: Whether to create a FIFO queue.
    :return: The URL of the created queue.
    :raises ClientError: If the queue creation fails.
    """
    try:
        # Add .fifo suffix for FIFO queues
        if is_fifo and not queue_name.endswith('.fifo'):
            queue_name += '.fifo'

        attributes = {}
        if is_fifo:
            attributes['FifoQueue'] = 'true'

        response = self.sqs_client.create_queue(
            QueueName=queue_name,
            Attributes=attributes
        )

        queue_url = response['QueueUrl']
        logger.info(f"Created queue: {queue_name} with URL: {queue_url}")
        return queue_url

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error creating queue {queue_name}: {error_code} - {e}")
        raise

```

- Einzelheiten zur API finden Sie [CreateQueue](#) in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
# 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__
```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie eine Standard-Warteschlange von Amazon SQS.

```
TRY.
    oo_result = lo_sqs->createqueue( iv_queue_name = iv_queue_name ).
    oo_result is returned for testing purposes. "
    MESSAGE 'SQS queue created.' TYPE 'I'.
    CATCH /aws1/cx_sqsqueuedeletedrecently.
    MESSAGE 'After deleting a queue, wait 60 seconds before creating another
queue with the same name.' TYPE 'E'.
    CATCH /aws1/cx_sqsqueueexists.
    MESSAGE 'A queue with this name already exists.' TYPE 'E'.
ENDTRY.
```

Erstellen Sie eine Amazon-SQS-Warteschlange, die auf eine Nachricht wartet.


```

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_queue_name = iv_queue_name
        it_attributes = lt_attributes ).
    MESSAGE 'SQS queue created.' TYPE 'I'.
    CATCH /aws1/cx_sqsqueuedeletedrecently.
    MESSAGE 'After deleting a queue, wait 60 seconds before creating another
queue with the same name.' TYPE 'E'.
    CATCH /aws1/cx_sqsqueue_name_exists.
    MESSAGE 'A queue with this name already exists.' TYPE 'E'.
ENDTRY.

```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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  
}
```

- Einzelheiten zur API finden Sie [CreateQueue](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **DeleteMessage** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteMessage` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- [Senden und Empfangen von Stapeln von Nachrichten](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie eine Nachricht aus einer Amazon-SQS-Warteschlange und löschen Sie sie anschließend.

```

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;
    }
}

```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

    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();  
}
```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Um eine Nachricht zu löschen

In diesem Beispiel wird die angegebene Nachricht gelöscht.

Befehl:

```
aws sqs delete-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

Ausgabe:

```
None.
```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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);
}
```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie Amazon-SQS-Nachrichten ab und löschen Sie sie.

```
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,
                })),
            }),
        );
    }
};
```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie Amazon-SQS-Nachrichten ab und löschen Sie sie.

```
// 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);
      }
    });
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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!")
    }
}
```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der API-Referenz zum AWS SDK für Kotlin.

PowerShell

Tools für V4 PowerShell

Beispiel 1: In diesem Beispiel wird die Nachricht mit dem angegebenen Empfangs-Handle aus der angegebenen Warteschlange gelöscht.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Einzelheiten zur API finden Sie unter [DeleteMessage AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird die Nachricht mit dem angegebenen Empfangs-Handle aus der angegebenen Warteschlange gelöscht.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Einzelheiten zur API finden Sie unter [DeleteMessage AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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

```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in AWS SDK for Python (Boto3) API Reference.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

TRY.
    lo_sqs->deletemessage(
        iv_queueurl = iv_queue_url
        iv_receipthandle = iv_receipt_handle ).
    MESSAGE 'Message deleted from SQS queue.' TYPE 'I'.
CATCH /aws1/cx_sqsinvalididformat.
    MESSAGE 'The specified receipt handle is not valid.' TYPE 'E'.
CATCH /aws1/cx_sqsreceipthandleisinv.
    MESSAGE 'The specified receipt handle is not valid for the current
version.' TYPE 'E'.
ENDTRY.

```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **DeleteMessageBatch** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteMessageBatch` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Process-S3-Ereignisbenachrichtigungen](#)
- [Veröffentlichen Sie Nachrichten in Warteschlangen](#)
- [Senden und Empfangen von Stapeln von Nachrichten](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
/// <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();
}

```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::SQS::SQSClient sqsClient(clientConfig);

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;
    }
}

```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So löschen Sie mehrere Nachrichten als Batch

In diesem Beispiel werden die angegebenen Nachrichten gelöscht.

Befehl:

```
aws sqs delete-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-batch.json
```

Eingabedatei (delete-message-batch.json):

```
[
  {
    "Id": "FirstMessage",
    "ReceiptHandle": "AQEB1mg1...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
    "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

Ausgabe:

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
}

```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der AWS SDK für Go API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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,
            })),
        }),
    );
}
```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der AWS SDK für JavaScript API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel werden zwei Nachrichten mit den angegebenen EmpfangsHandles aus der angegebenen Warteschlange gelöscht.

```
$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 = "AQEBq0VY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
    $deleteMessageRequest2
```

Ausgabe:

Failed	Successful
-----	-----
{}	{Request1, Request2}

- Einzelheiten zur API finden Sie unter [DeleteMessageBatch AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden zwei Nachrichten mit den angegebenen Empfangs-Handles aus der angegebenen Warteschlange gelöscht.

```
$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 = "AQEBq0VY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
    $deleteMessageRequest2
```

Ausgabe:

Failed	Successful
-----	-----
{}	{Request1, Request2}

- Einzelheiten zur API finden Sie unter [DeleteMessageBatch AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. [GitHub](#) Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
```

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)

    def delete_messages(self, queue_url: str, messages: List[Dict[str, Any]]) ->
bool:
    """
    Delete messages from an SQS queue in batches.

    :param queue_url: The URL of the queue.
    :param messages: List of messages to delete.
    :return: True if successful.
    :raises ClientError: If deleting messages fails.
    """
    try:
        if not messages:
            return True

        # Build delete entries for batch delete
        delete_entries = []
        for i, message in enumerate(messages):
            delete_entries.append({
                'Id': str(i),
```

```

        'ReceiptHandle': message['ReceiptHandle']
    })

    # Delete messages in batches of 10 (SQS limit)
    batch_size = 10
    for i in range(0, len(delete_entries), batch_size):
        batch = delete_entries[i:i + batch_size]

        response = self.sqs_client.delete_message_batch(
            QueueUrl=queue_url,
            Entries=batch
        )

        # Check for failures
        if 'Failed' in response and response['Failed']:
            for failed in response['Failed']:
                logger.warning(f"Failed to delete message: {failed}")

    logger.info(f"Deleted {len(messages)} messages from {queue_url}")
    return True

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error deleting messages: {error_code} - {e}")
    raise

```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in AWS SDK for Python (Boto3) API Reference.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

TRY.
    oo_result = lo_sqs->deletemessagebatch(      " oo_result is returned for
testing purposes. "
        iv_queueurl = iv_queue_url
        it_entries = it_entries ).
    MESSAGE 'Messages deleted from SQS queue.' TYPE 'I'.
CATCH /aws1/cx_sqsbtcentidsnotdist00.
    MESSAGE 'Two or more batch entries in the request have the same ID.' TYPE
'E'.
CATCH /aws1/cx_sqsemtypbatchrequest.
    MESSAGE 'The batch request does not contain any entries.' TYPE 'E'.
CATCH /aws1/cx_sqsinvbatchentryid.
    MESSAGE 'The ID of a batch entry in a batch request is not valid.' TYPE
'E'.
CATCH /aws1/cx_sqstoomanyentriesin00.
    MESSAGE 'The batch request contains more entries than allowed.' TYPE 'E'.
ENDTRY.

```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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.")
}
```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **DeleteQueue** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteQueue` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Veröffentlichen Sie Nachrichten in Warteschlangen](#)
- [Senden und Empfangen von Stapeln von Nachrichten](#)
- [Verwenden der Amazon SQS Java Messaging Library, um mit der JMS-Schnittstelle zu arbeiten](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Eine Warteschlange unter Verwendung ihrer URL löschen

```
/// <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;
}
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

        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();
    }
}

```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So löschen Sie eine Warteschlange

In diesem Beispiel wird die angegebene Warteschlange gelöscht.

Befehl:

```
aws sqs delete-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

Ausgabe:

```
None.
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
}
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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);
    }
}
}
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie eine Amazon-SQS-Warteschlange.

```
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;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie eine Amazon-SQS-Warteschlange.

```
// 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);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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!")
    }
}
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der API-Referenz zum AWS SDK für Kotlin.

PowerShell

Tools für V4 PowerShell

Beispiel 1: In diesem Beispiel wird die angegebene Warteschlange gelöscht.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Einzelheiten zur API finden Sie unter [DeleteQueue AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird die angegebene Warteschlange gelöscht.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Einzelheiten zur API finden Sie unter [DeleteQueue AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
```

```
class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
```

```
Create an SqsWrapper instance using a default boto3 client.

:return: An instance of this class.
"""
sqs_client = boto3.client('sqs')
return cls(sqs_client)

def delete_queue(self, queue_url: str) -> bool:
    """
    Delete an SQS queue.

    :param queue_url: The URL of the queue to delete.
    :return: True if successful.
    :raises ClientError: If the queue deletion fails.
    """
    try:
        self.sqs_client.delete_queue(QueueUrl=queue_url)

        logger.info(f"Deleted queue: {queue_url}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')

        if error_code == 'AWS.SimpleQueueService.NonExistentQueue':
            logger.warning(f"Queue not found: {queue_url}")
            return True # Already deleted
        else:
            logger.error(f"Error deleting queue: {error_code} - {e}")
            raise
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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)
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
TRY.
    lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
    MESSAGE 'SQS queue deleted' TYPE 'I'.
ENDTRY.
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
}
```

- Einzelheiten zur API finden Sie [DeleteQueue](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **GetQueueAttributes** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetQueueAttributes` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Process-S3-Ereignisbenachrichtigungen](#)

- [Veröffentlichen Sie Nachrichten in Warteschlangen](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
/// <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;
}
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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;
```



```
}
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So rufen Sie die Attribute einer Warteschlange ab

In diesem Beispiel werden alle Attribute der angegebenen Warteschlange abgerufen.

Befehl:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

Ausgabe:

```
{
  "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"
  }
}
```

In diesem Beispiel werden ausschließlich die maximalen Timeout-Attribute für Nachrichtengrößen und Sichtbarkeit der angegebenen Warteschlange abgerufen.

Befehl:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-names MaximumMessageSize VisibilityTimeout
```

Ausgabe:

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
}

```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der AWS SDK für Go API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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;
};
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der AWS SDK für JavaScript API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel werden Details zu den Attributen für die angegebene Warteschlange aufgelistet.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Ausgabe:

```
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":"2012-10-17",
  "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":

      "SendMessageFromMyQueue","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]...}

```

Beispiel 2: In diesem Beispiel werden nur die angegebenen Attribute für die angegebene Warteschlange separat aufgelistet.

```

Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

```

Ausgabe:

```

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":"2012-10-17",
  "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":

      "SendMessageFromMyQueue","Effect":"Allow","Principal":
      {"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
      arn:aws:sqs:us-
      east-1:80398EXAMPLE:MyQueue"}]}
Attributes                           : [[MaximumMessageSize, 262144],
  [VisibilityTimeout, 30]]

```

- Einzelheiten zur API finden Sie unter [GetQueueAttributes AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden Details zu den Attributen für die angegebene Warteschlange aufgelistet.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Ausgabe:

```

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":"2012-10-17",
  "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":

  "SendMessageFromMyQueue","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]...}

```

Beispiel 2: In diesem Beispiel werden nur die angegebenen Attribute für die angegebene Warteschlange separat aufgelistet.

```

Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

```

Ausgabe:

```

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":"2012-10-17",
      "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":

      "SendMessageFromMyQueue", "Effect":"Allow", "Principal":
    {"AWS":"80398EXAMPLE"}, "Action":"SQS:SendMessage", "Resource":
      "arn:aws:sqs:us-
    east-1:80398EXAMPLE:MyQueue"}]]}
Attributes
    : {[MaximumMessageSize, 262144],
      [VisibilityTimeout, 30]}

```

- Einzelheiten zur API finden Sie unter [GetQueueAttributes AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

```



```
@classmethod
def from_client(cls) -> 'SqsWrapper':
    """
    Create an SqsWrapper instance using a default boto3 client.

    :return: An instance of this class.
    """
    sqs_client = boto3.client('sqs')
    return cls(sqs_client)

def get_queue_arn(self, queue_url: str) -> str:
    """
    Get the ARN of an SQS queue.

    :param queue_url: The URL of the queue.
    :return: The ARN of the queue.
    :raises ClientError: If getting queue attributes fails.
    """
    try:
        response = self.sqs_client.get_queue_attributes(
            QueueUrl=queue_url,
            AttributeNames=['QueueArn']
        )

        queue_arn = response['Attributes']['QueueArn']
        logger.info(f"Queue ARN for {queue_url}: {queue_arn}")
        return queue_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error getting queue ARN: {error_code} - {e}")
        raise
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in AWS SDK for Python (Boto3) API Reference.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
    }
}
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **GetQueueUrl** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetQueueUrl` verwendet wird.

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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.");
}
}
```

- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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();
}

```

- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So rufen Sie eine Warteschlangen-URL ab

In diesem Beispiel wird die URL der angegebenen Warteschlange abgerufen.

Befehl:

```
aws sqs get-queue-url --queue-name MyQueue
```

Ausgabe:

```
{
```

```
"QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
GetQueueUrlResponse getQueueUrlResponse = sqsClient
    .getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
    return getQueueUrlResponse.queueUrl();
```

- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die URL für eine Amazon-SQS-Warteschlange ab.

```
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;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die URL für eine Amazon-SQS-Warteschlange ab.

```
// 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);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der AWS SDK für JavaScript API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel wird die URL der Warteschlange mit dem angegebenen Namen aufgelistet.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Ausgabe:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [GetQueueUrl AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird die URL der Warteschlange mit dem angegebenen Namen aufgelistet.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Ausgabe:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [GetQueueUrl AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
```

- Einzelheiten zur API finden Sie [GetQueueUrl](#) in AWS SDK for Python (Boto3) API Reference.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

TRY.
    oo_result = lo_sqs->getqueueurl( iv_queue_name = iv_queue_name ).
    oo_result is returned for testing purposes. "
    MESSAGE 'Queue URL retrieved.' TYPE 'I'.
CATCH /aws1/cx_sqsqueue_does_not_exist.
    MESSAGE 'The requested queue does not exist.' TYPE 'E'.
ENDTRY.

```

- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung von **ListDeadLetterSourceQueues** mit einer CLI

Die folgenden Code-Beispiele zeigen, wie `ListDeadLetterSourceQueues` verwendet wird.

CLI

AWS CLI

So listen Sie Quellwarteschlangen für unzustellbare Nachrichten auf

In diesem Beispiel werden die Warteschlangen aufgeführt, die der angegebenen Quellwarteschlange für unzustellbare Nachrichten zugeordnet sind.

Befehl:

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Ausgabe:

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

```
]
}
```

- Einzelheiten zur API finden Sie [ListDeadLetterSourceQueues](#) in der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel werden alle Warteschlangen aufgeführt, die URLs sich auf die angegebene Warteschlange als Warteschlange für unzustellbare Nachrichten verlassen.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Ausgabe:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Einzelheiten zur API finden Sie unter [ListDeadLetterSourceQueues AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden alle Warteschlangen aufgeführt, die URLs sich auf die angegebene Warteschlange als Warteschlange für unzustellbare Nachrichten verlassen.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Ausgabe:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Einzelheiten zur API finden Sie unter [ListDeadLetterSourceQueues AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **ListQueues** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie ListQueues verwendet wird.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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;
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So listen Sie Warteschlangen auf

In diesem Beispiel werden alle Warteschlangen aufgelistet.

Befehl:

```
aws sqs list-queues
```

Ausgabe:

```
{
  "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"
  ]
}
```

In diesem Beispiel werden nur Warteschlangen aufgelistet, die mit „My“ beginnen.

Befehl:

```
aws sqs list-queues --queue-name-prefix My
```

Ausgabe:

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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)
        }
    }
}
```

```
}  
}  
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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);  
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie Ihre Amazon-SQS-Warteschlangen auf.

```
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;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie Ihre Amazon-SQS-Warteschlangen auf.

```
// 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);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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)
        }
    }
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der API-Referenz zum AWS SDK für Kotlin.

PowerShell

Tools für V4 PowerShell

Beispiel 1: In diesem Beispiel werden alle Parameter aufgelistet.

```
Get-SQSQueue
```

Ausgabe:

```
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
```

Beispiel 2: In diesem Beispiel werden alle Warteschlangen aufgelistet, die mit dem angegebenen Namen beginnen.

```
Get-SQSQueue -QueueNamePrefix My
```

Ausgabe:

```
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
```

- Einzelheiten zur API finden Sie unter [ListQueues AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden alle Parameter aufgelistet.

```
Get-SQSQueue
```

Ausgabe:

```
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
```

Beispiel 2: In diesem Beispiel werden alle Warteschlangen aufgelistet, die mit dem angegebenen Namen beginnen.

```
Get-SQSQueue -QueueNamePrefix My
```

Ausgabe:

```
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
```

- Einzelheiten zur API finden Sie unter [ListQueues AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. [GitHub](#) Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
```

- Einzelheiten zur API finden Sie [ListQueues](#) in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

Note

Es gibt noch mehr dazu. [GitHub](#) Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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']
  )
end
```

```

    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

```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die erste Amazon-SQS-Warteschlange ab, die in der Region aufgeführt ist.

```
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())
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
TRY.
    oo_result = lo_sqs->listqueues( ).          " oo_result is returned for
testing purposes. "
    MESSAGE 'Retrieved list of queues.' TYPE 'I'.
ENDTRY.
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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)
    }
}
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung von **PurgeQueue** mit einer CLI

Die folgenden Code-Beispiele zeigen, wie PurgeQueue verwendet wird.

CLI

AWS CLI

So bereinigen Sie eine Warteschlange

In diesem Beispiel werden alle Nachrichten in der angegebenen Warteschlange gelöscht.

Befehl:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue
```

Ausgabe:

```
None .
```

- Einzelheiten zur API finden Sie [PurgeQueue](#) in der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel werden alle Nachrichten aus der angegebenen Warteschlange gelöscht.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [PurgeQueue AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden alle Nachrichten aus der angegebenen Warteschlange gelöscht.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [PurgeQueue AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **ReceiveMessage** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ReceiveMessage` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Verwalten großer Nachrichten mit S3](#)
- [Process-S3-Ereignisbenachrichtigungen](#)
- [Veröffentlichen Sie Nachrichten in Warteschlangen](#)
- [Senden und Empfangen von Stapeln von Nachrichten](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Nachrichten aus einer Warteschlange unter Verwendung ihrer URL empfangen

```
/// <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;
    }

```

Empfangen Sie eine Nachricht aus einer Amazon-SQS-Warteschlange und löschen Sie sie anschließend.

```

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;
    }
}

```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

    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();
}

```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So empfangen Sie eine Nachricht

In diesem Beispiel werden bis zu 10 verfügbare Nachrichten empfangen und alle verfügbaren Attribute zurückgegeben.

Befehl:

```
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
```

Ausgabe:

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEBzbVv...fqNzFw==",
      "MD5ofBody": "1000f835...a35411fa",
      "MD5ofMessageAttributes": "9424c491...26bc3ae7",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "ApproximateFirstReceiveTimestamp": "1442428276921",
        "SenderId": "AIDAIKMSNQ7EXAMPLE",
        "ApproximateReceiveCount": "5",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        },
        "City": {
          "DataType": "String",
          "StringValue": "Any City"
        }
      }
    }
  ]
}
```



```
}
```

In diesem Beispiel wird die nächste verfügbare Nachricht empfangen und nur die `SentTimestamp` Attribute `SenderId` und sowie das `PostalCode` Nachrichtenattribut zurückgegeben.

Befehl:

```
aws sqs receive-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names SenderId SentTimestamp --message-attribute-names PostalCode
```

Ausgabe:

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEB6nR4...HzlvZQ==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "b8e89563...e088e74f",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "SenderId": "AIDAIKMSNQ7TEXAMPLE",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        }
      }
    }
  ]
}
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) unter AWS CLI Befehlsreferenz.

Go

SDK für Go V2

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
}
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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;
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie eine Nachricht von einer Amazon-SQS-Warteschlange.

```
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,
        })),
      }),
    );
  }
};
```

Empfangen Sie mithilfe der Long-Poll-Unterstützung eine Nachricht aus einer Amazon-SQS-Warteschlange.

```
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;
};
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie mithilfe der Long-Poll-Unterstützung eine Nachricht aus einer Amazon-SQS-Warteschlange.

```
// 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);  
    }  
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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)  
        }  
    }  
}
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der API-Referenz zum AWS SDK für Kotlin.

PowerShell

Tools für V4 PowerShell

Beispiel 1: In diesem Beispiel werden Informationen für die nächsten 10 Nachrichten aufgelistet, die in der angegebenen Warteschlange empfangen werden sollen. Die Informationen enthalten Werte für die angegebenen Nachrichtenattribute, sofern sie existieren.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName  
  StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-  
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Ausgabe:

```
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==
```

- Einzelheiten zur API finden Sie unter [ReceiveMessage AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden Informationen für die nächsten 10 Nachrichten aufgelistet, die in der angegebenen Warteschlange empfangen werden sollen. Die Informationen enthalten Werte für die angegebenen Nachrichtenattribute, sofern sie existieren.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName  
  StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-  
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Ausgabe:

```
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==

```

- Einzelheiten zur API finden Sie unter [ReceiveMessage AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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

```

```

class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)

    def receive_messages(self, queue_url: str, max_messages: int = 10) ->
List[Dict[str, Any]]:
        """
        Receive messages from an SQS queue.

        :param queue_url: The URL of the queue to receive messages from.
        :param max_messages: Maximum number of messages to receive (1-10).
        :return: List of received messages.

```

```

        :raises ClientError: If receiving messages fails.
        """
        try:
            # Ensure max_messages is within valid range
            max_messages = max(1, min(10, max_messages))

            response = self.sqs_client.receive_message(
                QueueUrl=queue_url,
                MaxNumberOfMessages=max_messages,
                WaitTimeSeconds=2, # Short polling
                MessageAttributeNames=['All']
            )

            messages = response.get('Messages', [])
            logger.info(f"Received {len(messages)} messages from {queue_url}")
            return messages

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error receiving messages: {error_code} - {e}")
            raise

```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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
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__

```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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(())
}
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. [GitHub](#) Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie eine Nachricht von einer Amazon-SQS-Warteschlange.

```
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.
```

Empfangen Sie mithilfe der Long-Poll-Unterstützung eine Nachricht aus einer Amazon-SQS-Warteschlange.

```
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.
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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("---")  
}
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung von **RemovePermission** mit einer CLI

Die folgenden Code-Beispiele zeigen, wie `RemovePermission` verwendet wird.

CLI

AWS CLI

So entfernen Sie eine Berechtigung

In diesem Beispiel wird die Berechtigung mit dem angegebenen Label aus der angegebenen Warteschlange entfernt.

Befehl:

```
aws sqs remove-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessageFromMyQueue
```

Ausgabe:

```
None.
```

- Einzelheiten zur API finden Sie [RemovePermission](#) in der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel werden die Berechtigungseinstellungen mit der angegebenen Bezeichnung aus der angegebenen Warteschlange entfernt.


```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [RemovePermission AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden die Berechtigungseinstellungen mit der angegebenen Bezeichnung aus der angegebenen Warteschlange entfernt.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [RemovePermission AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **SendMessage** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `SendMessage` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- [Verwalten großer Nachrichten mit S3](#)
- [Senden und Empfangen von Stapeln von Nachrichten](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie eine Amazon-SQS-Warteschlange und senden Sie eine Nachricht an diese.

```
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;
    }
}

```

- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

    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();
}

```

- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So senden Sie eine Nachricht

In diesem Beispiel wird eine Nachricht mit dem angegebenen Nachrichtentext, der angegebenen Verzögerungszeit und Nachrichtenattributen an die angegebene Warteschlange gesendet.

Befehl:

```

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
```

Eingabedatei (send-message.json):

```
{  
  "City": {  
    "DataType": "String",  
    "StringValue": "Any City"  
  },  
  "Greeting": {  
    "DataType": "Binary",  
    "BinaryValue": "Hello, World!"  
  },  
  "Population": {  
    "DataType": "Number",  
    "StringValue": "1250800"  
  }  
}
```

Ausgabe:

```
{  
  "MD5ofMessageBody": "51b0a325...39163aa0",  
  "MD5ofMessageAttributes": "00484c68...59e48f06",  
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"  
}
```

- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Es folgen zwei Beispiele für die SendMessage-Operation:

- Senden einer Nachricht mit Text und einer Verzögerung
- Senden einer Nachricht mit Text und Nachrichtenattributen

Senden einer Nachricht mit Text und einer Verzögerung.

```
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);
        }
    }
}

```

Senden einer Nachricht mit Text und Nachrichtenattributen.

```

/**
 * <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 thumbnailPath Filesystem path of the image.
    * @param queueUrl      URL of the SQS queue.
    */
    static void sendMessageWithAttributes(Path thumbnailPath, 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(thumbnailPath)))
                    .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);
    }
}
```

- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie eine Nachricht an eine Amazon-SQS-Warteschlange.

```
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",
      },
    },
  });
  await client.send(command);
}
```

```
    },  
    },  
    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;  
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS SDK für JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Senden Sie eine Nachricht an eine Amazon-SQS-Warteschlange.

```
// 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);
    }
});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS SDK für JavaScript API-Referenz.

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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.")
    }
}
```

- Einzelheiten zur API finden Sie [SendMessage](#) in der API-Referenz zum AWS SDK für Kotlin.

PowerShell

Tools für V4 PowerShell

Beispiel 1: In diesem Beispiel wird eine Nachricht mit den angegebenen Attributen und dem angegebenen Nachrichtentext an die angegebene Warteschlange gesendet, wobei die Zustellung der Nachricht um 10 Sekunden verzögert wird.

```
$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
```

Ausgabe:

MD5ofMessageAttributes	MD5ofMessageBody	MessageId
-----	-----	-----
1d3e51347bc042efbdf6dda31EXAMPLE c739-4d0c-818b-1820eEXAMPLE	51b0a3256d59467f973009b73EXAMPLE	c35fed8f-

- Einzelheiten zur API finden Sie unter [SendMessage AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird eine Nachricht mit den angegebenen Attributen und dem angegebenen Nachrichtentext an die angegebene Warteschlange gesendet, wobei die Zustellung der Nachricht um 10 Sekunden verzögert wird.

```
$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
```

Ausgabe:

MD5ofMessageAttributes	MD5ofMessageBody	MessageId
-----	-----	-----
1d3e51347bc042efbdf6dda31EXAMPLE c739-4d0c-818b-1820eEXAMPLE	51b0a3256d59467f973009b73EXAMPLE	c35fed8f-

- Einzelheiten zur API finden Sie unter [SendMessage AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
```

- Einzelheiten zur API finden Sie [SendMessage](#) in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

Note

Es gibt noch mehr dazu. [GitHub](#) Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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__

```

- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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(())
    }
}

```

- Einzelheiten zur API finden Sie [SendMessage](#) in der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

TRY.
    oo_result = lo_sqs->sendmessage(
        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.

```

- Einzelheiten zur API finden Sie [SendMessage](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **SendMessageBatch** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `SendMessageBatch` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- [Senden und Empfangen von Stapeln von Nachrichten](#)

CLI

AWS CLI

So senden Sie mehrere Nachrichten als Batch

In diesem Beispiel werden 2 Nachrichten mit den angegebenen Nachrichtentexten, Verzögerungszeiten und Nachrichtenattributen an die angegebene Warteschlange gesendet.

Befehl:

```
aws sqs send-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-batch.json
```

Eingabedatei (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"
    }
}
}

```

```
}
]
```

Ausgabe:

```
{
  "Successful": [
    {
      "MD50fMessageBody": "203c4a38...7943237e",
      "MD50fMessageAttributes": "10809b55...baf283ef",
      "Id": "FuelReport-0001-2015-09-16T140731Z",
      "MessageId": "d175070c-d6b8-4101-861d-adeb3EXAMPLE"
    },
    {
      "MD50fMessageBody": "2cf0159a...c1980595",
      "MD50fMessageAttributes": "55623928...ae354a25",
      "Id": "FuelReport-0002-2015-09-16T140930Z",
      "MessageId": "f9b7d55d-0570-413e-b9c5-a9264EXAMPLE"
    }
  ]
}
```

- Einzelheiten zur API finden Sie [SendMessageBatch](#) in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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);
```

- Einzelheiten zur API finden Sie [SendMessageBatch](#) in der AWS SDK for Java 2.x API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel werden zwei Nachrichten mit den angegebenen Attributen und Nachrichtentexten an die angegebene Warteschlange gesendet. Die Zustellung verzögert sich um 15 Sekunden für die erste Nachricht und um 10 Sekunden für die zweite Nachricht.

```
$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
```

Ausgabe:

```
Failed      Successful
-----
{}          {FirstMessage, SecondMessage}
```

- Einzelheiten zur API finden Sie unter [SendMessageBatch AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel werden zwei Nachrichten mit den angegebenen Attributen und Nachrichtentexten an die angegebene Warteschlange gesendet. Die Zustellung verzögert sich um 15 Sekunden für die erste Nachricht und um 10 Sekunden für die zweite Nachricht.

```
$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

```

Ausgabe:

```

Failed      Successful
-----
{}          {FirstMessage, SecondMessage}

```

- Einzelheiten zur API finden Sie unter [SendMessageBatch AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. [GitHub](#) Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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

```

- Einzelheiten zur API finden Sie [SendMessageBatch](#) in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK für Ruby

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
```

- Einzelheiten zur API finden Sie [SendMessageBatch](#) in der AWS SDK für Ruby API-Referenz.

SAP ABAP

SDK für SAP ABAP

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
TRY.
    oo_result = lo_sqs->sendmessagebatch(          " oo_result is returned for
testing purposes. "
        iv_queueurl = iv_queue_url
        it_entries = it_messages ).
    MESSAGE 'Messages sent to SQS queue.' TYPE 'I'.
CATCH /aws1/cx_sqsbtcentidsnotdist00.
    MESSAGE 'Two or more batch entries in the request have the same ID.' TYPE
'E'.
CATCH /aws1/cx_sqsbatchreqtoolong.
    MESSAGE 'The length of all the messages put together is more than the
limit.' TYPE 'E'.
CATCH /aws1/cx_sqsemptybatchrequest.
    MESSAGE 'The batch request does not contain any entries.' TYPE 'E'.
CATCH /aws1/cx_sqsinvbatchentryid.
    MESSAGE 'The ID of a batch entry in a batch request is not valid.' TYPE
'E'.
CATCH /aws1/cx_sqstoomanyentriesin00.
    MESSAGE 'The batch request contains more entries than allowed.' TYPE 'E'.
CATCH /aws1/cx_sqsunsupportedop.
    MESSAGE 'Operation not supported.' TYPE 'E'.
ENDTRY.
```

- Einzelheiten zur API finden Sie [SendMessageBatch](#) in der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **SetQueueAttributes** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `SetQueueAttributes` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- [Veröffentlichen Sie Nachrichten in Warteschlangen](#)

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Richtlinienattribut einer Warteschlange für ein Thema festlegen

```
/// <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;
}

```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

    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();
}

```

Konfigurieren Sie eine Warteschlange für unzustellbare Nachrichten.

```
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();
}

```

Konfigurieren Sie eine Amazon-SQS-Warteschlange für die Verwendung einer Langabfrage.

```

    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();
}

```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So legen Sie Warteschlangenattribute fest

In diesem Beispiel wird für die angegebene Warteschlange eine Zustellungsverzögerung von 10 Sekunden, eine maximale Nachrichtengröße von 128 KB (128 KB x 1 024 Byte), eine Aufbewahrungszeit für Nachrichten von 3 Tagen (3 Tage x 24 Stunden x 60 Minuten x 60 Sekunden), eine Wartezeit für den Empfang von Nachrichten von 20 Sekunden, und ein standardmäßiges Sichtbarkeits-Timeout von 60 Sekunden festgelegt. In diesem Beispiel wird der angegebenen Warteschlange für unzustellbare Nachrichten außerdem eine maximale Empfangszahl von 1 000 Nachrichten zugewiesen.

Befehl:

```
aws sqs set-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-attributes.json
```

Eingabedatei (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"
}
```

Ausgabe:

None.

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der AWS CLI Befehlsreferenz.

Go

SDK für Go V2

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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

```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Konfigurieren Sie ein Amazon SQS für die serverseitige Verschlüsselung (SSE) mit einem benutzerdefinierten KMS-Schlüssel.

```

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();
    }
}

```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der AWS SDK for Java 2.x API-Referenz.

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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;
};
```

Konfigurieren Sie eine Amazon-SQS-Warteschlange für die Verwendung einer Langabfrage.

```
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;
};

```

Konfigurieren Sie eine Warteschlange für unzustellbare Nachrichten.

```

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;
};

```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der AWS SDK für JavaScript API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Dieses Beispiel zeigt, wie Sie eine Richtlinie festlegen, mit der eine Warteschlange für ein SNS-Thema abonniert wird. Wenn eine Nachricht in einem Thema veröffentlicht wird, wird eine Nachricht an die abonnierte Warteschlange gesendet.

```
# 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
$policy = @"
{
  "Version":"2012-10-17",
  "Id": "$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 }
```

Beispiel 2: In diesem Beispiel werden die angegebenen Attribute für die angegebene Warteschlange festgelegt.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
"131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Einzelheiten zur API finden Sie unter [SetQueueAttributes AWS -Tools für PowerShell](#) Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Dieses Beispiel zeigt, wie Sie eine Richtlinie festlegen, mit der eine Warteschlange für ein SNS-Thema abonniert wird. Wenn eine Nachricht in einem Thema veröffentlicht wird, wird eine Nachricht an die abonnierte Warteschlange gesendet.

```
# 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
$policy = @"
{
  "Version":"2012-10-17",
  "Id": "$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 }

```

Beispiel 2: In diesem Beispiel werden die angegebenen Attribute für die angegebene Warteschlange festgelegt.

```

Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
    "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

```

- Einzelheiten zur API finden Sie unter [SetQueueAttributes AWS -Tools für PowerShell](#) Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Richtlinienattribut einer Warteschlange für ein Thema festlegen

```

class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

```

```

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """
        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)

    def set_queue_policy_for_topic(self, queue_arn: str, topic_arn: str,
        queue_url: str) -> bool:
        """
        Set the queue policy to allow SNS to send messages to the queue.

        :param queue_arn: The ARN of the SQS queue.
        :param topic_arn: The ARN of the SNS topic.
        :param queue_url: The URL of the SQS queue.
        :return: True if successful.
        :raises ClientError: If setting the queue policy fails.
        """
        try:
            # Create policy that allows SNS to send messages to the queue
            policy = {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "sns.amazonaws.com"
                        },
                        "Action": "sqs:SendMessage",
                        "Resource": queue_arn,
                        "Condition": {
                            "ArnEquals": {
                                "aws:SourceArn": topic_arn
                            }
                        }
                    }
                ]
            }

```

```

        ]
    }

    self.sqs_client.set_queue_attributes(
        QueueUrl=queue_url,
        Attributes={
            'Policy': json.dumps(policy)
        }
    )

    logger.info(f"Set queue policy for {queue_url} to allow messages from
{topic_arn}")
    return True

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error setting queue policy: {error_code} - {e}")
    raise

```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in AWS SDK for Python (Boto3) API Reference.

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

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.")  
}
```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der API-Referenz zum AWS SDK für Swift.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Szenarien für die Verwendung von Amazon SQS AWS SDKs

Die folgenden Codebeispiele zeigen Ihnen, wie Sie gängige Szenarien in Amazon SQS mit AWS SDKs implementieren. Diese Szenarien zeigen Ihnen, wie Sie bestimmte Aufgaben durch den Aufruf mehrerer Funktionen innerhalb von Amazon SQS oder in Kombination mit anderen AWS-Services ausführen können. Jedes Szenario enthält einen Link zum vollständigen Quell-Code, wo Sie Anleitungen zum Einrichten und Ausführen des Codes finden.

Szenarien zielen auf eine mittlere Erfahrungsebene ab, um Ihnen zu helfen, Service-Aktionen im Kontext zu verstehen.

Beispiele

- [Entwickeln einer Webanwendung, die Nachrichten mithilfe von Amazon SQS sendet und abrufen](#)
- [Erstellen einer Messenger-Anwendung mit Step Functions](#)
- [Erstellen Sie eine Amazon-Textextract-Explorer-Anwendung](#)
- [Mit einem SDK erstellen und in einem FIFO-Amazon-SNS-Thema veröffentlichen AWS](#)
- [Erkennen Sie Personen und Objekte in einem Video mit Amazon Rekognition mithilfe eines SDK AWS](#)
- [Verwalten Sie große Amazon SQS SQS-Nachrichten mithilfe von Amazon S3 mit einem SDK AWS](#)

- [Empfangen und Verarbeiten Amazon S3 S3-Ereignisbenachrichtigungen mithilfe eines AWS SDK](#)
- [Veröffentlichen Sie Amazon SNS SNS-Nachrichten mithilfe eines SDK in Amazon SQS SQS-Warteschlangen AWS](#)
- [Senden und Empfangen von Nachrichtenstapeln mit Amazon SQS mithilfe eines SDK AWS](#)
- [Verwenden Sie das AWS Message Processing Framework for .NET, um Amazon SQS SQS-Nachrichten zu veröffentlichen und zu empfangen](#)
- [Verwenden der Amazon SQS Java Messaging Library, um mit der Java-Message-Service\(JMS\)-Schnittstelle zu arbeiten](#)
- [Arbeiten Sie mit Warteschlangen-Tags und Amazon SQS mithilfe eines SDK AWS](#)

Entwickeln einer Webanwendung, die Nachrichten mithilfe von Amazon SQS sendet und abrufen

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon SQS eine Nachrichtenanwendung erstellen können.

Java

SDK für Java 2.x

Zeigt, wie Sie die Amazon-SQS-API verwenden können, um eine Spring-REST-API zu entwickeln, die Nachrichten sendet und abrufen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK für Kotlin

Zeigt, wie Sie die Amazon-SQS-API verwenden können, um eine Spring-REST-API zu entwickeln, die Nachrichten sendet und abrufen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon SQS

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen einer Messenger-Anwendung mit Step Functions

Das folgende Codebeispiel zeigt, wie eine AWS Step Functions Messenger-Anwendung erstellt wird, die Nachrichtendatensätze aus einer Datenbanktabelle abrufen.

Python

SDK für Python (Boto3)

Zeigt, wie AWS SDK für Python (Boto3) mit AWS Step Functions eine Messenger-Anwendung erstellt wird, die Nachrichtendatensätze aus einer Amazon DynamoDB-Tabelle abrufen und sie mit Amazon Simple Queue Service (Amazon SQS) senden. Die Zustandsmaschine ist mit einer AWS Lambda Funktion integriert, mit der die Datenbank nach nicht gesendeten Nachrichten durchsucht werden kann.

- Erstellen Sie einen Zustandsautomaten, der Nachrichtendatensätze aus einer Amazon-DynamoDB-Tabelle abrufen und aktualisiert.
- Aktualisieren Sie die Definition des Zustandsautomaten, um auch Nachrichten an Amazon Simple Queue Service (Amazon SQS) zu senden.
- Starten und stoppen Sie Ausführungen des Zustandsautomaten.
- Stellen Sie vom Zustandsautomaten aus über Serviceintegrationen eine Verbindung zu Lambda, DynamoDB und Amazon SQS her.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB

- Lambda
- Amazon SQS
- Step Functions

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen Sie eine Amazon-Texttract-Explorer-Anwendung

Die folgenden Code-Beispiele zeigen, wie man die Amazon-Texttract-Ausgabe in einer interaktiven Anwendung untersuchen kann.

JavaScript

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript eine React-Anwendung erstellen, die Amazon Texttract verwendet, um Daten aus einem Dokumentbild zu extrahieren und auf einer interaktiven Webseite anzuzeigen. Dieses Beispiel wird in einem Webbrowser ausgeführt und erfordert eine authentifizierte Amazon-Cognito-Identität für Anmeldeinformationen. Es verwendet Amazon Simple Storage Service (Amazon S3) zur Speicherung und fragt für Benachrichtigungen eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, die ein Amazon Simple Notification Service (Amazon SNS)-Thema abonniert hat.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Texttract

Python

SDK für Python (Boto3)

Zeigt, wie Sie AWS SDK für Python (Boto3) mit Amazon Textract Text-, Formular- und Tabellenelemente in einem Dokumentbild erkennen können. Das Eingabe-Image und die Amazon-Textract-Ausgabe werden in einer Tkinter-Anwendung angezeigt, mit der Sie die erkannten Elemente untersuchen können.

- Senden Sie ein Dokument-Image an Amazon Textract und untersuchen Sie die Ausgabe erkannter Elemente.
- Senden Sie Images direkt an Amazon Textract oder über einen Amazon Simple Storage Service (Amazon S3)-Bucket.
- Verwenden Sie asynchron APIs , um einen Job zu starten, der nach Abschluss des Jobs eine Benachrichtigung in einem Amazon Simple Notification Service (Amazon SNS) -Thema veröffentlicht.
- Stellen Sie eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, um eine Meldung zum Abschluss des Auftrags zu erhalten.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Mit einem SDK erstellen und in einem FIFO-Amazon-SNS-Thema veröffentlichen AWS

Die folgenden Code-Beispiele zeigen, wie man ein FIFO-Amazon-SNS-Thema erstellt.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Dieses Beispiel

- erstellt ein Amazon-SNS-FIFO-Thema, zwei Amazon SQS-FIFO-Warteschlangen und eine Standard-Warteschlange.
- abonniert die Warteschlangen für das Thema und veröffentlicht eine Nachricht zu dem Thema.

Der [Test](#) überprüft den Eingang der Nachricht in jeder Warteschlange. Das [vollständige Beispiel](#) zeigt auch das Hinzufügen von Zugriffsrichtlinien und löscht die Ressourcen am Ende.

```
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 be 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);
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
 - [CreateTopic](#)
 - [Veröffentlichen](#)
 - [Abonnieren](#)

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie ein Amazon-SNS-FIFO-Thema, abonnieren Sie eine Amazon-SQS-FIFO- und eine Standard-Warteschlange für das Thema und veröffentlichen Sie eine Nachricht zu dem Thema.

```
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
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
 - [CreateTopic](#)
 - [Veröffentlichen](#)
 - [Abonnieren](#)

SAP ABAP

SDK für SAP ABAP

 Note

Es gibt noch mehr dazu. [GitHub](#) Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie ein FIFO-Thema, abonnieren Sie eine Amazon-SQS-FIFO-Warteschlange für das Thema und veröffentlichen Sie eine Nachricht zu einem Amazon-SNS-Thema.

```
" Creates a FIFO topic. "
DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsm_w=>tt_topicattributesmap.
DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsm_w=>ts_topicattributesmap_maprow.
ls_tpc_attributes-key = 'FifoTopic'.
ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsm_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_snssubscriptionlmt00.
        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.

```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für SAP ABAP.
 - [CreateTopic](#)
 - [Veröffentlichen](#)
 - [Abonnieren](#)

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erkennen Sie Personen und Objekte in einem Video mit Amazon Rekognition mithilfe eines SDK AWS

Die folgenden Code-Beispiele zeigen, wie man Personen und Objekte in einem Video mit Amazon Rekognition erkennt.

Java

SDK für Java 2.x

Zeigt, wie man die Amazon-Rekognition-Java-API verwendet, um eine App zu erstellen, die Gesichter und Objekte in Videos erkennt, die sich in einem Amazon Simple Storage Service (Amazon S3)-Bucket befinden. Die App sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK für Python (Boto3)

Verwenden Sie Amazon Rekognition, um Gesichter, Objekte und Personen in Videos zu erkennen, indem Sie asynchrone Erkennungsaufträge starten. In diesem Beispiel wird Amazon Rekognition auch so konfiguriert, dass es ein Amazon Simple Notification Service (Amazon SNS)-Thema benachrichtigt, wenn Aufträge abgeschlossen sind, und eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange bei dem Thema abonniert. Wenn die Warteschlange eine Meldung über einen Job erhält, wird der Job abgerufen und die Ergebnisse werden ausgegeben.

Dieses Beispiel lässt sich am besten auf ansehen GitHub. Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwalten Sie große Amazon SQS SQS-Nachrichten mithilfe von Amazon S3 mit einem SDK AWS

Das folgende Codebeispiel zeigt, wie Sie die erweiterte Clientbibliothek von Amazon SQS verwenden, um mit großen Amazon-SQS-Nachrichten zu arbeiten.

Java

SDK für Java 2.x

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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;
    }
}
```

- Weitere Informationen finden Sie im [AWS SDK for Java 2.x -Entwicklerhandbuch](#).
- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
 - [CreateBucket](#)
 - [PutBucketLifecycleConfiguration](#)
 - [ReceiveMessage](#)
 - [SendMessage](#)

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Empfangen und Verarbeiten Amazon S3 S3-Ereignisbenachrichtigungen mithilfe eines AWS SDK

Das folgende Codebeispiel zeigt, wie Sie mit S3-Ereignisbenachrichtigungen auf objektorientierte Weise arbeiten.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Dieses Beispiel zeigt, wie Sie ein S3-Benachrichtigungsereignis mithilfe von Amazon SQS verarbeiten.

```
/**
 * 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);
}
}

```


- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
 - [DeleteMessageBatch](#)
 - [GetQueueAttributes](#)
 - [PutBucketNotificationConfiguration](#)
 - [ReceiveMessage](#)

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Veröffentlichen Sie Amazon SNS SNS-Nachrichten mithilfe eines SDK in Amazon SQS SQS-Warteschlangen AWS

Die folgenden Code-Beispiele veranschaulichen Folgendes:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
/// <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 messageId = 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($"{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;
    }
}

```

Erstellen Sie eine Klasse, die Amazon-SQS-Operationen einschließt.

```

    /// <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;
}
}

```

Erstellen Sie eine Klasse, die Amazon-SNS-Operationen einschließt.

```

/// <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;
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für .NET - API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)

- [SetQueueAttributes](#)
- [Abonnieren](#)
- [Unsubscribe](#)

C++

SDK für C++

Note

Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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 ostream;
        ostream << "Enter a name for " << (first ? "an" : "the next")
              << " SQS queue. ";
        queueName = askQuestion(ostream.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 ostream;
            ostream << "Filter messages for \"" << queueName
                << "\"'s subscription to the topic \""
                << topicName << "\"? (y/n)";

            // Add filter if user answers yes.
            if (askYesNoQuestion(ostream.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();
}

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für C++ -API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)

- [Unsubscribe](#)

Go

SDK für Go V2

 Note

Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
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
message? (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\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))
}

```

Definieren Sie eine Struktur, die in diesem Beispiel verwendete Amazon-SNS-Aktionen umschließt.

```

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 subscribe 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
}

```

Definieren Sie eine Struktur, die in diesem Beispiel verwendete Amazon-SQS-Aktionen umschließt.

```

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
}

```

Bereinigen von Ressourcen.

```

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)
        }
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)

- [GetQueueAttributes](#)
- [Veröffentlichen](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Abonnieren](#)
- [Unsubscribe](#)

Java

SDK für Java 2.x

Note

Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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> queueAttrs = response.attributesAsStrings();
        for (Map.Entry<String, String> queueAtt : queueAttrs.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 "";
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)
 - [Unsubscribe](#)

JavaScript

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Dies ist der Einstiegspunkt für dieses Szenario.

```
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();
};
```

Der obige Code stellt die erforderlichen Abhängigkeiten bereit und startet das Szenario. Der nächste Abschnitt enthält den Großteil des Beispiels.

```
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();
    }
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)
 - [Unsubscribe](#)

Kotlin

SDK für Kotlin

Note

Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für Kotlin.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)

- [GetQueueAttributes](#)
- [Veröffentlichen](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Abonnieren](#)
- [Unsubscribe](#)

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
class TopicsAndQueuesScenario:
    """Manages the Topics and Queues feature scenario."""

    DASHES = "-" * 80

    def __init__(self, sns_wrapper: SnsWrapper, sqs_wrapper: SqsWrapper) -> None:
        """
        Initialize the Topics and Queues scenario.

        :param sns_wrapper: SnsWrapper instance for SNS operations.
        :param sqs_wrapper: SqsWrapper instance for SQS operations.
        """
        self.sns_wrapper = sns_wrapper
        self.sqs_wrapper = sqs_wrapper

        # Scenario state
        self.use_fifo_topic = False
        self.use_content_based_deduplication = False
        self.topic_name = None
        self.topic_arn = None
```

```
self.queue_count = 2
self.queue_urls = []
self.subscription_arns = []
self.tones = ["cheerful", "funny", "serious", "sincere"]

def run_scenario(self) -> None:
    """Run the Topics and Queues feature scenario."""
    print(self.DASHES)
    print("Welcome to messaging with topics and queues.")
    print(self.DASHES)
    print(f"""
    In this scenario, you will create an SNS topic and subscribe
    {self.queue_count} SQS queues to the topic.
    You can select from several options for configuring the topic and the
    subscriptions for the queues.
    You can then post to the topic and see the results in the queues.
    """)

    try:
        # Setup Phase
        print(self.DASHES)
        self._setup_topic()
        print(self.DASHES)

        self._setup_queues()
        print(self.DASHES)

        # Demonstration Phase
        self._publish_messages()
        print(self.DASHES)

        # Examination Phase
        self._poll_queues_for_messages()
        print(self.DASHES)

        # Cleanup Phase
        self._cleanup_resources()
        print(self.DASHES)

    except Exception as e:
        logger.error(f"Scenario failed: {e}")
        print(f"There was a problem with the scenario: {e}")
        print("\nInitiating cleanup...")
        try:
```

```

        self._cleanup_resources()
    except Exception as cleanup_error:
        logger.error(f"Error during cleanup: {cleanup_error}")

    print("Messaging with topics and queues scenario is complete.")
    print(self.DASHES)

def _setup_topic(self) -> None:
    """Set up the SNS topic to be used with the queues."""
    print("SNS topics can be configured as FIFO (First-In-First-Out).")
    print("FIFO topics deliver messages in order and support deduplication
and message filtering.")
    print()

    self.use_fifo_topic = q.ask("Would you like to work with FIFO topics? (y/
n): ", q.is_yesno)

    if self.use_fifo_topic:
        print(self.DASHES)
        self.topic_name = q.ask("Enter a name for your SNS topic: ",
q.non_empty)
        print("Because you have selected a FIFO topic, '.fifo' must be
appended to the topic name.")
        print()

        print(self.DASHES)
        print("""
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.
""")

        self.use_content_based_deduplication = q.ask(
            "Use content-based deduplication instead of entering a
deduplication ID? (y/n): ",
            q.is_yesno
        )

```

```
        else:
            self.topic_name = q.ask("Enter a name for your SNS topic: ",
q.non_empty)

        print(self.DASHES)

        # Create the topic
        self.topic_arn = self.sns_wrapper.create_topic(
            self.topic_name,
            self.use_fifo_topic,
            self.use_content_based_deduplication
        )

        print(f"Your new topic with the name {self.topic_name}")
        print(f"  and Amazon Resource Name (ARN) {self.topic_arn}")
        print(f"  has been created.")
        print()

    def _setup_queues(self) -> None:
        """Set up the SQS queues and subscribe them to the topic."""
        print(f"Now you will create {self.queue_count} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.")

        for i in range(self.queue_count):
            queue_name = q.ask(f"Enter a name for SQS queue #{i+1}: ",
q.non_empty)

            if self.use_fifo_topic and i == 0:
                print("Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.")

            # Create the queue
            queue_url = self.sqs_wrapper.create_queue(queue_name,
self.use_fifo_topic)
            self.queue_urls.append(queue_url)

            print(f"Your new queue with the name {queue_name}")
            print(f"  and queue URL {queue_url}")
            print(f"  has been created.")
            print()

            if i == 0:
                print("The queue URL is used to retrieve the queue ARN,")
                print("which is used to create a subscription.")
```

```

        print(self.DASHES)

    # Get queue ARN
    queue_arn = self.sqs_wrapper.get_queue_arn(queue_url)

    if i == 0:
        print("An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue,")
        print("enabling it to receive messages from an SNS topic.")

    # Set queue policy to allow SNS to send messages
    self.sqs_wrapper.set_queue_policy_for_topic(queue_arn,
self.topic_arn, queue_url)

    # Set up message filtering if using FIFO
    subscription_arn = self._setup_subscription_with_filter(i, queue_arn,
queue_name)
    self.subscription_arns.append(subscription_arn)

    def _setup_subscription_with_filter(self, queue_index: int, queue_arn: str,
queue_name: str) -> str:
        """Set up subscription with optional message filtering."""
        filter_policy = None

        if self.use_fifo_topic:
            print(self.DASHES)
            if queue_index == 0:
                print("Subscriptions to a FIFO topic can have filters.")
                print("If you add a filter to this subscription, then only the
filtered messages")
                print("will be received in the queue.")
                print()
                print("For information about message filtering,")
                print("see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html")
                print()
                print("For this example, you can filter messages by a TONE
attribute.")

            use_filter = q.ask(f"Filter messages for {queue_name}'s subscription
to the topic? (y/n): ", q.is_yesno)

            if use_filter:
                filter_policy = self._create_filter_policy()

```

```

        subscription_arn = self.sns_wrapper.subscribe_queue_to_topic(
            self.topic_arn, queue_arn, filter_policy
        )

        print(f"The queue {queue_name} has been subscribed to the topic {self.topic_name}")
        print(f"    with the subscription ARN {subscription_arn}")

        return subscription_arn

    def _create_filter_policy(self) -> str:
        """Create a message filter policy based on user selections."""
        print(self.DASHES)
        print("You can filter messages by one or more of the following TONE attributes.")

        filter_selections = []
        selection_number = 0

        while True:
            print("Enter a number to add a TONE filter, or enter 0 to stop adding filters.")
            for i, tone in enumerate(self.tones, 1):
                print(f"    {i}. {tone}")

            selection = q.ask("Your choice: ", q.is_int, q.in_range(0, len(self.tones)))

            if selection == 0:
                break
            elif selection > 0 and self.tones[selection - 1] not in filter_selections:
                filter_selections.append(self.tones[selection - 1])
                print(f"Added '{self.tones[selection - 1]}' to filter list.")

            if filter_selections:
                filters = {"tone": filter_selections}
                return json.dumps(filters)
        return None

    def _publish_messages(self) -> None:
        """Publish messages to the topic with various options."""
        print("Now we can publish messages.")

```



```
keep_sending = True
while keep_sending:
    print()
    message = q.ask("Enter a message to publish: ", q.non_empty)

    message_group_id = None
    deduplication_id = None
    tone_attribute = None

    if self.use_fifo_topic:
        print("Because you are using a FIFO topic, you must set a message
group ID.")
        print("All messages within the same group will be received in the
order they were published.")
        print()
        message_group_id = q.ask("Enter a message group ID for this
message: ", q.non_empty)

        if not self.use_content_based_deduplication:
            print("Because you are not using content-based
deduplication,")
            print("you must enter a deduplication ID.")
            deduplication_id = q.ask("Enter a deduplication ID for this
message: ", q.non_empty)

        # Ask about tone attribute
        add_attribute = q.ask("Add an attribute to this message? (y/n):
", q.is_yesno)
        if add_attribute:
            print("Enter a number for an attribute:")
            for i, tone in enumerate(self.tones, 1):
                print(f" {i}. {tone}")

            selection = q.ask("Your choice: ", q.is_int, q.in_range(1,
len(self.tones)))
            if 1 <= selection <= len(self.tones):
                tone_attribute = self.tones[selection - 1]

    # Publish the message
    message_id = self.sns_wrapper.publish_message(
        self.topic_arn,
        message,
        tone_attribute,
```

```

        deduplication_id,
        message_group_id
    )

    print(f"Message published with ID: {message_id}")

    keep_sending = q.ask("Send another message? (y/n): ", q.is_yesno)

def _poll_queues_for_messages(self) -> None:
    """Poll all queues for messages and display results."""
    for i, queue_url in enumerate(self.queue_urls):
        print(f"Polling queue #{i+1} at {queue_url} for messages...")

        q.ask("Press Enter to continue...")

        messages = self._poll_queue_for_messages(queue_url)

        if messages:
            print(f"{len(messages)} message(s) were received by queue #{i+1}")

            for j, message in enumerate(messages, 1):
                print(f"  Message {j}:")
                # Parse the SNS message body to get the actual message
                try:
                    sns_message = json.loads(message['Body'])
                    actual_message = sns_message.get('Message',
message['Body'])

                    print(f"    {actual_message}")
                except (json.JSONDecodeError, KeyError):
                    print(f"    {message['Body']}")

            # Delete the messages
            self.sqs_wrapper.delete_messages(queue_url, messages)
            print(f"Messages deleted from queue #{i+1}")
        else:
            print(f"No messages received by queue #{i+1}")

        print(self.DASHES)

def _poll_queue_for_messages(self, queue_url: str) -> List[Dict[str, Any]]:
    """Poll a single queue for messages."""
    all_messages = []
    max_polls = 3 # Limit polling to avoid infinite loops

```

```

    for poll_count in range(max_polls):
        messages = self.sqs_wrapper.receive_messages(queue_url, 10)

        if messages:
            all_messages.extend(messages)
            print(f" Received {len(messages)} messages in poll {poll_count +
1}")

            # Small delay between polls
            time.sleep(1)
        else:
            print(f" No messages in poll {poll_count + 1}")
            break

    return all_messages

def _cleanup_resources(self) -> None:
    """Clean up all resources created during the scenario."""
    print("Cleaning up resources...")

    # Delete queues
    for i, queue_url in enumerate(self.queue_urls):
        if queue_url:
            delete_queue = q.ask(f"Delete queue #{i+1} with URL {queue_url}?
(y/n): ", q.is_yesno)
            if delete_queue:
                try:
                    self.sqs_wrapper.delete_queue(queue_url)
                    print(f"Deleted queue #{i+1}")
                except Exception as e:
                    print(f"Error deleting queue #{i+1}: {e}")

    # Unsubscribe from topic
    for i, subscription_arn in enumerate(self.subscription_arns):
        if subscription_arn:
            try:
                self.sns_wrapper.unsubscribe(subscription_arn)
                print(f"Unsubscribed subscription #{i+1}")
            except Exception as e:
                print(f"Error unsubscribing #{i+1}: {e}")

    # Delete topic
    if self.topic_arn:
        delete_topic = q.ask(f"Delete topic {self.topic_name}? (y/n): ",
q.is_yesno)

```

```

        if delete_topic:
            try:
                self.sns_wrapper.delete_topic(self.topic_arn)
                print(f"Deleted topic {self.topic_name}")
            except Exception as e:
                print(f"Error deleting topic: {e}")

    print("Resource cleanup complete.")

```

Erstellen Sie Klassen, die Amazon SNS- und Amazon SQS SQS-Operationen für die Verwendung im Szenario zusammenfassen.

```

class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def create_topic(
        self,
        topic_name: str,
        is_fifo: bool = False,
        content_based_deduplication: bool = False
    ) -> str:
        """

```

```

        Create an SNS topic.

        :param topic_name: The name of the topic to create.
        :param is_fifo: Whether to create a FIFO topic.
        :param content_based_deduplication: Whether to use content-based
deduplication for FIFO topics.
        :return: The ARN of the created topic.
        :raises ClientError: If the topic creation fails.
        """
    try:
        # Add .fifo suffix for FIFO topics
        if is_fifo and not topic_name.endswith('.fifo'):
            topic_name += '.fifo'

        attributes = {}
        if is_fifo:
            attributes['FifoTopic'] = 'true'
            if content_based_deduplication:
                attributes['ContentBasedDeduplication'] = 'true'

        response = self.sns_client.create_topic(
            Name=topic_name,
            Attributes=attributes
        )

        topic_arn = response['TopicArn']
        logger.info(f"Created topic: {topic_name} with ARN: {topic_arn}")
        return topic_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error creating topic {topic_name}: {error_code} -
{e}")
        raise

    def subscribe_queue_to_topic(
        self,
        topic_arn: str,
        queue_arn: str,
        filter_policy: Optional[str] = None
    ) -> str:
        """
        Subscribe an SQS queue to an SNS topic.

```

```

:param topic_arn: The ARN of the SNS topic.
:param queue_arn: The ARN of the SQS queue.
:param filter_policy: Optional JSON filter policy for message filtering.
:return: The ARN of the subscription.
:raises ClientError: If the subscription fails.
"""
try:
    attributes = {}
    if filter_policy:
        attributes['FilterPolicy'] = filter_policy

    response = self.sns_client.subscribe(
        TopicArn=topic_arn,
        Protocol='sqs',
        Endpoint=queue_arn,
        Attributes=attributes
    )

    subscription_arn = response['SubscriptionArn']
    logger.info(f"Subscribed queue {queue_arn} to topic {topic_arn}")
    return subscription_arn

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error subscribing queue to topic: {error_code} - {e}")
    raise

def publish_message(
    self,
    topic_arn: str,
    message: str,
    tone_attribute: Optional[str] = None,
    deduplication_id: Optional[str] = None,
    message_group_id: Optional[str] = None
) -> str:
    """
    Publish a message to an SNS topic.

    :param topic_arn: The ARN of the SNS topic.
    :param message: The message content to publish.
    :param tone_attribute: Optional tone attribute for message filtering.
    :param deduplication_id: Optional deduplication ID for FIFO topics.

```

```

:param message_group_id: Optional message group ID for FIFO topics.
:return: The message ID of the published message.
:raises ClientError: If the message publication fails.
"""
try:
    publish_args = {
        'TopicArn': topic_arn,
        'Message': message
    }

    # Add message attributes if tone is specified
    if tone_attribute:
        publish_args['MessageAttributes'] = {
            'tone': {
                'DataType': 'String',
                'StringValue': tone_attribute
            }
        }

    # Add FIFO-specific parameters
    if message_group_id:
        publish_args['MessageGroupId'] = message_group_id

    if deduplication_id:
        publish_args['MessageDeduplicationId'] = deduplication_id

    response = self.sns_client.publish(**publish_args)

    message_id = response['MessageId']
    logger.info(f"Published message to topic {topic_arn} with ID:
{message_id}")
    return message_id

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error publishing message to topic: {error_code} -
{e}")
    raise

def unsubscribe(self, subscription_arn: str) -> bool:
    """
    Unsubscribe from an SNS topic.

```

```
    :param subscription_arn: The ARN of the subscription to remove.
    :return: True if successful.
    :raises ClientError: If the unsubscribe operation fails.
    """
    try:
        self.sns_client.unsubscribe(SubscriptionArn=subscription_arn)

        logger.info(f"Unsubscribed: {subscription_arn}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')

        if error_code == 'NotFound':
            logger.warning(f"Subscription not found: {subscription_arn}")
            return True  # Already unsubscribed
        else:
            logger.error(f"Error unsubscribing: {error_code} - {e}")
            raise

def delete_topic(self, topic_arn: str) -> bool:
    """
    Delete an SNS topic.

    :param topic_arn: The ARN of the topic to delete.
    :return: True if successful.
    :raises ClientError: If the topic deletion fails.
    """
    try:
        self.sns_client.delete_topic(TopicArn=topic_arn)

        logger.info(f"Deleted topic: {topic_arn}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')

        if error_code == 'NotFound':
            logger.warning(f"Topic not found: {topic_arn}")
            return True  # Already deleted
        else:
            logger.error(f"Error deleting topic: {error_code} - {e}")
            raise
```



```

def list_topics(self) -> list:
    """
    List all SNS topics in the account using pagination.

    :return: List of topic ARNs.
    :raises ClientError: If listing topics fails.
    """
    try:
        topics = []
        paginator = self.sns_client.get_paginator('list_topics')

        for page in paginator.paginate():
            topics.extend([topic['TopicArn'] for topic in page.get('Topics',
[])])

        logger.info(f"Found {len(topics)} topics")
        return topics

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        if error_code == 'AuthorizationError':
            logger.error("Authorization error listing topics - check IAM
permissions")
        else:
            logger.error(f"Error listing topics: {error_code} - {e}")
        raise

class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:
        """
        Initialize the SqsWrapper.

        :param sqs_client: A Boto3 Amazon SQS client.
        """
        self.sqs_client = sqs_client

    @classmethod
    def from_client(cls) -> 'SqsWrapper':
        """

```

```

        Create an SqsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sqs_client = boto3.client('sqs')
        return cls(sqs_client)

def create_queue(self, queue_name: str, is_fifo: bool = False) -> str:
    """
    Create an SQS queue.

    :param queue_name: The name of the queue to create.
    :param is_fifo: Whether to create a FIFO queue.
    :return: The URL of the created queue.
    :raises ClientError: If the queue creation fails.
    """
    try:
        # Add .fifo suffix for FIFO queues
        if is_fifo and not queue_name.endswith('.fifo'):
            queue_name += '.fifo'

        attributes = {}
        if is_fifo:
            attributes['FifoQueue'] = 'true'

        response = self.sqs_client.create_queue(
            QueueName=queue_name,
            Attributes=attributes
        )

        queue_url = response['QueueUrl']
        logger.info(f"Created queue: {queue_name} with URL: {queue_url}")
        return queue_url

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error creating queue {queue_name}: {error_code} - {e}")
        raise

def get_queue_arn(self, queue_url: str) -> str:
    """

```

```

    Get the ARN of an SQS queue.

    :param queue_url: The URL of the queue.
    :return: The ARN of the queue.
    :raises ClientError: If getting queue attributes fails.
    """
    try:
        response = self.sqs_client.get_queue_attributes(
            QueueUrl=queue_url,
            AttributeNames=['QueueArn']
        )

        queue_arn = response['Attributes']['QueueArn']
        logger.info(f"Queue ARN for {queue_url}: {queue_arn}")
        return queue_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error getting queue ARN: {error_code} - {e}")
        raise

def set_queue_policy_for_topic(self, queue_arn: str, topic_arn: str,
queue_url: str) -> bool:
    """
    Set the queue policy to allow SNS to send messages to the queue.

    :param queue_arn: The ARN of the SQS queue.
    :param topic_arn: The ARN of the SNS topic.
    :param queue_url: The URL of the SQS queue.
    :return: True if successful.
    :raises ClientError: If setting the queue policy fails.
    """
    try:
        # Create policy that allows SNS to send messages to the queue
        policy = {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "sns.amazonaws.com"
                    },
                    "Action": "sqs:SendMessage",

```

```

        "Resource": queue_arn,
        "Condition": {
            "ArnEquals": {
                "aws:SourceArn": topic_arn
            }
        }
    ]
}

self.sqs_client.set_queue_attributes(
    QueueUrl=queue_url,
    Attributes={
        'Policy': json.dumps(policy)
    }
)

logger.info(f"Set queue policy for {queue_url} to allow messages from
{topic_arn}")
return True

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error setting queue policy: {error_code} - {e}")
    raise

def receive_messages(self, queue_url: str, max_messages: int = 10) ->
List[Dict[str, Any]]:
    """
    Receive messages from an SQS queue.

    :param queue_url: The URL of the queue to receive messages from.
    :param max_messages: Maximum number of messages to receive (1-10).
    :return: List of received messages.
    :raises ClientError: If receiving messages fails.
    """
    try:
        # Ensure max_messages is within valid range
        max_messages = max(1, min(10, max_messages))

        response = self.sqs_client.receive_message(
            QueueUrl=queue_url,
            MaxNumberOfMessages=max_messages,

```

```

        WaitTimeSeconds=2, # Short polling
        MessageAttributeNames=['All']
    )

    messages = response.get('Messages', [])
    logger.info(f"Received {len(messages)} messages from {queue_url}")
    return messages

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error receiving messages: {error_code} - {e}")
    raise

def delete_messages(self, queue_url: str, messages: List[Dict[str, Any]]) ->
bool:
    """
    Delete messages from an SQS queue in batches.

    :param queue_url: The URL of the queue.
    :param messages: List of messages to delete.
    :return: True if successful.
    :raises ClientError: If deleting messages fails.
    """
    try:
        if not messages:
            return True

        # Build delete entries for batch delete
        delete_entries = []
        for i, message in enumerate(messages):
            delete_entries.append({
                'Id': str(i),
                'ReceiptHandle': message['ReceiptHandle']
            })

        # Delete messages in batches of 10 (SQS limit)
        batch_size = 10
        for i in range(0, len(delete_entries), batch_size):
            batch = delete_entries[i:i + batch_size]

            response = self.sqs_client.delete_message_batch(
                QueueUrl=queue_url,
                Entries=batch
            )

```

```

        )

        # Check for failures
        if 'Failed' in response and response['Failed']:
            for failed in response['Failed']:
                logger.warning(f"Failed to delete message: {failed}")

        logger.info(f"Deleted {len(messages)} messages from {queue_url}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error deleting messages: {error_code} - {e}")
        raise

def delete_queue(self, queue_url: str) -> bool:
    """
    Delete an SQS queue.

    :param queue_url: The URL of the queue to delete.
    :return: True if successful.
    :raises ClientError: If the queue deletion fails.
    """
    try:
        self.sqs_client.delete_queue(QueueUrl=queue_url)

        logger.info(f"Deleted queue: {queue_url}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')

        if error_code == 'AWS.SimpleQueueService.NonExistentQueue':
            logger.warning(f"Queue not found: {queue_url}")
            return True # Already deleted
        else:
            logger.error(f"Error deleting queue: {error_code} - {e}")
            raise

def list_queues(self, queue_name_prefix: Optional[str] = None) -> List[str]:
    """
    List all SQS queues in the account using pagination.

```

```

:param queue_name_prefix: Optional prefix to filter queue names.
:return: List of queue URLs.
:raises ClientError: If listing queues fails.
"""
try:
    queue_urls = []
    paginator = self.sqs_client.get_paginator('list_queues')

    page_params = {}
    if queue_name_prefix:
        page_params['QueueNamePrefix'] = queue_name_prefix

    for page in paginator.paginate(**page_params):
        queue_urls.extend(page.get('QueueUrls', []))

    logger.info(f"Found {len(queue_urls)} queues")
    return queue_urls

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    if error_code == 'AccessDenied':
        logger.error("Access denied listing queues - check IAM
permissions")
    else:
        logger.error(f"Error listing queues: {error_code} - {e}")
    raise

def send_message(self, queue_url: str, message_body: str, **kwargs) -> str:
    """
    Send a message to an SQS queue.

    :param queue_url: The URL of the queue.
    :param message_body: The message content.
    :param kwargs: Additional message parameters (DelaySeconds,
MessageAttributes, etc.).
    :return: The message ID.
    :raises ClientError: If sending the message fails.
    """
    try:
        send_params = {
            'QueueUrl': queue_url,
            'MessageBody': message_body,
            **kwargs

```

```
}

response = self.sqs_client.send_message(**send_params)

message_id = response['MessageId']
logger.info(f"Sent message to {queue_url} with ID: {message_id}")
return message_id

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error sending message: {error_code} - {e}")
    raise
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)
 - [Unsubscribe](#)

Swift

SDK für Swift

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
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)
        }
    }
}

```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Swift.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)
 - [Unsubscribe](#)

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Senden und Empfangen von Nachricht stapeln mit Amazon SQS mithilfe eines SDK AWS

Die folgenden Code-Beispiele veranschaulichen Folgendes:

- Erstellen einer Amazon SQS-Warteschlange
- Senden Sie Stapel von Nachrichten an die Warteschlange.
- Empfangen Sie Stapel von Nachrichten aus der Warteschlange.
- Löschen Sie Stapel von Nachrichten aus der Warteschlange.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Wie in den folgenden Beispielen gezeigt, können Sie Batch-Nachrichtenoperationen mit Amazon SQS auf zwei verschiedene Arten mit dem AWS SDK for Java 2.x durchführen.

`SendRecvBatch.java` verwendet explizite Batch-Operationen. Sie erstellen Nachrichtenstapel manuell und rufen `sendMessageBatch()` und `deleteMessageBatch()` direkt auf. Sie bearbeiten auch Batch-Antworten, einschließlich fehlgeschlagener Nachrichten. Dieser Ansatz gibt Ihnen die volle Kontrolle über die Batchgröße und die Fehlerbehandlung. Es erfordert jedoch mehr Code, um die Logik der Batchverarbeitung zu verwalten.

`SimpleProducerConsumer.java` verwendet die `SqsAsyncBatchManager` High-Level-Bibliothek für die automatische Batchverarbeitung von Anfragen. Sie führen individuelle Aufrufe von `sendMessage()` und `deleteMessage()` durch und verwenden dieselben Methodensignaturen wie der `StandardClient`. Das SDK puffert diese Aufrufe automatisch und sendet sie als Batch-Operationen. Dieser Ansatz erfordert nur minimale Codeänderungen und bietet gleichzeitig Leistungsvorteile bei der Batchverarbeitung.

Verwenden Sie die explizite Batchverarbeitung, wenn Sie eine genaue Kontrolle über die Batchzusammensetzung und die Fehlerbehandlung benötigen. Verwenden Sie die automatische Batchverarbeitung, wenn Sie die Leistung mit minimalen Codeänderungen optimieren möchten.

`SendRecvBatch.java` — Verwendet explizite Batch-Operationen mit Nachrichten.

```
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)
            .numberOfMessages(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 — Verwendet automatisches Batching von Nachrichten.

```

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:
 * - batchSize: 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)
                    .numberOfMessages(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.  
    }  
}  
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
 - [CreateQueue](#)
 - [DeleteMessage](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [ReceiveMessage](#)
 - [SendMessage](#)
 - [SendMessageBatch](#)

Python

SDK für Python (Boto3)

Note

Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie Funktionen, um Amazon-SQS-Nachrichtenfunktionen zu umschließen.

```
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

```

Verwenden Sie die Wrapper-Funktionen, um Nachrichten stapelweise zu senden und zu empfangen.

```

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)
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
 - [CreateQueue](#)
 - [DeleteMessage](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [ReceiveMessage](#)
 - [SendMessage](#)
 - [SendMessageBatch](#)

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie das AWS Message Processing Framework for.NET, um Amazon SQS SQS-Nachrichten zu veröffentlichen und zu empfangen

Das folgende Codebeispiel zeigt, wie Sie mithilfe des AWS Message Processing Framework for.NET Anwendungen erstellen, die Amazon SQS SQS-Nachrichten veröffentlichen und empfangen.

.NET

SDK für .NET

Stellt ein Tutorial für das AWS Message Processing Framework for .NET bereit. Das Tutorial erstellt eine Webanwendung, mit der der Benutzer Amazon-SQS-Nachricht veröffentlichen kann, sowie eine Befehlszeilenanwendung, die die Nachricht empfängt.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im [vollständigen Tutorial](#) im AWS SDK für .NET Entwicklerhandbuch und im Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon SQS

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden der Amazon SQS Java Messaging Library, um mit der Java-Message-Service(JMS)-Schnittstelle zu arbeiten

Das folgende Codebeispiel zeigt, wie Sie die Amazon SQS Java Messaging Library verwenden, um mit der JMS-Schnittstelle zu arbeiten.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Die folgenden Beispiele funktionieren mit standardmäßigen Amazon-SQS-Warteschlangen und beinhalten:

- Senden einer Testnachricht
- Synchrones Empfangen von Nachrichten
- Asynchrones Empfangen von Nachrichten
- Empfangen von Nachrichten im CLIENT_ACKNOWLEDGE-Modus.
- Empfangen von Nachrichten im UNORDERED_ACKNOWLEDGE-Modus.
- Verwenden von Spring, um Abhängigkeiten einzufügen.
- Eine Dienstprogramm-Klasse, die allgemeine Methoden bereitstellt, die in den anderen Beispielen verwendet werden.

Weitere Informationen zur Verwendung von JMS mit Amazon SQS finden Sie im [Entwicklerhandbuch für Amazon SQS](#).

Senden einer Testnachricht

```
/**
 * 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 JMSEException If there is a problem connecting to or sending
 messages to the queue
 */
public static void doSendTextMessage() throws JMSEException {
```

```

        // 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 (JMSEException e) {
        LOGGER.error("Failed sending message: {}", e.getMessage(), e);
    }
}

```

Synchrones Empfangen von Nachrichten

```

/**
 * 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 JMSEException If there is a problem connecting to or receiving
 * messages from the queue
 */
public static void doReceiveMessageSync() throws JMSEException {
    // 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 (JMSEException e) {
        LOGGER.error("Error receiving from SQS: {}", e.getMessage(), e);
    }
}

```

Asynchrones Empfangen von Nachrichten

```

/**
 * 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 JMSEException If there is a problem connecting to or receiving
messages from the queue
 */
public static void doReceiveMessageAsync() throws JMSEException {
    // 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(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 (JMSEException e) {
                LOGGER.error("Error processing message: {}",
e.getMessage());
            }
        });
        // Start receiving incoming messages.
        connection.start();
        LOGGER.info("Waiting for messages...");
    } catch (JMSEException 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" );
}

```

Empfangen von Nachrichten im CLIENT_ACKNOWLEDGE-Modus.

```

/**
 * 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 JMSEException If there is a problem with the messaging operations
*/
public static void doReceiveMessagesSyncClientAcknowledge() throws
JMSEException {
    // 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 JMSEException If there is an error creating or sending the message
 */
private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSEException {
    // 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 JMSEException If there is an error receiving, processing, or
    acknowledging the message
    */
    private static void receiveMessage(MessageConsumer consumer, boolean
    acknowledge) throws JMSEException {
        // 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();
        }
    }
}

```

Empfangen von Nachrichten im UNORDERED_ACKNOWLEDGE-Modus.

```

/**
 * 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 JMSEException      If a JMS-related error occurs during message
operations
*/
public static void doReceiveMessagesUnorderedAcknowledge() throws
JMSEException {
    // 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 JMSEException If message creation or sending fails
 */
private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSEException {
    // 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 JMSEException If message reception or acknowledgment fails
    */
    private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSEException {
        // 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();
        }
    }
}

```

Verwenden von Spring, um Abhängigkeiten einzufügen.

```

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 (JMSEException e) {
            LOGGER.error("Error receiving from SQS.", e);
        }
    }
}

```

Spring-Bean-Definitionen.

```

<?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>
```

Eine Dienstprogramm-Klasse, die allgemeine Methoden bereitstellt, die in den anderen Beispielen verwendet werden.

```
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 JMSEException If there is an error accessing or creating the queue
    */
    public static void ensureQueueExists(SQSConnection connection, String
    queueName, Long visibilityTimeout) throws JMSEException {
        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 (JMSEException 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 JMSEException If there is an error reading the message content
     */
    public static void handleMessage(Message message) throws JMSEException {
        // 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 (JMSEException e) {
                    LOGGER.error("Error processing message: {}", e.getMessage());
                }
            });
            // Start receiving incoming messages.
            connection.start();
        } catch (JMSEException 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 JMSEException If there is an error during the cleanup process
     */
    public static void cleanUpExample(String queueName, Long visibilityTimeout)
throws JMSEException {
        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 (JMSEException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        }
    };
}
}

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
 - [CreateQueue](#)
 - [DeleteQueue](#)

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Arbeiten Sie mit Warteschlangen-Tags und Amazon SQS mithilfe eines SDK AWS

Das folgende Codebeispiel zeigt, wie Sie Tagging-Operationen mit Amazon SQS durchführen.

Java

SDK für Java 2.x

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Im folgenden Beispiel werden Tags für eine Warteschlange erstellt, Tags aufgelistet und ein Tag entfernt.

```
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);
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
 - [ListQueueTags](#)
 - [TagQueue](#)
 - [UntagQueue](#)

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Serverless-Beispiele für Amazon SQS

Die folgenden Codebeispiele zeigen, wie Sie Amazon SQS mit AWS SDKs verwenden.

Beispiele

- [Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser](#)
- [Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser](#)

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Die folgenden Codebeispiele zeigen, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von .NET.

```
// 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.DefaultLambdaJsonSerializer))]

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
        }
        catch { }
    }
}
```

```

        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 für Go V2

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von Go.

```

// 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 für Java 2.x

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von Java.

```
// 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 für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. JavaScript

```
// 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;
  }
}
```

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. TypeScript

```
// 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 für PHP

 Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von PHP.

```
// 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 für Python (Boto3)

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von Python.

```
# 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 für Ruby

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von Ruby.

```
# 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 für Rust

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von Rust.

```
// 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
}
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Die folgenden Codebeispiele zeigen, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet

die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

.NET

SDK für .NET

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batchelementfehlern mit Lambda unter Verwendung von .NET.

```
// 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.DefaultLambdaJsonSerializer),
    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 für Go V2

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batchelementfehlern mit Lambda unter Verwendung von Go.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "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 len(message.Body) > 0 {
            // Your message processing condition here
            fmt.Printf("Successfully processed message: %s\n", message.Body)
        } else {
            // Message processing failed
            fmt.Printf("Failed to process message %s\n", message.MessageId)
            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 für Java 2.x

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei SQS-Batchelementen mit Lambda unter Verwendung von Java.

```
// 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>();

        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 für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von JavaScript

```
// 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}`);
}

```

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von TypeScript

```

// 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 für PHP

Note

Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batchelementfehlern mit Lambda unter Verwendung von PHP.

```

// 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 für Python (Boto3)

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei SQS-Batchelementen mit Lambda unter Verwendung von Python.

```

# 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:
        print(f"Processed message: {record['body']}")
    except Exception as e:
        batch_item_failures.append({"itemIdentifier":
record['messageId']})

sqs_batch_response["batchItemFailures"] = batch_item_failures
return sqs_batch_response
```

Ruby

SDK für Ruby

Note

Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei SQS-Batchelementen mit Lambda unter Verwendung von Ruby.

```
# 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
  end
end
```

```

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
end
end

```

Rust

SDK für Rust

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei SQS-Batchelementen mit Lambda unter Verwendung von Rust.

```

// 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  
    }  
}
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter [Amazon SQS mit einem AWS SDK verwenden](#). Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Behebung von Problemen in Amazon SQS

Dieses Thema enthält Hinweise zur Behebung häufiger Fehler und Probleme, die bei der Verwendung der Amazon SQS-Konsole, der Amazon SQS SQS-API oder anderer Tools mit Amazon SQS auftreten können. Wenn Sie auf ein Problem stoßen, das hier nicht aufgeführt ist, können Sie die Schaltfläche Feedback auf dieser Seite verwenden, um es zu melden.

Weitere Tipps zur Fehlerbehebung und Antworten auf häufig gestellte Supportfragen finden Sie im [AWS - Wissenscenter](#).

Themen

- [Behebung eines Fehlers „Zugriff verweigert“ in Amazon SQS](#)
- [Behebung von Amazon SQS SQS-API-Fehlern](#)
- [Behebung von Problemen mit der Amazon SQS SQS-Warteschlange für unzustellbare Briefe und DLQ Redrive](#)
- [Behebung von Problemen mit der FIFO-Drosselung in Amazon SQS](#)
- [Problembehandlung bei Nachrichten, die bei einem Amazon SQS ReceiveMessage SQS-API-Aufruf nicht zurückgegeben wurden](#)
- [Behebung Amazon SQS SQS-Netzwerkfehlern](#)
- [Problembehandlung bei Amazon Simple Queue Service-Warteschlangen mit AWS X-Ray](#)

Behebung eines Fehlers „Zugriff verweigert“ in Amazon SQS

Die folgenden Themen behandeln die häufigsten Ursachen AccessDenied oder AccessDeniedException Fehler Amazon SQS SQS-API-Aufrufen. Weitere Informationen zur Behebung dieser Fehler finden Sie unter [Wie behebe ich "" - oder AccessDenied "AccessDeniedException" -Fehler Amazon SQS SQS-API-Aufrufen?](#) im AWS Knowledge Center-Leitfaden.

Beispiele für Fehlermeldungen:

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

- oder -


```
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 SQS-Warteschlangenrichtlinie und IAM-Richtlinie

Gehen Sie wie folgt vor, um zu überprüfen, ob der Anforderer über die erforderlichen Berechtigungen zur Durchführung eines Amazon SQS SQS-Vorgangs verfügt:

- Identifizieren Sie den IAM-Prinzipal, der den Amazon SQS SQS-API-Aufruf durchführt. Wenn der IAM-Principal von demselben Konto stammt, müssen entweder die Amazon SQS SQS-Warteschlangenrichtlinie oder die AWS Identity and Access Management (IAM) -Richtlinie Berechtigungen enthalten, um den Zugriff für die Aktion explizit zuzulassen.
- Wenn der Principal eine IAM-Entität ist:
 - Sie können Ihren IAM-Benutzer oder Ihre IAM-Rolle identifizieren, indem Sie die obere rechte Ecke von überprüfen oder den AWS-Managementkonsole Befehl verwenden. [aws sts get-caller-identity](#)
 - Prüfen Sie die IAM-Richtlinien im Zusammenhang mit dem IAM-Benutzer oder der IAM-Rolle. Sie können eine der folgenden Methoden verwenden:
 - [Testen Sie IAM-Richtlinien mit dem IAM Policy Simulator.](#)
 - Überprüfen der verschiedenen [IAM-Richtlinientypen](#)
 - [Bearbeiten Sie gegebenenfalls Ihre IAM-Benutzerrichtlinie.](#)
 - Überprüfen Sie die Warteschlangenrichtlinie und [bearbeiten Sie](#) sie gegebenenfalls.
- Wenn der Principal ein AWS Service ist, muss die Amazon SQS SQS-Warteschlangenrichtlinie den Zugriff explizit zulassen.
- Wenn es sich bei dem Principal um einen kontoübergreifenden Principal handelt, müssen sowohl die Amazon SQS SQS-Warteschlangenrichtlinie als auch die IAM-Richtlinie den Zugriff explizit zulassen.
- Wenn die Richtlinie ein Bedingungelement verwendet, überprüfen Sie, ob die Bedingung den Zugriff einschränkt.

⚠ Important

Eine ausdrückliche Ablehnung in einer der Richtlinien hat Vorrang vor einer ausdrücklichen Zulassung. Hier sind einige grundlegende Beispiele für [Amazon SQS SQS-Richtlinien](#).

AWS Key Management Service Berechtigungen

Wenn in Ihrer Amazon SQS SQS-Warteschlange [serverseitige Verschlüsselung \(SSE\)](#) aktiviert ist und ein Kunde verwaltet wird AWS KMS key, müssen sowohl Produzenten als auch Verbrauchern Berechtigungen erteilt werden. Um zu überprüfen, ob eine Warteschlange verschlüsselt ist, können Sie das [GetQueueAttributesKmsMasterKeyId](#)API-Attribut oder die Warteschlangenkonsolle unter Verschlüsselung verwenden.

- Erforderliche [Berechtigungen für Produzenten](#):

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "<Key ARN>"
}
```

- Erforderliche [Berechtigungen für Verbraucher](#):

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "<Key ARN>"
}
```

- Erforderliche Berechtigungen für den [kontoübergreifenden Zugriff](#):

```
{
  "Effect": "Allow",
  "Action": [
    "kms:DescribeKey",

```

```
"kms:Decrypt",
"kms:ReEncrypt",
"kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

Wählen Sie eine der folgenden Optionen, um die Verschlüsselung für eine Amazon SQS SQS-Warteschlange zu aktivieren:

- [SSE-Amazon SQS](#) (Verschlüsselungsschlüssel, der vom Amazon SQS-Service erstellt und verwaltet wird.)
- [AWS verwalteter Standardschlüssel](#) (alias/aws/sqs)
- [Kundenverwalteter Schlüssel](#)

Wenn Sie jedoch einen AWS-verwalteten [KMS-Schlüssel](#) verwenden, können Sie die Standardschlüsselrichtlinie nicht ändern. Verwenden Sie daher den vom Kunden verwalteten Schlüssel, um Zugriff auf andere Dienste und kontenübergreifende Konten zu gewähren. Auf diese Weise können Sie die Schlüsselrichtlinie bearbeiten.

VPC-Endpunktrichtlinie

Wenn Sie [über einen Amazon Virtual Private Cloud \(Amazon VPC\) -Endpunkt auf Amazon SQS zugreifen, muss die Amazon SQS VPC-Endpunktrichtlinie](#) den Zugriff zulassen. Sie können eine Richtlinie für Amazon VPC-Endpunkte für Amazon SQS erstellen, in der Sie Folgendes angeben können:

1. Prinzipal, der die Aktionen ausführen kann.
2. Aktionen, die ausgeführt werden können
3. Die Ressourcen, für die Aktionen ausgeführt werden können.

Im folgenden Beispiel legt die VPC-Endpunktrichtlinie fest, dass der IAM-Benutzer Nachrichten an die Amazon *MyUser* SQS SQS-Warteschlange senden darf. *MyQueue* Anderen Aktionen, IAM-Benutzern und Amazon SQS-Ressourcen wird der Zugriff über den VPC-Endpunkt verweigert.

```
{
  "Statement": [{
```

```
"Action": ["sqs:SendMessage"],
"Effect": "Allow",
"Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
"Principal": {
  "AWS": "arn:aws:iam:123456789012:user/MyUser"
}
}]
}
```

Richtlinie zur Dienstkontrolle der Organisation

Wenn Sie zu einer Organisation AWS-Konto gehören, können AWS Organizations Richtlinien Sie daran hindern, auf Ihre Amazon SQS SQS-Warteschlangen zuzugreifen. Standardmäßig blockieren AWS Organizations Richtlinien keine Anfragen an Amazon SQS. Stellen Sie jedoch sicher, dass Ihre AWS Organizations Richtlinien nicht so konfiguriert wurden, dass sie den Zugriff auf Amazon SQS SQS-Warteschlangen blockieren. Anweisungen, wie Sie Ihre AWS Organizations Richtlinien überprüfen können, finden Sie im AWS Organizations Benutzerhandbuch unter [Alle Richtlinien auflisten](#).

Behebung von Amazon SQS SQS-API-Fehlern

Die folgenden Themen behandeln die häufigsten Fehler, die bei Amazon SQS SQS-API-Aufrufen auftreten, und deren Behebung.

QueueDoesNotExist Fehler

Dieser Fehler wird zurückgegeben, wenn der Amazon SQS-Service die angegebene Warteschlange für die Amazon SQS SQS-Aktion nicht finden kann.

Mögliche Ursachen und Abhilfemaßnahmen:

- Falsche Region: Überprüfen Sie die Amazon SQS SQS-Client-Konfiguration, um sicherzustellen, dass Sie die richtige Region auf dem Client konfiguriert haben. Wenn Sie keine Region auf dem Client konfigurieren, AWS CLI wählt das SDK oder die Region aus der [Konfigurationsdatei](#) oder der Umgebungsvariablen aus. Wenn das SDK in der Konfigurationsdatei keine Region findet, setzt das SDK die Region standardmäßig auf us-east-1.
- Die Warteschlange wurde möglicherweise kürzlich gelöscht: Wenn die Warteschlange vor dem API-Aufruf gelöscht wurde, gibt der API-Aufruf diesen Fehler zurück. CloudTrailSuchen Sie nach [DeleteQueue](#) Vorgängen, die vor dem Zeitpunkt des Fehlers aufgetreten sind.

- Probleme mit Berechtigungen: Wenn der anfragende Benutzer oder die Rolle AWS Identity and Access Management (IAM) nicht über die erforderlichen Berechtigungen verfügt, wird möglicherweise die folgende Fehlermeldung angezeigt:

```
The specified queue does not exist or you do not have access to it.
```

Überprüfen Sie die Berechtigungen und führen Sie den API-Aufruf mit den richtigen Berechtigungen durch.

Weitere Informationen zur Behebung des QueueDoesNotExist Fehlers finden Sie unter [Wie behebe ich den QueueDoesNotExist Fehler, wenn ich API-Aufrufe an meine Amazon SQS-Warteschlange](#) tätige? im AWS Knowledge Center-Leitfaden.

InvalidAttributeValue Fehler

Dieser Fehler wird zurückgegeben, wenn die Amazon SQS SQS-Warteschlangenressourcenrichtlinie oder Eigenschaften mit einer falschen Richtlinie oder einem falschen Prinzipal aktualisiert werden.

Mögliche Ursachen und Abhilfemaßnahmen:

- Ungültige Ressourcenrichtlinie: Stellen Sie sicher, dass die Ressourcenrichtlinie alle erforderlichen Felder enthält. Weitere Informationen finden Sie unter [Referenz zu IAM-JSON-Richtlinienelementen](#) und [Validierung von IAM-Richtlinien](#). Sie können den [IAM-Richtliniengenerator](#) auch verwenden, um eine Amazon SQS SQS-Ressourcenrichtlinie zu erstellen und zu testen. Stellen Sie sicher, dass die Richtlinie im JSON-Format vorliegt.
- Ungültiger Prinzipal: Stellen Sie sicher, dass das Principal Element in der Ressourcenrichtlinie vorhanden ist und dass der Wert gültig ist. Wenn Ihr Amazon SQS Principal SQS-Ressourcenrichtlinienelement eine IAM-Entität enthält, stellen Sie sicher, dass die Entität existiert, bevor Sie die Richtlinie verwenden. Amazon SQS validiert die Ressourcenrichtlinie und sucht nach der IAM-Entität. Wenn die IAM-Entität nicht existiert, erhalten Sie eine Fehlermeldung. Verwenden Sie [und](#), um IAM-Entitäten zu bestätigen. [GetRoleGetUser](#) APIs

Weitere Informationen zur Behebung eines InvalidAttributeValue Fehlers finden Sie unter [Wie behebe ich den QueueDoesNotExist Fehler, wenn ich API-Aufrufe an meine Amazon SQS-Warteschlange](#) tätige? im AWS Knowledge Center-Leitfaden.

ReceiptHandle Fehler

Bei einem [DeleteMessage](#) API-Aufruf `InvalidParameterValue` kann der Fehler `ReceiptHandleIsInvalid` oder zurückgegeben werden, wenn die Empfangsnummer falsch oder abgelaufen ist.

- `ReceiptHandleIsInvalid` Fehler: Wenn das Beleg-Handle falsch ist, erhalten Sie eine Fehlermeldung, die dem folgenden Beispiel ähnelt:

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:  
The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- `InvalidParameterValue` Fehler: Wenn das Beleg-Handle abgelaufen ist, erhalten Sie eine Fehlermeldung, die dem folgenden Beispiel ähnelt:

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:  
Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The  
receipt handle has expired.
```

Mögliche Ursachen und Abhilfemaßnahmen:

Das Empfangs-Handle wird für jede empfangene Nachricht erstellt und ist nur für den Zeitraum gültig, in dem die Sichtbarkeit überschritten wird. Wenn das Sichtbarkeits-Timeout abläuft, wird die Nachricht in der Warteschlange für Verbraucher sichtbar. Wenn Sie die Nachricht erneut vom Verbraucher erhalten, erhalten Sie eine neue Empfangsnummer. Um Fehler bei der Verarbeitung falscher oder abgelaufener Belege zu vermeiden, verwenden Sie die richtige Empfangsnummer, um die Nachricht innerhalb des Zeitlimits für die Sichtbarkeit der Amazon SQS SQS-Warteschlange zu löschen.

Weitere Informationen zur Behebung eines `ReceiptHandle` Fehlers finden Sie unter [Wie behebe ich die Fehler "ReceiptHandleIsInvalid" und "InvalidParameterValue", wenn ich den Amazon SQS DeleteMessage SQS-API-Aufruf verwende?](#) im AWS Knowledge Center-Leitfaden.

Behebung von Problemen mit der Amazon SQS DLQ- Warteschlange für unzustellbare Briefe und DLQ Redrive

In den folgenden Themen werden die häufigsten Ursachen für Amazon SQS DLQ- und DLQ Redrive-Probleme sowie deren Behebung behandelt. Weitere Informationen finden Sie unter [Wie behebe ich Probleme mit Amazon SQS DLQ Redrive?](#) im AWS Knowledge Center-Leitfaden.

DLQ-Probleme

Erfahren Sie mehr über häufig auftretende DLQ-Probleme und wie Sie sie lösen können.

Durch das Anzeigen von Nachrichten mithilfe der Konsole werden Nachrichten in eine Warteschlange für unzustellbare Nachrichten verschoben

Amazon SQS zählt das Anzeigen einer Nachricht in der Konsole im Rahmen der zugehörigen Redrive-Richtlinie der Warteschlange. Wenn Sie also eine Nachricht in der Konsole so oft anzeigen, wie es in der Redrive-Richtlinie der entsprechenden Warteschlange angegeben ist, wird die Nachricht in die Warteschlange für unzustellbare Nachrichten der entsprechenden Warteschlange verschoben.

Führen Sie einen der folgenden Schritte aus, um dieses Verhalten anzupassen:

- Erhöhen Sie die Einstellung Maximum Receives für die Redrive-Richtlinie der entsprechenden Warteschlange.
- Rufen Sie Nachrichten der betroffenen Warteschlange nicht in der Konsole auf.

Die Werte für **NumberOfMessagesSent** und **NumberOfMessagesReceived** für eine Warteschlange für unzustellbare Nachrichten stimmen nicht überein

Wenn Sie eine Nachricht manuell an eine Warteschlange für unzustellbare Nachrichten senden, wird sie in der Metrik [NumberOfMessagesSent](#) erfasst. Wenn eine Nachricht jedoch aufgrund eines fehlgeschlagenen Verarbeitungsversuchs an eine Warteschlange für unzustellbare Nachrichten gesendet wird, wird sie von dieser Metrik nicht erfasst. Daher ist es möglich, dass die Werte von **NumberOfMessagesSent** und [NumberOfMessagesReceived](#) unterschiedlich sind.

Ein Redrive für Warteschlangen mit unerlaubten Briefen erstellen und konfigurieren

Um die Warteschlange für unzustellbare Nachrichten erneut zu aktivieren, müssen Sie Amazon SQS die entsprechenden [Berechtigungen](#) für den Empfang von Nachrichten aus der Warteschlange

für unzustellbare Briefe und das Senden von Nachrichten an die Zielwarteschlange einrichten. Wenn Sie nicht über die richtigen Berechtigungen verfügen, kann die Aufgabe zur Neuübertragung von Warteschlangen mit unerlaubten Briefen fehlschlagen. Sie können den Status Ihrer Aufgabe zum erneuten Versand von Nachrichten anzeigen, um die Probleme zu beheben, und es erneut versuchen.

Behandlung von Fehlern bei Nachrichten in Standard- und FIFO-Warteschlangen

[Standardwarteschlangen](#) verarbeiten weiterhin Nachrichten bis zum Ablauf der [Aufbewahrungsfrist](#). Durch diese kontinuierliche Verarbeitung wird die Wahrscheinlichkeit minimiert, dass die Warteschlange durch nicht verbrauchte Nachrichten blockiert wird. Eine große Anzahl von Nachrichten, die der Verbraucher wiederholt nicht löscht, kann die Kosten in die Höhe treiben und die Hardware zusätzlich belasten. Um die Kosten niedrig zu halten, verschieben Sie fehlgeschlagene Nachrichten in die Warteschlange für unzustellbare Nachrichten.

Standardwarteschlangen ermöglichen auch eine hohe Anzahl von Nachrichten während des Fluges. Wenn der Großteil Ihrer Nachrichten nicht verarbeitet werden kann und nicht in eine Warteschleife mit unzustellbaren Briefen geschickt wird, kann sich Ihre Geschwindigkeit bei der Nachrichtenverarbeitung verlangsamen. Um die Effizienz Ihrer Warteschlange aufrechtzuerhalten, stellen Sie sicher, dass Ihre Anwendung die Nachrichtenverarbeitung korrekt handhabt.

[FIFO-Warteschlangen](#) sorgen dafür, dass Nachrichten genau einmal verarbeitet werden, indem diese nacheinander einer Nachrichtengruppe entnommen werden. Daher kann der Verbraucher zwar weiterhin geordnete Nachrichten aus einer anderen Nachrichtengruppe abrufen, die erste Nachrichtengruppe ist jedoch so lange nicht verfügbar, bis die Nachricht, die die Warteschlange blockiert, erfolgreich verarbeitet oder in eine Warteschlange mit unzustellbaren Nachrichten verschoben wurde.

Darüber hinaus ermöglichen FIFO-Warteschlangen eine geringere Anzahl von Nachrichten während der Übertragung. Um zu verhindern, dass Ihre FIFO-Warteschlange durch eine Nachricht blockiert wird, stellen Sie sicher, dass Ihre Anwendung die Nachrichtenverarbeitung korrekt handhabt.

Weitere Informationen erhalten Sie unter [Amazon SQS SQS-Nachrichtenkontingente](#) und [Bewährte Methoden für Amazon SQS](#).

Probleme mit DLQ-Redrive

Erfahren Sie mehr über häufig auftretende Probleme mit DLQ-Redrive und wie Sie diese lösen können.

AccessDenied Problem mit der Genehmigung

Der AccessDenied Fehler tritt auf, wenn das DLQ-Redrive fehlschlägt, weil die AWS Identity and Access Management (IAM-) Entität nicht über die erforderlichen Berechtigungen verfügt.

Beispiel für eine Fehlermeldung:

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

Die folgenden API-Berechtigungen sind erforderlich, um DLQ-Redrive-Anfragen zu stellen:

Um eine Nachrichten-Redrive zu starten:

- Berechtigungen für die Warteschlange „Unzustellbarer Briefe“:
 - `sqs:StartMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `kms:Decrypt`— Wenn entweder die Warteschlange für unzustellbare Nachrichten oder die ursprüngliche Quellwarteschlange verschlüsselt sind.
- Berechtigungen für die Zielwarteschlange:
 - `sqs:SendMessage`
 - `kms:GenerateDataKey`— Wenn die Zielwarteschlange verschlüsselt ist.
 - `kms:Decrypt`— Wenn die Zielwarteschlange verschlüsselt ist.

Um eine Nachricht abzuberechnen, die gerade bearbeitet wird, wiederholen Sie den Vorgang:

- Berechtigungen für die Warteschlange „Unzustellbare Briefe“:
 - `sqs:CancelMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `kms:Decrypt`— Wenn entweder die Warteschlange für unzustellbare Nachrichten oder die ursprüngliche Quellwarteschlange verschlüsselt sind.

So zeigen Sie den Verschiebungsstatus einer Nachricht an:

- Berechtigungen für die Warteschlange „Unzustellbare Briefe“:
 - `sqs:ListMessageMoveTasks`
 - `sqs:GetQueueAttributes`

NonExistentQueue-Fehler

Der `NonExistentQueue` Fehler tritt auf, wenn die Amazon SQS SQS-Quellwarteschlange nicht existiert oder gelöscht wurde. Suchen Sie nach einer vorhandenen Amazon SQS-Warteschlange und wechseln Sie erneut zu einer vorhandenen Amazon SQS-Warteschlange.

Beispiel für eine Fehlermeldung:

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource Fehler

Der `CouldNotDetermineMessageSource` Fehler tritt auf, wenn Sie versuchen, einen DLQ-Redrive mit den folgenden Szenarien zu starten:

- Eine Amazon SQS SQS-Nachricht, die mit [SendMessage](#) API direkt an den DLQ gesendet wird.
- Eine Nachricht vom Thema oder der AWS Lambda Funktion Amazon Simple Notification Service (Amazon SNS) mit konfiguriertem DLQ.

Um diesen Fehler zu beheben, wählen Sie `Redrive to a custom destination`, wenn Sie den Redrive starten. Geben Sie dann den ARN der Amazon SQS SQS-Warteschlange ein, um alle Nachrichten von der DLQ in die Zielwarteschlange zu verschieben.

Beispiel für eine Fehlermeldung:

```
Failed: CouldNotDetermineMessageSource
```

Behebung von Problemen mit der FIFO-Drosselung in Amazon SQS

Standardmäßig unterstützen FIFO-Warteschlangen 300 Transaktionen pro Sekunde und API-Aktion für [SendMessageReceiveMessage](#), und [DeleteMessage](#). Bei Anfragen über 300 TPS wird der `ThrottlingException` Fehler auch dann angezeigt, wenn Nachrichten in der Warteschlange verfügbar sind. Um dies zu beheben, können Sie die folgenden Methoden verwenden:

- [Aktivierung eines hohen Durchsatzes für FIFO-Warteschlangen in Amazon SQS](#).
- Verwenden Sie die Amazon SQS SQS-API-Batch-Aktionen `SendMessageBatch` und `ChangeMessageVisibilityBatch` um das TPS-Limit von bis zu 3.000 Nachrichten pro Sekunde pro API-Aktion zu erhöhen und die Kosten zu senken. `DeleteMessageBatch` Stellen Sie für die `ReceiveMessage` API den `MaxNumberOfMessages` Parameter so ein, dass er bis zu zehn Nachrichten pro Transaktion empfängt. Weitere Informationen finden Sie unter [Amazon-SQS-Stapelaktionen](#).
- Folgen Sie bei FIFO-Warteschlangen mit hohem Durchsatz den Empfehlungen zur [Optimierung der Partitionsnutzung](#). Senden Sie Nachrichten mit derselben Nachrichtengruppe IDs stapelweise. Löschen Sie Nachrichten oder ändern Sie die Timeout-Werte für die Nachrichtensichtbarkeit in Stapeln mit Empfangsnummern aus denselben `ReceiveMessage` API-Anfragen.
- Erhöhen Sie die Anzahl der [MessageGroupId](#) Einzelwerte. Dies ermöglicht eine gleichmäßige Verteilung auf die FIFO-Warteschlangenpartitionen. Weitere Informationen finden Sie unter [Verwenden der Amazon SQS SQS-Nachrichtengruppen-ID](#).

Weitere Informationen finden Sie unter [Warum gibt meine Amazon SQS FIFO-Warteschlange nicht alle Nachrichten oder Nachrichten in anderen Nachrichtengruppen zurück?](#) im AWS Knowledge Center-Leitfaden.

Problembehandlung bei Nachrichten, die bei einem Amazon SQS `ReceiveMessage` SQS-API-Aufruf nicht zurückgegeben wurden

In den folgenden Themen werden die häufigsten Ursachen behandelt, warum eine Amazon SQS SQS-Nachricht möglicherweise nicht an Verbraucher zurückgesendet wird, und wie diese Probleme behoben werden können. Weitere Informationen finden Sie unter [Warum kann ich keine Nachrichten aus meiner Amazon SQS SQS-Warteschlange empfangen?](#) im AWS Knowledge Center-Leitfaden.

Leere Warteschlange

Um festzustellen, ob eine Warteschlange leer ist, rufen Sie die [ReceiveMessage](#) API mithilfe von langen Abfragen auf. Sie können auch die `ApproximateNumberOfMessagesDelayed` CloudWatch Metriken

`ApproximateNumberOfMessagesVisible` `ApproximateNumberOfMessagesNotVisible`, und verwenden. Wenn alle Metrikwerte für mehrere Minuten auf 0 gesetzt sind, wird die Warteschlange als leer betrachtet.

Während des Fluges wurde das Limit erreicht

Wenn Sie Long Polling verwenden und das In-Flight-Limit der Warteschlange (standardmäßig 120000) überschritten wird, gibt Amazon SQS keine Fehlermeldungen zurück, die die Kontingentgrenzen überschreiten.

Verzögerung der Nachricht

Wenn die Amazon SQS SQS-Warteschlange als [Verzögerungswarteschlange](#) konfiguriert ist oder die Nachrichten mit [Nachrichtentimern](#) gesendet wurden, sind die Nachrichten erst sichtbar, wenn die Verzögerungszeit abgelaufen ist. Um zu überprüfen, ob eine Warteschlange als Verzögerungswarteschlange konfiguriert ist, verwenden Sie das [GetQueueAttributes](#) `DelaySeconds` API-Attribut oder verwenden Sie die Warteschlangenkonsolle unter Delivery Delay. Prüfen Sie [ApproximateNumberOfMessagesDelayed](#) CloudWatch anhand der Metrik, ob Nachrichten verzögert sind.

Die Nachricht ist im Flug

Wenn ein anderer Nutzer die Nachricht abgefragt hat, ist die Nachricht während des [Sichtbarkeitszeitraums](#) entweder im Umlauf oder unsichtbar. Bei den zusätzlichen Umfragen wird möglicherweise ein leerer Empfang zurückgegeben. Überprüfen Sie die [ApproximateNumberOfMessagesVisible](#) CloudWatch Metrik, um die Anzahl der Nachrichten zu ermitteln, die für den Empfang verfügbar sind. Wenn bei FIFO-Warteschlangen eine Nachricht mit der Nachrichtengruppen-ID gesendet wird, werden keine Nachrichten mehr zurückgegeben, es sei denn, Sie löschen die Nachricht oder sie wird sichtbar. Das liegt daran, dass die [Reihenfolge der Nachrichten](#) in einer FIFO-Warteschlange auf Nachrichtengruppenebene beibehalten wird.

Abfragemethode

Wenn Sie [Short-Polling](#) verwenden, nimmt ([WaitTimeSeconds](#)ist 0) Amazon SQS eine Stichprobe einer Teilmenge seiner Server ab und gibt nur Nachrichten von diesen Servern zurück. Daher erhalten Sie die Nachrichten möglicherweise nicht, auch wenn sie für den Empfang verfügbar sind. Bei nachfolgenden Umfrageanfragen werden die Nachrichten zurückgegeben.

Wenn Sie [lange Abfragen verwenden, fragt](#) Amazon SQS alle Server ab und sendet eine Antwort, nachdem mindestens eine verfügbare Nachricht erfasst wurde, und zwar bis zur angegebenen Höchstanzahl. Wenn der Wert für zu niedrig `ReceiveMessageWaitTimeSeconds`ist, erhalten Sie möglicherweise nicht alle verfügbaren Nachrichten.

Behebung Amazon SQS SQS-Netzwerkfehlern

Die folgenden Themen behandeln die häufigsten Ursachen für Netzwerkprobleme in Amazon SQS und deren Behebung.

ETIMOUT error

Der ETIMOUT Fehler tritt auf, wenn der Client keine TCP-Verbindung zu einem Amazon SQS SQS-Endpunkt herstellen kann.

Fehlerbehebung

- Überprüfen Sie die Netzwerkverbindung

Testen Sie Ihre Netzwerkverbindung zu Amazon SQS, indem Sie Befehle wie `telnet` ausführen.

Example: `telnet sqs.us-east-1.amazonaws.com 443`

- Überprüfen Sie die Netzwerkeinstellungen
 - Stellen Sie sicher, dass Ihre lokalen Firewallregeln, Routen und Zugriffskontrolllisten (ACLs) Datenverkehr auf dem von Ihnen verwendeten Port zulassen.
 - Die Regeln der Sicherheitsgruppe für ausgehenden Datenverkehr (ausgehenden Verkehr) müssen den Datenverkehr zum Port 80 oder 443 zulassen.
 - Die Netzwerk-ACL-Regeln für ausgehenden Verkehr (ausgehenden Verkehr) müssen den Datenverkehr zum TCP-Port 80 oder 443 zulassen.
 - Die Netzwerk-ACL-Regeln für eingehenden Datenverkehr (Eingang) müssen den Datenverkehr auf den TCP-Ports 1024-65535 zulassen.

- Amazon Elastic Compute Cloud (Amazon EC2) -Instances, die eine Verbindung zum öffentlichen Internet herstellen, müssen über eine [Internetverbindung](#) verfügen.
- Endpunkte der Amazon Virtual Private Cloud (Amazon VPC)

Wenn Sie über einen Amazon VPC-Endpoint auf Amazon SQS zugreifen, muss die Sicherheitsgruppe der Endgeräte eingehenden Datenverkehr zur Sicherheitsgruppe des Clients auf Port 443 zulassen. Die Netzwerk-ACL, die dem Subnetz des VPC-Endpunkts zugeordnet ist, muss diese Konfiguration haben:

- Die Netzwerk-ACL-Regeln für ausgehenden Datenverkehr (Ausgang) müssen den Datenverkehr auf den TCP-Ports 1024-65535 (kurzlebige Ports) zulassen.
- Die Netzwerk-ACL-Regeln für eingehenden Datenverkehr (Eingang) müssen Datenverkehr auf Port 443 zulassen.

Außerdem muss die Amazon SQS VPC-Endpoint AWS Identity and Access Management (IAM) - Richtlinie den Zugriff zulassen. Die folgende Beispiel-VPC-Endpunktrichtlinie legt fest, dass der IAM-Benutzer *MyUser* Nachrichten an die Amazon SQS SQS-Warteschlange senden darf. *MyQueue* Anderen Aktionen, IAM-Benutzern und Amazon SQS-Ressourcen wird der Zugriff über den VPC-Endpoint verweigert.

```
{
  "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

Der UnknownHostException Fehler tritt auf, wenn die Host-IP-Adresse nicht ermittelt werden konnte.

Fehlerbehebung

Verwenden Sie das nslookup Hilfsprogramm, um die dem Hostnamen zugeordnete IP-Adresse zurückzugeben:

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI oder SDK für ältere Python-Endpunkte:

```
nslookup <region>.queue.amazonaws.com
```

Wenn Sie eine erfolglose Ausgabe erhalten haben, folgen Sie den Anweisungen [unter Wie funktioniert DNS und wie behebe ich teilweise oder zeitweise auftretende DNS-Fehler?](#) im AWS Knowledge Center-Leitfaden.

Wenn Sie eine gültige Ausgabe erhalten haben, handelt es sich wahrscheinlich um ein Problem auf Anwendungsebene. Probieren Sie die folgenden Methoden aus, um Probleme auf Anwendungsebene zu beheben:

- Starten Sie Ihre Anwendung neu.
- Vergewissern Sie sich, dass Ihre Java-Anwendung keinen fehlerhaften DNS-Cache hat. Wenn möglich, konfigurieren Sie Ihre Anwendung so, dass sie der DNS-TTL entspricht. Weitere Informationen finden Sie unter [JVM-TTL für DNS-Namenssuchen einrichten](#).

Weitere Informationen zur Behebung von Netzwerkfehlern finden Sie unter [Wie behebe ich die Verbindungsfehler „ETIMEDOUT“ und „UnknownHostException“ von Amazon SQS?](#) im AWS Knowledge Center-Leitfaden.

Problembehandlung bei Amazon Simple Queue Service-Warteschlangen mit AWS X-Ray

AWS X-Ray sammelt Daten über Anfragen, die Ihre Anwendung bearbeitet, und ermöglicht es Ihnen, Daten anzuzeigen und zu filtern, um potenzielle Probleme und Optimierungsmöglichkeiten zu identifizieren. Für jede verfolgte Anfrage an Ihre Anwendung können Sie detaillierte Informationen zu der Anfrage, der Antwort und den Aufrufen einsehen, die Ihre Anwendung an nachgelagerte AWS Ressourcen, Microservices, Datenbanken und das HTTP-Web sendet. APIs

Um AWS X-Ray Trace-Header über Amazon SQS zu senden, können Sie einen der folgenden Schritte ausführen:

- Verwenden Sie X-Amzn-Trace-Id-[Nachverfolgungs-Header](#).
- Verwenden Sie das AWSTraceHeader-[Nachrichtensystemattribut](#).

Um Daten über Fehler und Latenz zu erfassen, müssen Sie den [AmazonSQS](#)-Client mit dem [AWS -X-Ray-SDK](#) instrumentieren.

Sie können die AWS X-Ray Konsole verwenden, um die Verbindungsübersicht zwischen Amazon SQS und anderen Services anzuzeigen, die Ihre Anwendung verwendet. Sie können mithilfe der Konsole auch Metriken, wie durchschnittliche Latenz- und Ausfallraten, anzuzeigen. Weitere Informationen finden Sie unter [Amazon SQS und AWS X-Ray](#) im AWS X-Ray -Entwicklerhandbuch.

Sicherheit in Amazon SQS

Dieser Abschnitt enthält Informationen zur Sicherheit, Authentifizierung und Zugriffskontrolle in Amazon SQS sowie zur Sprache der Zugriffsrichtlinie von Amazon SQS.

Themen

- [Datenschutz in Amazon SQS](#)
- [Identity and Access Management in Amazon SQS](#)
- [Protokollierung und Überwachung in Amazon SQS](#)
- [Compliance-Validierung für Amazon SQS](#)
- [Ausfallsicherheit bei Amazon SQS](#)
- [Infrastruktursicherheit in Amazon SQS](#)
- [Bewährte Methoden für die Sicherheit in Amazon SQS](#)

Datenschutz in Amazon SQS

Das AWS [Modell](#) der gilt für den Datenschutz in Amazon Simple Queue Service. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter [Häufig gestellte Fragen zum Datenschutz](#) . Informationen zum Datenschutz in Europa finden Sie im [General Data Protection Regulation \(GDPR\) Center](#).

Aus Datenschutzgründen empfehlen wir, dass Sie Ihre AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.

- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter [Arbeiten mit CloudTrail Pfaden](#) im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-3](#).

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit Amazon SQS oder anderen Geräten arbeiten und die Konsole AWS CLI, API oder AWS SDKs AWS-Services verwenden. Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Die folgenden Abschnitte enthalten Informationen zum Datenschutz in Amazon SQS.

Datenverschlüsselung in Amazon SQS

Datenschutz bezieht sich auf Daten in Übertragung (wenn sie zu Amazon SQS oder von diesem geschickt werden), ebenso wie auf ruhende Daten (die in Amazon-SQS-Rechenzentren auf Datenträgern gespeichert sind). Sie können Daten mithilfe von Secure Sockets Layer (SSL) oder einer clientseitigen Verschlüsselung während der Übertragung schützen. Standardmäßig speichert Amazon SQS Nachrichten und Dateien mit Festplattenverschlüsselung. Sie können Daten im Ruhezustand schützen, indem Sie Amazon SQS auffordern, Ihre Nachrichten zu verschlüsseln, bevor Sie sie im verschlüsselten Dateisystem in seinen Rechenzentren speichern. Amazon SQS empfiehlt die Verwendung von SSE für eine optimierte Datenverschlüsselung.

Themen

- [Verschlüsselung im Ruhezustand in Amazon SQS](#)

- [Amazon SQS Schlüsselmanagement](#)

Verschlüsselung im Ruhezustand in Amazon SQS

Server-side Mit Verschlüsselung (SSE) können Sie vertrauliche Daten in verschlüsselten Warteschlangen übertragen. SSE schützt den Inhalt von Nachrichten in Warteschlangen mithilfe von SQS-managed Verschlüsselungsschlüsseln (SSE-SQS) oder Schlüsseln, die in der AWS Key Management Service (SSE-KMS) verwaltet werden. Informationen zur Verwaltung von SSE mithilfe von finden Sie unter: AWS-Managementkonsole

- [Konfiguration SSE-SQS für eine Warteschlange \(Konsole\)](#)
- [Konfiguration SSE-KMS für eine Warteschlange \(Konsole\)](#)

Informationen zur Verwaltung von SSE mithilfe der [GetQueueAttributes](#) Aktionen AWS SDK für Java (und [CreateQueueSetQueueAttributes](#), und) finden Sie in den folgenden Beispielen:

- [Serverseitige Verschlüsselung mit Amazon SQS SQS-Warteschlangen verwenden](#)
- [Konfiguration von KMS-Berechtigungen für AWS-Services](#)

SSE verschlüsselt Nachrichten, sobald sie bei Amazon SQS eingehen. Die Nachrichten werden verschlüsselt gespeichert. Amazon SQS entschlüsselt Nachrichten nur, wenn sie an einen autorisierten Konsumenten gesendet werden.

Important

Alle Anfragen zu Warteschlangen mit aktiviertem SSE müssen HTTPS und [Signature Version 4](#) verwenden.

Eine [verschlüsselte Warteschlange](#), die den Standardschlüssel (AWS verwalteter KMS-Schlüssel für Amazon SQS) verwendet, kann keine Lambda-Funktion in einer anderen aufrufen. AWS-Konto

Einige Funktionen von AWS Diensten, die mithilfe der AWS -Security-Token-Service [AssumeRole](#) Aktion Benachrichtigungen an Amazon SQS senden können, sind mit SSE kompatibel, funktionieren jedoch nur mit Standardwarteschlangen:

- [Auto Scaling Lifecycle Hooks](#)
- [AWS Lambda Dead-Letter](#) -Warteschlangen

Weitere Informationen zur Kompatibilität anderer Services mit verschlüsselten Warteschlangen finden Sie unter [Konfigurieren Sie KMS-Berechtigungen für AWS service](#) und in Ihrer Service-Dokumentation.

AWS KMS kombiniert sichere, hochverfügbare Hardware und Software zu einem für die Cloud skalierten Schlüsselverwaltungssystem. Wenn Sie Amazon SQS mit verwenden AWS KMS, werden die [Datenschlüssel](#), die Ihre Nachrichtendaten verschlüsseln, ebenfalls verschlüsselt und zusammen mit den Daten gespeichert, die sie schützen.

Vorteile von AWS KMS:

- Sie können [AWS KMS keys](#) selbst erstellen und verwalten.
- Sie können auch den AWS verwalteten KMS-Schlüssel für Amazon SQS verwenden, der für jedes Konto und jede Region einzigartig ist.
- Die AWS KMS Sicherheitsstandards können Ihnen dabei helfen, die Compliance-Anforderungen im Zusammenhang mit der Verschlüsselung zu erfüllen.

Weitere Informationen finden Sie unter [Was ist AWS Key Management Service?](#) im AWS Key Management Service -Entwicklerhandbuch.

Verschlüsselungsumfang

SSE verschlüsselt den Nachrichtentext in einer Amazon-SQS-Warteschlange.

Folgendes wird von SSE nicht verschlüsselt:

- Warteschlangen-Metadaten (Name und Attribute der Warteschlange)
- Metadaten der Nachrichten (ID, Zeitstempel und Attribute)
- Per-queue Metriken

Durch die Verschlüsselung sind die Nachrichteninhalte für nicht autorisierte oder anonyme Benutzer nicht zugänglich. Wenn SSE aktiviert ist, werden anonyme SendMessage- und ReceiveMessage-Anfragen an die verschlüsselte Warteschlange abgewiesen. Die bewährten Sicherheitsmethoden von Amazon SQS raten davon ab, anonyme Anfragen zu verwenden. Wenn Sie anonyme Anfragen an

eine Amazon-SQS-Warteschlange senden möchten, stellen Sie sicher, dass Sie SSE deaktivieren. Dies wirkt sich nicht auf die normale Funktion von Amazon SQS aus:

- Eine Nachricht ist nur dann verschlüsselt, wenn sie gesendet wurde, nachdem die Verschlüsselung einer Warteschlange aktiviert wurde. Amazon SQS verschlüsselt keine Nachrichten, die sich bereits in der Queue befinden.
- Verschlüsselte Nachrichten bleiben auch dann verschlüsselt, wenn die Verschlüsselung der Warteschlange deaktiviert wird.

Das Verschieben einer Nachricht in eine [Warteschlange für unzustellbare Nachrichten](#) wirkt sich nicht auf ihre Verschlüsselung aus:

- Wenn Amazon SQS eine Nachricht aus einer verschlüsselten Quellwarteschlange in eine unverschlüsselte Warteschlange für unzustellbare Nachrichten verschiebt, bleibt die Nachricht verschlüsselt.
- Wenn Amazon SQS eine Nachricht aus einer unverschlüsselten Quellwarteschlange in eine verschlüsselte Warteschlange für unzustellbare Nachrichten verschiebt, bleibt die Nachricht unverschlüsselt.

Wichtige Begriffe

Die folgenden wichtigen Begriffe vermitteln Ihnen ein besseres Verständnis für die Funktionalität von SSE. Detaillierte Beschreibungen finden Sie in der [Amazon-Simple-Queue-Service-API-Referenz](#).

Datenschlüssel

Der Schlüssel (DEK), der dafür zuständig ist, den Inhalt von Amazon-SQS-Nachrichten zu verschlüsseln.

Weitere Informationen finden Sie unter [Datenschlüssel](#) im AWS Key Management Service - Entwicklerhandbuch im AWS Encryption SDK -Entwicklerhandbuch.

Data key reuse period (Wiederverwendungszeitraum für den Datenschlüssel)

Die Zeitspanne in Sekunden, für die Amazon SQS einen Datenschlüssel wiederverwenden kann, um Nachrichten zu verschlüsseln oder zu entschlüsseln, bevor er erneut anruft. AWS KMS Eine Ganzzahl stellt Sekunden dar, und zwar zwischen 60 Sekunden (1 Minute) und 86 400 Sekunden (24 Stunden). Der Standardwert ist 300 (5 Minuten). Weitere Informationen finden Sie unter [Grundlegendes zum Wiederverwendungszeitraum für den Datenschlüssel](#).

 Note

In dem unwahrscheinlichen Fall, dass keine Verbindung hergestellt werden kann AWS KMS, verwendet Amazon SQS weiterhin den zwischengespeicherten Datenschlüssel, bis eine Verbindung wiederhergestellt ist.

KMS-Schlüssel-ID

Der Alias, Alias-ARN, die Schlüssel-ID oder der Schlüssel-ARN eines AWS verwalteten KMS-Schlüssels oder eines benutzerdefinierten KMS-Schlüssels — in Ihrem Konto oder in einem anderen Konto. Während der Alias des AWS verwalteten KMS-Schlüssels für Amazon SQS immer ist `alias/aws/sqs`, kann der Alias eines benutzerdefinierten KMS-Schlüssels beispielsweise sein `alias/MyAlias`. Sie können diese KMS-Schlüssel zum Schutz der Nachrichten in Amazon-SQS-Warteschlangen verwenden.

 Note

Beachten Sie Folgendes:

- Wenn Sie keinen benutzerdefinierten KMS-Schlüssel angeben, verwendet Amazon SQS den AWS verwalteten KMS-Schlüssel für Amazon SQS.
- Wenn Sie AWS-Managementkonsole zum ersten Mal den AWS verwalteten KMS-Schlüssel für Amazon SQS für eine Warteschlange angeben, AWS KMS wird der AWS verwaltete KMS-Schlüssel für Amazon SQS erstellt.
- Alternativ können Sie, wenn Sie die `SendMessageBatch` Aktion `SendMessage` oder zum ersten Mal in einer Warteschlange mit aktivierter SSE verwenden, der AWS verwaltete KMS-Schlüssel für Amazon SQS AWS KMS erstellt.

Sie können KMS-Schlüssel erstellen, die Richtlinien definieren, die steuern, wie KMS-Schlüssel verwendet werden können, und die Verwendung von KMS-Schlüsseln mithilfe des Abschnitts „Vom Kunden verwaltete Schlüssel“ der AWS KMS Konsole oder der [CreateKey](#) AWS KMS Aktion überprüfen. Weitere Informationen zum Erstellen von [KMS-Schlüsseln](#) finden Sie unter [Erstellen von Schlüsseln](#) im AWS Key Management Service -Entwicklerhandbuch. Weitere Beispiele für KMS-Schlüsselkennungen finden Sie [KeyId](#) in der AWS Key Management Service API-Referenz. Weitere Informationen zum Suchen von KMS-Schlüssel-IDs finden Sie unter [Suchen der Schlüssel-ID und des ARNs](#) im AWS Key Management Service -Entwicklerhandbuch.

 Important

Für die Nutzung AWS KMS fallen zusätzliche Gebühren an. Weitere Informationen finden Sie unter [Schätzung AWS KMS Kosten](#) und [Preise zu AWS Key Management Service](#).

Envelope-Verschlüsselung

Die Sicherheit Ihrer verschlüsselten Daten hängt teilweise vom Schutz des Datenschlüssels ab, der sie entschlüsseln kann. Amazon SQS verwendet den KMS-Schlüssel, um den Datenschlüssel zu verschlüsseln. Anschließend wird der verschlüsselte Datenschlüssel zusammen mit der verschlüsselten Nachricht gespeichert. Diese Vorgehensweise der Verwendung eines KMS-Schlüssels zum Verschlüsseln von Datenschlüsseln wird als Umschlagverschlüsselung bezeichnet.

Weitere Informationen zur [Envelope-Verschlüsselung](#) finden im AWS Encryption SDK Entwicklerhandbuch.

Amazon SQS Schlüsselverwaltung

Amazon SQS lässt sich in das AWS Key Management Service (KMS) integrieren, um [KMS-Schlüssel](#) für serverseitige Verschlüsselung (SSE) zu verwalten. SSE-Informationen und Definitionen zur Schlüsselverwaltung finden Sie unter [Verschlüsselung im Ruhezustand in Amazon SQS](#). Amazon SQS verwendet KMS-Schlüssel, um die Datenschlüssel zu validieren und zu sichern, mit denen die Nachrichten ver- und entschlüsselt werden. Die folgenden Abschnitte enthalten Informationen über das Arbeiten mit KMS-Schlüsseln und Datenschlüsseln im Amazon-SQS-Service.

Konfigurieren AWS KMS Berechtigungen

Jeder KMS-Schlüssel muss über eine Schlüsselrichtlinie verfügen. Beachten Sie, dass Sie die Schlüsselrichtlinie eines AWS verwalteten KMS-Schlüssels für Amazon SQS nicht ändern können. Die Richtlinie für diesen KMS-Schlüssel beinhaltet Berechtigungen für alle Prinzipale im Konto (die zur Verwendung von Amazon SQS berechtigt sind), verschlüsselte Warteschlangen zu verwenden.

Amazon SQS unterscheidet Anrufer anhand ihrer AWS Anmeldeinformationen, unabhängig davon, ob sie unterschiedliche AWS Konten, IAM-Benutzer oder IAM-Rollen verwenden. Darüber hinaus wird dieselbe IAM-Rolle mit unterschiedlichen Bereichsrichtlinien als separate Anforderer behandelt. Bei der Verwendung von IAM-Rollensitzungen werden jedoch keine unterschiedlichen Anforderer erstellt, wenn nur die Variation unter `RoleSessionName` Beibehaltung der gleichen IAM-Rollen-

und Bereichsrichtlinien vorgenommen wird. Vermeiden Sie es daher, sich bei der Angabe AWS KMS wichtiger Richtlinienprinzipale ausschließlich auf `RoleSessionName` Unterschiede zu verlassen, da diese Sitzungen als derselbe Anforderer behandelt werden.

Alternativ können Sie die erforderlichen Berechtigungen in einer IAM-Richtlinie angeben, die den Prinzipalen zugewiesen ist, die verschlüsselte Nachrichten produzieren und konsumieren. Weitere Informationen finden Sie unter [Verwenden von IAM-Richtlinien mit AWS KMS](#) im AWS Key Management Service -Entwicklerhandbuch.

Note

Sie können zwar globale Berechtigungen für das Senden an und Empfangen von Amazon SQS konfigurieren, AWS KMS erfordert jedoch die ausdrückliche Benennung des vollständigen ARN der KMS-Schlüssel in bestimmten Regionen im `Resource` Abschnitt einer IAM-Richtlinie.

Konfigurieren Sie KMS-Berechtigungen für AWS service

Verschiedene AWS Dienste dienen als Ereignisquellen, die Ereignisse an Amazon SQS SQS-Warteschlangen senden können. Damit diese Ereignisquellen mit verschlüsselten Warteschlangen arbeiten können, müssen Sie einen vom Kunden verwalteten KMS-Schlüssel erstellen und der Schlüsselrichtlinie Berechtigungen hinzufügen, damit der Service die erforderlichen AWS KMS API-Methoden verwenden kann. Führen Sie zum Konfigurieren der Berechtigungen die folgenden Schritte durch.

Warning

Wenn Sie den KMS-Schlüssel für die Verschlüsselung Ihrer Amazon SQS SQS-Nachrichten ändern, beachten Sie, dass bestehende Nachrichten, die mit dem alten KMS-Schlüssel verschlüsselt wurden, mit diesem Schlüssel verschlüsselt bleiben. Um diese Nachrichten zu entschlüsseln, müssen Sie den alten KMS-Schlüssel beibehalten und sicherstellen, dass seine Schlüsselrichtlinie Amazon SQS die Berechtigungen für `kms:Decrypt` und `kms:GenerateDataKey` gewährt. Stellen Sie nach der Aktualisierung auf einen neuen KMS-Schlüssel zur Verschlüsselung neuer Nachrichten sicher, dass alle vorhandenen Nachrichten, die mit dem alten KMS-Schlüssel verschlüsselt wurden, verarbeitet und aus der Warteschlange entfernt werden, bevor Sie den alten KMS-Schlüssel löschen oder deaktivieren.

1. Erstellen Sie einen vom Kunden verwalteten KMS-Schlüssel. Weitere Informationen finden Sie unter [Erstellen von Schlüsseln](#) im AWS Key Management Service -Entwicklerhandbuch.
2. Damit die AWS Dienstereignisquelle die Methoden `kms:Decrypt` und `kms:GenerateDataKey` API verwenden kann, fügen Sie der KMS-Schlüsselrichtlinie die folgende Anweisung hinzu.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "service.amazonaws.com"
    },
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": "*"
  }]
}
```

Ersetzen Sie „service“ im obigen Beispiel durch den Servicenamen der Ereignisquelle. Zu den Ereignisquellen gehören die folgenden Services.

Ereignisquelle	Service-Name
CloudWatch Amazon-Veranstaltungen	events.amazonaws.com
Amazon-S3-Ereignis-Benachrichtigungen	s3.amazonaws.com
Amazon-SNS-Themenabonnements	sns.amazonaws.com

3. [Konfigurieren Sie eine vorhandene SSE-Warteschlange](#) mit dem ARN Ihres KMS-Schlüssels.
4. Stellen Sie der verschlüsselten Warteschlange den ARN der Ereignisquelle zur Verfügung.

Konfiguration AWS KMS Genehmigungen für Produzenten

Wenn der [Zeitraum für die Wiederverwendung des Datenschlüssels](#) abgelaufen ist, löst der nächste Aufruf des Produzenten von `SendMessage` oder `SendMessageBatch` auch Aufrufe von `kms:Decrypt` und `kms:GenerateDataKey` aus. Der Aufruf von `kms:Decrypt` dient dazu, die Integrität des neuen Datenschlüssels vor dessen Verwendung zu überprüfen. Daher muss der Produzent über die Berechtigungen `kms:Decrypt` und `kms:GenerateDataKey` für den KMS-Schlüssel verfügen.

Fügen Sie der IAM-Richtlinie des Produzenten die folgende Anweisung hinzu. Denken Sie daran, die richtigen ARN-Werte für die Schlüsselressource und die Warteschlangenressource zu verwenden.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:us-east-2:123456789012:key/111112222233333"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage"
      ],
      "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
    }
  ]
}
```

Konfiguration AWS KMS Genehmigungen für Verbraucher

Wenn der Zeitraum für die Wiederverwendung des Datenschlüssels abläuft, löst der nächste Aufruf des Konsumenten von `ReceiveMessage` ebenfalls einen Aufruf von `kms:Decrypt` aus, um die Integrität des neuen Datenschlüssels vor dessen Verwendung zu überprüfen. Deshalb muss der

Konsument über die Berechtigung `kms:Decrypt` für alle KMS-Schlüssel verfügen, die für die Verschlüsselung von Nachrichten in der angegebenen Warteschlange verwendet werden. Wenn es sich bei der Warteschlange um eine [Warteschlange für unzustellbare Nachrichten](#) handelt, muss der Konsument zusätzlich die Berechtigung `kms:Decrypt` für jeden KMS-Schlüssel haben, mit dem die Nachrichten in der Quellwarteschlange verschlüsselt werden. Fügen Sie der IAM-Richtlinie des Konsumenten die folgende Anweisung hinzu. Denken Sie daran, die richtigen ARN-Werte für die Schlüsselressource und die Warteschlangenressource zu verwenden.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-2:123456789012:key/11111222223333"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
    }
  ]
}
```

Konfiguration AWS KMS Berechtigungen mit verwirrtem stellvertretendem Schutz

Wenn der Prinzipal in einer Schlüsselrichtlinie ein [AWS -Service-Prinzipal](#) ist, können Sie die globalen Zustandsschlüssel [aws:SourceArn](#) oder [aws:SourceAccount](#) zum Schutz vor dem [Confused Deputy Scenario](#) verwenden. Um diese globalen Bedingungsschlüssel zu verwenden, legen Sie den Wert auf den Amazon-Ressourcennamen (ARN) oder das Konto der Ressource fest, die verschlüsselt wird. Wenn Sie den ARN der Ressource nicht kennen, verwenden Sie stattdessen `aws:SourceAccount`.

In dieser KMS-Schlüsselrichtlinie darf eine bestimmte Ressource aus einem Service, der dem Konto 111122223333 gehört, KMS für die Aktionen Decrypt und GenerateDataKey Aktionen aufrufen, die während der SSE-Nutzung von Amazon SQS auftreten.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sqs.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": [
            "arn:aws:sqs:us-west-1:111122223333:resource"
          ]
        }
      }
    }
  ]
}
```

Bei Verwendung von SSE-fähigen Amazon-SQS-Warteschlangen unterstützen die folgenden Services `aws:SourceArn`:

- Amazon SNS
- Amazon S3
- CloudWatch Events
- AWS Lambda
- CodeBuild
- Kundenprofile Connect

- AWS Auto Scaling
- Amazon Chime

Grundlegendes zum Wiederverwendungszeitraum für den Datenschlüssel

Der [Zeitraum für die Wiederverwendung des Datenschlüssels](#) definiert die maximale Dauer für Amazon SQS zur Wiederverwendung desselben Datenschlüssels. Endet der Zeitraum der Datenschlüssel-Wiederverwendung, generiert Amazon SQS einen neuen Datenschlüssel. Beachten Sie die folgenden Richtlinien zum Wiederverwendungszeitraum.

- Ein kürzerer Wiederverwendungszeitraum bietet mehr Sicherheit, führt jedoch zu mehr Anrufen AWS KMS, wodurch Gebühren anfallen können, die über das kostenlose Kontingent hinausgehen.
- Obwohl der Datenschlüssel für die Verschlüsselung und Entschlüsselung separat zwischengespeichert wird, gilt der Wiederverwendungszeitraum für beide Kopien des Datenschlüssels.
- Wenn der Zeitraum für die Wiederverwendung von Datenschlüsseln endet, löst der nächste Aufruf von `SendMessage` oder `SendMessageBatch` in der Regel einen Aufruf der AWS KMS `GenerateDataKey` Methode aus, um einen neuen Datenschlüssel abzurufen. Außerdem lösen die nächsten Aufrufe jeweils einen Aufruf von `Decrypt` um die Integrität des Datenschlüssels zu überprüfen, bevor er verwendet wird. `SendMessage` `ReceiveMessage`
- [Prinzipale](#) (AWS-Konten oder Benutzer) verwenden keine gemeinsamen Datenschlüssel (Nachrichten, die von eindeutigen Prinzipalen gesendet werden, erhalten immer eindeutige Datenschlüssel). Daher entspricht die Anzahl der Anrufe an AWS KMS ein Vielfaches der Anzahl der eindeutigen Prinzipale, die während der Wiederverwendung von Datenschlüsseln verwendet wurden.

Schätzung AWS KMS Kosten

Um Kosten vorherzusagen und Ihre AWS Rechnung besser zu verstehen, möchten Sie vielleicht wissen, wie oft Amazon SQS Ihren KMS-Schlüssel verwendet.

Note

Mit der nachstehenden Formel erhalten Sie eine gute Vorstellung davon, welche Kosten auf Sie zukommen. Allerdings können die tatsächlichen Kosten aufgrund der verteilten Struktur von Amazon SQS höher liegen.

Zur Berechnung der Anzahl der API-Anfragen (R) pro Warteschlange verwenden Sie folgende Formel:

$$R = (B / D) * (2 * P + C)$$

B ist der Abrechnungszeitraum (in Sekunden).

D ist der [Zeitraum für die Wiederverwendung des Datenschlüssels](#) (in Sekunden).

P ist die Anzahl der produzierenden [Prinzipale](#), die Nachrichten an die Amazon-SQS-Warteschlange senden.

C ist die Anzahl der konsumierenden Prinzipale, die Nachrichten aus der Amazon-SQS-Warteschlange erhalten.

 Important

Für produzierende Prinzipale entstehen in der Regel doppelt so hohe Kosten wie für konsumierende Prinzipale. Weitere Informationen finden Sie unter [Grundlegendes zum Wiederverwendungszeitraum für den Datenschlüssel](#).

Die Kosten erhöhen sich, wenn der Produzent und der Verbraucher unterschiedliche - Benutzer haben.

Es folgen Beispielberechnungen: Preisinformationen finden Sie unter [AWS Key Management Service -Preise](#).

Beispiel 1: Berechnung der Anzahl AWS KMS API-Aufrufe für 2 Principals und 1 Warteschlange

In diesem Beispiel wird Folgendes angenommen:

- Der Abrechnungszeitraum ist 1. bis 31. Januar (2 678 400 Sekunden).
- Der Wiederverwendungszeitraum für den Datenschlüssel ist auf 5 Minuten (300 Sekunden) eingestellt.
- Es ist 1 Warteschlange vorhanden.
- Es gibt 1 produzierenden und 1 konsumierenden Prinzipal.

$$(2,678,400 / 300) * (2 * 1 + 1) = 26,784$$

Beispiel 2: Berechnung der Anzahl von AWS KMS API-Aufrufe für mehrere Produzenten und Verbraucher und 2 Warteschlangen

In diesem Beispiel wird Folgendes angenommen:

- Der Abrechnungszeitraum ist 1. bis 28. Februar (2 419 200 Sekunden).
- Der Wiederverwendungszeitraum für den Datenschlüssel ist auf 24 Stunden (86 400 Sekunden) eingestellt.
- Es gibt 2 Warteschlangen.
- Die erste Warteschlange hat 3 produzierende Prinzipale und 1 konsumierenden Prinzipal.
- Die zweite Warteschlange hat 5 produzierende und 2 konsumierende Prinzipale.

$$(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532$$

AWS KMS Fehler

Wenn Sie mit Amazon SQS und arbeiten AWS KMS, können Fehler auftreten. In den folgenden Referenzen werden die Fehler und möglichen Lösungen zur Fehlerbehebung beschrieben.

- [Häufige AWS KMS -Fehler](#)
- [AWS KMS -Entschlüsselungsfehler](#)
- [AWS KMS GenerateDataKey Fehler](#)

Datenschutz im Netzwerkverkehr in Amazon SQS

Ein Amazon Virtual Private Cloud (Amazon VPC)-Endpunkt für Amazon SQS ist eine logische Einheit innerhalb einer VPC, die nur Konnektivität mit Amazon SQS ermöglicht. Die VPC leitet Anforderungen an Amazon SQS weiter und Antworten an die VPC zurück. In den folgenden Abschnitten finden Sie Informationen zum Arbeiten mit VPC-Endpunkten und zum Erstellen von VPC-Endpunktrichtlinien.

Endpunkte von Amazon Virtual Private Cloud für Amazon SQS

Wenn Sie Amazon VPC zum Hosten Ihrer AWS Ressourcen verwenden, können Sie eine Verbindung zwischen Ihrer VPC und Amazon SQS herstellen. Sie können diese Verbindung zum Senden von Nachrichten an Ihre Amazon-SQS-Warteschlangen ganz ohne das öffentliche Internet verwenden.

Mit Amazon VPC können Sie AWS Ressourcen in einem benutzerdefinierten virtuellen Netzwerk starten. Mit einer VPC können Sie Netzwerkeinstellungen, wie IP-Adressbereich, Subnetze, Routing-Tabellen und Netzwerk-Gateways, steuern. Weitere Informationen zu VPCs finden Sie im [Amazon-VPC-Benutzerhandbuch](#).

Um Ihre VPC mit Amazon SQS zu verbinden, müssen Sie zunächst einen Schnittstellen-VPC-Endpunkt definieren, der eine Verbindung zwischen Ihrer VPC mit anderen AWS -Services ermöglicht. Der Endpunkt bietet zuverlässige, skalierbare Konnektivität zu Amazon SQS, ohne dass ein Internet-Gateway, eine Network Address Translation (NAT)-Instance oder eine VPN-Verbindung erforderlich ist. Weitere Informationen finden Sie unter [Tutorial: Senden einer Nachricht an eine Amazon-SQS-Warteschlange aus Amazon Virtual Private Cloud](#) und [Beispiel 5: Verweigerung des Zugriffs, wenn dieser nicht von einem VPC-Endpunkt aus erfolgt](#) in diesem Handbuch und unter [Interface-VPC-Endpunkte \(AWS PrivateLink\)](#) im Amazon-VPC-Benutzerhandbuch.

Important

- Sie können Amazon Virtual Private Cloud nur mit HTTPS-Amazon-SQS-Endpunkten verwenden.
- Wenn Sie Amazon SQS für das Senden von Nachrichten von Amazon VPC konfigurieren, müssen Sie privates DNS aktivieren und Endpunkte im Format `sqs.us-east-2.amazonaws.com` oder `sqs.us-east-2.api.aws` für den Dual-Stack-Endpunkt angeben.
- Amazon SQS unterstützt auch FIPS-Endpunkte PrivateLink mithilfe des `com.amazonaws.region.sqs-fips` Endpoint Service. Sie können im folgenden Format eine Verbindung zu FIPS-Endpunkten herstellen. `sqs-fips.region.amazonaws.com`
- Bei Verwendung des Dual-Stack-Endpunkts in Amazon Virtual Private Cloud werden Anfragen über IPv4 und IPv6 gesendet.
- Ein privates DNS unterstützt keine Legacy-Endpunkte wie z. B. `queue.amazonaws.com` oder `us-east-2.queue.amazonaws.com`.

Erstellen einer Amazon-VPC-Endpunkt-Richtlinie für Amazon SQS

Sie können eine Richtlinie für Amazon-VPC-Endpunkte für Amazon SQS erstellen, in der Sie Folgendes angeben:

- Prinzipal, der die Aktionen ausführen kann.

- Aktionen, die ausgeführt werden können
- Die Ressourcen, für die Aktionen ausgeführt werden können.

Weitere Informationen finden Sie unter [Steuerung des Zugriffs auf Services mit VPC-Endpunkten](#) im Amazon VPC-Benutzerhandbuch.

Im folgenden Beispiel für eine VPC-Endpunkt-Richtlinie wird angegeben, dass der MyUser-Benutzer Nachrichten an die Amazon-SQS-Warteschlange MyQueue senden darf.

```
{
  "Statement": [{
    "Action": ["sqs:SendMessage"],
    "Effect": "Allow",
    "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

Folgendes wird abgelehnt:

- Andere Amazon-SQS-API-Aktionen, z. B. `sqs:CreateQueue` und `sqs>DeleteQueue`.
- Andere -Benutzer und -Regeln, die versuchen, diesen VPC-Endpunkt zu verwenden.
- MyUser Senden von Nachrichten an eine andere Amazon-SQS-Warteschlange.

Note

Der Benutzer kann nach wie vor andere Amazon-SQS-API-Aktionen von außerhalb der VPC verwenden. Weitere Informationen finden Sie unter [Beispiel 5: Verweigerung des Zugriffs, wenn dieser nicht von einem VPC-Endpunkt aus erfolgt](#).

Stellen Sie mithilfe von Dual-stack (IPv4- und IPv6-) Endpunkten eine Connect zu Amazon SQS her

Dual-stack Endpunkte unterstützen sowohl IPv4- als auch IPv6-Verkehr. Wenn Sie eine Anfrage an einen Dual-Stack-Endpunkt stellen, wird die Endpunkt-URL in eine IPv4- oder eine IPv6-

Adresse aufgelöst. [Weitere Informationen zu Dual-Stack- und FIPS-Endpunkten finden Sie im SDK-Referenzhandbuch.](#)

Amazon SQS unterstützt regionale Dual-Stack-Endpunkte, was bedeutet, dass Sie die AWS Region als Teil des Endpunktnamens angeben müssen. Dual-stack Endpunktnamen verwenden die folgende Benennungskonvention: `sqs.Region.amazonaws.com` Beispielsweise ist der Dual-Stack-Endpunktname für die Region `eu-west-1` `sqs.eu-west-1.amazonaws.com`.

Die vollständige Liste der Amazon SQS Endpunkte finden Sie in der [AWS Allgemeinen Referenz](#).

Identity and Access Management in Amazon SQS

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service, den Zugriff auf AWS Ressourcen sicher zu kontrollieren. IAM-Administratoren steuern, wer authentifiziert (angemeldet) und autorisiert (im Besitz von Berechtigungen) ist, Amazon-SQS-Ressourcen zu nutzen. IAM ist ein Programm AWS-Service, das Sie ohne zusätzliche Kosten nutzen können.

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe [Beheben von Identitäts- und Zugriffsfehlern bei Amazon Simple Queue Service](#)).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe [So funktioniert Amazon Simple Notification Service mit IAM](#)).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe [Best Practices für Richtlinien](#)).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung

oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter [So melden Sie sich bei Ihrem AWS-Konto an](#) im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter [AWS Signature Version 4 for API requests](#) im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter [Tasks that require root user credentials](#) im IAM-Benutzerhandbuch.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen aus einer Identitätsquelle zugreift. Verbundene Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter [Was ist IAM Identity Center?](#) im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein [IAM-Benutzer](#) ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu](#) können.

Eine [IAM-Gruppe](#) spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter [Anwendungsfälle für IAM-Benutzer](#) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine [IAM-Rolle](#) ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie [von einer Benutzer- zu einer IAM-Rolle \(Konsole\) wechseln](#) AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter [Methoden, um eine Rolle zu übernehmen](#) im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM](#) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter [Übersicht über JSON-Richtlinien](#) im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcen und unter welchen Bedingungen durchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter [Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien](#) im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind)

sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter [Choose between managed policies and inline policies](#) im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie [einen Prinzipal angeben](#).

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- **Berechtigungsgrenzen** – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter [Berechtigungsgrenzen für IAM-Entitäten](#) im -IAM-Benutzerhandbuch.
- **Richtlinien zur Dienstkontrolle (SCPs)** — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter [Service-Kontrollrichtlinien](#) im AWS Organizations -Benutzerhandbuch.
- **Richtlinien zur Ressourcenkontrolle (RCPs)** — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter [Richtlinien zur Ressourcenkontrolle \(RCPs\)](#).
- **Sitzungsrichtlinien** – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter [Sitzungsrichtlinien](#) im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter [Bewertungslogik für Richtlinien](#) im IAM-Benutzerhandbuch.

Übersicht über die Zugriffsverwaltung in Amazon SQS

Jede AWS Ressource gehört einem AWS-Konto, und die Berechtigungen zum Erstellen oder Zugreifen auf eine Ressource werden durch Berechtigungsrichtlinien geregelt. Ein Kontoadministrator kann IAM-Identitäten (d. h. Benutzern, Gruppen und Rollen) Berechtigungsrichtlinien zuweisen. Manche Services (wie etwa Amazon SQS) unterstützen auch die Zuweisung von Berechtigungsrichtlinien zu Ressourcen.

Note

Ein Kontoadministrator (oder Administratorbenutzer) ist ein Benutzer mit Administratorberechtigungen. Weitere Informationen finden Sie unter [Bewährte Methoden für IAM](#) im IAM-Benutzerhandbuch.

Beim Erteilen von Berechtigungen geben Sie an, wer die Berechtigungen erhält, für welche Ressourcen die Berechtigungen gelten und welche Aktionen an dieser Ressource gestattet werden sollen.

Ressourcen und Abläufe von Amazon Simple Queue Service

In Amazon SQS ist die einzige Ressource die Warteschlange. In einer Richtlinie identifizieren Sie die Ressource, für welche die Richtlinie gilt, mithilfe eines Amazon-Ressourcennamens (ARN). Der folgenden Ressource ist ein eindeutiger ARN zugewiesen:

Ressourcentyp	ARN-Format
Warteschlange	<code>arn:aws:sqs: <i>region</i>:<i>account_id</i>:<i>queue_name</i></code>

Es folgen Beispiele des ARN-Formats für Warteschlangen:

- Ein ARN für eine Warteschlange mit dem Namen `my_queue` in der Region USA Ost (Ohio), die zum AWS Konto 123456789012 gehört:

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- Ein ARN für eine Warteschlange mit dem Namen `my_queue` in den verschiedenen Regionen, die von Amazon SQS unterstützt werden:

```
arn:aws:sqs:*:123456789012:my_queue
```

- Eine ARN, der `*` oder `?` als Platzhalter für den Warteschlangennamen verwendet. In den folgenden Beispielen entspricht der ARN allen Warteschlangen mit dem Präfix `my_prefix_`:

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

Sie können den ARN-Wert für eine vorhandene Warteschlange abrufen, indem Sie die Aktion [GetQueueAttributes](#) aufrufen. Der Wert des `QueueArn`-Attributs ist der ARN der Warteschlange. Weitere Informationen zu ARNs finden Sie unter [IAM ARNs im IAM-Benutzerhandbuch](#).

Amazon SQS bietet eine Reihe von Aktionen für die Arbeit mit der Warteschlangenressource. Weitere Informationen finden Sie unter [Amazon-SQS-API-Berechtigungen: Referenz für Aktionen und Ressourcen](#).

Grundlegendes zum Eigentum an Ressourcen

Das AWS-Konto besitzt die Ressourcen, die im Konto erstellt wurden, unabhängig davon, wer die Ressourcen erstellt hat. Genauer gesagt ist der Ressourceneigentümer das AWS-Konto der Prinzipal-Entität (d. h. das Root-Konto, ein Benutzer oder eine IAM-Rolle), die die Anfrage zur Erstellung der Ressource authentifiziert. Die Funktionsweise wird anhand der folgenden Beispiele deutlich:

- Wenn Sie Ihre Root-Kontoanmeldedaten verwenden, AWS-Konto um eine Amazon SQS-Warteschlange zu erstellen, sind Sie AWS-Konto der Eigentümer der Ressource (in Amazon SQS ist die Ressource die Amazon SQS SQS-Warteschlange).
- Wenn Sie in Ihrer Datenbank einen Benutzer erstellen AWS-Konto und diesem Benutzer die Berechtigung zum Erstellen einer Warteschlange erteilen, kann der Benutzer die Warteschlange erstellen. Eigentümer der Warteschlangenressource ist jedoch Ihr AWS-Konto (zu dem der Benutzer gehört).
- Wenn Sie eine IAM-Rolle in Ihrem AWS-Konto mit den Berechtigungen zum Erstellen einer Amazon SQS SQS-Warteschlange erstellen, kann jeder, der die Rolle übernehmen kann, eine Warteschlange erstellen. Ihnen AWS-Konto (zu der die Rolle gehört) gehört die Warteschlangenressource.

Verwaltung des Zugriffs auf -Ressourcen

Eine Berechtigungsrichtlinie beschreibt die Konten zugewiesenen Berechtigungen. Im folgenden Abschnitt werden die verfügbaren Optionen zum Erstellen von Berechtigungsrichtlinien erläutert.

Note

Dieser Abschnitt behandelt die Verwendung von IAM im Zusammenhang mit Amazon SQS. Er enthält keine detaillierten Informationen über den IAM-Service. Eine umfassende IAM-Dokumentation finden Sie unter [Was ist IAM?](#) im IAM-Benutzerhandbuch. Für Informationen über die Syntax und Beschreibungen von [AWS -IAM-Richtlinien](#) lesen Sie die IAM-Richtlinienreferenz im IAM-Benutzerhandbuch.

Richtlinien, die einer IAM-Identität zugeordnet sind, werden als identitätsbasierte Richtlinien (IAM-Richtlinien) bezeichnet, während Richtlinien, die einer Ressource zugeordnet sind, ressourcenbasierte Richtlinien genannt werden.

Identitätsbasierte Richtlinien

Es gibt zwei Möglichkeiten, Ihren Benutzern Berechtigungen für Ihre Amazon-SQS-Warteschlangen zu erteilen: das Amazon-SQS-Richtliniensystem und das IAM-Richtliniensystem. Sie können entweder eines der Systeme oder beide verwenden, um Benutzern oder Rollen Richtlinien anzufügen. In den meisten Fällen erzielen Sie mit beiden Systemen dasselbe Ergebnis. Sie können z. B. Folgendes tun:

- Einem Benutzer oder einer Gruppe in Ihrem Konto eine Berechtigungsrichtlinie zuweisen – Wenn Sie einem Benutzer Berechtigungen zur Erstellung einer Amazon-SQS-Warteschlange erteilen möchten, können Sie dem Benutzer oder der Gruppe, zu der er gehört, eine Berechtigungsrichtlinie zuweisen.
- Eine Berechtigungsrichtlinie an einen Benutzer in einem anderen anhängen AWS-Konto— Sie können eine Berechtigungsrichtlinie an einen Benutzer in einem anderen anhängen AWS-Konto , damit dieser mit einer Amazon SQS SQS-Warteschlange interagieren kann. Kontoübergreifende Berechtigungen gelten jedoch nicht für die folgenden Aktionen:

Kontoübergreifende Berechtigungen gelten nicht für die folgenden Aktionen:

- [AddPermission](#)
- [CancelMessageMoveTask](#)

- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

Um Zugriff auf diese Aktionen zu gewähren, muss der Benutzer demselben AWS-Konto Benutzer angehören, dem die Amazon SQS SQS-Warteschlange gehört.

- Einer Rolle eine Berechtigungsrichtlinie zuordnen (kontoübergreifende Berechtigungen gewähren) — Um kontoübergreifende Berechtigungen für eine SQS-Warteschlange zu gewähren, müssen Sie sowohl IAM- als auch ressourcenbasierte Richtlinien kombinieren:

1. In Konto A (dem die Warteschlange gehört):

- Hängen Sie eine ressourcenbasierte Richtlinie an die SQS-Warteschlange an. Diese Richtlinie muss dem Prinzipal in Konto B (z. [SendMessage](#) oder einer [ReceiveMessage](#) IAM-Rolle) ausdrücklich die erforderlichen Berechtigungen (z. B.,) gewähren.

2. Erstellen Sie in Konto A eine IAM-Rolle:

- Eine Vertrauensrichtlinie, die es Konto B oder A ermöglicht, die Rolle AWS-Service zu übernehmen.

 Note

Wenn Sie möchten, dass ein AWS-Service (wie Lambda oder EventBridge) die Rolle übernimmt, geben Sie den Dienstprinzipal (z. B. `lambda.amazonaws.com`) in der Vertrauensrichtlinie an.

- Eine identitätsbasierte Richtlinie, die der angenommenen Rolle Berechtigungen zur Interaktion mit der Warteschlange gewährt.

3. Erteilen Sie in Konto B die Erlaubnis, die Rolle in Konto A zu übernehmen.

Sie müssen die Zugriffsrichtlinie der Warteschlange so konfigurieren, dass der kontoübergreifende Prinzipal zulässig ist. Identitätsbasierte IAM-Richtlinien allein reichen für den kontoübergreifenden Zugriff auf SQS-Warteschlangen nicht aus.

Weitere Informationen zum Delegieren von Berechtigungen mithilfe von IAM finden Sie unter [Zugriffsverwaltung](#) im IAM-Benutzerhandbuch.

Amazon SQS arbeitet zwar mit IAM-Richtlinien, verfügt jedoch über eine eigene Richtlinieninfrastruktur. Sie können eine Amazon SQS SQS-Richtlinie mit einer Warteschlange verwenden, um anzugeben, welche AWS Konten Zugriff auf die Warteschlange haben. Sie können die Art des Zugriffs sowie Bedingungen festlegen (z. B. eine Bedingung, mit der Berechtigungen für `SendMessage` und für `ReceiveMessage` erteilt werden, wenn die Anforderung vor dem 31. Dezember 2010 gestellt wurde). Die spezifischen Aktionen, für die Sie Berechtigungen erteilen können, gelten für eine Untergruppe der gesamten Liste der Amazon-SQS-Aktionen. Wenn Sie eine Amazon-SQS-Richtlinie schreiben und für * „alle Amazon-SQS-Aktionen zulassen“ festlegen, bedeutet dies, dass alle Aktionen dieser Untergruppe von einzelnen Benutzern ausgeführt werden können.

Das folgende Diagramm veranschaulicht das Konzept einer dieser grundlegenden Amazon-SQS-Richtlinien, die für die Untergruppe der Aktionen gelten. Die Richtlinie gilt für `queue_xyz` und gibt AWS Konto 1 und AWS Konto 2 die Erlaubnis, jede der zulässigen Aktionen mit der angegebenen Warteschlange zu verwenden.

 Note

Die Ressource in der Richtlinie ist wie folgt angegeben: `123456789012/queue_xyz`. Dabei `123456789012` handelt es sich um die AWS Konto-ID des Kontos, dem die Warteschlange gehört.

SQS Policy on queue_xyz

Allow who:

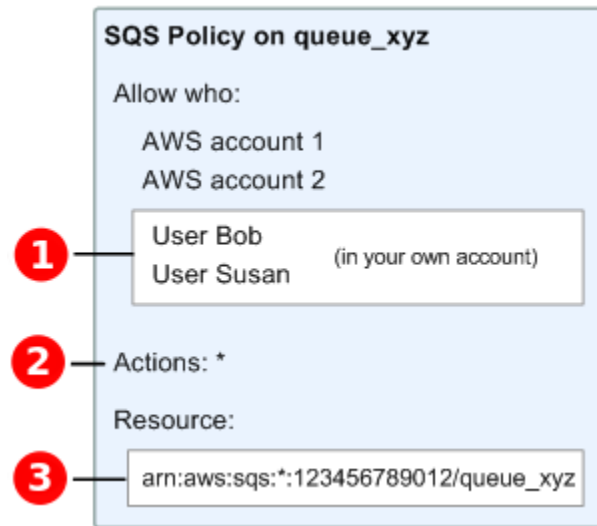
AWS account 1
AWS account 2

Actions: *

Resource:

`123456789012/queue_xyz`

Mit der Einführung von IAM und den Konzepten von Users und Amazon Resource Names (ARNs) haben sich einige Dinge an den SQS-Richtlinien geändert. Im folgenden Diagramm und in der Tabelle werden die Änderungen beschreiben.



1

Informa

zur Erteilung von Berechtigungen an Benutzer mit unterschiedlichen Konten finden Sie unter [Tutorial: Kontenübergreifender Delegieren des Zugriffs mithilfe von IAM-Rollen im IAM-Benutzerhandbuch](#).

AWS

2

Die Untergruppe der Aktionen in * wurde erweitert. Eine Liste zulässiger Aktionen finden Sie unter [Amazon-SQS-API-Berechtigungen: Referenz für Aktionen und Ressourcen](#).

3

Sie können die Ressource mithilfe des Amazon-Ressourcennamens (ARN) angeben. Dies entspricht dem standardmäßigen Verfahren zum Festlegen von Ressourcen in IAM-Richtlinien. Weitere Informationen zum ARN-Format für Amazon-SQS-Warteschlangen finden Sie unter [Ressourcen und Abläufe von Amazon Simple Queue Service](#).

Gemäß der Amazon SQS SQS-Richtlinie im obigen Diagramm kann beispielsweise jeder, der die Sicherheitsanmeldedaten für AWS Konto 1 oder AWS Konto 2 besitzt, darauf zugreifenqueue_xyz. Außerdem haben die Benutzer Bob und Susan in Ihrem eigenen AWS -Konto (Benutzer-ID 123456789012) Zugriff auf die Warteschlange.

Vor der Einführung von IAM übertrug Amazon SQS automatisch die vollständige Kontrolle über die Warteschlange auf den jeweiligen Ersteller der Warteschlange (d. h. Zugriff auf alle möglichen

Amazon-SQS-Aktionen für diese Warteschlange). Dies trifft nur noch zu, wenn der Ersteller AWS - Sicherheitsanmeldeinformationen verwendet. Jeder Benutzer, der über Berechtigungen zum Erstellen einer Warteschlange verfügt, muss zudem Berechtigungen zur Verwendung anderer Amazon-SQS-Aktionen haben, um Aktionen für die erstellten Warteschlangen ausführen zu können.

Es folgt ein Beispiel für eine Richtlinie, mit der ein Benutzer zwar alle Amazon-SQS-Aktionen verwenden kann, jedoch für Warteschlangen, deren Namen mit dem Präfix der Literalzeichenfolge `bob_queue_` versehen sind.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
  }]
}
```

Weitere Informationen finden Sie unter [Richtlinien mit Amazon SQS verwenden](#) und [Identitäten \(Benutzer, Gruppen und Rollen\)](#) im IAM-Benutzerhandbuch.

Angeben der Richtlinienelemente: Aktionen, Effekte, Ressourcen und Prinzipale

Für jede [Amazon-Simple-Queue-Service-Ressource](#) definiert der Service eine Reihe von [Aktionen](#). Zur Erteilung von Berechtigungen für diese Aktionen definiert Amazon SQS eine Reihe von Aktionen, die Sie in einer Richtlinie angeben können.

Note

Für das Durchführen einer Aktion können Berechtigungen für mehrere Aktionen erforderlich sein. Bei der Erteilung von Berechtigungen für bestimmte Aktionen geben Sie auch die Ressource an, für die die Aktionen zugelassen oder verweigert werden.

Grundlegende Richtlinienelemente:

- **Ressource** – In einer Richtlinie wird der Amazon-Ressourcenname (ARN) zur Identifizierung der Ressource verwendet, für die die Richtlinie gilt.
- **Aktion** – Mit Aktionsschlüsselwörtern geben Sie die Ressourcenaktionen an, die Sie zulassen oder verweigern möchten. Die `sqs:CreateQueue`-Berechtigung erteilt dem Benutzer zum Beispiel Berechtigungen zum Ausführen der Amazon-Simple-Queue-Service-Aktion `CreateQueue`.
- **Auswirkung** – Die von Ihnen festgelegte Auswirkung, wenn der Benutzer die jeweilige Aktion anfordert – entweder „allow“ (Zugriffserlaubnis) oder „deny“ (Zugriffsverweigerung). Wenn Sie den Zugriff auf eine Ressource nicht ausdrücklich gestatten (""), wird er automatisch verweigert. Sie können den Zugriff auf eine Ressource auch explizit verweigern. So können Sie sicherstellen, dass Benutzer nicht darauf zugreifen können, auch wenn der Zugriff durch eine andere Richtlinie gestattet wird.
- **Prinzipal** – In identitätsbasierten Richtlinien (IAM-Richtlinien) ist der Benutzer, dem die Richtlinie zugewiesen ist, automatisch der Prinzipal. In ressourcenbasierten Richtlinien müssen Sie den Benutzer, das Konto, den Service oder die sonstige Entität angeben, die die Berechtigungen erhalten soll (gilt nur für ressourcenbasierte Richtlinien).

Weitere Informationen zur Syntax und zu Beschreibungen von Amazon-SQS-Richtlinien finden Sie in der [AWS -IAM-Richtlinienreferenz](#) im IAM-Benutzerhandbuch.

Eine Tabellenliste mit allen Amazon-Simple-Queue-Service-Aktionen und den Ressourcen, für die diese gelten, finden Sie unter [Amazon-SQS-API-Berechtigungen: Referenz für Aktionen und Ressourcen](#).

So funktioniert Amazon Simple Notification Service mit IAM

Bevor Sie mit IAM den Zugriff auf Amazon SQS verwalten können, sollten Sie sich darüber informieren, welche IAM-Features Sie mit Amazon SQS verwenden können.

IAM-Features, die Sie mit Amazon Simple Queue Service verwenden können

IAM-Feature	Amazon-SQS-Unterstützung
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Ja
Richtlinienaktionen	Ja

IAM-Feature	Amazon-SQS-Unterstützung
Richtlinienressourcen	Ja
Richtlinienbedingungsschlüssel (servicespezifisch)	Ja
ACLs	Nein
ABAC (Tags in Richtlinien)	Teilweise
Temporäre Anmeldeinformationen	Ja
Forward Access Sessions (FAS)	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Nein

Einen allgemeinen Überblick darüber, wie Amazon SQS und andere AWS Services mit den meisten IAM-Funktionen funktionieren, finden Sie im [AWS IAM-Benutzerhandbuch unter Services, die mit IAM funktionieren](#).

Zugriffskontrolle

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Dienste, die Unterstützung ACLs bieten. AWS WAF Weitere Informationen finden Sie unter [Übersicht über ACLs die Zugriffskontrollliste \(ACL\)](#) im Amazon Simple Storage Service Developer Guide.

Note

Es ist wichtig zu verstehen, dass alle ihre Berechtigungen an Benutzer unter ihren Konten delegieren AWS-Konten können. Durch den kontoübergreifenden Zugriff können Sie den Zugriff auf Ihre AWS Ressourcen gemeinsam nutzen, ohne zusätzliche Benutzer verwalten zu müssen. Weitere Informationen über die Verwendung des kontoübergreifenden Zugriffs finden Sie unter [Enabling Cross-Account Access](#) im IAM-Benutzerhandbuch.

Weitere Informationen zu inhaltsübergreifenden Berechtigungen und Bedingungsschlüsseln in den benutzerdefinierten Amazon-SQS-Richtlinien finden Sie unter [Einschränkungen der benutzerdefinierten Amazon SQS SQS-Richtlinien](#).

Identitätsbasierte Richtlinien für Amazon SQS

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter [Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien](#) im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der [IAM-Referenz für JSON-Richtlinienelemente](#) im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien für Amazon SQS

Beispiele für identitätsbasierte Amazon-SQS-Richtlinien finden Sie unter [Best Practices für Richtlinien](#).

Ressourcenbasierte Richtlinien in Amazon SQS

Unterstützt ressourcenbasierte Richtlinien: Ja

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie [einen Prinzipal angeben](#). Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Weitere Informationen finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM](#) im IAM-Benutzerhandbuch.

Richtlinienaktionen für Amazon SQS

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Eine Liste der Amazon-SQS-Aktionen finden Sie unter [Von Amazon Simple Queue Service definierte Ressourcen](#) in der Service-Autorisierungs-Referenz.

Richtlinienaktionen in Amazon SQS verwenden das folgende Präfix vor der Aktion:

```
sqs
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

```
"Action": [  
    "sqs:action1",  
    "sqs:action2"  
]
```

Beispiele für identitätsbasierte Amazon-SQS-Richtlinien finden Sie unter [Best Practices für Richtlinien](#).

Richtlinienressourcen für Amazon SQS

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen [Amazon-Ressourcennamen \(ARN\)](#) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*" 
```

Eine Liste der Amazon SQS SQS-Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter [Von Amazon Simple Queue Service definierte Aktionen in der Service](#) Authorization Reference. Um zu erfahren, mit welchen Aktionen Sie den ARN jeder Ressource angeben können, lesen Sie [Von Amazon Simple Queue Service definierte Ressourcen](#).

Beispiele für identitätsbasierte Amazon-SQS-Richtlinien finden Sie unter [Best Practices für Richtlinien](#).

Richtlinien-Bedingungsschlüssel für Amazon SQS

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die [Bedingungsoperatoren](#) verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter [Kontextschlüssel für AWS globale Bedingungen](#) im IAM-Benutzerhandbuch.

Eine Liste der Amazon-SQS-Bedingungsschlüssel finden Sie unter [Bedingungsschlüssel für Amazon Simple Queue Service](#) in der Service-Autorisierungs-Referenz. Informationen dazu, mit welchen Aktionen und Ressourcen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter [Von Amazon Simple Queue Service definierte Ressourcen](#).

Beispiele für identitätsbasierte Amazon-SQS-Richtlinien finden Sie unter [Best Practices für Richtlinien](#).

ACLs bei Amazon SQS

Unterstützt ACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

ABAC mit Amazon SQS

Unterstützt ABAC (Tags in Richtlinien): Teilweise

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen, auch als Tags bezeichnet, definiert werden. Sie können Tags an IAM-Entitäten und AWS -Ressourcen anhängen und dann ABAC-Richtlinien entwerfen, um Operationen zu ermöglichen, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungelement einer [Richtlinie Tag-Informationen](#) an, indem Sie die Schlüssel `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter [Definieren von Berechtigungen mit ABAC-Autorisierung](#) im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe [Attributbasierte Zugriffskontrolle \(ABAC\)](#) verwenden im IAM-Benutzerhandbuch.

Verwenden temporärer Anmeldeinformationen mit Amazon SQS

Unterstützt temporäre Anmeldeinformationen: Ja

Temporäre Anmeldeinformationen ermöglichen den kurzfristigen Zugriff auf AWS Ressourcen und werden automatisch erstellt, wenn Sie einen Verbund verwenden oder die Rollen wechseln. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter [Temporäre Anmeldeinformationen in IAM](#) und [AWS-Services , die mit IAM funktionieren](#) im IAM-Benutzerhandbuch.

Zugriffssitzungen für Amazon SQS weiterleiten

Unterstützt Forward Access Sessions (FAS): Ja

Forward Access Sessions (FAS) verwenden die Berechtigungen des Principals, der einen aufruft AWS-Service, in Kombination mit der Anfrage, Anfragen AWS-Service an nachgelagerte Dienste zu stellen. Einzelheiten zu den Richtlinien für FAS-Anforderungen finden Sie unter [Zugriffssitzungen weiterleiten](#).

Servicerollen für Amazon SQS

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine [IAM-Rolle](#), die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service](#) im IAM-Benutzerhandbuch.

Warning

Das Ändern der Berechtigungen für eine Servicerolle könnte die Funktionalität von Amazon SQS beeinträchtigen. Bearbeiten Sie Servicerollen nur, wenn Amazon SQS dazu Anleitungen gibt.

Serviceverknüpfte Rollen für Amazon SQS

Unterstützt serviceverknüpfte Rollen: Ja

Eine serviceverknüpfte Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

Details zum Erstellen oder Verwalten von serviceverknüpften Rollen finden Sie unter [AWS -Services, die mit IAM funktionieren](#). Suchen Sie in der Tabelle nach einem Service mit einem Yes in der Spalte Service-linked role (Serviceverknüpfte Rolle). Wählen Sie den Link Yes (Ja) aus, um die Dokumentation für die serviceverknüpfte Rolle für diesen Service anzuzeigen.

Amazon SQS SQS-Updates für AWS verwaltete Richtlinien

Um Benutzern, Gruppen und Rollen Berechtigungen hinzuzufügen, ist es einfacher, von AWS verwaltete Richtlinien zu verwenden, als selbst Richtlinien zu schreiben. Es erfordert Zeit und Fachwissen, um [von Kunden verwaltete IAM-Richtlinien zu erstellen](#), die Ihrem Team nur die benötigten Berechtigungen bieten. Um schnell loszulegen, können Sie unsere von AWS verwalteten Richtlinien verwenden. Diese Richtlinien decken häufige Anwendungsfälle ab und sind in Ihrem AWS -Konto verfügbar. Weitere Informationen zu AWS verwalteten Richtlinien finden Sie unter [AWS Verwaltete Richtlinien](#) im IAM-Benutzerhandbuch.

AWS Dienste verwalten und aktualisieren AWS verwaltete Richtlinien. Sie können die Berechtigungen in AWS verwalteten Richtlinien nicht ändern. Dienste fügen einer AWS verwalteten Richtlinie gelegentlich zusätzliche Berechtigungen hinzu, um neue Funktionen zu unterstützen. Diese Art von Update betrifft alle Identitäten (Benutzer, Gruppen und Rollen), an welche die Richtlinie angehängt ist. Es ist sehr wahrscheinlich, dass Dienste eine AWS verwaltete Richtlinie aktualisieren, wenn eine neue Funktion eingeführt wird oder wenn neue Operationen verfügbar werden. Dienste entfernen keine Berechtigungen aus einer AWS verwalteten Richtlinie, sodass durch Richtlinienaktualisierungen Ihre bestehenden Berechtigungen nicht beeinträchtigt werden.

AWS Unterstützt außerdem verwaltete Richtlinien für Jobfunktionen, die sich über mehrere Dienste erstrecken. Die ReadOnlyAccess AWS verwaltete Richtlinie bietet beispielsweise schreibgeschützten Zugriff auf alle AWS Dienste und Ressourcen. Wenn ein Dienst eine neue Funktion startet, werden nur Leseberechtigungen für neue Operationen und Ressourcen AWS hinzugefügt. Eine Liste und Beschreibungen der Richtlinien für Auftragsfunktionen finden Sie in [Verwaltete AWS -Richtlinien für Auftragsfunktionen](#) im IAM-Leitfaden.

AWS verwaltete Richtlinie: Amazon SQSFull Access

Sie können die AmazonSQSFullAccess-Richtlinie an Ihre Amazon-SQS-Identitäten anfügen. Diese Richtlinie gewährt Berechtigungen, die vollen Zugriff auf Amazon SQS ermöglichen.

Die Berechtigungen für diese Richtlinie finden Sie unter [Amazon SQSFull Access](#) in der Referenz für AWS verwaltete Richtlinien.

AWS verwaltete Richtlinie: Amazon SQSRead OnlyAccess

Sie können die AmazonSQSReadOnlyAccess-Richtlinie an Ihre Amazon-SQS-Identitäten anfügen. Diese Richtlinie gewährt Berechtigungen, die einen schreibgeschützten Zugriff auf Amazon SQS erlauben.

Die Berechtigungen für diese Richtlinie finden Sie unter [Amazon SQSRead OnlyAccess](#) in der Referenz zu AWS verwalteten Richtlinien.

AWS verwaltete Richtlinie: SQSUnlock QueuePolicy

Wenn Sie Ihre Warteschlangenrichtlinie für ein Mitgliedskonto falsch konfiguriert haben, um allen Benutzern den Zugriff auf Ihre Amazon SQS SQS-Warteschlange zu verweigern, können Sie die SQSUnlockQueuePolicy AWS verwaltete Richtlinie verwenden, um die Warteschlange zu entsperren.

Weitere Informationen zum Entfernen einer falsch konfigurierten Warteschlangenrichtlinie, die allen Prinzipalen den Zugriff auf eine Amazon SQS SQS-Warteschlange verweigert, finden Sie unter [Ausführen einer privilegierten Aufgabe auf einem AWS Organizations Mitgliedskonto](#) im IAM-Benutzerhandbuch.

Amazon SQS SQS-Updates für AWS verwaltete Richtlinien

Sehen Sie sich Details zu Aktualisierungen der AWS verwalteten Richtlinien für Amazon SQS an, seit dieser Service begonnen hat, diese Änderungen zu verfolgen. Um automatische Warnungen über Änderungen an dieser Seite zu erhalten, abonnieren Sie den RSS-Feed auf der Amazon-SQS-[Dokumentverlauf](#)-Seite.

Änderungen	Beschreibung	Date
SQSUnlockQueuePolicy	Amazon SQS hat eine neue AWS verwaltete Richtlinie hinzugefügt, mit der eine Warteschlange entsperrt und eine falsch konfigurierte Warteschlangenrichtlinie entfernt wirdSQSUnlock QueuePolicy , die allen Prinzipalen den Zugriff auf eine Amazon SQS SQS-Warteschlange verweigert.	15. November 2024
AmazonSQSReadOnlyAccess	Amazon SQS hat die ListQueueTags Aktion	20. Juni 2024

Änderungen	Beschreibung	Date
	hinzugefügt, die alle Tags abrufen, die mit einer bestimmten Amazon SQS SQS-Warteschlange verknüpft sind. Sie ermöglicht es Ihnen, die Schlüssel-Wert-Paare einzusehen, die der Warteschlange aus organisatorischen oder Metadaten-Gründen zugewiesen wurden. Diese Aktion ist mit dem <code>ListQueueTags</code>-API-Vorgang verknüpft.	
AmazonSQSReadOnlyAccess	Amazon SQS hat eine neue Aktion hinzugefügt, mit der Sie die neuesten Aufgaben zur Nachrichtenverschiebung (bis zu 10) in einer bestimmten Quellwarteschlange auflisten können. Diese Aktion ist mit dem ListMessageMoveTasks-API-Vorgang verknüpft.	9. Juni 2023

Beheben von Identitäts- und Zugriffsfehlern bei Amazon Simple Queue Service

Diagnostizieren und beheben Sie mithilfe der folgenden Informationen gängige Probleme, die bei der Verwendung von Amazon SQS und IAM auftreten können.

Ich bin nicht autorisiert, eine Aktion in Amazon SQS auszuführen

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen einer Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um die Aktion durchführen zu können.

Der folgende Beispielfehler tritt auf, wenn der `mateojackson`-Benutzer versucht, die Konsole zum Anzeigen von Details zu einer fiktiven `my-example-widget`-Ressource zu verwenden, jedoch nicht über `sqs:GetWidget`-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
sqs:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Mateo-Richtlinie aktualisiert werden, damit er mit der `sqs:GetWidget`-Aktion auf die `my-example-widget`-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Ausführung der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an Amazon SQS übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in Amazon SQS auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine Amazon SQS SQS-Ressourcen ermöglichen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen dazu, ob Amazon SQS diese Features unterstützt, finden Sie unter [So funktioniert Amazon Simple Notification Service mit IAM](#).
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen in AWS-Konten Ihrem Besitz gewähren können, finden Sie im IAM-Benutzerhandbuch unter [Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto, dem Sie gehören](#).
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie [AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte](#).
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter [Gewähren von Zugriff für extern authentifizierte Benutzer \(Identitätsverbund\)](#) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM](#) im IAM-Benutzerhandbuch.

Ich möchte meine Warteschlange entsperren

Wenn Sie zu einer Organisation AWS-Konto gehören, können AWS Organizations Richtlinien Sie daran hindern, auf Amazon SQS SQS-Ressourcen zuzugreifen. Standardmäßig blockieren AWS Organizations Richtlinien keine Anfragen an Amazon SQS. Stellen Sie jedoch sicher, dass Ihre AWS Organizations Richtlinien nicht so konfiguriert wurden, dass sie den Zugriff auf Amazon SQS SQS-Warteschlangen blockieren. Anweisungen, wie Sie Ihre AWS Organizations Richtlinien überprüfen können, finden Sie im AWS Organizations Benutzerhandbuch unter [Alle Richtlinien auflisten](#).

Wenn Sie Ihre Warteschlangenrichtlinie für ein Mitgliedskonto falsch konfiguriert haben, um allen Benutzern den Zugriff auf Ihre Amazon SQS SQS-Warteschlange zu verweigern, können Sie die

Warteschlange außerdem entsperren, indem Sie eine privilegierte Sitzung für das Mitgliedskonto in IAM starten. Sobald Sie eine privilegierte Sitzung gestartet haben, können Sie die falsch konfigurierte Warteschlangenrichtlinie löschen, um wieder Zugriff auf die Warteschlange zu erhalten. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Ausführen einer privilegierten Aufgabe für ein AWS Organizations Mitgliedskonto](#).

Richtlinien mit Amazon SQS verwenden

In diesem Thema finden Sie Beispiele für identitätsbasierte Richtlinien, in denen ein Kontoadministrator den IAM-Identitäten (Benutzern, Gruppen und Rollen) Berechtigungsrichtlinien zuweisen kann.

Important

Wir empfehlen Ihnen, zunächst die einführenden Themen zu lesen, in denen die Grundkonzepte und die für Sie verfügbaren Optionen zum Verwalten des Zugriffs auf Ihre Amazon-Simple-Queue-Service-Ressourcen erläutert werden. Weitere Informationen finden Sie unter [Übersicht über die Zugriffsverwaltung in Amazon SQS](#).

Mit Ausnahme von `ListQueues` unterstützen alle Amazon-SQS-Aktionen Berechtigungen auf Ressourcenebene. Weitere Informationen finden Sie unter [Amazon-SQS-API-Berechtigungen: Referenz für Aktionen und Ressourcen](#).

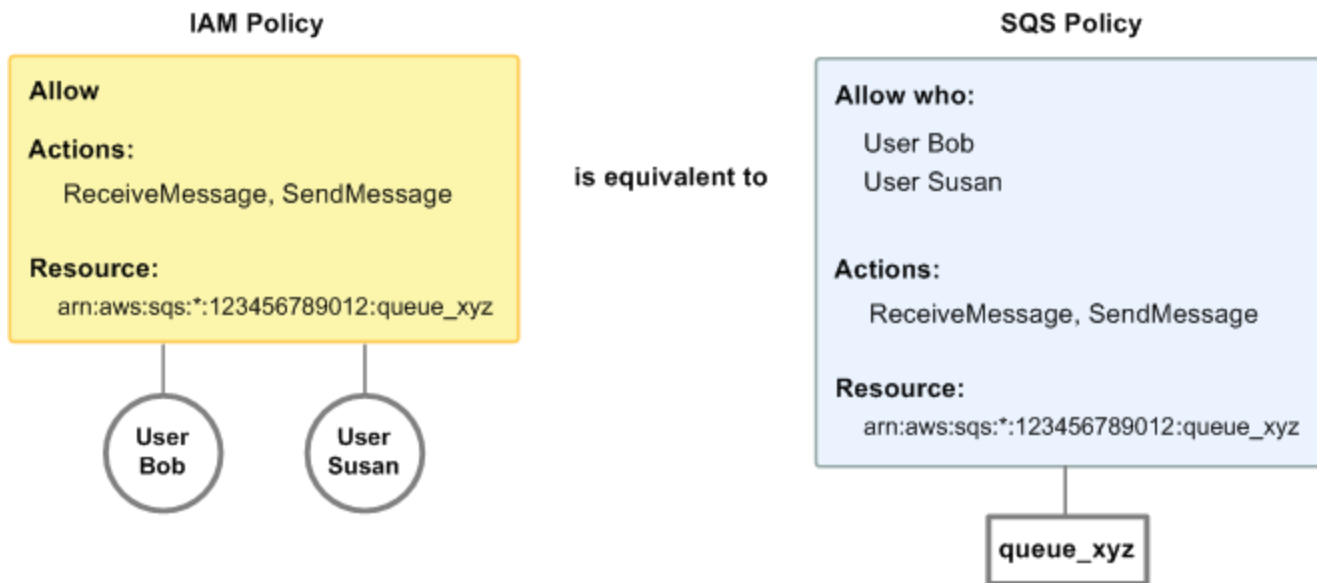
Verwenden von Amazon-SQS- und IAM-Richtlinien

Es gibt zwei Möglichkeiten, Ihren Benutzern Berechtigungen für Ihre Amazon SQS SQS-Ressourcen zu erteilen: mithilfe des Amazon SQS SQS-Richtliniensystems (ressourcenbasierte Richtlinien) und mithilfe des IAM-Richtliniensystems (identitätsbasierte Richtlinien). Sie können eine oder beide Methoden verwenden, mit Ausnahme der `ListQueues` Aktion, bei der es sich um eine regionale Genehmigung handelt, die nur in einer IAM-Richtlinie festgelegt werden kann.

Die folgende Abbildung zeigt eine IAM-Richtlinie und eine entsprechende Amazon-SQS-Richtlinie. Die IAM-Richtlinie gewährt die Rechte an Amazon SQS `ReceiveMessage` und `SendMessage` Aktionen für die Warteschlange, die `queue_xyz` in Ihrem AWS Konto aufgerufen wurde, und die Richtlinie gilt für Benutzer mit den Namen Bob und Susan (Bob und Susan verfügen über die in der Richtlinie angegebenen Berechtigungen). Diese Amazon-SQS-Richtlinie erteilt Bob und Susan Berechtigungen zum Ausführen der Aktionen `ReceiveMessage` und `SendMessage` für dieselbe Warteschlange.

Note

Das folgende Beispiel zeigt einfache Richtlinien ohne Bedingungen. Sie können eine bestimmte Bedingung in einer der Richtlinien angeben und erhalten dasselbe Ergebnis.



Es gibt einen wesentlichen Unterschied zwischen IAM- und Amazon SQS SQS-Richtlinien: Mit dem Amazon SQS SQS-Richtliniensystem können Sie anderen AWS Konten Berechtigungen erteilen, während dies bei IAM nicht der Fall ist.

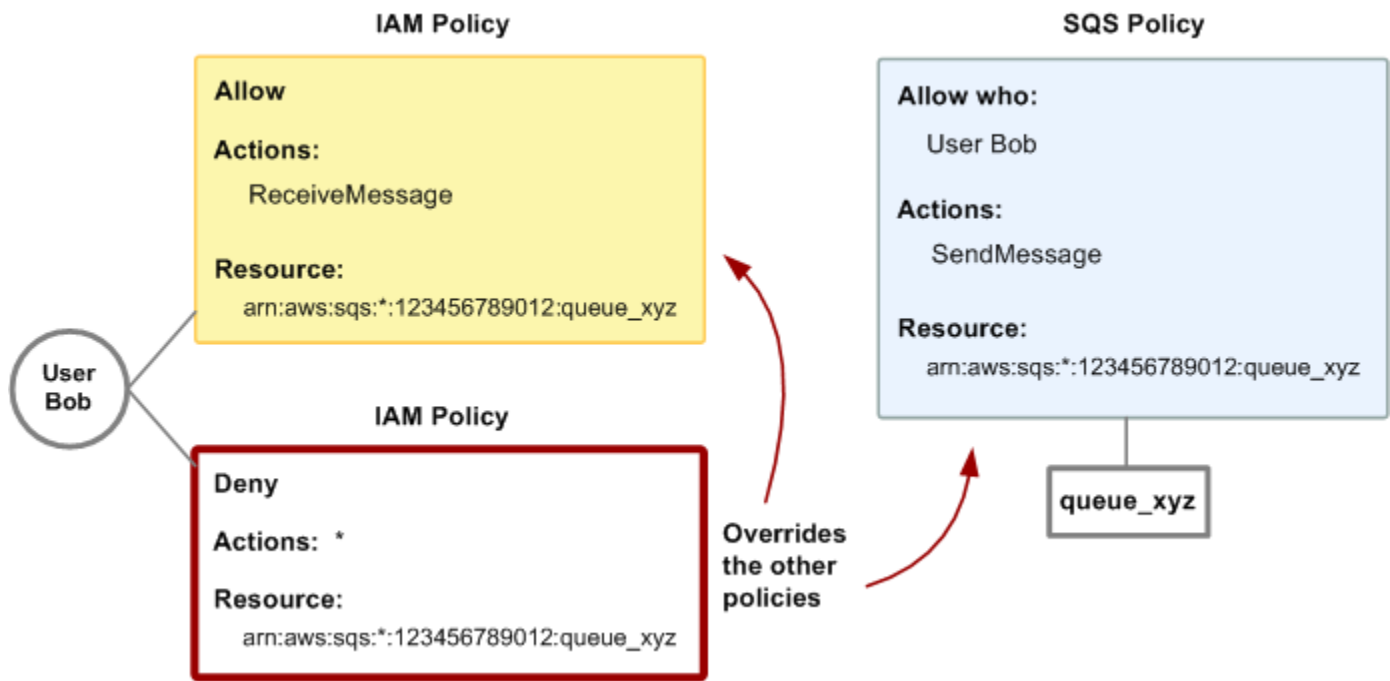
Sie entscheiden, wie Sie beide Systeme zum Verwalten von Berechtigungen verwenden. Die folgenden Beispiele zeigen, wie die beiden Richtliniensysteme zusammenarbeiten.

- Im ersten Beispiel verfügt Bob sowohl über eine IAM-Richtlinie als auch über eine Amazon-SQS-Richtlinie, die für sein Konto gelten. Die IAM-Richtlinie erteilt seinem Konto die Berechtigung für die Aktion `ReceiveMessage` für `queue_xyz`, während die Amazon-SQS-Richtlinie sein Konto berechtigt, die Aktion `SendMessage` für dieselbe Warteschlange auszuführen. Das folgende Diagramm verdeutlicht das Konzept.



Wenn Bob eine `ReceiveMessage`-Anforderung an `queue_xyz` sendet, lässt die IAM-Richtlinie die Aktion zu. Wenn Bob eine `SendMessage`-Anforderung an `queue_xyz` sendet, lässt die Amazon-SQS-Richtlinie die Aktion zu.

- Im zweiten Beispiel missbraucht Bob seinen Zugriff auf `queue_xyz`, sodass es nötig wird, seinen gesamten Zugriff auf die Warteschlange zu verweigern. Der einfachste Weg ist, eine Richtlinie hinzuzufügen, die ihm den Zugriff auf alle Aktionen für die Warteschlange verweigert. Diese Richtlinie setzt die beiden anderen außer Kraft, da ein explizites `deny` ein `allow` immer überschreibt. Weitere Informationen zur Richtlinienauswertungslogik finden Sie unter [Verwenden von benutzerdefinierten Richtlinien mit der Sprache der Zugriffsrichtlinie von Amazon SQS](#). Das folgende Diagramm verdeutlicht das Konzept.



Sie können der Amazon-SQS-Richtlinie auch eine Anweisung hinzufügen, die Bob alle Zugriffe auf die Warteschlange verweigert. Dies hat die gleiche Auswirkung wie das Hinzufügen einer IAM-Richtlinie, die Bob den Zugriff auf die Warteschlange verweigert. Beispiele von Richtlinien, die Amazon-SQS-Aktionen und -Ressourcen abdecken, finden Sie unter [Grundlegende Beispiele für Amazon-SQS-Richtlinien](#). Weitere Informationen zum Erstellen von Amazon-SQS-Richtlinien finden Sie unter [Verwenden von benutzerdefinierten Richtlinien mit der Sprache der Zugriffsrichtlinie von Amazon SQS](#).

Erforderliche Berechtigungen zur Verwendung der Amazon-SQS-Konsole

Ein Benutzer, der mit der Amazon-SQS-Konsole arbeiten möchte, muss über die Mindestmenge an Berechtigungen verfügen, die es ihm erlauben, die Amazon-SQS-Warteschlangen im AWS-Konto des Benutzers zu verwenden. Beispielsweise muss der Benutzer über die Berechtigung zum Aufruf der Aktion `ListQueues` verfügen, um Warteschlangen aufzulisten, oder über die Berechtigung zum Aufruf der Aktion `CreateQueue`, um Warteschlangen erstellen zu können. Zusätzlich zu den Amazon-SQS-Berechtigungen erfordert die Konsole zum Abonnieren einer Amazon-SQS-Warteschlange für ein Amazon-SNS-Thema Berechtigungen für Amazon-SNS-Aktionen.

Wenn Sie eine IAM-Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole möglicherweise nicht wie vorgesehen für Benutzer mit dieser IAM-Richtlinie.

Sie müssen Benutzern, die nur die Aktionen AWS CLI oder Amazon SQS aufrufen, keine Mindestberechtigungen für die Konsole gewähren.

Beispiele für identitätsbasierte Richtlinien für Amazon SQS

Benutzer und Rollen besitzen standardmäßig keine Berechtigungen zum Erstellen oder Ändern von Amazon-SQS-Ressourcen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter [Erstellen von IAM-Richtlinien \(Konsole\)](#) im IAM-Benutzerhandbuch.

Einzelheiten zu den von Amazon SQS definierten Aktionen und Ressourcentypen, einschließlich des Formats ARNs für die einzelnen Ressourcentypen, finden Sie unter [Aktionen, Ressourcen und Bedingungsschlüssel für Amazon Simple Queue Service in der Service](#) Authorization Reference.

Note

Wenn Sie Lebenszyklus-Hooks für Amazon EC2 Auto Scaling konfigurieren, müssen Sie keine Richtlinie schreiben, um Nachrichten an eine Amazon-SQS-Warteschlange zu senden. Weitere Informationen finden Sie unter [Amazon EC2 Auto Scaling Lifecycle Hooks](#) im Amazon EC2-Benutzerhandbuch.

Best Practices für Richtlinien

Identitätsbasierte Richtlinien können festlegen, ob jemand Amazon-SQS-Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder daraus löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter [Von AWS verwaltete Richtlinien](#) oder [Von AWS verwaltete Richtlinien für Auftragsfunktionen](#) im IAM-Benutzerhandbuch.

- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter [Richtlinien und Berechtigungen in IAM](#) im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter [IAM-JSON-Richtlinienelemente: Bedingung](#) im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter [Richtlinienvvalidierung mit IAM Access Analyzer](#) im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter [Sicherer API-Zugriff mit MFA](#) im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter [Best Practices für die Sicherheit in IAM](#) im IAM-Benutzerhandbuch.

Verwenden der Amazon-SQS-Konsole

Um auf die Konsole von Amazon Simple Queue Service zugreifen zu können, müssen Sie über einen Mindestsatz von Berechtigungen verfügen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den Amazon SQS SQS-Ressourcen in Ihrem AWS-Konto aufzulisten und anzuzeigen. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen

Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die die Benutzer ausführen möchten.

Um sicherzustellen, dass Benutzer und Rollen weiterhin die Amazon SQS SQS-Konsole verwenden können, fügen Sie den Entitäten auch die `AmazonSQSReadOnlyAccess` AWS verwaltete Amazon SQS Richtlinie hinzu. Weitere Informationen finden Sie unter [Hinzufügen von Berechtigungen zu einem Benutzer](#) im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der API AWS CLI oder AWS .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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": "*"
}
]
}

```

Erlauben Sie einem Benutzer, Warteschlangen zu erstellen

Im folgenden Beispiel wird eine Richtlinie für den Benutzer Bob erstellt, mit der er zwar auf alle Amazon-SQS-Aktionen zugreifen kann, aber nur mit Warteschlangen, deren Namen mit dem Präfix der Literalzeichenfolge `alice_queue_` versehen sind.

Amazon SQS gewährt dem Ersteller einer Warteschlange nicht automatisch Berechtigungen zum Verwenden der Warteschlange. Daher müssen wir Bob explizit Berechtigungen zum Verwenden aller Amazon-SQS-Aktionen zusätzlich zur Aktion `CreateQueue` in der IAM-Richtlinie erteilen.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
  }]
}

```

Erlauben Sie Entwicklern, Nachrichten in eine gemeinsam genutzte Warteschlange zu schreiben

Im folgenden Beispiel erstellen wir eine Gruppe für Entwickler und fügen eine Richtlinie hinzu, die es der Gruppe ermöglicht, die Amazon SQS `SendMessage` SQS-Aktion zu verwenden, aber nur mit der Warteschlange, die zu der angegebenen gehört AWS-Konto und benannt `MyCompanyQueue` ist.

JSON

```

{

```



```
"Version": "2012-10-17",  
"Statement": [{  
  "Effect": "Allow",  
  "Action": "sqs:SendMessage",  
  "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"  
}]  
}
```

Sie können `*` anstelle von `SendMessage` verwenden, um die folgenden Aktionen auf einen Prinzipal auf einer gemeinsamen Warteschlange zu gewähren: `ChangeMessageVisibility`, `DeleteMessage`, `GetQueueAttributes`, `GetQueueUrl`, `ReceiveMessage` und `SendMessage`.

Note

Obwohl `*` den von den anderen Berechtigungstypen bereitgestellten Zugriff beinhaltet, werden Berechtigungen von Amazon SQS als separat betrachtet. Es ist beispielsweise möglich, einem Benutzer sowohl die Berechtigung `*` als auch die Berechtigung `SendMessage` zu erteilen, obwohl ein `*` den von `SendMessage` bereitgestellten Zugriff gewährt.

Dieses Konzept gilt auch, wenn Sie eine Berechtigung entfernen. Wenn einem Prinzipal lediglich eine `*`-Berechtigung gewährt wurde, verfügt er durch die Anforderung zum Entfernen einer `SendMessage`-Berechtigung nicht ausschließlich über eine `alles außer`-Berechtigung. Stattdessen hat die Anforderung keine Auswirkungen, da der Prinzipal keine explizite `SendMessage`-Berechtigung besitzt. Wenn dem Prinzipal lediglich die `ReceiveMessage`-Berechtigung erteilt werden soll, fügen Sie die `ReceiveMessage`-Berechtigung hinzu und entfernen Sie die `*`-Berechtigung.

Erlauben Sie Managern, die allgemeine Größe von Warteschlangen zu ermitteln

Im folgenden Beispiel erstellen wir eine Gruppe für Manager und fügen eine Richtlinie hinzu, die es der Gruppe ermöglicht, die Amazon SQS `GetQueueAttributes` SQS-Aktion mit allen Warteschlangen zu verwenden, die zu dem angegebenen AWS Konto gehören.

JSON

```
{  
  "Version": "2012-10-17",
```

```
"Statement": [{  
  "Effect": "Allow",  
  "Action": "sqs:GetQueueAttributes",  
  "Resource": "*" }  
]
```

Erlauben Sie einem Partner, Nachrichten an eine bestimmte Warteschlange zu senden

Sie können diese Aufgabe mit einer Amazon-SQS- oder einer IAM-Richtlinie ausführen. Wenn Ihr Partner über eine verfügt AWS-Konto, ist es möglicherweise einfacher, eine Amazon SQS SQS-Richtlinie zu verwenden. Jeder Benutzer im Unternehmen des Partners, der über die AWS Sicherheitsanmeldedaten verfügt, kann jedoch Nachrichten an die Warteschlange senden. Wenn Sie den Zugriff auf einen bestimmten Benutzer oder eine Anwendung beschränken möchten, müssen Sie den Partner wie einen Benutzer in Ihrem eigenen Unternehmen behandeln und eine IAM-Richtlinie anstelle einer Amazon-SQS-Richtlinie verwenden.

Dieses Beispiel führt die folgenden Aktionen aus:

1. Erstellen Sie eine Gruppe WidgetCo , die das Partnerunternehmen repräsentiert.
2. Erstellt einen Benutzer für den jeweiligen Benutzer oder die Anwendung des Unternehmens des Partners, der bzw. die Zugriff erfordert.
3. Fügen Sie den Benutzer zur Gruppe hinzu.
4. Fügt eine Richtlinie an, mit der die Gruppe nur Zugriff auf die Aktion SendMessage ausschließlich für die Warteschlange mit dem Namen WidgetPartnerQueue erhält.

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [{  
    "Effect": "Allow",  
    "Action": "sqs:SendMessage",  
    "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"  
  }]  
}
```

Grundlegende Beispiele für Amazon-SQS-Richtlinien

Dieser Abschnitt zeigt Richtlinienbeispiele für allgemeine Amazon-SQS-Anwendungsfälle.

Während Sie dem Benutzer die Richtlinien zuweisen, können Sie die Konsole verwenden, um die Auswirkungen der einzelnen Richtlinien zu überprüfen. Zunächst verfügt der Benutzer über keine Berechtigungen und kann in der Konsole keine Aktionen ausführen. Während Sie dem Benutzer Richtlinien zuweisen, können Sie überprüfen, ob der Benutzer die verschiedenen Aktionen in der Konsole ausführen kann.

Note

Es wird empfohlen, zwei Browserfenster zu verwenden: eines, um Berechtigungen zu erteilen, und das andere, um sich AWS-Managementkonsole mit den Anmeldeinformationen des Benutzers anzumelden, um die Berechtigungen zu überprüfen, während Sie sie dem Benutzer gewähren.

Beispiel 1: Erteilen Sie einem Benutzer eine Berechtigung AWS-Konto

Die folgende Beispielrichtlinie gewährt AWS-Konto Number 111122223333 die SendMessage Berechtigung für die Warteschlange, die 444455556666/queue1 in der Region USA Ost (Ohio) benannt ist.

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"
  }]
}
```

```
}
```

Beispiel 2: Erteilen Sie einer Person zwei Berechtigungen AWS-Konto

Die folgende Beispielrichtlinie gewährt der genannten Warteschlange 111122223333 SendMessage sowohl die AWS-Konto Nummer als auch die ReceiveMessage Berechtigung 444455556666/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"
  }]
}
```

Beispiel 3: Erteilen Sie zwei Personen alle Berechtigungen AWS-Konten

Die folgende Beispielrichtlinie gewährt zwei verschiedene AWS-Konten Nummern (111122223333 und 444455556666) die Erlaubnis, alle Aktionen zu verwenden, für die Amazon SQS gemeinsamen Zugriff für die Warteschlange gewährt, die 123456789012/queue1 in der Region USA Ost (Ohio) benannt ist.

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"
}]
}
```

Beispiel 4: Einer Rolle und einem Benutzernamen kontenübergreifende Berechtigungen erteilen

Die folgende Beispielrichtlinie gewährt `role1` einer Person `username1` unter der AWS-Konto Nummer `111122223333` kontoübergreifend die Erlaubnis, alle Aktionen zu verwenden, für die Amazon SQS gemeinsamen Zugriff auf die Warteschlange gewährt, die `123456789012/queue1` in der Region USA Ost (Ohio) benannt ist.

Kontoübergreifende Berechtigungen gelten nicht für die folgenden Aktionen:

- [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"
  }]
}
```

Beispiel 5: Allen Benutzern eine Berechtigung erteilen

Mit der folgenden Beispielrichtlinie wird allen Benutzern (anonymen Benutzern) die Berechtigung `ReceiveMessage` für die Warteschlange mit dem Namen `111122223333/queue1` erteilt.

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"
  }]
}
```

Beispiel 6: Allen Benutzern eine zeitlich begrenzte Berechtigung erteilen

Das folgende Beispiel gewährt die Berechtigung `ReceiveMessage` allen Benutzern (anonymen Benutzern) der Warteschlange mit dem Namen `111122223333/queue1`, aber nur zwischen 12:00 Uhr und 15:00 Uhr am 31. Januar 2009.

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"
      }
    }
  }]
}
```

Beispiel 7: Allen Benutzern in einem CIDR-Bereich sämtliche Berechtigungen erteilen

Die folgende Beispielrichtlinie erteilt allen Benutzern (anonymen Benutzern) die Berechtigung zur Verwendung aller möglichen Amazon-SQS-Aktionen, die für die Warteschlange mit dem Namen `111122223333/queue1` gemeinsam genutzt werden können, jedoch nur, wenn die Anfrage aus dem `192.0.2.0/24`-CIDR-Bereich kommt.

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"
    }
  }
}]
}

```

Beispiel 8: Berechtigungen für Benutzer in verschiedenen CIDR-Bereichen über Zulassungslisten und Sperrlisten

Die folgende Beispielrichtlinie enthält zwei Anweisungen:

- Die erste Anweisung gewährt allen Benutzer (anonymen Benutzern) im CIDR-Bereich 192.0.2.0/24 (mit Ausnahme von 192.0.2.188) die Berechtigung zur Verwendung der Aktion SendMessage für die Warteschlange mit dem Namen 111122223333/queue1.
- Die zweite Anweisung verwehrt allen Benutzern (anonyme Benutzer) im CIDR-Bereich 12.148.72.0/23 die Nutzung der Warteschlange.

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"
      }
    }
  }
}]
}
```

Verwenden von benutzerdefinierten Richtlinien mit der Sprache der Zugriffsrichtlinie von Amazon SQS

Um grundlegende Berechtigungen (wie [SendMessage](#) oder [ReceiveMessage](#)) nur auf der Grundlage einer AWS-Konto ID zu gewähren, müssen Sie keine benutzerdefinierte Richtlinie schreiben. Verwenden Sie stattdessen die Amazon SQS [AddPermission](#) SQS-Aktion.

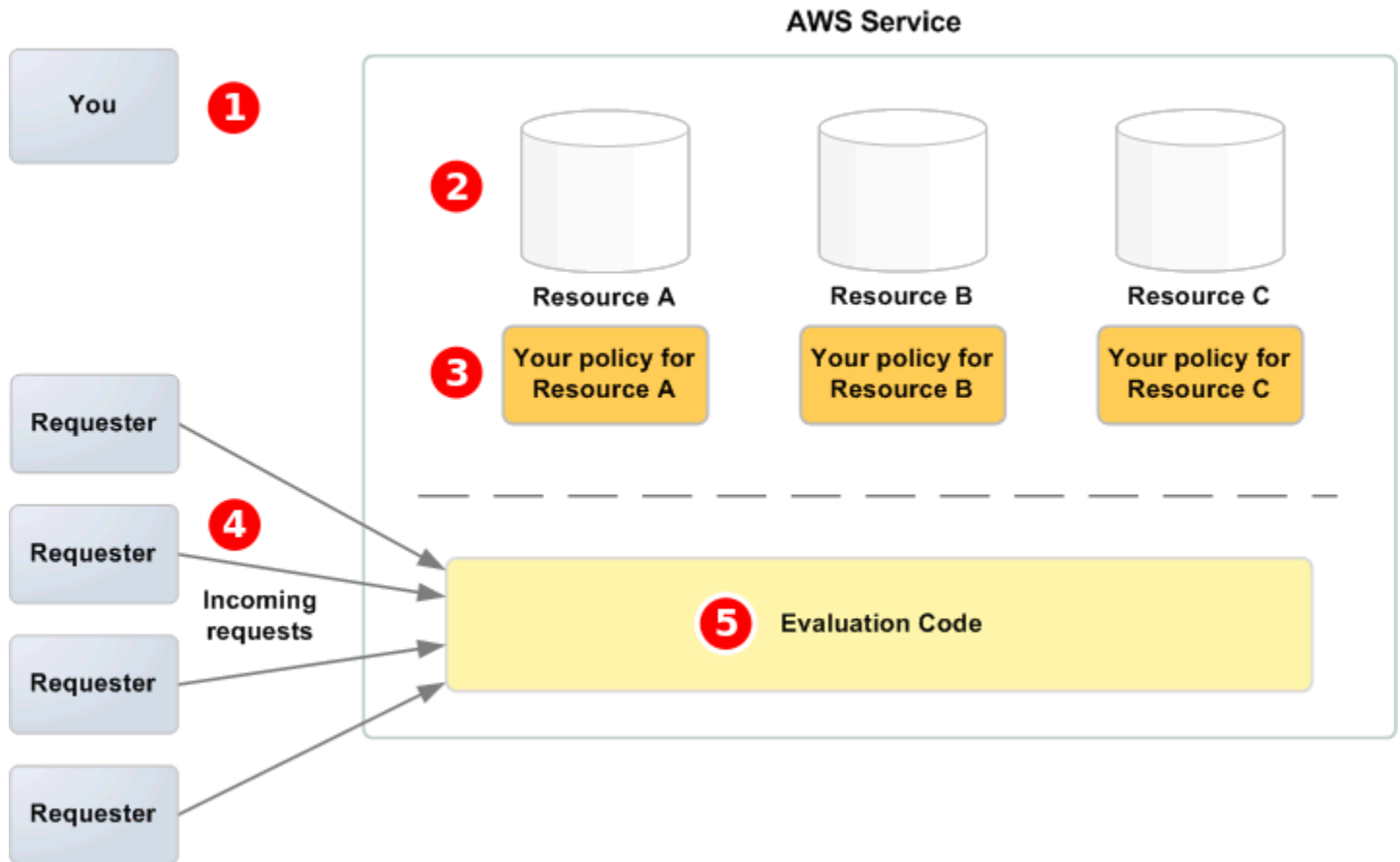
Um den Zugriff auf der Grundlage bestimmter Bedingungen wie der Anforderungszeit oder der IP-Adresse des Anforderers zuzulassen oder zu verweigern, müssen Sie eine benutzerdefinierte Amazon SQS SQS-Richtlinie erstellen und diese mithilfe der [SetQueueAttributes](#) Aktion hochladen.

Themen

- [Amazon-SQS-Zugriffskontrollarchitektur](#)
- [Prozess-Workflow für die Amazon-SQS-Zugriffskontrolle](#)
- [Die wichtigsten Konzepte der Sprache der Zugriffsrichtlinie von Amazon SQS](#)
- [Bewertungslogik der Sprache der Zugriffsrichtlinie von Amazon SQS](#)
- [Beziehungen zwischen expliziten und standardmäßigen Ablehnungen in der Sprache der Zugriffsrichtlinie von Amazon SQS](#)
- [Einschränkungen der benutzerdefinierten Amazon SQS SQS-Richtlinien](#)
- [Beispiele für eine benutzerdefinierte Sprache der Zugriffsrichtlinie von Amazon SQS](#)

Amazon-SQS-Zugriffskontrollarchitektur

Das folgende Diagramm beschreibt die Zugriffskontrolle für Ihre Amazon-SQS-Ressourcen.



1 Sie selbst als Ressourceneigentümer.

2 Ihre im AWS Service enthaltenen Ressourcen (z. B. Amazon SQS SQS-Warteschlangen).

3 Ihre Richtlinien. Es empfiehlt sich eine Richtlinie pro Ressource. Der AWS Service bietet eine API, mit der Sie Ihre Richtlinien hochladen und verwalten können.

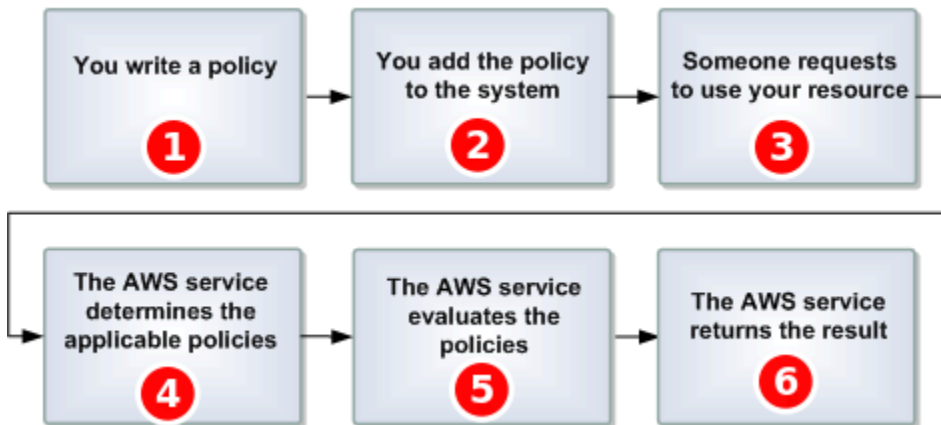
4 Anforderer und deren eingehende Anforderungen an den AWS -Service.

5

Der Code für die Auswertung der Sprache der Zugriffsrichtlinie. Dies ist der Codesatz innerhalb des AWS Dienstes, der eingehende Anfragen anhand der geltenden Richtlinien bewertet und bestimmt, ob dem Antragsteller Zugriff auf die Ressource gewährt wird.

Prozess-Workflow für die Amazon-SQS-Zugriffskontrolle

Das folgende Diagramm beschreibt den allgemeinen Workflow der Zugriffskontrolle der Sprache der Zugriffsrichtlinie von Amazon SQS.

**1**

Sie schreiben eine Amazon-SQS-Richtlinie für Ihre Warteschlange.

2

Sie laden Ihre Richtlinie auf AWS hoch. Der AWS Dienst stellt eine API bereit, die Sie zum Hochladen Ihrer Richtlinien verwenden. Sie verwenden beispielsweise die Amazon-SQS-Aktion `SetQueueAttributes` zum Hochladen einer Richtlinie für eine bestimmte Amazon-SQS-Warteschlange.

3

Jemand sendet eine Anforderung zur Verwendung Ihrer Amazon-SQS-Warteschlange.

4

Amazon SQS prüft alle verfügbaren Amazon-SQS-Richtlinien und bestimmt, welche zutreffend sind.

5

Amazon SQS bewertet die Richtlinien und bestimmt, ob der Anforderer Ihre Warteschlange verwenden darf.

6

Basierend auf dem Ergebnis der Richtlinienbewertung gibt Amazon SQS entweder einen Access denied-Fehler an den Anforderer zurück oder verarbeitet die Anforderung.

Die wichtigsten Konzepte der Sprache der Zugriffsrichtlinie von Amazon SQS

Zum Schreiben Ihrer eigenen Richtlinien müssen Sie mit [JSON](#) und einer Reihe von wichtigen Konzepten vertraut sein.

Sobald Sie die Details auf dieser Seite überprüft haben, klicken Sie auf

Das Ergebnis einer [Statement](#), bei der [Effect \(Effekt\)](#) auf allow festgelegt ist.

Action (Aktion)

Die Aktivität, zu der [Auftraggeber](#) berechtigt ist, in der Regel eine Anforderung an AWS.

Default-deny

Das Ergebnis einer [Statement](#), die weder über die Einstellung [Sobald Sie die Details auf dieser Seite überprüft haben, klicken Sie auf](#) noch [Explicit-deny](#) verfügt.

Bedingung

Jede Einschränkung oder jedes Detail zu einer [Berechtigung](#). Bedingungen sind in der Regel auf das Datum, die Uhrzeit und IP-Adressen bezogen.

Effect (Effekt)

Das Ergebnis, die die [Statement](#) einer [Richtlinie](#) zu Auswertungszeit zurückgeben soll. Sie geben den Wert deny oder allow an, wenn Sie die Richtlinienanweisung schreiben. Bei der Richtlinienbewertung sind drei Ergebnisse möglich: [Default-deny](#), [Sobald Sie die Details auf dieser Seite überprüft haben, klicken Sie auf](#) und [Explicit-deny](#).

Explicit-deny

Das Ergebnis einer [Statement](#), bei der [Effect \(Effekt\)](#) auf deny festgelegt ist.

Bewertung

Der Prozess, mit dem Amazon SQS ermittelt, ob eine eingehende Anforderung basierend auf einer [Richtlinie](#) verweigert oder erlaubt werden soll.

Aussteller

Der Benutzer, der eine [Richtlinie](#) erstellt, um einer Ressource Berechtigungen zu erteilen. Der Emittent ist per Definition immer der Eigentümer der Ressource. AWS erlaubt Amazon SQS SQS-Benutzern nicht, Richtlinien für Ressourcen zu erstellen, die ihnen nicht gehören.

Key (Schlüssel)

Das besondere Merkmal, das die Grundlage für die Einschränkung des Zugriffs bildet.

Berechtigung

Das Konzept, den Zugriff auf eine Ressource mit einer [Bedingung](#) und einem [Key \(Schlüssel\)](#) zuzulassen oder abzulehnen.

Richtlinie

Das Dokument dient als Container für eine oder mehrere [Anweisungen](#).



Amazon SQS verwendet die Richtlinie, um zu bestimmen, ob einem Benutzer der Zugriff auf eine Ressource gewährt werden soll.

Auftraggeber

Der Benutzer, der die [Berechtigung](#) in der [Richtlinie](#) erhält.

Ressource

Das Objekt, auf das die [Auftraggeber](#)-Anforderungen zugreifen.

Statement

Die formelle Beschreibung einer einzelnen Berechtigung, die in der Sprache der Zugriffsrichtlinie als Teil eines umfassenderen [Richtlinie](#)-Dokuments verfasst ist.

Auftraggeber

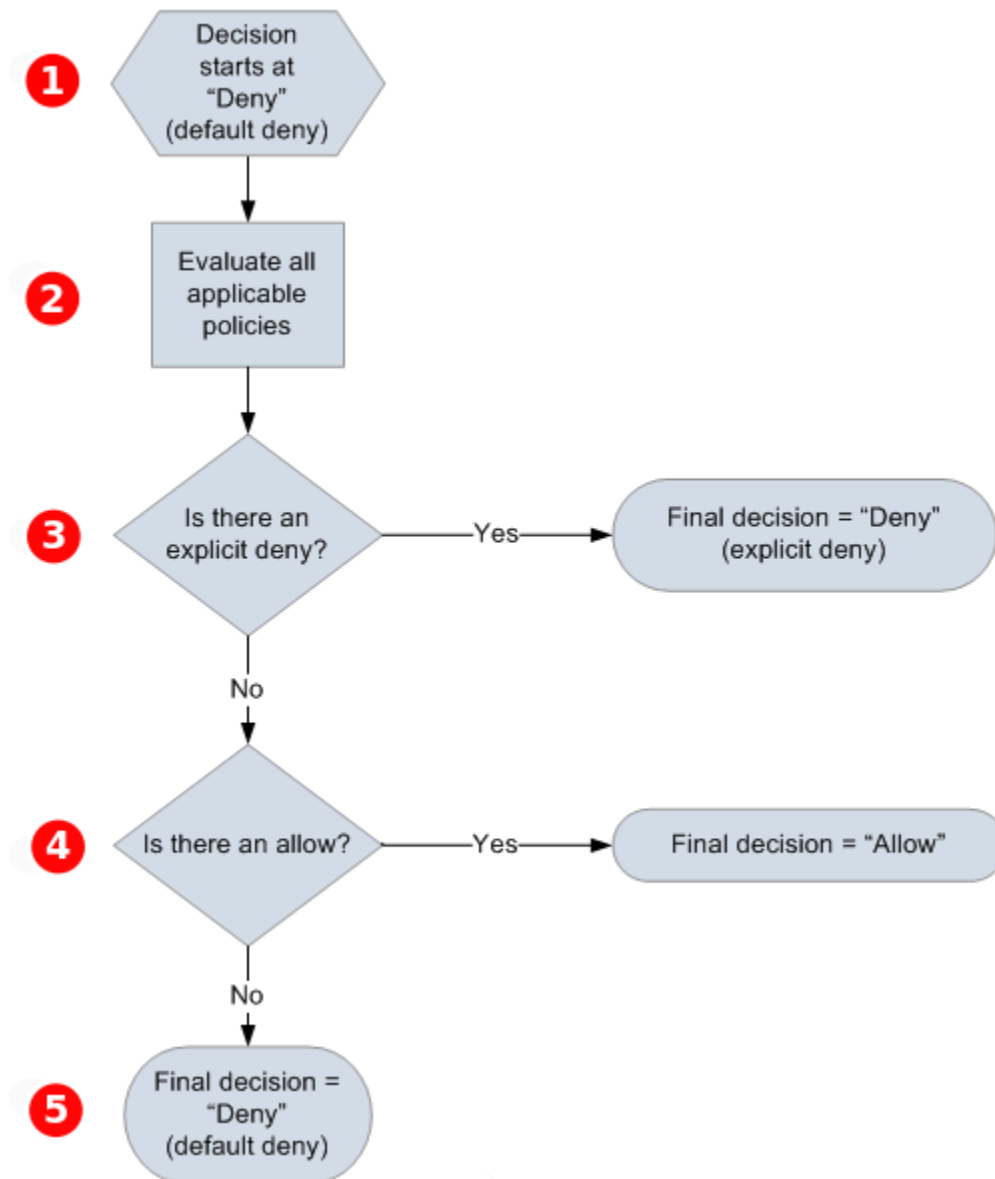
Der Benutzer, der eine Anforderung für den Zugriff auf eine [Ressource](#) sendet.

Bewertungslogik der Sprache der Zugriffsrichtlinie von Amazon SQS

Zum Zeitpunkt der Auswertung bestimmt Amazon SQS, ob eine Anforderung von einem anderen Benutzer als dem Ressourceneigentümer zugelassen oder abgelehnt werden soll. Die Auswertungslogik unterliegt mehreren Grundregeln:

- Standardmäßig werden alle Anforderungen zur Verwendung Ihrer Ressourcen, die nicht von Ihnen stammen, verweigert.
- Ein [Sobald Sie die Details auf dieser Seite überprüft haben, klicken Sie auf](#) setzt jedes [Default-deny](#) außer Kraft.
- Ein [Explicit-deny](#) setzt jedes allow außer Kraft.
- Es spielt keine Rolle, in welcher Reihenfolge die Richtlinien ausgewertet werden.

Im folgenden Diagramm wird detailliert beschrieben, wie Amazon SQS Entscheidungen im Hinblick auf Zugriffsberechtigungen trifft.



1 Die Entscheidung beginnt mit default-deny.

2 Es werden alle Richtlinien ausgewertet, die auf die Anforderung anwendbar sind (basierend auf der Ressource, dem Prinzipal, der Aktion und den Bedingungen). Es spielt keine Rolle, in welcher Reihenfolge der Durchführungscode die Richtlinien auswertet.

3 Der Durchführungscode sucht nach einer explicit-deny-Anweisung, die für die Anforderung gelten

könnte. Wird eine Anweisung gefunden, gibt der Durchführungscode die Entscheidung deny zurück und der Prozess wird beendet.

4

Wenn keine explicit-deny-Anweisung gefunden wird, wird nach allow-Anweisungen gesucht, die auf die Anfrage zutreffen können. Wenn auch nur eine explizite Zugriffserlaubnis gefunden wird, wird allow zurückgegeben und der Prozess beendet (der Service setzt die Verarbeitung der Anforderung fort).

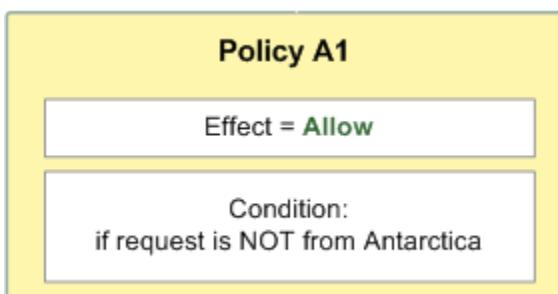
5

Wenn keine allow-Anweisung gefunden wird, ist die endgültige Entscheidung deny (da es kein explicit-deny bzw. allow gab, gilt dies als default-deny).

Beziehungen zwischen expliziten und standardmäßigen Ablehnungen in der Sprache der Zugriffsrichtlinie von Amazon SQS

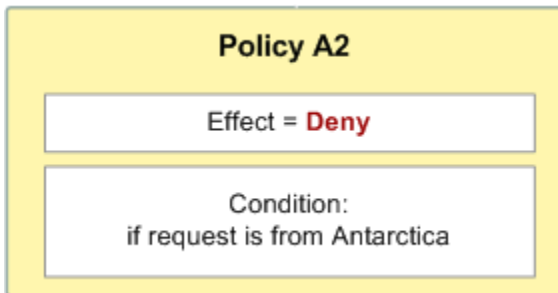
Wenn eine Amazon-SQS-Richtlinie nicht direkt auf eine Anforderung anwendbar ist, führt dies im Ergebnis zu [Default-deny](#). Wenn ein Benutzer z. B. die Berechtigung zur Nutzung von Amazon SQS anfordert, aber die einzige für den Benutzer anwendbare Richtlinie DynamoDB verwenden kann, ist das Ergebnis default-deny.

Wenn eine Bedingung in einer Anweisung nicht erfüllt ist, hat die Anforderung default-deny zur Folge. Wenn alle Bedingungen einer Anweisung erfüllt sind, hat dies für diese Anforderung abhängig vom Wert des Elements [Effect \(Effekt\)](#) entweder [Sobald Sie die Details auf dieser Seite überprüft haben, klicken Sie auf](#) oder [Explicit-deny](#) zur Folge. In Richtlinien ist nicht festgelegt, was geschieht, wenn eine Bedingung nicht erfüllt ist, daher ist das Ergebnis in diesem Fall default-deny. Angenommen, Sie möchten Anforderungen, die aus der Antarktis stammen, ablehnen. Sie erstellen die Richtlinie A1, um Zugriff nur dann zu gewähren, wenn Anforderungen von außerhalb der Antarktis kommen. Die Amazon-SQS-Richtlinie ist in der folgenden Abbildung dargestellt.



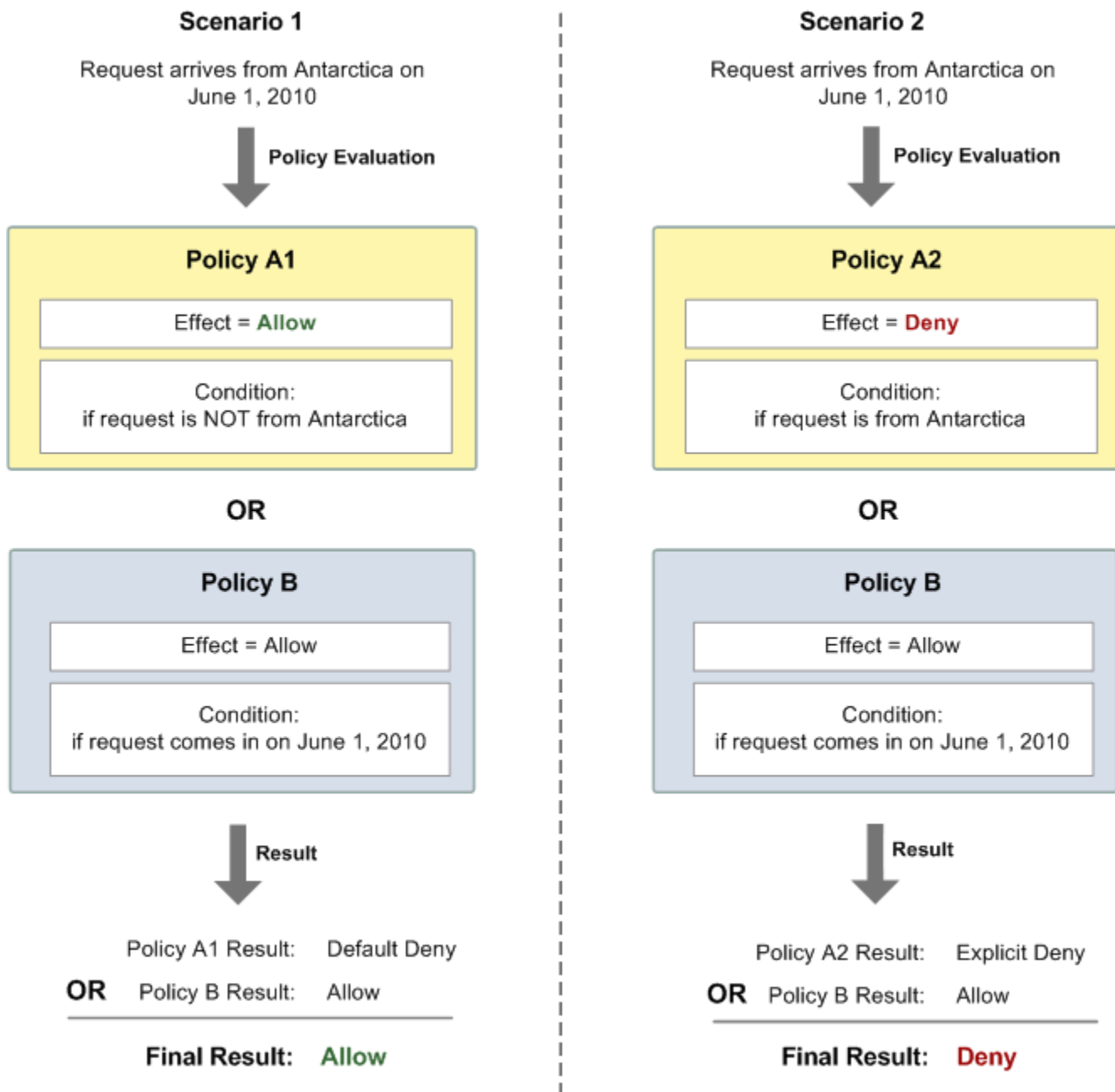
Wenn ein Benutzer eine Anforderung aus den USA sendet, wird die Bedingung erfüllt (die Anforderung stammt nicht aus der Antarktis), und die Anforderung hat allow zur Folge. Wenn ein

Benutzer allerdings eine Anforderung aus der Antarktis sendet, ist die Bedingung nicht erfüllt und die Anforderung führt standardmäßig zu default-deny. Sie können das Ergebnis von default-deny ändern, indem Sie Richtlinie A2 erstellen, die einer Anforderung den Zugriff ausdrücklich verweigert, wenn sie aus der Antarktis stammt. Diese Richtlinie ist in der folgenden Abbildung dargestellt.



Wenn ein Benutzer eine Anforderung aus der Antarktis sendet, ist die Bedingung erfüllt und die Anforderung führt zu explicit-deny.

Der Unterschied zwischen default-deny und explicit-deny ist wichtig, da allow zwar eine standardmäßige Ablehnung, aber keine explizite Zugriffsverweigerung außer Kraft setzen kann. Beispiel: Richtlinie B lässt Anforderungen zu, wenn sie am 1. Juni 2010 eingehen. Das folgende Diagramm vergleicht die Kombination dieser Richtlinie mit Richtlinie A1 und Richtlinie A2.



In Szenario 1 liefert Richtlinie A1 das Ergebnis default-deny und Richtlinie B allow, weil die Richtlinie Anforderungen zulässt, die am 1. Juni 2010 eingehen. Das allow aus Richtlinie B überschreibt default-deny aus Richtlinie A1 und die Anforderung wird zugelassen.

In Szenario 2 hat Richtlinie B2 explicit-deny und Richtlinie B allow zur Folge. Das allow aus Richtlinie B wird durch explicit-deny aus Richtlinie A2 überschrieben und die Anforderung wird abgelehnt.

Einschränkungen der benutzerdefinierten Amazon SQS SQS-Richtlinien

Kontoübergreifender Zugriff

Kontoübergreifende Berechtigungen gelten nicht für die folgenden Aktionen:

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

Bedingungsschlüssel

Derzeit unterstützt Amazon SQS nur einen eingeschränkten Teilbereich der [in IAM verfügbaren Bedingungsschlüssel](#). Weitere Informationen finden Sie unter [Amazon-SQS-API-Berechtigungen: Referenz für Aktionen und Ressourcen](#).

Beispiele für eine benutzerdefinierte Sprache der Zugriffsrichtlinie von Amazon SQS

Nachfolgend sind Beispiele typischer Amazon-SQS-Zugriffsrichtlinien aufgeführt.

Beispiel 1: Einem Konto eine Berechtigung erteilen

Das folgende Beispiel für eine Amazon-SQS-Richtlinie gewährt AWS-Konto 111122223333 die Berechtigung zum Senden an und zum Empfangen von queue2, in Eigentümerschaft von AWS-Konto 444455556666.

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"
  }]
}
```

Beispiel 2: Einem oder mehreren Konten eine Berechtigung erteilen

Das folgende Beispiel für eine Amazon SQS SQS-Richtlinie gewährt eine oder mehrere AWS-Konten Zugriff auf Warteschlangen, die Ihrem Konto gehören, für einen bestimmten Zeitraum. Es ist erforderlich, diese Richtlinie zu erstellen und mit der Aktion [SetQueueAttributes](#) zu Amazon SQS hochzuladen, da die Aktion [AddPermission](#) bei der Zugriffserteilung für eine Warteschlange keine Angabe einer Zeitbeschränkung erlaubt.

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"
        }
    }
}
}]
}

```

Beispiel 3: Berechtigung für Anforderungen von Amazon-EC2-Instances erteilen

Das folgende Beispiel für eine Amazon-SQS-Richtlinie erteilt den Zugriff für Anforderungen, die von Amazon-EC2-Instances stammen. Dieses Beispiel basiert auf dem Beispiel "[Beispiel 2: Einem oder mehreren Konten eine Berechtigung erteilen](#)": Es beschränkt den Zugriff auf die Zeit vor dem 30. Juni 2009 12.00 Uhr (UTC) und auf den IP-Adressbereich 203.0.113.0/24. Es ist erforderlich, diese Richtlinie zu erstellen und mit der Aktion [SetQueueAttributes](#) in Amazon SQS hochzuladen, da die Aktion [AddPermission](#) bei der Zugriffserteilung für eine Warteschlange keine Angabe einer IP-Adressbeschränkung erlaubt.

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"
    }
  }
}
}]
}

```

Beispiel 4: Zugriff für ein bestimmtes Konto verweigern

Das folgende Beispiel einer Amazon SQS SQS-Richtlinie verweigert einen bestimmten AWS-Konto Zugriff auf Ihre Warteschlange. Dieses Beispiel baut auf dem Beispiel ["Beispiel 1: Einem Konto eine Berechtigung erteilen"](#) auf: Es verweigert den Zugriff auf die angegebene Datei. AWS-Konto Es ist erforderlich, diese Richtlinie zu erstellen und mit der Aktion [SetQueueAttributes](#) zu Amazon SQS hochzuladen, da die Aktion [AddPermission](#) keine Zugriffsverweigerung für eine Warteschlange erlaubt (sie lässt nur die Erteilung des Zugriffs auf eine Warteschlange zu).

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"
}

```

```
}]
}
```

Beispiel 5: Verweigerung des Zugriffs, wenn dieser nicht von einem VPC-Endpoint aus erfolgt

Das folgende Beispiel für eine Amazon-SQS-Richtlinie beschränkt den Zugriff auf queue1:

111122223333 kann die Aktionen [SendMessage](#) und [ReceiveMessage](#) nur von der VPC-Endpoint-ID vpce-1a2b3c4d aus durchführen (angegeben mit der `aws:sourceVpce`-Bedingung). Weitere Informationen finden Sie unter [Endpunkte von Amazon Virtual Private Cloud für Amazon SQS](#).

Note

- Die `aws:sourceVpce`-Bedingung benötigt keine ARN für die VPC-Endpoint-Ressource, sondern nur die VPC-Endpoint-ID.
- Sie können das folgende Beispiel so abändern, dass alle Aktionen auf einen spezifischen VPC-Endpoint eingeschränkt werden, indem Sie in der zweiten Anweisung alle Amazon-SQS-Aktionen (`sqs:*`) verweigern. Durch eine solche Richtlinienanweisung würde jedoch festgelegt, dass alle Aktionen (einschließlich von zum Ändern von Warteschlangen-Berechtigungen erforderlichen administrativen Aktionen) über den spezifischen, in der Richtlinie definierten VPC-Endpoint erfolgen müssen; hierdurch würde dem Benutzer in der Zukunft das Ändern von Warteschlangen-Berechtigungen unmöglich gemacht.

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"
    }
  }
}
]
```

Verwenden von temporären Sicherheitsanmeldeinformationen mit Amazon SQS

Mit IAM können Sie nicht nur Benutzer mit eigenen Sicherheitsanmeldedaten erstellen, sondern auch jedem Benutzer temporäre Sicherheitsanmeldedaten zuweisen, sodass der Benutzer auf Ihre AWS Dienste und Ressourcen zugreifen kann. Sie können Benutzer verwalten, die AWS-Konten haben. Sie können auch Benutzer für Ihr System verwalten, die noch keine haben AWS-Konten (Verbundbenutzer). Darüber hinaus können Anwendungen, die Sie für den Zugriff auf Ihre AWS Ressourcen erstellen, auch als „Benutzer“ betrachtet werden.

Sie können diese temporären Sicherheitsanmeldeinformationen für das Erstellen von Anforderungen an Amazon SQS verwenden. Die API-Bibliotheken berechnen anhand dieser Anmeldeinformationen den notwendigen Signaturwert, um Ihre Anforderung zu authentifizieren. Wenn Sie beim Senden von Anfragen abgelaufene Anmeldeinformationen verwenden, lehnt Amazon SQS die Anfrage ab.

Note

Sie können keine Richtlinie auf der Basis von temporären Anmeldeinformationen festlegen.

Voraussetzungen

1. Verwenden Sie IAM zum Erstellen von temporären Sicherheitsanmeldeinformationen:
 - Sicherheits-Token
 - Access Key ID
 - Secret Access Key
2. Bereiten Sie die zu signierende Zeichenfolge mit der temporären Zugriffsschlüssel-ID und dem Sicherheits-Token vor.
3. Verwenden Sie den temporären geheimen Zugriffsschlüssel anstelle Ihres eigenen geheimen Zugriffsschlüssels, um Ihre Abfrage-API-Anforderung zu signieren.

Note

Wenn Sie die signierte Abfrage-API senden, verwenden Sie die temporäre Zugriffsschlüssel-ID anstelle Ihrer eigenen Zugriffsschlüssel-ID und schließen Sie das Sicherheits-Token ein. Weitere Informationen zur IAM-Unterstützung für temporäre Sicherheitsanmeldedaten finden Sie unter [Granting Temporary Access to Your AWS Resources](#) im IAM-Benutzerhandbuch.

So rufen Sie eine Amazon-SQS-Abfrage-API-Aktion mit temporären Sicherheitsanmeldeinformationen auf

1. Fordern Sie ein temporäres Sicherheitstoken an mit AWS Identity and Access Management. Weitere Informationen finden Sie unter [Erstellen temporärer Sicherheitsanmeldeinformationen für IAM-Benutzer](#) im IAM-Benutzerhandbuch.

IAM gibt ein Sicherheits-Token, eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel zurück.

2. Bereiten Sie Ihre Abfrage mit der temporären Zugriffsschlüssel-ID anstelle Ihrer eigenen Zugriffsschlüssel-ID vor und schließen Sie das Sicherheits-Token ein. Signieren Sie Ihre Anforderung mit dem temporären geheimen Zugriffsschlüssel anstelle Ihres eigenen Zugriffsschlüssels.
3. Senden Sie die signierte Abfragezeichenfolge mit der temporären Zugriffsschlüssel-ID und dem Sicherheits-Token.

Das folgende Beispiel zeigt, wie Sie temporäre Sicherheitsanmeldeinformationen zum Authentifizieren einer Amazon-SQS-Anforderung verwenden. Die Struktur von **AUTHPARAMS** hängt davon ab, wie Sie Ihre API-Anforderung signieren. Weitere Informationen finden Sie unter [AWS API-Anfragen signieren](#) in der Allgemeinen Referenz zu Amazon Web Services.

```
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=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE  
&Version=2012-11-05  
&AUTHPARAMS
```

Das folgende Beispiel verwendet temporäre Sicherheitsanmeldeinformationen, um zwei Nachrichten mit der SendMessageBatch-Aktion zu senden.

```
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
```

Zugriffsverwaltung für verschlüsselte Amazon SQS SQS-Warteschlangen mit Richtlinien für geringste Rechte

Sie können Amazon SQS verwenden, um vertrauliche Daten zwischen Anwendungen auszutauschen, indem Sie serverseitige Verschlüsselung (SSE) verwenden, die in [AWS Key Management Service \(KMS\)](#) integriert ist. Mit der Integration von Amazon SQS und können Sie

die Schlüssel AWS KMS, die Amazon SQS schützen, sowie die Schlüssel, die Ihre anderen AWS Ressourcen schützen, zentral verwalten.

Mehrere AWS Dienste können als Ereignisquellen dienen, die Ereignisse an Amazon SQS senden. Um einer Ereignisquelle den Zugriff auf die verschlüsselte Amazon SQS SQS-Warteschlange zu ermöglichen, müssen Sie die Warteschlange mit einem [vom Kunden verwalteten Schlüssel](#) AWS KMS konfigurieren. Verwenden Sie dann die Schlüsselrichtlinie, damit der Service die erforderlichen AWS KMS API-Methoden verwenden kann. Der Service benötigt außerdem Berechtigungen zur Authentifizierung des Zugriffs, damit die Warteschlange Ereignisse senden kann. Sie können dies erreichen, indem Sie eine Amazon-SQS-Richtlinie verwenden. Dabei handelt es sich um eine ressourcenbasierte Richtlinie, mit der Sie den Zugriff auf die Amazon-SQS-Warteschlange und ihre Daten kontrollieren können.

Die folgenden Abschnitte enthalten Informationen darüber, wie Sie den Zugriff auf Ihre verschlüsselte Amazon SQS SQS-Warteschlange mithilfe der Amazon SQS SQS-Richtlinie und der AWS KMS Schlüsselrichtlinie kontrollieren können. Die Richtlinien in diesem Handbuch helfen Ihnen dabei, die [geringsten Berechtigungen](#) zu erlangen.

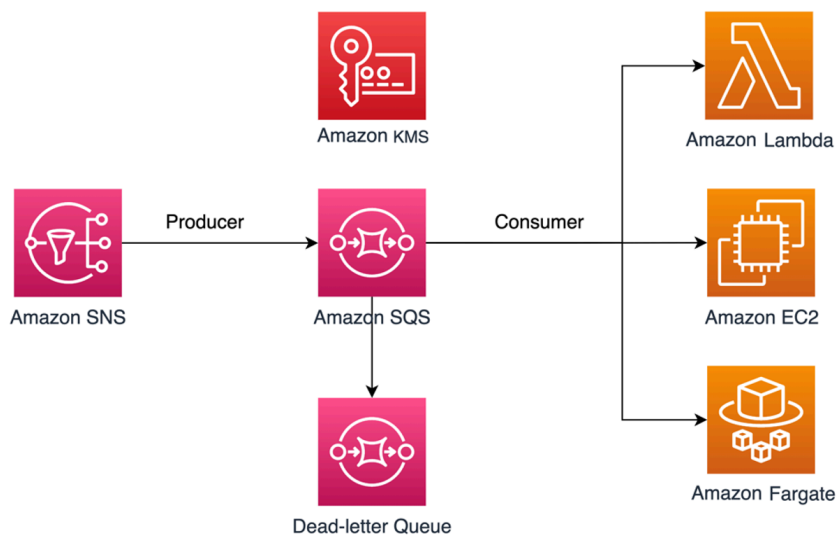
In diesem Leitfaden wird auch beschrieben, wie ressourcenbasierte Richtlinien das [Confused-Deputy-Problem](#) mithilfe der globalen IAM-Bedingungskontextschlüssel [aws:SourceArn](#), [aws:SourceAccount](#) und [aws:PrincipalOrgID](#) lösen.

Themen

- [-Übersicht](#)
- [Schlüsselrichtlinie mit den geringsten Berechtigungen für Amazon SQS](#)
- [Amazon-SQS-Richtlinienerklärungen für die Warteschlange für unzustellbare Nachrichten](#)
- [Vermeidung des serviceübergreifenden Confused-Deputy-Problems](#)
- [Verwenden von IAM Access Analyzer, um den kontoübergreifenden Zugriff zu überprüfen](#)

-Übersicht

In diesem Thema werden wir Sie durch einen häufigen Anwendungsfall führen, um zu veranschaulichen, wie Sie die Schlüsselrichtlinie und die Amazon-SQS-Warteschlangenrichtlinie erstellen können. Dieser Anwendungsfall wird im folgenden Bild veranschaulicht.



In diesem Beispiel ist der Nachrichtenproduzent ein [Amazon Simple Notification Service \(SNS\)](#)-Thema, das so konfiguriert ist, dass es per Fanout Nachrichten in Ihre verschlüsselte Amazon-SQS-Warteschlange überträgt. Der Nachrichtenkonsument ist ein Computing-Service, z. B. eine [AWS Lambda](#)-Funktion, eine [Amazon Elastic Compute Cloud \(EC2\)](#)-Instance oder ein [AWS Fargate](#)-Container. Ihre Amazon-SQS-Warteschlange ist dann so konfiguriert, dass fehlgeschlagene Nachrichten an eine [Warteschlange für unzustellbare Nachrichten \(DLQ\)](#) gesendet werden. Dies ist nützlich für das Debuggen Ihrer Anwendung oder Ihres Messaging-Systems, da DLQs Sie nicht konsumierte Nachrichten isolieren können, um festzustellen, warum ihre Verarbeitung nicht erfolgreich war. In der in diesem Thema definierten Lösung wird ein Computing-Service wie eine Lambda-Funktion verwendet, um Nachrichten zu verarbeiten, die in der Amazon-SQS-Warteschlange gespeichert sind. Wenn sich der Nachrichtenverbraucher in einer Virtual Private Cloud (VPC) befindet, können Sie mit der in diesem Handbuch enthaltenen [DenyReceivingIfNotThroughVPCE](#)-Richtlinienanweisung den Nachrichtenempfang auf diese spezifische VPC beschränken.

Note

Dieses Handbuch enthält nur die erforderlichen IAM-Berechtigungen in Form von Richtlinienanweisungen. Um die Richtlinie zu erstellen, müssen Sie die Anweisungen zu Ihrer Amazon SQS SQS-Richtlinie oder Ihrer AWS KMS wichtigsten Richtlinie hinzufügen. Dieses Handbuch enthält keine Anweisungen zum Erstellen der Amazon SQS SQS-Warteschlange oder des AWS KMS Schlüssels. Anweisungen zum Erstellen dieser Ressourcen finden Sie unter [Erstellen einer Amazon-SQS-Warteschlange](#) und [Erstellen von Schlüsseln](#).

Die in diesem Handbuch definierte Amazon-SQS-Richtlinie unterstützt nicht die direkte Weiterleitung von Nachrichten an dieselbe oder eine andere Amazon-SQS-Warteschlange.

Schlüsselrichtlinie mit den geringsten Berechtigungen für Amazon SQS

In diesem Abschnitt beschreiben wir die erforderlichen Berechtigungen mit den geringsten Rechten AWS KMS für den vom Kunden verwalteten Schlüssel, den Sie zum Verschlüsseln Ihrer Amazon SQS SQS-Warteschlange verwenden. Mit diesen Genehmigungen können Sie den Zugriff nur auf die vorgesehenen Entitäten beschränken und gleichzeitig die geringsten Berechtigungen implementieren. Die Schlüsselrichtlinie muss aus den folgenden Richtlinienanweisungen bestehen, die wir im Folgenden ausführlich beschreiben:

- [Erteilen Sie Administratorrechte für den Schlüssel AWS KMS](#)
- [Gewährt Lesezugriff auf die wichtigsten Metadaten](#)
- [Gewährt Amazon SNS KMS-Berechtigungen, um Nachrichten für die Warteschlange zu veröffentlichen](#)
- [Konsumenten gestatten, Nachrichten aus der Warteschlange zu entschlüsseln](#)

Erteilen Sie Administratorrechte für den Schlüssel AWS KMS

Um einen AWS KMS Schlüssel zu erstellen, müssen Sie AWS KMS Administratorberechtigungen für die IAM-Rolle bereitstellen, die Sie für die Bereitstellung des AWS KMS Schlüssels verwenden. Diese Administratorberechtigungen sind in der folgenden AllowKeyAdminPermissions-Richtlinienanweisung definiert. Wenn Sie diese Aussage zu Ihrer AWS KMS Schlüsselrichtlinie hinzufügen, achten Sie darauf, sie durch den Amazon-Ressourcennamen (ARN) der IAM-Rolle zu *<admin-role ARN>* ersetzen, die für die Bereitstellung des AWS KMS Schlüssels, die Verwaltung des AWS KMS Schlüssels oder beides verwendet wurde. Dies kann die IAM-Rolle Ihrer Bereitstellungs-Pipeline oder die [Administratorrolle für Ihre Organisation](#) in Ihren [AWS - Organisationen](#) sein.

```
{
  "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

In einer AWS KMS Schlüsselrichtlinie muss der Wert des Resource Elements sein*, was „dieser AWS KMS Schlüssel“ bedeutet. Das Sternchen (*) kennzeichnet den AWS KMS Schlüssel, dem die Schlüsselrichtlinie zugeordnet ist.

Gewährt Lesezugriff auf die wichtigsten Metadaten

Um anderen IAM-Rollen schreibgeschützten Zugriff auf Ihre wichtigsten Metadaten zu gewähren, fügen Sie die `AllowReadAccessToKeyMetaData`-Anweisung zu Ihrer Schlüsselrichtlinie hinzu. Mit der folgenden Anweisung können Sie beispielsweise alle AWS KMS Schlüssel in Ihrem Konto zu Prüfungszwecken auflisten. Diese Anweisung gewährt dem AWS Root-Benutzer nur Lesezugriff auf die Schlüsselmetadaten. Daher kann jeder IAM-Prinzipal in dem Konto auf die Schlüsselmetadaten zugreifen, wenn seine identitätsbasierten Richtlinien über die in der folgenden Anweisung aufgeführten Berechtigungen verfügen: `kms:Describe*`, `kms:Get*` und `kms:List*`. Stellen Sie sicher, dass Sie es durch Ihre eigenen Informationen `<account-ID>` ersetzen.

```

{
  "Sid": "AllowReadAccesssToKeyMetaData",
  "Effect": "Allow",

```

```
"Principal": {
  "AWS": [
    "arn:aws:iam::<accountID>:root"
  ]
},
"Action": [
  "kms:Describe*",
  "kms:Get*",
  "kms:List*"
],
"Resource": "*"
}
```

Gewährt Amazon SNS KMS-Berechtigungen, um Nachrichten für die Warteschlange zu veröffentlichen

Damit Ihr Amazon-SNS-Thema Nachrichten in Ihrer verschlüsselten Amazon-SQS-Warteschlange veröffentlichen kann, fügen Sie die AllowSNSToSendToSQS-Richtlinienanweisung zu Ihrer Schlüsselrichtlinie hinzu. Diese Erklärung erteilt Amazon SNS die Erlaubnis, den AWS KMS Schlüssel zur Veröffentlichung in Ihrer Amazon SQS SQS-Warteschlange zu verwenden. Stellen Sie sicher, dass Sie es durch Ihre eigenen Informationen *<account-ID>* ersetzen.

 Note

Die Condition in der Erklärung angegebene Einschränkung beschränkt den Zugriff nur auf den Amazon SNS SNS-Service auf demselben AWS Konto.

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
}
```

```

"Condition": {
  "StringEquals": {
    "aws:SourceAccount": "<account-id>"
  }
}
}

```

Konsumenten gestatten, Nachrichten aus der Warteschlange zu entschlüsseln

Die folgende AllowConsumersToReceiveFromTheQueue-Anweisung gewährt dem Amazon-SQS-Nachrichtenkonsumenten die erforderlichen Berechtigungen zum Entschlüsseln von Nachrichten, die aus der verschlüsselten Amazon-SQS-Warteschlange empfangen wurden. Wenn Sie die Richtlinienanweisung anhängen, *<consumer's runtime role ARN>* ersetzen Sie sie durch den ARN der IAM-Laufzeitrolle des Nachrichtenkonsumenten.

```

{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}

```

Richtlinie für Amazon SQS mit den geringsten Berechtigungen

Dieser Abschnitt führt Sie durch die Amazon-SQS-Warteschlangenrichtlinien mit den geringsten Berechtigungen für den in diesem Handbuch behandelten Anwendungsfall (z. B. Amazon SNS zu Amazon SQS). Die definierte Richtlinie soll unbeabsichtigten Zugriff verhindern, indem eine Kombination aus den beiden Anweisungen Deny und Allow verwendet wird. Die Allow-Anweisungen gewähren Zugriff auf die intendierte(n) Entität(en). Die Deny-Anweisungen verhindern, dass andere unbeabsichtigte Entitäten auf die Amazon-SQS-Warteschlange zugreifen, während die beabsichtigte Entität innerhalb der Richtlinienbedingung ausgeschlossen wird.

Die Amazon-SQS-Richtlinie umfasst die folgenden Anweisungen, die wir im Folgenden ausführlich beschreiben:

- [Beschränken der Amazon-SQS-Verwaltungsberechtigungen](#)
- [Beschränken der Amazon-SQS-Warteschlangenaktionen der angegebenen Organisation](#)
- [Erteilen von Amazon-SQS-Berechtigungen für Konsumenten](#)
- [Erzwingen der Verschlüsselung von Daten während der Übertragung](#)
- [Einschränkung der Nachrichtenübertragung auf ein bestimmtes Amazon-SNS-Thema](#)
- [\(Optional\) Einschränken des Nachrichtenempfangs auf einen bestimmten VPC-Endpunkt](#)

Beschränken der Amazon-SQS-Verwaltungsberechtigungen

Die folgende `RestrictAdminQueueActions`-Richtlinienanweisung beschränkt die Amazon-SQS-Verwaltungsberechtigungen nur auf die IAM-Rolle(n), die Sie für die Bereitstellung der Warteschlange, die Verwaltung der Warteschlange oder für beide verwenden. Ersetzen Sie *<placeholder values>* durch Ihre eigenen Informationen. Geben Sie den ARN der IAM-Rolle an, die für die Bereitstellung der Amazon SQS SQS-Warteschlange verwendet wurde, sowie den ARNs aller Administratorrollen, die über Amazon SQS SQS-Verwaltungsberechtigungen verfügen sollten.

```
{
  "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>"
      ]
    }
  }
}
```

Beschränken der Amazon-SQS-Warteschlangenaktionen der angegebenen Organisation

Verwenden Sie die folgende Anweisungen, um Ihre Amazon-SQS-Ressourcen vor externem Zugriff (Zugriff durch eine Entität außerhalb Ihrer [AWS -Organisation](#)) zu schützen. Diese Anweisung beschränkt den Zugriff auf die Amazon-SQS-Warteschlange auf die Organisation, die Sie in der Condition angeben. Stellen Sie sicher, dass Sie es durch den ARN der IAM-Rolle *<SQS queue ARN>* ersetzen, die für die Bereitstellung der Amazon SQS SQS-Warteschlange verwendet wurde *<org-id>*, und dann durch Ihre Organisations-ID.

```
{
  "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>"
      ]
    }
  }
}
```

Erteilen von Amazon-SQS-Berechtigungen für Konsumenten

Um Nachrichten aus der Amazon-SQS-Warteschlange zu empfangen, müssen Sie dem Nachrichtenverbraucher die erforderlichen Berechtigungen erteilen. Die folgende Richtlinienanweisung gewährt dem von Ihnen angegebenen Konsumenten die erforderlichen Berechtigungen, um Nachrichten aus der Amazon-SQS-Warteschlange zu konsumieren. Achten Sie beim Hinzufügen der Anweisung zu Ihrer Amazon SQS SQS-Richtlinie darauf, sie durch den *<consumer's IAM runtime role ARN>* ARN der vom Verbraucher verwendeten IAM-

Laufzeitrolle und durch den ARN der IAM-Rolle zu ersetzen *<SQS queue ARN>*, die für die Bereitstellung der Amazon SQS SQS-Warteschlange verwendet wurde.

```
{
  "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>"
}
```

Um zu verhindern, dass andere Entitäten Nachrichten aus der Amazon-SQS-Warteschlange erhalten, fügen Sie die `DenyOtherConsumersFromReceiving`-Anweisung zu der Amazon-SQS-Warteschlangenrichtlinie hinzu. Diese Anweisung beschränkt den Nachrichtenverbrauch auf den von Ihnen angegebenen Konsumenten, so dass keine anderen Konsumenten Zugriff haben, selbst wenn ihnen ihre Identitätsberechtigungen Zugriff gewähren würden. Stellen Sie sicher, dass Sie *<SQS queue ARN>* und durch Ihre eigenen Informationen *<consumer's runtime role ARN>* ersetzen.

```
{
  "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>"
  }
}
}

```

Erzwingen der Verschlüsselung von Daten während der Übertragung

Die folgende DenyUnsecureTransport-Richtlinienanweisung verpflichtet die Konsumenten und Produzenten, sichere Kanäle (TLS-Verbindungen) zu verwenden, um Nachrichten aus der Amazon-SQS-Warteschlange zu senden und zu empfangen. Stellen Sie sicher, dass Sie es *<SQS queue ARN>* durch den ARN der IAM-Rolle ersetzen, die für die Bereitstellung der Amazon SQS SQS-Warteschlange verwendet wurde.

```

{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}

```

Einschränkung der Nachrichtenübertragung auf ein bestimmtes Amazon-SNS-Thema

Die folgende AllowSNSToSendToTheQueue-Richtlinienanweisung ermöglicht dem angegebenen Amazon-SNS-Thema, Nachrichten an die Amazon-SQS-Warteschlange zu senden. Stellen Sie sicher, dass Sie es durch den ARN der IAM-Rolle *<SQS queue ARN>* ersetzen, die für die Bereitstellung der Amazon SQS SQS-Warteschlange verwendet wurde *<SNS topic ARN>*, und durch das Amazon SNS SNS-Thema ARN.

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

Die folgende `DenyAllProducersExceptSNSFromSending`-Richtlinienanweisung verhindert, dass andere Produzenten Nachrichten an die Warteschlange senden. Ersetzen Sie `<SQS queue ARN>` und durch Ihre `<SNS topic ARN>` eigenen Informationen.

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(Optional) Einschränken des Nachrichtenempfangs auf einen bestimmten VPC-Endpunkt

Um den Empfang von Nachrichten nur auf einen bestimmten [VPC-Endpunkt](#) zu beschränken, fügen Sie Ihrer Amazon-SQS-Warteschlangenrichtlinie die folgende Richtlinienanweisung hinzu. Diese Anweisung verhindert, dass ein Nachrichtenkonsument Nachrichten aus der Warteschlange

empfängt, es sei denn, die Nachrichten stammen vom gewünschten VPC-Endpunkt. *<SQS queue ARN>* Ersetzen Sie durch den ARN der IAM-Rolle, die für die Bereitstellung der Amazon SQS SQS-Warteschlange verwendet wurde, und *<vpce_id>* durch die ID des VPC-Endpunkts.

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

Amazon-SQS-Richtlinienerklärungen für die Warteschlange für unzustellbare Nachrichten

Fügen Sie Ihrer DLQ-Zugriffsrichtlinie die folgenden Richtlinienerklärungen hinzu, die anhand ihrer Anweisungs-ID gekennzeichnet sind:

- RestrictAdminQueueActions
- DenyQueueActionsOutsideOrg
- AllowConsumersToReceiveFromTheQueue
- DenyOtherConsumersFromReceiving
- DenyUnsecureTransport

Zusätzlich zum Hinzufügen der obigen Richtlinienanweisungen zu Ihrer DLQ-Zugriffsrichtlinie sollten Sie auch eine Anweisung hinzufügen, um die Nachrichtenübertragung an Amazon-SQS-Warteschlangen einzuschränken, wie im folgenden Abschnitt beschrieben.

Einschränken der Nachrichtenübertragung auf Amazon-SQS-Warteschlangen

Um den Zugriff nur auf Amazon-SQS-Warteschlangen von demselben Konto aus zu beschränken, fügen Sie der DLQ-Warteschlangenrichtlinie die folgende DenyAnyProducersExceptSQS-Richtlinienanweisung hinzu. Diese Anweisung beschränkt die Nachrichtenübertragung nicht auf eine

bestimmte Warteschlange, da Sie die DLQ bereitstellen müssen, bevor Sie die Hauptwarteschlange erstellen, so dass Sie den Amazon-SQS-ARN nicht kennen, wenn Sie die DLQ erstellen. Wenn Sie den Zugriff auf nur eine Amazon-SQS-Warteschlange beschränken möchten, ändern Sie den `aws:SourceArn` in der Condition zu dem ARN Ihrer Amazon-SQS-Quellwarteschlange, wenn Sie diesen kennen.

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

Important

Die in diesem Handbuch definierten Amazon-SQS-Warteschlangenrichtlinien beschränken die `sqs:PurgeQueue`-Aktion nicht auf eine oder mehrere bestimmte IAM-Rolle(n). Die `sqs:PurgeQueue`-Aktion ermöglicht Ihnen, alle Nachrichten in der Amazon-SQS-Warteschlange zu löschen. Sie können diese Aktion auch verwenden, um Änderungen am Nachrichtenformat vorzunehmen, ohne die Amazon-SQS-Warteschlange zu ersetzen. Beim Debuggen einer Anwendung können Sie die Amazon-SQS-Warteschlange leeren, um potenziell fehlerhafte Nachrichten zu entfernen. Beim Testen der Anwendung können Sie ein hohes Nachrichtenvolumen durch die Amazon-SQS-Warteschlange leiten und dann die Warteschlange leeren, um neu zu beginnen, bevor Sie mit der Produktion beginnen. Der Grund dafür, dass diese Aktion nicht auf eine bestimmte Rolle beschränkt wird, liegt darin, dass diese Rolle bei der Bereitstellung der Amazon-SQS-Warteschlange möglicherweise nicht bekannt ist. Sie müssen diese Berechtigung zur identitätsbasierten Richtlinie der Rolle hinzufügen, um die Warteschlange löschen zu können.

Vermeidung des serviceübergreifenden Confused-Deputy-Problems

Das [Confused-Deputy-Problem](#) ist ein Sicherheitsproblem, bei dem eine Entität, die nicht über die Berechtigung zum Ausführen einer Aktion verfügt, eine Entität mit größeren Rechten zwingen kann, die Aktion auszuführen. Um dies zu verhindern, AWS stellt Tools bereit, mit denen Sie Ihr Konto schützen können, wenn Sie Dritten (auch als kontoübergreifender Dienst bezeichnet) oder anderen AWS Diensten (bekannt als dienstübergreifender Zugriff) Zugriff auf Ressourcen in Ihrem Konto gewähren. Die Richtlinienanweisungen in diesem Abschnitt können Ihnen helfen, das serviceübergreifende Confused-Deputy-Problem zu vermeiden.

Ein serviceübergreifender Identitätswechsel kann auftreten, wenn ein Service (der Anruf-Service) einen anderen Service anruft (den aufgerufenen Service). Der Aufruf-Service kann so manipuliert werden, dass er seine Berechtigungen verwendet, um auf die Ressourcen eines anderen Kunden zu reagieren, auf die er sonst nicht zugreifen dürfte. Um sich vor diesem Problem zu schützen, verwenden die in diesem Beitrag definierten ressourcenbasierten Richtlinien die globalen IAM-Bedingungsschlüssel [aws:SourceArn](#), [aws:SourceAccount](#) und [aws:PrincipalOrgID](#). Dadurch werden die Berechtigungen eines Dienstes auf eine bestimmte Ressource, ein bestimmtes Konto oder eine bestimmte Organisation in AWS Organizations beschränkt.

Verwenden von IAM Access Analyzer, um den kontoübergreifenden Zugriff zu überprüfen

Sie können [AWS IAM Access Analyzer](#) verwenden, um Ihre Amazon SQS SQS-Warteschlangenrichtlinien und AWS KMS Schlüsselrichtlinien zu überprüfen und Sie zu benachrichtigen, wenn eine Amazon SQS SQS-Warteschlange oder ein AWS KMS Schlüssel Zugriff auf eine externe Entität gewährt. IAM Access Analyzer hilft bei der Identifizierung von [Ressourcen](#) in Ihrer Organisation und Ihren Konten, die außerhalb Ihrer Vertrauenszone weitergegeben werden. Bei dieser Vertrauenszone kann es sich um ein AWS Konto oder die Organisation innerhalb von AWS Organizations handeln, die Sie bei der Aktivierung von IAM Access Analyzer angeben.

IAM Access Analyzer identifiziert Ressourcen, die mit externen Prinzipalen gemeinsam genutzt werden, indem er die ressourcenbasierten Richtlinien in Ihrer Umgebung anhand von logischer Argumentation analysiert. AWS Für jede Instance einer Ressource, die außerhalb Ihrer Zone gemeinsam genutzt wird, erstellt Access Analyzer eine Erkenntnis. Die [Ergebnisse](#) enthalten Informationen über den Zugang und den externen Prinzipal, dem dieser gewährt wurde. Sie können die Ergebnisse prüfen, um festzustellen, ob der Zugriff beabsichtigt und sicher oder ob er unbeabsichtigt ist und ein Sicherheitsrisiko darstellt. Überprüfen Sie bei unbeabsichtigtem Zugriff die betroffene Richtlinie und korrigieren Sie diese. In diesem [Blogbeitrag](#) finden Sie weitere Informationen darüber, wie AWS IAM Access Analyzer unbeabsichtigte Zugriffe auf Ihre Ressourcen erkennt. AWS

Weitere Informationen zu AWS IAM Access Analyzer finden Sie in der [AWS IAM Access Analyzer-Dokumentation](#).

Amazon-SQS-API-Berechtigungen: Referenz für Aktionen und Ressourcen

Wenn Sie [Zugriffskontrolle](#) einrichten und Berechtigungsrichtlinien erstellen, die Sie einer IAM-Identität anfügen können, verwenden Sie die folgende Tabelle als Referenz. Die enthält jede Amazon Simple Queue Service-Aktion, die entsprechenden Aktionen, für die Sie Berechtigungen zur Ausführung der Aktion erteilen können, und die AWS Ressource, für die Sie die Berechtigungen erteilen können.

Die Aktionen geben Sie im Feld Action und den Wert für die Ressource im Feld Resource der Richtlinie an. Um eine Aktion anzugeben, verwenden Sie das Präfix `sqs:` gefolgt vom Namen der Aktion (z. B. `sqs:CreateQueue`).

Derzeit unterstützt Amazon SQS die [in IAM verfügbaren globalen Bedingungskontextschlüssel](#).

Amazon-Simple-Queue-Service-API und erforderliche Berechtigungen für Aktionen

[AddPermission](#)

Aktion(en): `sqs:AddPermission`

Ressource: `arn:aws:sqs:region:account_id:queue_name`

[ChangeMessageVisibility](#)

Aktion(en): `sqs:ChangeMessageVisibility`

Ressource: `arn:aws:sqs:region:account_id:queue_name`

[ChangeMessageVisibilityBatch](#)

Aktion(en): `sqs:ChangeMessageVisibilityBatch`

Ressource: `arn:aws:sqs:region:account_id:queue_name`

[CreateQueue](#)

Aktion(en): `sqs:CreateQueue`

Ressource: `arn:aws:sqs:region:account_id:queue_name`

[DeleteMessage](#)

Aktion(en): `sqs:DeleteMessage`

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

DeleteMessageBatch

Aktion(en): sqs:DeleteMessageBatch

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

DeleteQueue

Aktion(en): sqs:DeleteQueue

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueAttributes

Aktion(en): sqs:GetQueueAttributes

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueUrl

Aktion(en): sqs:GetQueueUrl

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListDeadLetterSourceQueues

Aktion(en): sqs>ListDeadLetterSourceQueues

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListQueues

Aktion(en): sqs>ListQueues

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListQueueTags

Aktion(en): sqs>ListQueueTags

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

PurgeQueue

Aktion(en): sqs:PurgeQueue

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ReceiveMessage

Aktion(en): sqs:ReceiveMessage

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

RemovePermission

Aktion(en): sqs:RemovePermission

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

SendMessage und SendMessageBatch

Aktion(en): sqs:SendMessage

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

SetQueueAttributes

Aktion(en): sqs:SetQueueAttributes

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

TagQueue

Aktion(en): sqs:TagQueue

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

UntagQueue

Aktion(en): sqs:UntagQueue

Ressource: arn:aws:sqs:*region*:*account_id*:*queue_name*

Protokollierung und Überwachung in Amazon SQS

Amazon Simple Queue Service ist in einen Service integriert [AWS CloudTrail](#), der eine Aufzeichnung der von einem Benutzer, einer Rolle oder einem ausgeführten Aktionen bereitstellt AWS-Service. CloudTrail erfasst alle API-Aufrufe für Amazon SQS als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der Amazon SQS SQS-Konsole und Code-Aufrufe der Amazon SQS SQS-API-Operationen. Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an Amazon SQS gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, den Zeitpunkt der Anfrage und weitere Details ermitteln.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Anmeldeinformationen des Root-Benutzers oder des Benutzers gestellt wurde.
- Die Anforderung wurde im Namen eines IAM-Identity-Center-Benutzers erstellt.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde.

CloudTrail ist in Ihrem aktiv AWS-Konto, wenn Sie das Konto erstellen, und Sie haben automatisch Zugriff auf den CloudTrail Event-Verlauf. Der CloudTrail Ereignisverlauf bietet eine einsehbare, durchsuchbare, herunterladbare und unveränderliche Aufzeichnung der aufgezeichneten Verwaltungsereignisse der letzten 90 Tage in einem AWS-Region. Weitere Informationen finden Sie im AWS CloudTrail Benutzerhandbuch unter [Arbeiten mit dem CloudTrail Ereignisverlauf](#). Für die Anzeige des Eventverlaufs CloudTrail fallen keine Gebühren an.

Für eine fortlaufende Aufzeichnung der Ereignisse in AWS-Konto den letzten 90 Tagen erstellen Sie einen Trail- oder [CloudTrailLake-Event-Datenspeicher](#).

CloudWatch Amazon-Alarme

Überwachen Sie eine einzelne Metrik über einen von Ihnen festgelegten Zeitraum und ergreifen Sie eine oder mehrere Aktionen auf der Grundlage des Werts der Metrik im Verhältnis zu einem definierten Schwellenwert über mehrere Zeiträume. Sie können beispielsweise einen CloudWatch Alarm so konfigurieren, dass er eine Benachrichtigung an ein Amazon SNS SNS-Thema sendet oder eine Aktion zum Senden einer Nachricht an eine Amazon SQS SQS-Warteschlange auslöst. CloudWatch Alarme führen keine Aktionen aus, nur weil sie sich in einem bestimmten Zustand befinden. Der Status muss sich ändern und für eine bestimmte Anzahl von Zeiträumen in diesem Zustand bleiben.

Weitere Informationen erhalten Sie unter [CloudWatch Alarme für Amazon SQS-Metriken erstellen](#) und [Alarme für Warteschlangen mit unbestätigten Briefen mithilfe von Amazon erstellen CloudWatch](#).

CloudWatch Amazon-Protokolle

Überwachen, speichern und greifen Sie auf Protokolldateien zu Amazon SQS zu, indem Sie Ihre Anwendungen oder Lambda-Funktionen konfigurieren, die Nachrichten verarbeiten, um

Protokolle an Logs zu CloudWatch senden. Sie können diese Protokolle verwenden, um die Nachrichtenverarbeitung zu analysieren, Probleme zu debuggen und die Leistung Ihrer Amazon SQS SQS-Workflows zu überwachen.

Weitere Informationen finden Sie unter [Protokollieren von Amazon Simple Queue Service API-Aufrufen mit AWS CloudTrail](#).

CloudWatch Amazon-Veranstaltungen

Verwenden Sie Amazon CloudWatch Events, um Änderungen oder bestimmte Ereignisse in Ihrer AWS Umgebung zu erkennen und sie an eine Amazon SQS-Warteschlange weiterzuleiten. Auf diese Weise können Sie Ereignisdaten erfassen, Workflows auslösen oder Ereignisse für die spätere Verarbeitung speichern.

Weitere Informationen finden Sie [Automatisieren von Benachrichtigungen von AWS Services an Amazon SQS mithilfe von Amazon EventBridge](#) in diesem Handbuch und [EventBridge die Entwicklung von Amazon CloudWatch Events](#) im EventBridge Amazon-Benutzerhandbuch.

AWS CloudTrail Logs

CloudTrail erfasst eine detaillierte Aufzeichnung der Aktionen, die in Amazon SQS von Benutzern, Rollen oder AWS-Services ausgeführt wurden. Diese Protokolle ermöglichen es Ihnen, API-Aufrufe wie [SendMessageReceiveMessage](#), oder nachzuverfolgen und wichtige Informationen zu liefern [DeleteQueue](#), z. B. wer die Anfrage gestellt hat, wann sie erfolgt ist und die ursprüngliche IP-Adresse.

Weitere Informationen finden Sie unter [Protokollieren von Amazon Simple Queue Service API-Aufrufen mit AWS CloudTrail](#).

AWS Trusted Advisor

Trusted Advisor verwendet Best Practices, die in der AWS Kundenbetreuung entwickelt wurden, um Sie bei der Optimierung Ihrer Amazon SQS SQS-Nutzung zu unterstützen. Es überprüft Ihre Amazon SQS SQS-Warteschlangen und bietet umsetzbare Empfehlungen zur Erhöhung der Sicherheit, Verbesserung der Zuverlässigkeit der Nachrichtenverarbeitung und Kostensenkung. Es kann beispielsweise vorgeschlagen werden, Warteschlangen für unzustellbare Nachrichten zu aktivieren oder Ihre Richtlinien für den Warteschlangenzugriff zu verbessern, um einen sicheren Betrieb zu gewährleisten.

Weitere Informationen finden Sie unter [AWS Trusted Advisor](#) im Support -Benutzerhandbuch.

CloudTrail Pfade

Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Alle mit dem erstellten Pfade AWS-Managementkonsole sind regionsübergreifend. Sie können mithilfe von AWS CLI einen Einzel-Region- oder einen Multi-Region-Trail erstellen. Es wird empfohlen, einen Trail mit mehreren Regionen zu erstellen, da Sie alle Aktivitäten AWS-Regionen in Ihrem Konto erfassen. Wenn Sie einen Einzel-Region-Trail erstellen, können Sie nur die Ereignisse anzeigen, die im AWS-Region des Trails protokolliert wurden. Weitere Informationen zu Trails finden Sie unter [Erstellen eines Trails für Ihr AWS-Konto](#) und [Erstellen eines Trails für eine Organisation](#) im AWS CloudTrail -Benutzerhandbuch.

Sie können eine Kopie Ihrer laufenden Verwaltungsereignisse kostenlos an Ihren Amazon S3 S3-Bucket senden, CloudTrail indem Sie einen Trail erstellen. Es fallen jedoch Amazon S3 S3-Speichergebühren an. Weitere Informationen zur CloudTrail Preisgestaltung finden Sie unter [AWS CloudTrail Preise](#). Informationen zu Amazon-S3-Preisen finden Sie unter [Amazon S3 – Preise](#).

CloudTrail Datenspeicher für Ereignisse in Lake

CloudTrail Mit Lake können Sie SQL-basierte Abfragen für Ihre Ereignisse ausführen. CloudTrail [Lake konvertiert bestehende Ereignisse im zeilenbasierten JSON-Format in das Apache ORC-Format](#). ORC ist ein spaltenförmiges Speicherformat, das für den schnellen Abruf von Daten optimiert ist. Die Ereignisse werden in Ereignisdatenspeichern zusammengefasst, bei denen es sich um unveränderliche Sammlungen von Ereignissen handelt, die auf Kriterien basieren, die Sie mit Hilfe von [erweiterten Ereignisselektoren](#) auswählen. Die Selektoren, die Sie auf einen Ereignisdatenspeicher anwenden, steuern, welche Ereignisse bestehen bleiben und für Sie zur Abfrage verfügbar sind. Weitere Informationen zu CloudTrail Lake finden Sie unter [Arbeiten mit AWS CloudTrail Lake](#) im AWS CloudTrail Benutzerhandbuch.

CloudTrail Für das Speichern und Abfragen von Ereignisdaten in Lake fallen Kosten an. Beim Erstellen eines Ereignisdatenspeichers wählen Sie die [Preisoption](#) aus, die für den Ereignisdatenspeicher genutzt werden soll. Die Preisoption bestimmt die Kosten für die Erfassung und Speicherung von Ereignissen sowie die standardmäßige und maximale Aufbewahrungsdauer für den Ereignisdatenspeicher. Weitere Informationen zur CloudTrail Preisgestaltung finden Sie unter [AWS CloudTrail Preise](#).

Protokollieren von Amazon Simple Queue Service API-Aufrufen mit AWS CloudTrail

CloudTrail ermöglicht es Ihnen, Amazon SQS SQS-Operationen mithilfe von zwei Ereignistypen zu protokollieren und zu überwachen: Datenereignisse und Verwaltungsereignisse. Dies macht es einfach, die Amazon SQS SQS-Aktivitäten in Ihrem Konto zu verfolgen und zu prüfen.

Amazon SQS SQS-Datenereignisse in CloudTrail

[Datenereignisse](#) liefern Informationen über die Ressourcenoperationen, die an oder in einer Ressource ausgeführt werden (z. B. das Senden von Nachrichten an ein Amazon SQS SQS-Objekt). Sie werden auch als Vorgänge auf Datenebene bezeichnet. Datenereignisse sind oft Aktivitäten mit hohem Volume. Protokolliert standardmäßig CloudTrail keine Datenereignisse. Der CloudTrail Ereignisverlauf zeichnet keine Datenereignisse auf.

Für Datenereignisse werden zusätzliche Gebühren fällig. Weitere Informationen zur CloudTrail Preisgestaltung finden Sie unter [AWS CloudTrail Preisgestaltung](#).

Sie können Datenereignisse für die Amazon SQS SQS-Ressourcentypen mithilfe der CloudTrail Konsole oder CloudTrail API-Operationen protokollieren. AWS CLI Weitere Informationen zum Protokollieren von Datenereignissen finden Sie unter [Protokollieren von Datenereignissen mit dem AWS-Managementkonsole](#) und [Protokollieren von Datenereignissen mit dem AWS Command Line Interface](#) im AWS CloudTrail -Benutzerhandbuch.

Um Amazon SQS SQS-Datenereignisse zu protokollieren CloudTrail, müssen Sie erweiterte Event-Selektoren verwenden, um die spezifischen Amazon SQS SQS-Ressourcen oder Aktionen zu konfigurieren, die Sie protokollieren möchten. Geben Sie den Ressourcentyp an, um Aktionen im Zusammenhang mit der [AWS::SQS::Queue](#) Warteschlange zu erfassen. Sie können Ihre Protokollierungseinstellungen noch weiter verfeinern, indem Sie Filter wie eventName (z. B. [SendMessage](#) Ereignisse) verwenden. Weitere Informationen finden Sie unter [AdvancedEventSelector](#) in der CloudTrail -API-Referenz.

Typ des Datenereignisses (Konsole)	resources.type-Wert	Daten APIs wurden protokolliert CloudTrail
Amazon-SQS-Warteschlange	AWS::SQS::Queue	• ChangeMessageVisibility • ChangeMessageVisibilityBatch

Typ des Datenereignisses (Konsole)	resources.type-Wert	Daten APIs wurden protokolliert CloudTrail
		• DeleteMessage • DeleteMessageBatch • GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

Verwenden Sie erweiterte Event-Selektoren, um Felder zu filtern und nur wichtige Ereignisse zu protokollieren. Weitere Informationen zu diesen Kontingenten finden Sie unter [AdvancedFieldSelector](#) in der AWS CloudTrail -API-Referenz.

Amazon SQS-Managementereignisse in CloudTrail

[Management-Ereignisse](#) bieten Informationen über Verwaltungsvorgänge, die mit Ressourcen in Ihrem AWS-Konto System ausgeführt werden. Sie werden auch als Vorgänge auf Steuerebene bezeichnet. CloudTrail protokolliert standardmäßig Verwaltungsereignisse.

Amazon SQS protokolliert die folgenden Vorgänge auf der Kontrollebene CloudTrail als Verwaltungsereignisse.

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTasks](#)
- [PurgeQueue](#)
- [RemovePermission](#)

- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

Beispiel für ein Amazon SQS SQS-Ereignis

Ein Ereignis stellt eine einzelne Anfrage aus einer beliebigen Quelle dar und enthält Informationen über den angeforderten API-Vorgang, Datum und Uhrzeit des Vorgangs, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass Ereignisse nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel zeigt ein CloudTrail Ereignis, das den SendMessage Vorgang demonstriert.

```
{
  "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": {
  "mD50fMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
  "mD50fMessageSystemAttributes": "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

Der `ListQueues` Vorgang ist ein Einzelfall, da er sich nicht auf eine bestimmte Ressource auswirkt. Daher enthält das ARN-Feld keinen Warteschlangennamen und verwendet stattdessen einen Platzhalter (*).

Informationen zu CloudTrail Datensatzinhalten finden Sie im AWS CloudTrail Benutzerhandbuch unter [CloudTrailDatensatzinhalte](#).

Überwachen Amazon SQS SQS-Warteschlangen mit CloudWatch

Amazon SQS und Amazon CloudWatch sind integriert, sodass Sie CloudWatch Metriken für Ihre Amazon SQS SQS-Warteschlangen anzeigen und analysieren können. [Sie können die Metriken Ihrer Warteschlangen über die Amazon SQS SQS-Konsole, die CloudWatch Konsole, mithilfe der oder mithilfe der AWS CLICloudWatch API anzeigen und analysieren](#). Sie können auch [CloudWatch Alarme für Amazon SQS-Metriken einrichten](#).

CloudWatch Metriken für Ihre Amazon SQS SQS-Warteschlangen werden automatisch erfasst und in Intervallen von einer CloudWatch Minute weitergeleitet. Diese Metriken werden für alle Warteschlangen erfasst, die den CloudWatch Richtlinien für aktive Benutzer entsprechen. CloudWatch betrachtet eine Warteschlange als bis zu sechs Stunden aktiv, wenn sie Nachrichten enthält oder wenn eine Aktion auf sie zugreift.

Wenn eine Amazon SQS-Warteschlange länger als sechs Stunden inaktiv ist, gilt der Amazon SQS-Service als inaktiv und stellt die Übermittlung von Metriken an den CloudWatch Service ein. Fehlende Daten oder Daten, die Null darstellen, können in den CloudWatch Kennzahlen für Amazon SQS für den Zeitraum, in dem Ihre Amazon SQS SQS-Warteschlange inaktiv war, nicht visualisiert werden.

Note

- Eine Amazon SQS SQS-Warteschlange kann aktiviert werden, wenn der Benutzer, der eine API für die Warteschlange aufruft, nicht autorisiert ist und die Anfrage fehlschlägt.
- Die Amazon SQS SQS-Konsole führt einen [GetQueueAttributes](#)API-Aufruf durch, wenn die Seite der Warteschlange geöffnet wird. Die `GetQueueAttributes` API-Anfrage aktiviert die Warteschlange.
- Bei CloudWatch Metriken kommt es zu einer Verzögerung von bis zu 15 Minuten, wenn eine Warteschlange aus einem inaktiven Zustand heraus aktiviert wird.
- Für die unter angegebenen Amazon SQS-Metriken fallen keine Gebühren an CloudWatch. Sie werden im Rahmen des Amazon-SQS-Service bereitgestellt.
- CloudWatch Metriken werden sowohl für Standard- als auch für FIFO-Warteschlangen unterstützt.

Zugreifen auf CloudWatch Metriken für Amazon SQS

Amazon SQS und Amazon CloudWatch sind integriert, sodass Sie CloudWatch Metriken für Ihre Amazon SQS SQS-Warteschlangen anzeigen und analysieren können. [Sie können die Metriken Ihrer Warteschlangen über die Amazon SQS SQS-Konsole, die CloudWatch Konsole, mithilfe der oder mithilfe der AWS CLICloudWatch API anzeigen und analysieren.](#) Sie können auch [CloudWatch Alarme für Amazon SQS-Metriken einrichten.](#)

Verwenden der Amazon-SQS-Konsole

Verwenden Sie die Amazon SQS SQS-Konsole, um auf Metriken für bis zu 10 Amazon SQS SQS-Warteschlangen zuzugreifen und diese zu analysieren.

1. Melden Sie sich bei der [Amazon-SQS-Konsole](#) an.
2. Aktivieren Sie in der Liste der Warteschlangen die Kontrollkästchen für die Warteschlangen, auf deren Metriken Sie zugreifen möchten. Sie können Metriken für bis zu 10 Warteschlangen anzeigen.
3. Wählen Sie die Registerkarte Überwachung.

Im Bereich SQS metrics werden verschiedene Diagramme angezeigt.

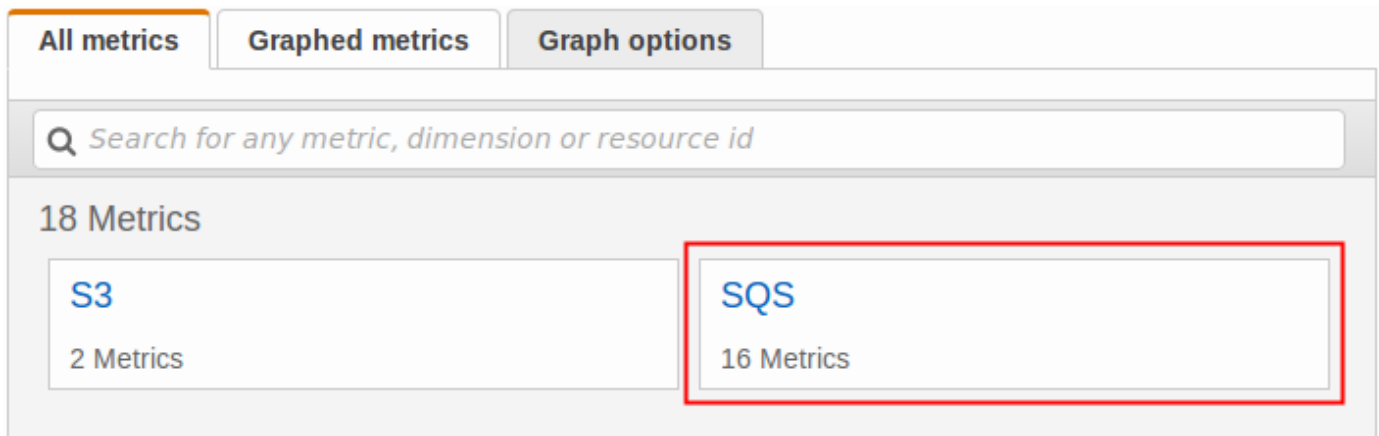
4. Wenn Sie wissen möchten, was ein bestimmtes Diagramm darstellt, zeigen Sie mit der Maus auf  neben dem gewünschten Diagramm. Weitere Informationen finden Sie auch unter [Verfügbare CloudWatch Metriken für Amazon SQS](#).
5. Um den Zeitraum für alle Diagramme gleichzeitig zu ändern, wählen Sie für Time Range den gewünschten Zeitraum aus (z. B. Last Hour).
6. Zum Anzeigen zusätzlicher Statistiken für ein einzelnes Diagramm wählen Sie das Diagramm aus.
7. Wählen Sie im Dialogfeld CloudWatch Monitoring Details (Überwachungsdetails) eine Statistic (Statistik) aus (z. B. Sum (Summe)). Eine Liste der unterstützten Statistiken finden Sie unter [Verfügbare CloudWatch Metriken für Amazon SQS](#).
8. Zum Ändern des Zeitraums und Zeitintervalls, die ein einzelnes Diagramm anzeigt (z. B. einen Zeitraum der letzten 24 Stunden anstelle der letzten 5 Minuten oder einen Zeitraum für jede Stunde anstatt alle 5 Minuten) wählen Sie bei aktivem Diagrammdialogfeld für Time Range den gewünschten Zeitraum aus (z. B. Last 24 Hours). Wählen Sie für Period den gewünschten Zeitraum innerhalb des angegebenen Zeitraums aus (z. B. 1 Hour). Wenn Sie mit dem Diagramm fertig sind, klicken Sie auf Close.

9. (Optional) Um mit zusätzlichen CloudWatch Funktionen zu arbeiten, wählen Sie auf der Registerkarte Überwachung die Option Alle CloudWatch Metriken anzeigen aus und folgen Sie dann den Anweisungen im Verfahren. [Verwenden der CloudWatch Amazon-Konsole](#)

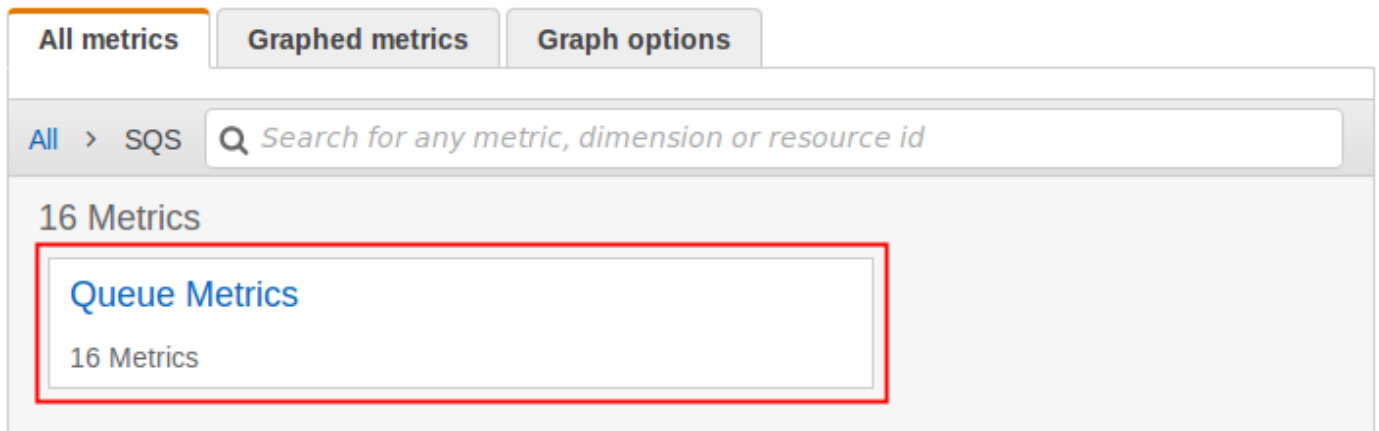
Verwenden der CloudWatch Amazon-Konsole

Verwenden Sie die CloudWatch Konsole, um auf Amazon SQS-Metriken zuzugreifen und diese zu analysieren.

1. Melden Sie sich bei der [CloudWatch-Konsole](#) an.
2. Wählen Sie im Navigationsbereich Metriken aus.
3. Wählen Sie den Namespace der SQS-Metrik aus.

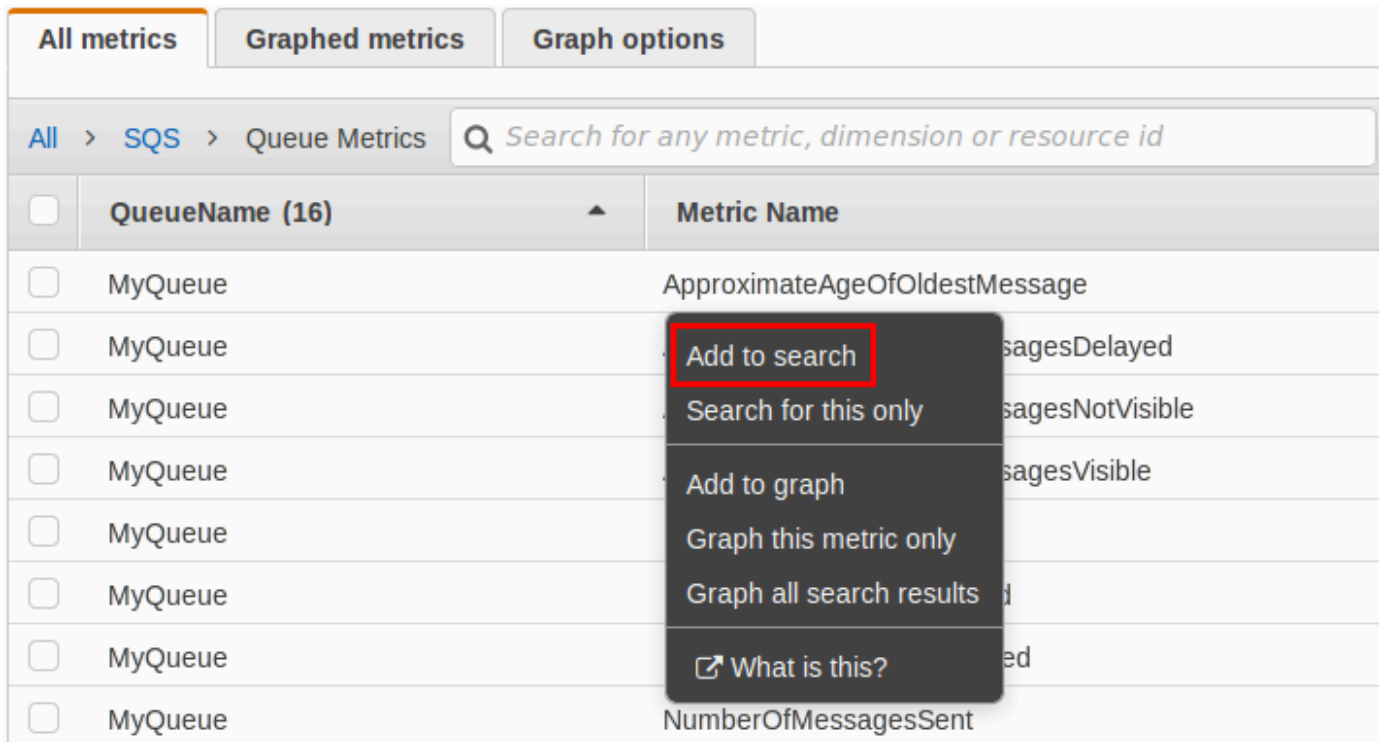


4. Wählen Sie die Metrikdimension Queue Metrics aus.



5. Sie können Ihre Amazon-SQS-Metriken jetzt analysieren:
 - Verwenden Sie die Spaltenüberschrift, um die Metriken zu sortieren.

- Um eine Metrik grafisch darzustellen, müssen Sie das Kontrollkästchen neben der Metrik aktivieren.
- Um nach Metrik zu filtern, müssen Sie den Metriknamen und anschließend Zur Suche hinzufügen auswählen.



Weitere Informationen und zusätzliche Optionen finden Sie unter [Graph Metrics](#) und [Using Amazon CloudWatch Dashboards](#) im CloudWatch Amazon-Benutzerhandbuch.

Mit dem AWS Command Line Interface

Um mit dem auf Amazon SQS-Metriken zuzugreifen AWS CLI, führen Sie den [get-metric-statistics](#) Befehl aus.

Weitere Informationen finden [Sie unter Get Statistics for a Metric](#) im CloudWatch Amazon-Benutzerhandbuch.

Verwenden der CloudWatch API

Verwenden Sie die [GetMetricStatistics](#) Aktion, um mithilfe der CloudWatch API auf Amazon SQS-Metriken zuzugreifen.

Weitere Informationen finden Sie unter [Get Statistics for a Metric](#) im CloudWatch Amazon-Benutzerhandbuch.

CloudWatch Alarme für Amazon SQS-Metriken erstellen

CloudWatch ermöglicht es Ihnen, Alarme auszulösen, wenn eine Metrik einen bestimmten Schwellenwert erreicht. Sie können beispielsweise einen Alarm für die `NumberOfMessagesSent`-Metrik erstellen. Beispiel: Wenn in einer Stunde mehr als 100 Nachrichten an die `MyQueue`-Warteschlange gesendet werden, wird eine E-Mail-Benachrichtigung gesendet. Weitere Informationen finden Sie unter [CloudWatch Amazon-Alarme erstellen](#) im CloudWatch Amazon-Benutzerhandbuch.

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die CloudWatch Konsole unter <https://console.aws.amazon.com/cloudwatch/>.
2. Wählen Sie Alarms und dann Create Alarm.
3. Wählen Sie im Abschnitt Select Metric (Metrik auswählen) im Dialogfeld Create Alarm (Alarm erstellen) die Optionen Browse Metrics (Metrik durchsuchen) und SQS aus.
4. Wählen Sie unter SQS > Queue Metrics den Metriknamen `QueueName` und den Metriknamen `NumberOfMessagesSent` aus, für den Sie einen Alarm einrichten möchten, und klicken Sie dann auf Weiter. Eine Liste der verfügbaren Metriken finden Sie unter [Verfügbare CloudWatch Metriken für Amazon SQS](#).

Im folgenden Beispiel handelt es sich bei der Auswahl um einen Alarm für die `NumberOfMessagesSent`-Metrik der `MyQueue`-Warteschlange. Der Alarm wird ausgelöst, wenn die Anzahl der gesendeten Nachrichten 100 überschreitet.

5. Gehen Sie im Abschnitt Define Alarm (Alarm festlegen) des Dialogfelds Create Alarm (Alarm erstellen) wie folgt vor:
 - a. Geben Sie im Feld Alarm Threshold (Alarmschwellenwert) Name und Description (Beschreibung) für den Alarm ein.
 - b. Legen Sie `is` auf `> 100` fest.
 - c. Setzen Sie `for` (für) auf den Wert 1 out of 1 datapoints (1 von 1 Datenpunkten).
 - d. Setzen Sie unter Alarm preview (Alarmvorschau) Period (Zeitraum) auf 1 Hour (1 Stunde).
 - e. Setzen Sie Statistic (Statistik) auf Standard, Sum (Summe).
 - f. Setzen Sie unter Actions (Aktionen) Whenever this alarm (Wann immer dieser Alarm ausgegeben wird) auf State is ALARM (Status lautet ALARM).

Wenn Sie eine Benachrichtigung senden CloudWatch möchten, wenn der Alarm ausgelöst wird, wählen Sie ein vorhandenes Amazon SNS SNS-Thema aus oder wählen Sie Neue Liste und geben Sie durch Kommas getrennte E-Mail-Adressen ein.

 Note

Wenn Sie ein neues Amazon-SNS-Thema erstellen, müssen die E-Mail-Adressen verifiziert werden, bevor die Empfänger Benachrichtigungen erhalten. Wenn es zu einer Änderung des Alarmstatus kommt, bevor die E-Mail-Adressen überprüft wurden, werden die Benachrichtigungen nicht versendet.

6. Wählen Sie Alarm erstellen aus.

Der Alarm ist erstellt.

Verfügbare CloudWatch Metriken für Amazon SQS

Amazon SQS sendet die folgenden Metriken an CloudWatch.

 Note

Bei einigen Metriken handelt es sich aufgrund der verteilten Architektur von Amazon SQS um ungefähre Ergebnisse. In den meisten Fällen sollte die Anzahl nahe an der tatsächlichen Anzahl von Nachrichten in der Warteschlange liegen.

Amazon-SQS-Metriken

Amazon SQS veröffentlicht automatisch Betriebsmetriken für [Amazon CloudWatch](#) unter dem AWS/SQS Namespace. Diese Metriken helfen Ihnen dabei, den Zustand und die Leistung der Warteschlange zu überwachen. Aufgrund der dezentralen Struktur von SQS sind viele Werte Näherungswerte, die jedoch für die meisten betrieblichen Entscheidungen genau genug sind.

 Note

- Alle Metriken geben nur dann nicht negative Werte aus, wenn die Warteschlange aktiv ist.

- Einige Metriken (z. B. `SentMessageSize`) werden erst ausgegeben, wenn mindestens eine Nachricht gesendet wurde.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
<code>ApproximateAgeOfOldestMessage</code>	Das Alter der ältesten unverarbeiteten Nachricht in der Warteschlange.	Sekunden	Wird gemeldet, wenn die Warteschlange mindestens eine aktive Nachricht enthält.	• Wenn bei Standardwarteschlangen eine Nachricht dreimal oder öfter empfangen und nicht gelöscht wird, verschiebt SQS sie an das Ende der Warteschlange. Die Metrik gibt dann das Alter der nächsten Nachricht wieder, die den Empfangsschwellenwert nicht überschritten hat. Diese Neuordnung erfolgt auch dann, wenn eine Redrive-Richtlinie gilt. • Poison-Pill-Meldungen (Nachrichten, die wiederholt empfangen, aber nie gelöscht wurden) werden von dieser Metrik ausgeschlossen, bis sie erfolgreich verarbeitet wurden. • Wenn eine Nachricht nach dem Überschreiten von in einen DLQ verschoben wird <code>maxReceiveCount</code>, wird das Alter zurückgesetzt. In diesem Fall spiegelt die DLQ-Metrik den Zeitpunkt wider, zu dem

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
				die Nachricht verschoben wurde, und nicht den Zeitpunkt, zu dem sie ursprünglich gesendet wurde. • FIFO-Warteschlangen ordnen Nachrichten nicht neu an, um die Reihenfolge beizubehalten. Eine fehlgeschlagene Nachricht blockiert ihre Nachrichtengruppe, bis sie gelöscht wird oder abläuft. Wenn ein DLQ konfiguriert ist, wird die Nachricht dorthin gesendet, nachdem der Empfangsschwellenwert erreicht wurde.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateNumberOfGroupsWithInflightMessages	Nur für FIFO. Die Anzahl der Nachrichtengruppen mit einer oder mehreren Nachrichten während der Übertragung.	Anzahl	Wird gemeldet, ob die FIFO-Warteschlange aktiv ist.	• Eine Nachricht gilt als in Bearbeitung befindlich, wenn sie von einem Verbraucher aus der Warteschlange empfangen, aber noch nicht gelöscht wurde oder abgelaufen ist. • Mit dieser Metrik können Sie Fehler beheben und den Durchsatz in der FIFO-Warteschlange optimieren. Hohe Werte deuten normalerweise auf eine starke Parallelität hin. • Wenn die Warteschlange einen großen Rückstand aufweist und dieser Wert niedrig bleibt, sollten Sie erwägen, die Anzahl der Verbraucher zu skalieren oder die Anzahl der aktiven Nachrichtengruppen zu erhöhen. • Informationen zu Durchsatz und Grenzwerten während der Übertragung finden Sie unter Amazon-SQS-Kontingente

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateNumberOfMessagesDelayed	Die Anzahl der Nachrichten in der Warteschlange, die verzögert werden und nicht sofort abgerufen werden können.	Anzahl	Wird gemeldet, ob verzögerte Nachrichten in der Warteschlange vorhanden sind.	- Gilt für Warteschlangen, die mit einer Standardverzögerung konfiguriert wurden, und für einzelne Nachrichten, die mit einem <code>DelaySeconds</code> Parameter gesendet wurden. - Verzögerte Nachrichten bleiben Benutzern verborgen, bis ihre Verzögerungszeit abgelaufen ist, was sich auf den wahrgenommenen Warteschlangenrückstand oder den Durchsatz auswirken kann.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateNumberOfMessagesNotVisible	Die Anzahl der Nachrichten während der Übertragung, die empfangen, aber noch nicht gelöscht wurden oder abgelaufen sind.	Anzahl	Wird gemeldet, ob Bordnachrichten vorhanden sind.	- Nachrichten gehen in den Status „Während des Fluges“ über, nachdem sie über die ReceiveMessage API an einen Verbraucher gesendet wurden. - Diese Nachrichten werden während des Sichtbarkeitszeitfensters vorübergehend vor anderen Verbrauchern verborgen. - Verwenden Sie diese Metrik, um Verzögerungen bei der Nachrichtenverarbeitung oder festgefahrene Nutzer nachzuverfolgen.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateNumberOfMessagesVisible	Die Anzahl der Nachrichten, die derzeit abgerufen und verarbeitet werden können.	Anzahl	Wird gemeldet, ob die Warteschlange aktiv ist.	• Spiegelt den aktuellen Verarbeitungsrückstand in der Warteschlange wider. • Es gibt keine feste Grenze für die Anzahl der Nachrichten, die sich ansammeln können, aber sie unterliegen der konfigurierten Aufbewahrungsfrist der Warteschlange. • Ein konstant hoher Wert kann auf eine unzureichende Anzahl von Verbrauchern oder auf eine festgefahrene Verarbeitungslogik hindeuten.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
NumberOfEmptyReceives ¹	Die Anzahl der ReceiveMessage API-Aufrufe, bei denen keine Nachrichten zurückgegeben wurden.	Anzahl	Wurde bei Empfangsvorgängen gemeldet.	• Diese Metrik kann dabei helfen, Ineffizienzen im Abfrageverhalten oder nicht ausgelastete Kundeninstanzen zu identifizieren. • Hohe Werte können auftreten, wenn die Warteschlange leer ist, der Verbraucher kurze Abfragen verwendet oder Nachrichten schneller verarbeitet als produziert werden. • Dies ist kein genauer Indikator für den Status der Warteschlange. Es spiegelt das Verhalten auf der Serviceseite wider und kann Wiederholungsversuche beinhalten.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
NumberOfDeduplicatedSentMessages	Nur für FIFO. Die Anzahl der gesendeten Nachrichten, die dedupliziert und nicht zur Warteschlange hinzugefügt wurden.	Anzahl	Wird gemeldet, wenn doppelte MessageDeduplicationId Werte oder Inhalte erkannt wurden.	• SQS dedupliziert Nachrichten basierend auf dem MessageDeduplicationId oder inhaltsbasierten Hashing (falls aktiviert). • Ein hoher Wert kann darauf hinweisen, dass ein Producer innerhalb des 5-minütigen Deduplizierungsfensters wiederholt dieselbe Nachricht sendet. • Verwenden Sie diese Metrik, um Fehler bei redundanter Herstellungslogik zu beheben oder zu überprüfen, ob die Deduplizierung wie vorgesehen funktioniert.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
NumberOfMessagesDeleted ¹	Die Anzahl der Nachrichten, die erfolgreich aus der Warteschlange gelöscht wurden.	Anzahl	Wird für jede Löschanforderung mit einer gültigen Empfangsnummer gemeldet.	• Diese Metrik zählt alle erfolgreichen Löschvorgänge — auch wenn dieselbe Nachricht mehr als einmal gelöscht wurde. • Zu den häufigsten Gründen für Werte gehören higher-than-expected: • Mehrfaches Löschen derselben Nachricht unter Verwendung verschiedener Empfangsnamen, nachdem das Sichtbarkeits-Timeout abgelaufen ist und die Nachricht erneut empfangen wurde. • Doppelte Löschungen, die dieselbe Empfangsnummer verwenden, geben trotzdem einen Erfolgsstatus zurück und erhöhen die Metrik. • Verwenden Sie diese Metrik, um den Erfolg der Nachrichtenverarbeitung nachzuverfolgen, aber behandeln Sie sie nicht als exakte Anzahl eindeutiger gelöschter Nachrichten.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
NumberOfMessagesReceived ¹	Die Anzahl der von der ReceiveMessageAPI zurückgegebenen Nachrichten.	Anzahl	Bei Empfangsvorgängen gemeldet.	- Dazu gehören alle Nachrichten, die an Verbraucher zurückgesendet wurden, einschließlich solcher, die später aufgrund des Ablaufs des Sichtbarkeits-TIMEOUTS in die Warteschlange zurückgeschickt werden. - Eine einzelne Nachricht kann mehrfach empfangen werden, wenn sie nicht gelöscht wird. Dies kann dazu führen, dass diese Metrik die Anzahl der gesendeten Nachrichten übersteigt. - Verwenden Sie diese Methode, um Kundenaktivitäten zu verfolgen, aber behandeln Sie sie nicht als die Anzahl der verarbeiteten einzelnen Nachrichten.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
<code>NumberOfMessagesSent</code> ¹	Die Anzahl der Nachrichten, die erfolgreich zu einer Warteschlange hinzugefügt wurden.	Anzahl	Wird für jeden erfolgreichen manuellen Versand gemeldet.	- Manuelle Anrufe an <code>SendMessage</code> oder <code>SendMessageBatch</code> werden gezählt, einschließlich solcher, die direkt auf einen DLQ abzielen. - Nachrichten, die automatisch in eine DLQ verschoben werden, nachdem sie den Wert überschritten haben, <code>maxReceiveCount</code> sind in dieser Metrik nicht enthalten. - Das <code>NumberOfMessagesSent</code> kann dazu führen, dass sie niedriger ist als <code>NumberOfMessagesReceived</code> — vor allem, wenn Redrive-Richtlinien viele Nachrichten DLQs hinter die Kulissen verschieben.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
SentMessageSize ¹	Die Größe der erfolgreich an die Warteschlange gesendeten Nachrichten.	Bytes	Wird erst ausgegeben, wenn mindestens eine Nachricht gesendet wurde.	• Diese Metrik wird erst in der CloudWatch Konsole angezeigt, wenn die Warteschlange ihre erste Nachricht erhält. • Verwenden Sie diese Metrik, um die Größe jeder Nachricht in Byte zu verfolgen. Dies ist nützlich, um Nutzlasttrends zu analysieren oder die Durchsatzkosten abzuschätzen. • Die maximale Nachrichtengröße für SQS beträgt 1 MiB.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateNumberOfNoisyGroups	Die Anzahl der Nachrichtengruppen, die in einer guten Warteschlange als störend eingestuft werden. Eine verrauschte Nachrichtengruppe steht für einen verrauschten Nachbarn einer Warteschlange mit mehreren Mandanten.	Anzahl	Ein nicht negativer Wert wird gemeldet, wenn die Warteschlange aktiv ist.	• Hilft bei der Identifizierung potenzieller Noisy-Neighbor-Probleme in Umgebungen mit mehreren Mandanten, indem Nachrichtengruppen nachverfolgt werden, die unverhältnismäßig viele Ressourcen verbrauchen. • Verwenden Sie diese Metrik, um Alarme einzurichten, die ausgelöst werden, wenn die Anzahl der verrauschten Gruppen Ihren akzeptablen Schwellenwert überschreitet, was auf mögliche Probleme mit der Warteschlangengerechtigkeit hinweist.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateNumberOfMessagesVisibleInQuietGroups	Die Anzahl der sichtbaren Nachrichten mit Ausnahme von Nachrichten aus Gruppen mit störenden Nachrichten.	Anzahl	Ein nicht negativer Wert wird gemeldet, wenn die Warteschlange aktiv ist .	- Bietet Einblick in den Warteschlangenrückstand für Nachrichtengruppen mit Standardgeschwindigkeit, ausgenommen Nachrichten von verrauschten Nachbarn. - Hilft dabei, den wahren Verarbeitungsrückstand für typische Nachrichtengruppen zu ermitteln, indem die Auswirkungen von störenden Nachbarn herausgefiltert werden.
ApproximateNumberOfMessagesNotVisibleInQuietGroups	Die Anzahl der laufenden Nachrichten, ausgenommen Nachrichten aus verrauschten Nachrichtengruppen.	Anzahl	Ein nicht negativer Wert wird gemeldet, wenn die Warteschlange aktiv ist .	- Verfolgt Nachrichten während der Übertragung (die verarbeitet, aber noch nicht gelöscht wurden) von Nachrichtengruppen, die sich gut verhalten haben. - Verwenden Sie diese Metrik, um den Verarbeitungsdurchsatz normaler Nachrichtengruppen zu überwachen und Verarbeitungsgengpässe zu erkennen, die nicht durch störende Nachbarn verursacht werden.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateNumberOfMessagesDelayedInQuietGroups	Die Anzahl der Nachrichten mit Ausnahme von Nachrichten aus verbrauchten Nachrichtenengruppen, die verzögert sind und nicht sofort gelesen werden können. Verzögerte Nachrichten treten auf, wenn die Warteschlange als Verzögerungswarteschlange konfiguriert ist oder wenn eine Nachricht mit einem Verzögerungsparameter gesendet wurde.	Anzahl	Ein nicht negativer Wert wird gemeldet, wenn die Warteschlange aktiv ist .	- Hilft bei der Überwachung des Rückstands verzögerter Nachrichten bei Nachrichtenengruppen mit normalem oder erwartetem Durchsatz (im Gegensatz zu Gruppen mit hohem Volumen oder hohen Datenvolumen) - Nützlich, um future Verarbeitungsanforderungen zu verstehen und die Kapazitätsplanung für typische Workloads zu planen.

Metrik	Description	Einheiten	Verhalten melden	Die wichtigsten Hinweise
ApproximateAgeOfOldestMessageInQuietGroups	Das Alter der ältesten nicht gelöschten Nachricht in der Warteschlange, ausgenommen Nachrichten aus verrauschten Nachrichtengruppen.	Sekunden	Ein nicht negativer Wert wird gemeldet, wenn die Warteschlange aktiv ist.	• Wird für die Überwachung der SLA-Einhaltung und die Erkennung von Verarbeitungsengpässen in Nachrichtengruppen mit normalem oder erwartetem Durchsatzmuster verwendet (im Gegensatz zu Nachrichtengruppen mit hohem Volumen oder verrauschten Nachrichten, die andernfalls die Metrik verzerren könnten). • Verwenden Sie diese Metrik, um Alarme für Timeouts bei der Nachricht enverarbeitung einzurichten, bei denen künstlich gealterte Nachrichten von störenden Nachbarn ignoriert werden.

¹ Diese Messwerte spiegeln die Aktivität auf Systemebene wider und können Wiederholungsversuche, Duplikate oder verzögerte Nachrichten beinhalten. Verwenden Sie keine Rohdaten, um den Status der Warteschlange in Echtzeit abzuschätzen, ohne das Verhalten im Nachrichtenlebenszyklus zu berücksichtigen.

Warteschlangen mit unbestätigten Nachrichten () und Metriken DLQs CloudWatch

Bei der Arbeit mit ist es wichtig zu verstehen DLQs, wie sich die Amazon SQS-Metriken verhalten:

- **NumberOfMessagesSent**— Diese Metrik verhält sich unterschiedlich für: DLQs
 - Manuelles Senden — Manuell an einen DLQ gesendete Nachrichten werden mit dieser Metrik erfasst.

- Automatisches Redrive — Nachrichten, die aufgrund von Verarbeitungsfehlern automatisch in eine DLQ verschoben wurden, werden von dieser Metrik nicht erfasst. Daher können die `NumberOfMessagesReceived` Metriken `NumberOfMessagesSent` und Unstimmigkeiten für aufweisen. DLQs
- Empfohlene Metrik für DLQs — Verwenden Sie die Metrik, um den Status einer DLQ zu überwachen. `ApproximateNumberOfMessagesVisible` Diese Metrik gibt die Anzahl der Nachrichten an, die derzeit in der DLQ verarbeitet werden können.

Faire Warteschlangen und Messwerte CloudWatch

Wenn Sie [faire Warteschlangen](#) verwenden, gibt Amazon SQS die folgenden zusätzlichen Messwerte aus:

- `ApproximateNumberOfNoisyGroups`
- `ApproximateNumberOfMessagesVisibleInQuietGroups`
- `ApproximateNumberOfMessagesNotVisibleInQuietGroups`
- `ApproximateNumberOfMessagesDelayedInQuietGroups`
- `ApproximateAgeOfOldestMessageInQuietGroups`

Note

Jede `QuietGroup` Metrik ist eine Teilmenge der entsprechenden Standardmetrik auf Warteschlangenebene, schließt jedoch Nachrichten von `Approximate Noisy-Neighbor`-Gruppen aus.

Verrauschte Gruppen

Eine Gruppe mit verrauschten Nachrichten steht für einen verrauschten Nachbarn einer Warteschlange mit mehreren Mandanten.

Ruhige Gruppen

Nachrichtengruppen, ausgenommen laute Gruppen.

Beobachtung des Verhaltens von SQS bei fairen Warteschlangen

Um die Auswirkungen fairer Warteschlangen in Amazon SQS zu überwachen, können Sie `Approximate..InQuietGroups` Metriken mit Standardmetriken auf Warteschlangenebene vergleichen. Bei einem starken Besucheraufkommen für einen bestimmten Mandanten können die allgemeinen Kennzahlen auf Warteschlangenebene Aufschluss über zunehmende Rückstände oder ein älteres Nachrichtenalter geben. Wenn Sie sich die stillen Gruppen jedoch isoliert ansehen, können Sie feststellen, dass die meisten nicht verrauschten Nachrichtengruppen oder Mandanten nicht betroffen sind, und eine Schätzung der Gesamtzahl der betroffenen Nachrichtengruppen vornehmen.

Diese neuen Metriken bieten zwar einen guten Überblick über das Verhalten von Amazon SQS SQS-Warteschlangen, es kann jedoch von Vorteil sein, zu verstehen, welcher bestimmte Mandant die Last verursacht. Mit [Amazon CloudWatch Contributor Insights](#) können Sie Kennzahlen zu den Top-N-Mitwirkenden, der Gesamtzahl der einzelnen Mitwirkenden und deren Nutzung einsehen. Dies ist besonders hilfreich in Szenarien, in denen Sie es mit Tausenden von Mietern zu tun haben, die sonst bei der Ausgabe herkömmlicher Kennzahlen zu Daten (und Kosten) mit hoher Kardinalität führen würden.

Ein Beispiel für die Überwachungskonfiguration für faire Warteschlangen finden Sie im Beispiel unter [GitHub](#).

Dimensionen für Amazon-SQS-Metriken

Die CloudWatch verwendeten Amazon SQS-Metriken haben eine einzige Dimension: **QueueName**. Alle Metrikdaten werden nach dem Namen der Warteschlange gruppiert und gefiltert.

Tipps zur Überwachung

Überwachen Sie SQS effektiv mithilfe wichtiger Kennzahlen und CloudWatch Alarme, um Warteschlangentrückstände zu erkennen, die Leistung zu optimieren und die Dienstgrenzen einzuhalten.

- Stellen Sie [CloudWatch Alarme](#) ein `ApproximateNumberOfMessagesVisible`, um den wachsenden Rückstand catch.
- Überwachen Sie `NumberOfEmptyReceives`, um die Abfragefrequenz zu optimieren und die API-Kosten zu senken.
- Wird `ApproximateNumberOfGroupsWithInflightMessages` in FIFO-Warteschlangen zur Diagnose von Durchsatzgrenzen verwendet.
- Informieren Sie sich über die [SQS-Kontingente](#), um sich mit metrischen Schwellenwerten und Servicebeschränkungen vertraut zu machen.

Compliance-Validierung für Amazon SQS

Informationen darüber, ob AWS-Service ein [AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt](#), finden Sie unter [Umfang nach Compliance-Programm AWS-Services unter](#) . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter [AWS Compliance-Programme AWS](#) .

Sie können Prüfberichte von Drittanbietern unter [herunterladen AWS Artifact](#). Weitere Informationen finden Sie unter [Berichte herunterladen unter](#) .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der [AWS Sicherheitsdokumentation](#).

Ausfallsicherheit bei Amazon SQS

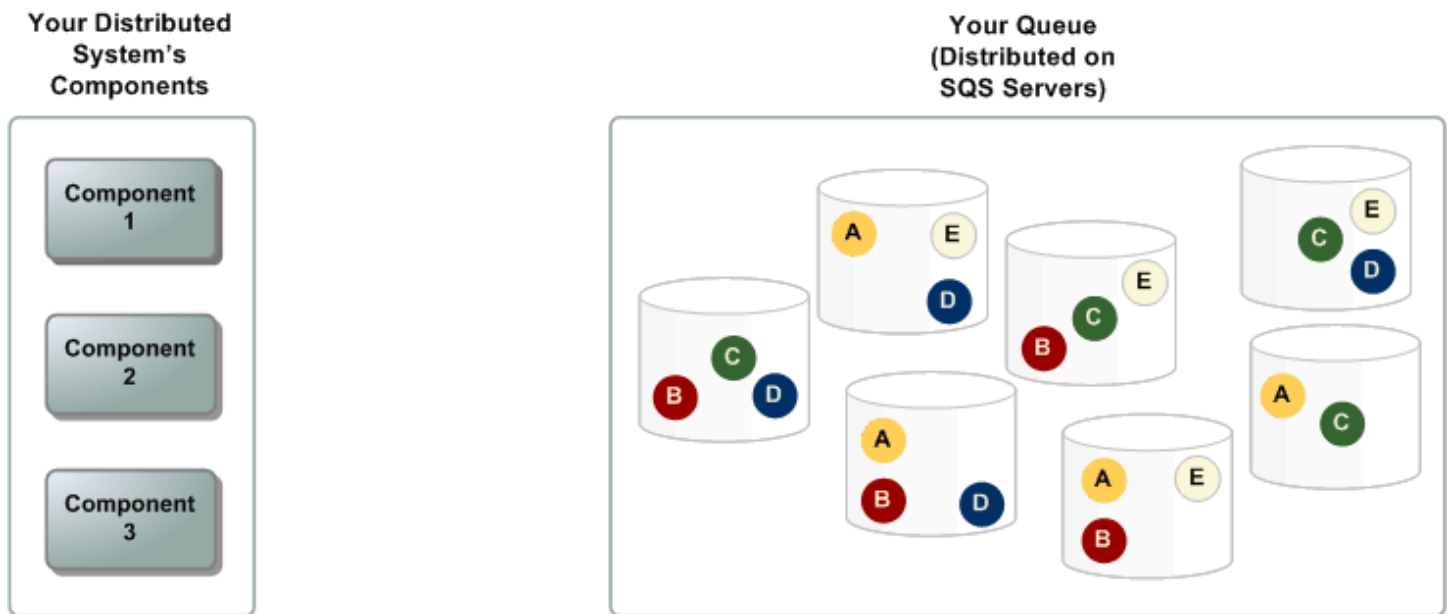
Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren. Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter [AWS Globale Infrastruktur](#).

Zusätzlich zur AWS globalen Infrastruktur bietet Amazon SQS verteilte Warteschlangen.

Verteilte Warteschlangen

Ein verteiltes Messaging-System besteht aus drei Hauptteilen: den Komponenten Ihres verteilten Systems, Ihrer Warteschlange (verteilt auf Amazon SQS-Servern) und den Nachrichten in der Warteschlange.

Im folgenden Szenario hat Ihr System mehrere Produzenten (Komponenten, die Nachrichten an die Warteschlange senden) und Konsumenten (Komponenten, die Nachrichten aus der Warteschlange empfangen). Die Warteschlange (die die Nachrichten A bis E enthält) speichert die Nachrichten redundant auf mehreren Amazon-SQS-Servern.



Infrastruktursicherheit in Amazon SQS

Als verwalteter Service ist Amazon SQS durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die im Whitepaper [Amazon Web Services: Sicherheitsprozesse im Überblick](#) beschrieben sind.

Sie verwenden AWS veröffentlichte API-Aktionen, um über das Netzwerk auf Amazon SQS zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Clients müssen außerdem Cipher Suites mit PFS (Perfect Forward Secrecy) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) unterstützen.

Anforderungen müssen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, die mit einem IAM-Prinzipal verknüpft sind. Alternativ können Sie die [AWS -Security-Token-Service](#) (AWS STS) verwenden, um temporäre Anmeldeinformationen für Signaturanforderungen zu generieren.

Sie können API-Aktionen von jedem Netzwerkstandort aus aufrufen. Amazon SQS unterstützt jedoch ressourcenbasierte Zugriffsrichtlinien, die Einschränkungen auf der Basis der Quell-IP-Adresse enthalten können. Sie können auch Amazon SQS SQS-Richtlinien verwenden, um den Zugriff von bestimmten Amazon VPC-Endpunkten oder bestimmten zu kontrollieren. VPCs Dadurch wird der Netzwerkzugriff auf eine bestimmte Amazon SQS SQS-Warteschlange effektiv nur von der spezifischen VPC innerhalb des Netzwerks isoliert. AWS Weitere Informationen finden Sie unter [Beispiel 5: Verweigerung des Zugriffs, wenn dieser nicht von einem VPC-Endpunkt aus erfolgt](#).

Bewährte Methoden für die Sicherheit in Amazon SQS

AWS bietet viele Sicherheitsfunktionen für Amazon SQS, die Sie im Zusammenhang mit Ihrer eigenen Sicherheitsrichtlinie überprüfen sollten. Im Folgenden werden bewährte vorbeugende Sicherheitsmethoden für Amazon SQS beschrieben.

Note

Die spezifischen Implementierungshinweise sind für häufige Anwendungsfälle und Implementierungen vorgesehen. Wir empfehlen Ihnen, diese bewährten Methoden im Kontext Ihres spezifischen Anwendungsfalls, der Architektur und des Bedrohungsmodells zu betrachten.

Sicherstellen, dass Warteschlangen nicht öffentlich zugänglich sind

Sofern Sie nicht ausdrücklich verlangen, dass jemand im Internet Ihre Amazon SQS SQS-Warteschlange lesen oder in sie schreiben kann, sollten Sie sicherstellen, dass Ihre Warteschlange nicht öffentlich zugänglich ist (für jeden auf der Welt oder für jeden authentifizierten AWS Benutzer zugänglich).

- Vermeiden Sie das Erstellen von Richtlinien mit auf "" festgelegtem Principal.
- Vermeiden Sie die Verwendung eines Platzhalters (*). Benennen Sie stattdessen einen oder mehrere bestimmte Benutzer.

Implementieren der geringstmöglichen Zugriffsrechte

Wenn Sie Berechtigungen erteilen, entscheiden Sie, wer sie erhält, für welche Warteschlangen die Berechtigungen gelten, und welche bestimmten API-Aktionen für diese Warteschlangen zugelassen werden. Die Implementierung von geringsten Privilegien ist wichtig, um Sicherheitsrisiken zu verringern und die Auswirkungen von Fehlern oder böswilligen Absichten zu reduzieren.

Folgen Sie den standardmäßigen Sicherheitshinweisen zur Erteilung von geringsten Privilegien. Das heißt: erteilen Sie nur die Berechtigungen, die zum Ausführen einer bestimmten Aufgabe erforderlich sind. Sie können dies mithilfe einer Kombination mehrerer Sicherheitsrichtlinien implementieren.

Amazon SQS verwendet das Producer-Consumer-Modell, für das drei Arten von Benutzerkontenzugriff erforderlich sind:

- Administratoren – Zugriff auf das Erstellen, Ändern und Löschen von Warteschlangen. Administratoren steuern auch Warteschlangenrichtlinien.
- Produzenten – Zugriff auf das Senden von Nachrichten an Warteschlangen.
- Konsumenten – Zugriff auf das Empfangen und Löschen von Nachrichten aus Warteschlangen.

Weitere Informationen finden Sie in den folgenden Abschnitten:

- [Identity and Access Management in Amazon SQS](#)
- [Amazon-SQS-API-Berechtigungen: Referenz für Aktionen und Ressourcen](#)
- [Verwenden von benutzerdefinierten Richtlinien mit der Sprache der Zugriffsrichtlinie von Amazon SQS](#)

Verwenden Sie IAM-Rollen für Anwendungen und AWS Services, die Amazon SQS SQS-Zugriff benötigen

Damit Anwendungen oder AWS Dienste wie Amazon EC2 auf Amazon SQS SQS-Warteschlangen zugreifen können, müssen sie in ihren AWS API-Anfragen gültige AWS Anmeldeinformationen verwenden. Da diese Anmeldeinformationen nicht automatisch rotiert werden, sollten Sie die AWS Anmeldeinformationen nicht direkt in der Anwendung oder EC2-Instance speichern.

Sie sollten mithilfe einer IAM-Rolle temporäre Anmeldeinformationen für Anwendungen und Services verwalten, die Zugriff auf Amazon SQS benötigen. Wenn Sie eine Rolle verwenden, müssen Sie keine langfristigen Anmeldeinformationen (wie einen Benutzernamen, ein Passwort und Zugriffsschlüssel) an eine EC2-Instance oder einen AWS Dienst wie z. B. verteilen. AWS Lambda Stattdessen stellt die Rolle temporäre Berechtigungen bereit, die Anwendungen verwenden können, wenn sie andere AWS Ressourcen aufrufen.

Weitere Informationen finden Sie unter [IAM-Rollen](#) und [Gängige Szenarien für Rollen: Benutzer, Anwendungen und Services](#) im IAM Benutzerhandbuch.

Implementieren serverseitiger Verschlüsselung

Probleme durch Datenlecks lassen sich verringern, indem Sie die Verschlüsselung im Ruhezustand verwenden. Dabei verschlüsseln Sie Ihre Nachrichten mithilfe eines Schlüssels, der an einem anderen Speicherort gespeichert ist als Ihre Nachrichten. Serverseitige Verschlüsselung (SSE) bietet Datenverschlüsselung im Ruhezustand. Amazon SQS verschlüsselt Ihre Daten auf Nachrichtenebene

bei der Speicherung und entschlüsselt die Nachrichten für Sie bei Zugriff darauf. SSE verwendet Schlüssel, die in AWS Key Management Service verwaltet werden. Solange Sie Ihre Anfrage authentifizieren und Zugriffsberechtigungen haben, gibt es keinen Unterschied zwischen dem Zugriff auf verschlüsselte und unverschlüsselte Warteschlangen.

Weitere Informationen erhalten Sie unter [Verschlüsselung im Ruhezustand in Amazon SQS](#) und [Amazon SQS Schlüsselverwaltung](#).

Erzwingen der Verschlüsselung von Daten während der Übertragung

Ohne HTTPS (TLS) kann ein netzwerkbasierter Angreifer den Netzwerkverkehr abhören oder ihn manipulieren, indem er einen Angriff wie verwendet. man-in-the-middle Erlauben Sie nur verschlüsselte Verbindungen über HTTPS (TLS) unter Verwendung der [aws:SecureTransport](#)-Bedingung in der Warteschlangenrichtlinie, um zu erzwingen, dass Anforderungen SSL verwenden.

Erwägen der Verwendung von VPC-Endpunkten für den Zugriff auf Amazon SQS

Verwenden Sie bei Warteschlangen, mit denen eine Interaktion erforderlich ist, die jedoch dem Internet absolut nicht zugänglich gemacht werden dürfen, VPC-Endpunkte, um den Zugriff nur auf die Hosts innerhalb einer bestimmten VPC in die Warteschlange festzulegen. Sie können Warteschlangenrichtlinien verwenden, um den Zugriff auf Warteschlangen von bestimmten Amazon VPC-Endpunkten oder von bestimmten aus zu kontrollieren. VPCs

Amazon-SQS-VPC-Endpunkte bieten zwei Möglichkeiten zur Kontrolle des Zugriffs auf Ihre Nachrichten:

- Sie können steuern, welche Anfragen, Benutzer oder Gruppen durch einen spezifischen VPC-Endpunkt erlaubt sind.
- Mithilfe einer Warteschlangenrichtlinie können Sie steuern, welche VPCs oder VPC-Endpunkte Zugriff auf Ihre Warteschlange haben.

Weitere Informationen erhalten Sie unter [Endpunkte von Amazon Virtual Private Cloud für Amazon SQS](#) und [Erstellen einer Amazon-VPC-Endpunkt-Richtlinie für Amazon SQS](#).

Verwandte Amazon-SQS-Ressourcen

Die folgende Tabelle enthält verwandte Ressourcen, die für die Arbeit mit diesem Service nützlich sind.

Ressource	Beschreibung
Amazon Simple Queue Service – API-Referenz	Beschreibungen der Aktionen, Parameter und Datentypen sowie eine Liste von Fehlern, die der Service zurückgibt.
Amazon SQS in der AWS CLI - Befehlsreferenz	Beschreibungen der AWS CLI Befehle, die Sie verwenden können, um mit Warteschlangen zu arbeiten.
Regionen und Endpunkte	Informationen zu Amazon-SQS-Regionen und -Endpunkten
Produktseite	Hauptwebsite für Informationen zu Amazon SQS.
Diskussionsforum	Ein auf der Community basierendes Forum, das für Entwickler eingerichtet wurde, um technische Fragen zu Amazon SWS zu klären.
AWS Informationen zum Premium-Support	Die wichtigste Webseite mit Informationen über AWS Premium Support, einen Support-Kanal mit schnellen Reaktionszeitenone-on-one, der Sie bei der Entwicklung und Ausführung von Anwendungen auf AWS Infrastrukturdiensten unterstützt.

Dokumentationsverlauf

In der folgenden Tabelle werden wichtige Änderungen am Amazon-Simple-Queue-Service-Entwicklerleitfaden seit Januar 2019 beschrieben. Um Benachrichtigungen über Aktualisierungen dieser Dokumentation zu erhalten, können Sie den [RSS-Feed](#) abonnieren.

Servicefunktionen werden manchmal schrittweise in den AWS Regionen eingeführt, in denen ein Service verfügbar ist. Wir aktualisieren diese Dokumentation nur für die erste Version. Wir stellen keine Informationen über die Verfügbarkeit von Regionen zur Verfügung und kündigen auch keine späteren Rollouts von Regionen an. Informationen zur regionalen Verfügbarkeit von Servicefunktionen und zum Abonnieren von Benachrichtigungen über Updates finden Sie unter [Was gibt's Neues bei AWS?](#) .

Änderung	Beschreibung	Datum
Unterstützung für Fair Queues	Unterstützung für faire Warteschlangen wurde für Standardwarteschlangen hinzugefügt.	21. Juli 2025
Unterstützung für Dual-Stack- (IPv4 und) Endpunkte hinzugefügt IPv6	Amazon SQS unterstützt jetzt Dual-Stack IPv4 - (und IPv6) Endpunkte, sodass auf Warteschlangen über beide IP-Protokolle zugegriffen werden kann.	17. April 2025
CloudTrail Integration für alle Amazon SQS APIs	CloudTrail Integration für alle Amazon SQS APIs hinzugefügt.	10. Januar 2025
SQSUnlockQueuePolicy	Amazon SQS hat eine neue AWS verwaltete Richtlinie hinzugefügt, mit der eine Warteschlange entsperrt und eine falsch konfigurierte Warteschlangenrichtlinie	15. November 2024

entfernt wird `SQSUnlockQueuePolicy` , die allen Prinzipalen den Zugriff auf eine Amazon SQS SQS-Warteschlange verweigert.

[AWS kms:Decrypt](#)

Amazon SQS benötigt keine `kms:Decrypt` Genehmigung mehr für die `SendMessage` API. Kunden benötigen jetzt nur noch die `kms:GenerateDataKey` Erlaubnis für den KMS-Schlüssel, der zum Verschlüsseln der Warteschlange verwendet wurde, benötigen aber weiterhin die `kms:Decrypt` Erlaubnis , anzurufen. `ReceiveMessage`

24. Juli 2024

[Aktualisierung der FIFO-Metriken](#)

Support für Amazon SQS FIFO-Metriken `NumberOfDuplicatedSentMessages` und zu diesen `ApproximateNumberOfGroupsWithInflightMessages` hinzugefügt.

3. Juli 2024

[ListQueueTags Aktion, die in der von Amazon SQSReadOnlyAccess verwalteten Richtlinie unterstützt wird](#)

Die von Amazon `SQSReadOnlyAccess` verwaltete Richtlinie unterstützt das Abrufen aller `TagsListQueueTags` , die mit einer bestimmten Amazon SQS SQS-Warteschlange verknüpft sind.

2. Mai 2024

<u>AWS JSON-Protokoll</u>	Stellen Sie API-Anfragen mithilfe des AWS JSON-Protokolls.	27. Juli 2023
<u>Neuer Abschnitt, in dem AWS verwaltete Richtlinien für Amazon SQS und Aktualisierungen dieser Richtlinien beschrieben werden</u>	Amazon SQS hat eine neue Aktion hinzugefügt, mit der Sie die neuesten Aufgaben zur Nachrichtenverschiebung (bis zu 10) in einer bestimmten Quellwarteschlange auflisten können. Diese Aktion ist mit dem <code>ListMessageMoveTasks</code> -API-Vorgang verknüpft.	7. Juni 2023
<u>Redrive der Warteschlange für unzustellbare Briefe mit APIs</u>	Konfigurieren Sie Neuansmeldungen von Warteschlangen mit Amazon SQS. APIs	7. Juni 2023
<u>ABAC für Amazon SQS</u>	Attributbasierte Zugriffskontrolle (ABAC) mit Warteschlangen-Tags für flexible und skalierbare Zugriffsberechtigungen.	10. November 2022
<u>Das FIFO-Limit für hohen Durchsatz wird erhöht</u>	Erhöhung der Standardkontingente für den FIFO-Hochdurchsatzmodus in kommerziellen Regionen sowie FIFO-Dokumentenoptimierung mit hohem Durchsatz.	20. Oktober 2022
<u>Standardmäßige serverseitige Verschlüsselung (SSE) ist verfügbar</u>	Serverseitige Verschlüsselung (SSE) mit standardmäßiger SQS-eigener Verschlüsselung (SSE-SQS).	26. September 2022

Unterstützung für Amazon SQS Confused Deputy Protection ist verfügbar	Confused Deputy Protection ermöglicht Ihnen, neue Header in Anfragen anzugeben, die bei Verwendung von Amazon SQS Managed SSE anhand der Bedingungen in der KMS-Richtlinie überprüft werden.	29. Dezember 2021
Managed SSE ist verfügbar	Amazon SQS Managed SSE (SSE-SQS) ist eine verwaltete serverseitige Verschlüsselung, die Amazon-SQS-eigene Verschlüsselungsschlüssel verwendet, um sensible Daten zu schützen, die über Nachrichtenwarteschlangen gesendet werden.	23. November 2021
Redrive von Warteschlangen für unzustellbare Nachrichten ist verfügbar	Amazon SQS unterstützt das Redrive von Warteschlangen für unzustellbare Nachrichten für Standard-Warteschlangen.	10. November 2021
Hoher Durchsatz für Nachrichten in FIFO-Warteschlangen ist verfügbar	Ein hoher Durchsatz für Amazon-SQS-FIFO-Warteschlangen ermöglicht eine höhere Anzahl von Transaktionen pro Sekunde (TPS) für Nachrichten in FIFO-Warteschlangen. Informationen zu Durchsatzkontingenten finden Sie unter Kontingente für Nachrichten .	27. Mai 2021

[Hoher Durchsatz für Nachrichten in FIFO-Warteschlangen ist in der Vorschauversion verfügbar](#)

Amazon-SQS-FIFO-Warteschlangen mit hohem Durchsatz sind als Vorschauversion verfügbar können sich noch ändern. Dieses Feature bietet eine höhere Anzahl von Transaktionen pro Sekunde (TPS) für Nachrichten in FIFO-Warteschlangen. Informationen zu Durchsatzkontingenten finden Sie unter [Kontingente für Nachrichten](#).

17. Dezember 2020

[Neues Amazon-SQS-Konsole-Design](#)

Um Entwicklungs- und Produktionsabläufe zu vereinfachen, bietet die Amazon-SQS-Konsole eine [neue Benutzerumgebung](#).

8. Juli 2020

[Amazon SQS unterstützt die Paginierung für ListQueues und listDeadLetterSourceQueues](#)

Sie können die maximale Anzahl von Ergebnissen angeben, die von einer [ListQueues](#) - oder [listDeadLetterSourceQueues](#)-Anfrage zurückgegeben werden sollen.

22. Juni 2020

[Amazon SQS unterstützt CloudWatch 1-Minuten-AWS-Metriken in allen AWS-Regionen außer den Regionen AWS GovCloud \(USA\)](#)

Die CloudWatch Ein-Minuten-Metrik für Amazon SQS ist in allen Regionen außer den AWS GovCloud (US) Regionen verfügbar.

9. Januar 2020

[Amazon SQS unterstützt CloudWatch 1-Minuten-Metriken](#)

Die einminütige CloudWatch Metrik für Amazon SQS ist derzeit nur in den folgenden Regionen verfügbar: USA Ost (Ohio), Europa (Irland), Europa (Stockholm) und Asien-Pazifik (Tokio).

25. November 2019

[AWS Lambda Auslöser für Amazon SQS FIFO-Warteschlangen sind verfügbar](#)

Sie können die in einer FIFO-Warteschlange Nachrichten so konfigurieren, dass eine Lambda-Funktion ausgelöst wird.

25. November 2019

[Serverseitige Verschlüsselung \(SSE\) für Amazon SQS ist in den chinesischen Regionen verfügbar](#)

SSE für Amazon SQS ist in den chinesischen Regionen verfügbar.

13. November 2019

[FIFO-Warteschlangen sind in der Region Naher Osten \(Bahrain\) verfügbar](#)

FIFO-Warteschlangen sind in der Region Naher Osten (Bahrain) verfügbar.

10. Oktober 2019

[Amazon Virtual Private Cloud \(Amazon VPC\) - Endpunkte für Amazon SQS sind in den Regionen AWS GovCloud \(USA Ost\) und AWS GovCloud \(USA West\) verfügbar](#)

Sie können Nachrichten von Amazon VPC in den Regionen AWS GovCloud (USA-Ost) und AWS GovCloud (US-West) an Ihre Amazon SQS-Warteschlangen senden.

5. September 2019

[Amazon SQS ermöglicht die Fehlerbehebung von Warteschlangen AWS X-Ray mithilfe von Nachrichtensystemattributen](#)

Sie können mithilfe von X-Ray Fehler bei Nachrichten beheben, die über Amazon-SQS-Warteschlangen weitergeleitet werden. Diese Version fügt `MessageSystemAttribute`-Anforderungsparameter zu den API-Aktionen `SendMessage` und `SendMessageBatch` (mit denen Sie X-Ray-Ablaufverfolgungs-Header über Amazon SQS senden können), das `AWSTraceHeader`-Attribut zu der API-Aktion [ReceiveMessage](#) und den Datentyp `MessageSystemAttributeValue` hinzu.

28. August 2019

[Sie können Amazon-SQS-Warteschlangen bei der Erstellung mit Tags markieren](#)

Sie können einen einzigen Amazon SQS SQS-API-Aufruf, eine AWS SDK-Funktion oder einen AWS Command Line Interface (AWS CLI) -Befehl verwenden, um gleichzeitig eine Warteschlange zu erstellen und ihre Tags anzugeben. Darüber hinaus unterstützt Amazon SQS die Schlüssel `aws:TagKeys` und `aws:RequestTag` AWS Identity and Access Management (IAM).

22. August 2019

[Der temporäre Warteschlangen-Client für Amazon SQS ist jetzt verfügbar](#)

Temporäre Warteschlangen helfen Ihnen, Entwicklungszeit und Bereitstellungskosten zu sparen, wenn Sie gängige Nachrichtenmuster wie Anfrage-Antwort verwenden. Sie können den [Temporären Warteschlangen-Client](#) verwenden, um kostengünstige, anwendungsverwaltete temporäre Warteschlangen mit hohem Durchsatz zu erstellen.

25. Juli 2019

[SSE für Amazon SQS ist in der Region AWS GovCloud \(USA-Ost\) verfügbar](#)

Serverseitige Verschlüsselung (SSE) für Amazon SQS ist in der Region AWS GovCloud (USA-Ost) verfügbar.

20. Juni 2019

[FIFO-Warteschlangen sind in den Regionen Asien-Pazifik \(Hongkong\), China \(Peking\), AWS GovCloud \(USA-Ost\) und AWS GovCloud \(US-West\) verfügbar](#)

FIFO-Warteschlangen sind in den Regionen Asien-Pazifik (Hongkong), China (Peking), AWS GovCloud (USA-Ost) und AWS GovCloud (US-West) verfügbar.

15. Mai 2019

[Amazon-VPC-Endpunkt-Richtlinien sind für Amazon SQS verfügbar](#)

Sie können Amazon-VPC-Endpunkt-Richtlinien für Amazon SQS erstellen.

4. April 2019

FIFO-Warteschlangen sind in den Regionen Europa (Stockholm) und China (Ningxia) verfügbar

FIFO-Warteschlangen sind in den Regionen Europa (Stockholm) und China (Ningxia) verfügbar.

14. März 2019

[FIFO-Warteschlangen sind in allen Regionen verfügbar, in denen Amazon SQS verfügbar ist](#)

FIFO-Warteschlangen sind in den Regionen USA Ost (Ohio), USA Ost (Nord-Virginia), USA West (Nordkalifornien), USA West (Oregon), Asien-Pazifik (Mumbai), Asien-Pazifik (Seoul), Asien-Pazifik (Singapur), Asien-Pazifik (Sydney), Asien-Pazifik (Tokio), Kanada (Zentral), Europa (Frankfurt), Europa (Irland), Europa (London), Europa (Paris) und Südamerika (São Paulo) verfügbar.

7. Februar 2019

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.