



Guida per gli sviluppatori

Amazon Simple Queue Service



Amazon Simple Queue Service: Guida per gli sviluppatori

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

I marchi e il trade dress di Amazon non possono essere utilizzati in relazione ad alcun prodotto o servizio che non sia di Amazon, in alcun modo che possa causare confusione tra i clienti, né in alcun modo che possa denigrare o screditare Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

Che cos'è Amazon SQS?	1
Vantaggi dell'utilizzo di Amazon SQS	1
Architettura di base	1
Code distribuite	2
Ciclo di vita del messaggio	2
Differenze tra Amazon SQS, Amazon MQ e Amazon SNS	4
Nozioni di base	6
Configurazione	6
Fase 1: creare un utente Account AWS e IAM	6
Fase 2: Concessione dell'accesso programmatico	8
Fase 3: ottieni un codice di esempio pronto per l'uso	10
Fasi successive	11
Comprendere le nozioni di base della console Amazon SQS	11
Tipi di coda	12
Implementazione di sistemi di richiesta e risposta in Amazon SQS	16
Creazione di una coda standard	16
Creazione di una coda	16
Invio di un messaggio utilizzando una coda standard	18
Creazione di una coda FIFO	19
Crea una coda	19
Invio di un messaggio utilizzando una coda FIFO	22
Attività comuni	22
Gestire una coda	24
Modifica di una coda	24
Ricezione ed eliminazione di un messaggio	24
Conferma che una coda è vuota	26
Eliminazione di una coda	27
Rimozione di una coda	28
Code standard	30
Consegna Amazon at-least-once SQS	31
Identificatori di code e messaggi	31
Identificatori per le code standard	31
Code FIFO	33
Termini chiave della coda FIFO	34

Logica di distribuzione FIFO	35
Invio di messaggi	36
Ricezione di messaggi	36
Tentativi di riesecuzione multipli	37
Note aggiuntive sul comportamento FIFO	38
Esempi per una migliore comprensione	38
Elaborazione "exactly-once"	39
Spostamento da una coda standard a una coda FIFO	40
Coda FIFO e comportamento di concorrenza Lambda	41
Raggruppamento dei messaggi in coda FIFO	42
Concorrenza Lambda con le code FIFO	42
Esempio di caso d'uso	42
Velocità di trasmissione effettiva elevata per le code FIFO	43
Casi d'uso	43
Partizioni e distribuzione dei dati	44
Abilitazione di un throughput elevato per le code FIFO	47
Identificatori di code e messaggi	48
Identificatori per le code FIFO	31
Identificatori aggiuntivi per code FIFO	49
Quote	51
Quote di coda FIFO	51
Quote di Amazon SQS	51
Quote di coda standard	53
Quote dei messaggi	54
Quote delle policy	61
Caratteristiche e funzionalità	62
Code di lettere non scritte	62
Utilizzo delle politiche per le code di lettere morte	63
Comprensione dei periodi di conservazione dei messaggi per le code di lettere scadute	63
Configurazione di una coda DLQ	64
Configurazione di un redrive della coda DLQ	64
CloudTrail requisiti di aggiornamento e autorizzazione	75
Creazione di allarmi per code di lettere scadenti con Amazon CloudWatch	79
Metadati dei messaggi per Amazon SQS	79
Attributi di messaggio	80
Attributi del sistema di messaggi	84

Risorse necessarie per l'elaborazione di messaggi	84
Elenca l'impaginazione delle code	85
Tag di allocazione dei costi	86
Short e long polling	87
Utilizzo dei messaggi con lo short polling	87
Utilizzo di messaggi con long polling	88
Differenze tra short e long polling	89
Timeout di visibilità	89
Casi d'uso del timeout di visibilità	90
Impostazione e regolazione del timeout di visibilità	90
Messaggi e quote in corso	91
Comprensione del timeout di visibilità nelle code standard e FIFO	91
Gestione degli errori	92
Modifica e interruzione del timeout di visibilità	92
Best practice	93
Code eque	93
Differenza rispetto alle code FIFO	95
Utilizzo di code eque	96
Metriche relative alle code eque CloudWatch	96
Code di ritardo	97
Code temporanee	98
Code virtuali	98
Modelli di messaggistica richiesta-risposta (code virtuali)	100
Scenario di esempio: elaborazione di una richiesta di accesso	101
Pulizia delle query	103
Timer messaggio	103
Accesso alle pipe EventBridge	104
Gestire messaggi di grandi dimensioni	105
Utilizzo della Extended Client Library per Java	106
Usare la libreria client estesa per Python	112
Configurazione di Amazon SQS	116
ABAC per Amazon SQS	116
Che cos'è ABAC?	116
Qual è il vantaggio di utilizzare ABAC in Amazon SQS?	117
Assegnazione di tag per il controllo degli accessi	118
Creazione di utenti IAM e code Amazon SQS	118

Test del controllo dell'accesso basato sugli attributi	121
Configurazione dei parametri della coda	122
Configurazione di una politica di accesso	124
Configurare SSE-SQS per una coda	125
Configurare SSE-KMS per una coda	126
Configurazione di tag per una coda	128
Iscrizione di una coda a un argomento	129
Abbonamenti per più account	130
Abbonamenti per più regioni	130
Configurazione di un trigger Lambda	131
Prerequisiti	132
Automatizzazione delle notifiche utilizzando EventBridge	133
Attributi di messaggio	133
Best practice	135
Gestione degli errori e messaggi problematici	135
Gestione degli errori di richiesta in Amazon SQS	135
Acquisizione di messaggi problematici in Amazon SQS	136
Configurazione della conservazione della coda di lettere non scritte in Amazon SQS	136
Deduplicazione e raggruppamento dei messaggi	137
Evitare l'elaborazione incoerente dei messaggi in Amazon SQS	137
Utilizzo dell'ID di deduplicazione messaggi	137
Utilizzo dell'ID gruppo di messaggi	140
Utilizzo dell'ID tentativo richiesta di ricezione	142
Elaborazione e tempistica dei messaggi	142
Elaborazione tempestiva dei messaggi in Amazon SQS	142
Configurazione di sondaggi lunghi in Amazon SQS	143
Utilizzo della modalità di polling appropriata in Amazon SQS	144
Esempi di SDK Java	146
Utilizzo della crittografia lato server	146
Aggiunta di SSE a una coda esistente	146
Disabilitazione di SSE per una coda	148
Creazione di una coda con SSE	148
Recupero degli attributi SSE	149
Configurare i tag	149
Come elencare i tag	149
Aggiunta o aggiornamento dei tag	150

Rimozione dei tag	150
Invio degli attributi del messaggio	151
Definizione degli attributi	151
Invio di un messaggio con attributi	153
Usando APIs	154
Effettuare richieste API di interrogazione utilizzando AWS il protocollo JSON	155
Costruzione di un endpoint	155
Effettuare una richiesta POST	156
Interpretazione delle risposte dell'API JSON di Amazon SQS	157
Protocollo Amazon SQS JSON AWS FAQs	158
Effettuazione di richieste API di interrogazione utilizzando il protocollo di AWS query	162
Costruzione di un endpoint	162
Effettuare una richiesta GET	162
Effettuare una richiesta POST	156
Interpretazione delle risposte dell'API XML di Amazon SQS	164
Autenticazione di richieste	165
Processo di autenticazione di base con HMAC-SHA	166
Parte 1: la richiesta dall'utente	167
Parte 2: La risposta di AWS	168
Azioni in batch	169
Raggruppamento delle azioni con messaggio	170
Abilitazione del buffering lato client e del batching delle richieste con Amazon SQS	171
Aumento del throughput utilizzando la scalabilità orizzontale e l'action batching con Amazon SQS	183
Lavorare con AWS SDKs	196
Usare JMS	198
Prerequisiti	198
Utilizzo della libreria di messaggistica Java	199
Creazione di una connessione JMS	200
Creazione di una coda Amazon SQS	200
Invio di messaggi in modo sincrono	201
Ricezione di messaggi in modo sincrono	203
Ricezione di messaggi in modo asincrono	204
Utilizzo della modalità di riconoscimento client	206
Utilizzo della modalità di riconoscimento non ordinata	207
Utilizzo del client JMS con altri client Amazon SQS	207

Esempi Java funzionanti per l'utilizzo di JMS con code standard	209
ExampleConfiguration.java	209
TextMessageSender.java	212
SyncMessageReceiver.java	213
AsyncMessageReceiver.java	215
SyncMessageReceiverClientAcknowledge.java	217
SyncMessageReceiverUnorderedAcknowledge.java	221
SpringExampleConfiguration.xml	224
SpringExample.java	226
ExampleCommon.java	228
Implementazioni JMS 1.1 supportate	230
Interfacce comuni supportate	230
Tipi di messaggi supportati	230
Modalità di riconoscimento del messaggio supportate	231
Intestazioni definite da JMS e proprietà riservate	231
Tutorial	233
Creazione di una coda Amazon SQS utilizzando CloudFormation	233
Invio di un messaggio da un VPC	235
Fase 1: Creazione di una coppia di chiavi di Amazon EC2	236
Fase 2: Creare risorse AWS	236
Fase 3: Verifica del fatto che l'istanza EC2 non è accessibile pubblicamente	237
Fase 4: Creazione di un endpoint Amazon VPC per Amazon SQS	238
Fase 5: Invio di un messaggio alla coda Amazon SQS	239
Esempi di codice	241
Nozioni di base	242
Hello Amazon SQS	243
Azioni	255
Scenari	431
Creare un'applicazione di messaggistica	432
Creazione di un'applicazione di messaggistica	433
Creazione di un'applicazione Amazon Textract explorer	434
Creazione e pubblicazione su un argomento FIFO	435
Rilevamento di persone e oggetti in un video	448
Gestire messaggi di grandi dimensioni con S3	449
Elaborare le notifiche di eventi S3	453
Pubblicazione di messaggi nelle code	457

Inviare e ricevere batch di messaggi	594
Usa il framework di elaborazione dei AWS messaggi per.NET con Amazon SQS	625
Utilizzare la raccolta di messaggistica Amazon SQS Java per utilizzare l'interfaccia JMS	626
Utilizzare i tag della coda	648
Esempi serverless	652
Invocazione di una funzione Lambda da un trigger Amazon SQS	652
Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS ..	661
Risoluzione dei problemi	672
Errore di accesso negato	672
Politica di coda di Amazon SQS e politica IAM	673
AWS Key Management Service (AWS KMS) autorizzazioni	674
Policy degli endpoint VPC	675
Politica di controllo dei servizi organizzativi	676
Errori API	676
QueueDoesNotExist errore	676
InvalidAttributeValue errore	677
ReceiptHandle errore	677
Problemi relativi al redrive DLQ e DLQ	678
Problemi DLQ	678
Problemi con DLQ-Redrive	680
Problemi di limitazione FIFO	682
Messaggi non restituiti per una chiamata API ReceiveMessage	682
Coda vuota	683
Limite di volo raggiunto	683
Ritardo dei messaggi	683
Il messaggio è in volo	683
Metodo di sondaggio	683
Errori di rete	684
ETIMEOUT error	684
UnknownHostException error	685
Risoluzione dei problemi delle code mediante X-Ray	686
Sicurezza	687
Protezione dei dati	687
Crittografia dei dati	688
Riservatezza del traffico Internet	700
Utilizzo di endpoint dual-stack per la connettività	702

Gestione dell'identità e degli accessi	703
Destinatari	703
Autenticazione con identità	703
Gestione dell'accesso tramite policy	705
Panoramica di	706
Come funziona Amazon Simple Queue Service con IAM	713
AWS politiche gestite	720
Risoluzione dei problemi	722
Utilizzo di policy	724
Registrazione di log e monitoraggio	772
Registrazione di chiamate API	775
Monitoraggio delle code	779
Convalida della conformità	799
Resilienza	800
Code distribuite	800
Sicurezza dell'infrastruttura	801
Best practice	801
Assicurarsi che le code non siano accessibili pubblicamente	801
Implementazione dell'accesso con privilegi minimi	802
Usa i ruoli IAM per applicazioni e AWS servizi che richiedono l'accesso ad Amazon SQS ...	802
Implementare la crittografia lato server	803
Applica la crittografia dei dati in transito	803
Prendere in considerazione l'utilizzo di endpoint VPC per accedere a Amazon SQS	803
Risorse correlate	805
Cronologia della documentazione	806
.....	dcccxv

What is Amazon Simple Queue Service?

Amazon Simple Queue Service (Amazon SQS) offre una coda ospitata internamente sicura, durevole e disponibile che consente di integrare e separare i componenti e i sistemi software distribuiti. Amazon SQS offre costrutti comuni come, ad esempio, [code DLQ](#) e [tag di allocazione dei costi](#). Fornisce un'API di servizi Web generica a cui puoi accedere utilizzando qualsiasi linguaggio di programmazione supportato dall' AWS SDK.

Vantaggi dell'utilizzo di Amazon SQS

- Sicurezza - [Puoi controllare](#) chi può inviare e ricevere messaggi da una coda Amazon SQS. Puoi scegliere di trasmettere dati sensibili proteggendo il contenuto dei messaggi nelle code utilizzando la crittografia lato server gestita (SSE) predefinita di Amazon SQS o utilizzando chiavi [SSE](#) personalizzate gestite in AWS Key Management Service (AWS KMS).
- Durabilità: per garantire la sicurezza dei tuoi messaggi, Amazon SQS li memorizza su più server. [Le code standard supportano il recapito dei at-least-once messaggi, mentre le code FIFO supportano l'elaborazione dei messaggi una sola volta e la modalità ad alta velocità.](#)
- Disponibilità: Amazon SQS utilizza un'[infrastruttura ridondante](#) per fornire l'accesso simultaneo ai messaggi e alta disponibilità per la produzione e l'utilizzo di messaggi.
- Scalabilità: Amazon SQS è in grado di elaborare ogni [richiesta di buffer](#) in modo indipendente e di ridimensionarsi in maniera trasparente per gestire eventuali picchi o aumenti di carico senza le relative istruzioni di provisioning.
- Affidabilità: Amazon SQS blocca i tuoi messaggi durante l'elaborazione, in modo tale che più produttori possono inviare e più consumatori possono ricevere messaggi contemporaneamente.
- Personalizzazione: le tue code non devono essere esattamente uguali, per esempio, è possibile impostare un [intervallo predefinito su una coda](#). Puoi archiviare il contenuto di messaggi di dimensioni superiori a 1 MiB utilizzando [Amazon Simple Storage Service \(Amazon S3\) o Amazon DynamoDB](#), con Amazon SQS che tiene premuto un puntatore all'oggetto Amazon S3, oppure puoi dividere un messaggio di grandi dimensioni in messaggi più piccoli.

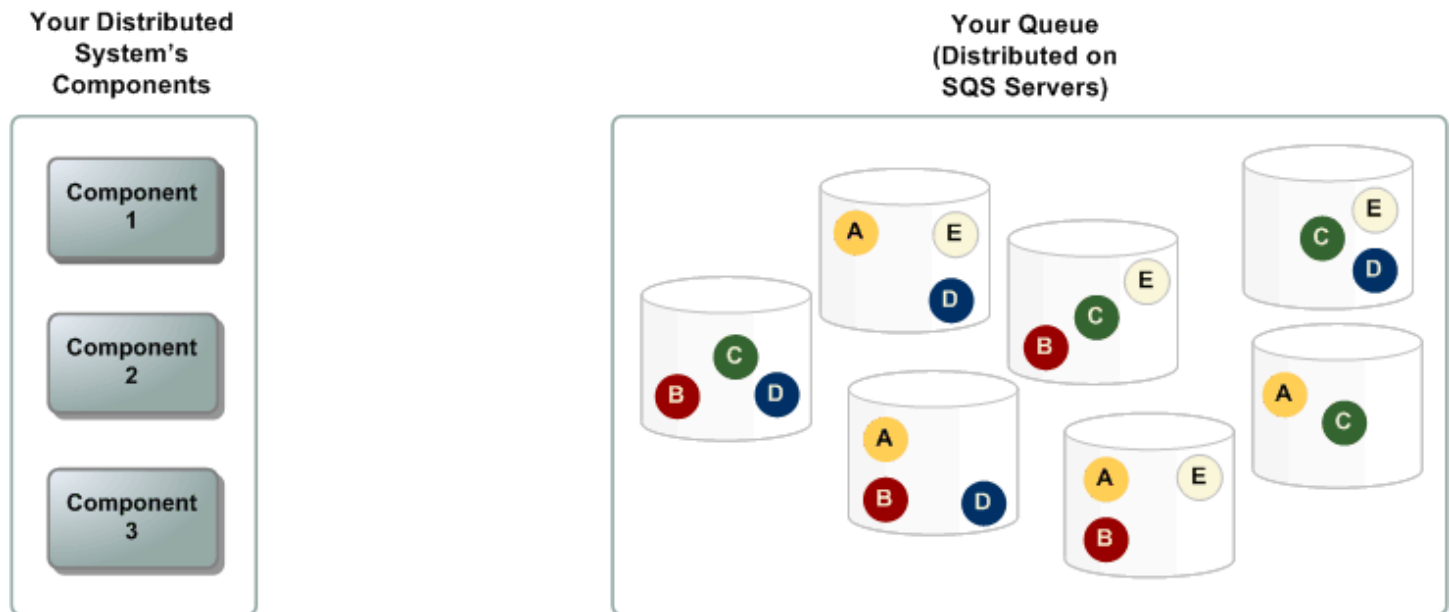
Architettura Amazon SQS di base

Questa sezione descrive i componenti di un sistema di messaggistica distribuito e spiega il ciclo di vita di un messaggio Amazon SQS.

Code distribuite

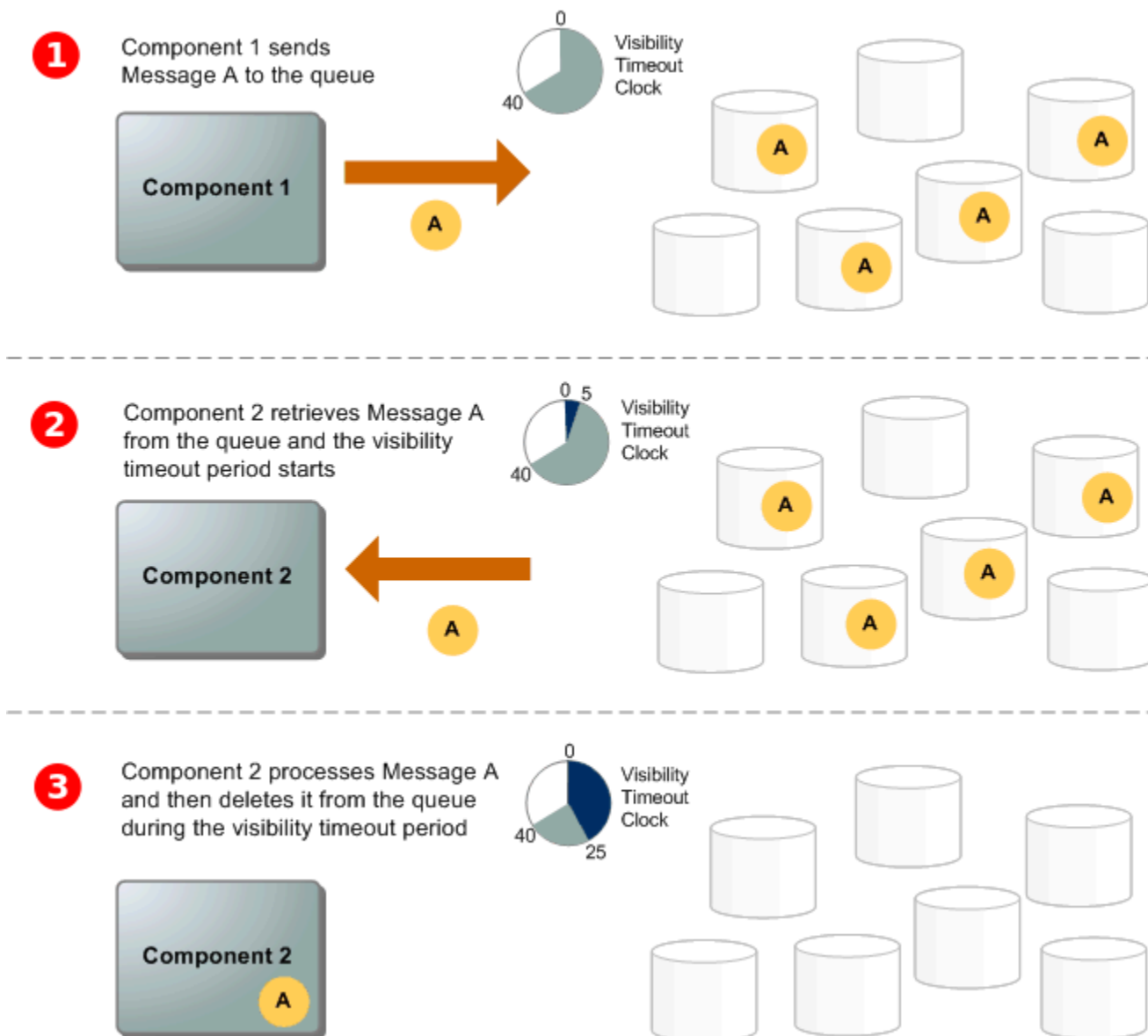
Un sistema di messaggistica distribuito si compone di tre parti principali: i componenti del sistema distribuito, la coda (distribuita sui server Amazon SQS) e i messaggi in coda.

Nel seguente scenario, il sistema dispone di diversi produttori, ossia componenti che inviano messaggi alla coda, e consumatori, componenti che ricevono messaggi dalla coda. La coda (che contiene i messaggi da A a E) archivia in modo ridondante i messaggi su più server Amazon SQS.



Ciclo di vita del messaggio

Lo scenario seguente descrive il ciclo di vita di un messaggio Amazon SQS in una coda, dalla creazione all'eliminazione.



1 Un produttore (Componente 1) invia il messaggio A a una coda e il messaggio viene distribuito sui server Amazon SQS in modo ridondante.

2 Quando un consumatore (Componente 2) è pronto per elaborare i messaggi, consuma i messaggi dalla coda e viene restituito il messaggio A. Mentre il messaggio A viene elaborato, rimane nella coda e non viene restituito alle richieste di ricezione successive per la durata del [timeout visibilità](#).

3

II

consumatore (Componente 2) elimina il messaggio A dalla coda per evitare che il messaggio venga ricevuto ed elaborato nuovamente allo scadere del timeout di visibilità.

Note

Amazon SQS elimina automaticamente i messaggi che sono stati in una coda per un periodo superiore quello massimo di conservazione. Il periodo predefinito per la conservazione dei messaggi è 4 giorni. Tuttavia, puoi impostare un valore compreso tra 60 secondi e 1.209.600 secondi (14 giorni) utilizzando l'azione [SetQueueAttributes](#).

Differenze tra Amazon SQS, Amazon MQ e Amazon SNS

Amazon SQS, Amazon [SNS](#) e [Amazon MQ](#) offrono servizi di messaggistica altamente scalabili easy-to-use e gestiti, ciascuno progettato per ruoli specifici all'interno di sistemi distribuiti. Ecco una panoramica avanzata delle differenze tra questi servizi:

Amazon SQS disaccoppia e ridimensiona sistemi e componenti software distribuiti come servizio di coda. In genere elabora i messaggi tramite un singolo abbonato, ideale per flussi di lavoro in cui l'ordine e la prevenzione delle perdite sono fondamentali. Per una distribuzione più ampia, l'integrazione di Amazon SQS con Amazon SNS consente [un modello di messaggistica fanout](#), inviando efficacemente i messaggi a più abbonati contemporaneamente.

Amazon SNS consente agli editori di inviare messaggi a più abbonati tramite argomenti, che fungono da canali di comunicazione. Gli abbonati ricevono messaggi pubblicati utilizzando un tipo di endpoint supportato, ad esempio [Amazon Amazon Data Firehose](#), [SQS](#), [Lambda](#), [HTTP](#), e-mail, notifiche push per dispositivi mobili e messaggi di testo mobili (SMS). Questo servizio è ideale per scenari che richiedono notifiche immediate, come il coinvolgimento degli utenti in tempo reale o i sistemi di allarme. Per evitare la perdita di messaggi quando gli abbonati sono offline, l'integrazione di Amazon SNS con i messaggi in coda di Amazon SQS garantisce una distribuzione coerente.

Amazon MQ [si adatta meglio alle aziende che desiderano migrare dai tradizionali broker di messaggistica, supportando protocolli di messaggistica standard come AMQP e MQTT, oltre ad Apache ActiveMQ e RabbitMQ](#). Offre compatibilità con i sistemi legacy che richiedono una messaggistica stabile e affidabile senza riconfigurazioni significative.

La tabella seguente fornisce una panoramica del tipo di risorsa di ciascun servizio:

Tipo di risorsa	Amazon SNS	Amazon SQS	Amazon MQ
Synchronous (Sincrona)	No	No	Sì
In modo asincrono	Sì	Sì	Sì
Queues	No	Sì	Sì
Messaggistica publish-subscribe	Sì	No	Sì
Broker di messaggi	No	No	Sì

Sia Amazon SQS che Amazon SNS sono consigliati per nuove applicazioni che possono trarre vantaggio da una scalabilità quasi illimitata e semplice. APIs In genere offrono soluzioni più convenienti per applicazioni ad alto volume con i loro prezzi. pay-as-you-go Consigliamo Amazon MQ per la migrazione di applicazioni da broker di messaggi esistenti che si basano sulla compatibilità con JMS o protocolli APIs come Advanced Message Queuing Protocol (AMQP), MQTT e Simple Text Oriented Message Protocol (STOMP). OpenWire

Nozioni di base su Amazon SQS

Questo argomento ti guida nell'uso della console Amazon SQS per creare e gestire code standard e code FIFO. Imparerai come navigare nella console, visualizzare gli attributi delle code e distinguere i tipi di coda. Le attività principali includono l'invio, la ricezione e la configurazione dei messaggi, la regolazione di parametri come il timeout di visibilità e la conservazione dei messaggi e la gestione dell'accesso alla coda tramite policy.

Argomenti

- [Configurazione di Amazon SQS](#)
- [Comprendere le nozioni di base della console Amazon SQS](#)
- [Tipi di code Amazon SQS](#)
- [Creazione di una coda Amazon SQS standard e invio di un messaggio](#)
- [Creazione di una coda FIFO Amazon SQS e invio di un messaggio](#)
- [Attività comuni per iniziare a usare Amazon SQS](#)

Configurazione di Amazon SQS

Prima di poter utilizzare Amazon SQS per la prima volta, devi completare i seguenti passaggi:

Fase 1: creare un utente Account AWS e IAM

Per accedere a qualsiasi AWS servizio, devi prima creare un [Account AWS](#) account Amazon.com che possa utilizzare AWS i prodotti. Puoi utilizzare il tuo Account AWS per visualizzare i report sulle attività e sull'utilizzo e per gestire l'autenticazione e l'accesso.

Per evitare di utilizzare l'utente Account AWS root per le azioni di Amazon SQS, è consigliabile creare un utente IAM per ogni persona che necessita di accesso amministrativo ad Amazon SQS.

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la <https://portal.aws.amazon.com/billing/registrazione>.

2. Segui le istruzioni online.

Nel corso della procedura di registrazione riceverai una telefonata o un messaggio di testo e ti verrà chiesto di inserire un codice di verifica attraverso la tastiera del telefono.

Quando ti iscrivi a un Account AWS, Utente root dell'account AWS viene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire [attività che richiedono l'accesso di un utente root](#).

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a <https://aws.amazon.com/> e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi [Console di gestione AWS](#) come proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

Per informazioni sull'accesso utilizzando un utente root, consulta la pagina [Accedere come utente root](#) nella Guida per l'utente di Accedi ad AWS .

2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

Per istruzioni, consulta [Abilitare un dispositivo MFA virtuale per l'utente Account AWS root \(console\)](#) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita il Centro identità IAM.

Per istruzioni, consulta [Abilitazione del AWS IAM Identity Center](#) nella Guida per l'utente di AWS IAM Identity Center .

2. Nel Centro identità IAM, assegna l'accesso amministrativo a un utente.

Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta [Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory](#) nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere come utente del Centro identità IAM, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente del Centro identità IAM.

Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta [AWS Accedere al portale di accesso](#) nella Guida per l'Accedi ad AWS utente.

Assegnazione dell'accesso ad altri utenti

1. Nel Centro identità IAM, crea un set di autorizzazioni conforme alla best practice per l'applicazione di autorizzazioni con il privilegio minimo.

Segui le istruzioni riportate nella pagina [Creazione di un set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

Per istruzioni, consulta [Aggiungere gruppi](#) nella Guida per l'utente di AWS IAM Identity Center .

Fase 2: Concessione dell'accesso programmatico

Per utilizzare le azioni di Amazon SQS (ad esempio, utilizzando Java o tramite AWS Command Line Interface), sono necessari un ID chiave di accesso e una chiave di accesso segreta.

Note

L'ID della chiave di accesso e la chiave di accesso segreta sono specifici di AWS Identity and Access Management. Non confonderle con le credenziali di altri AWS servizi, come le coppie di chiavi Amazon EC2.

Gli utenti hanno bisogno di un accesso programmatico se vogliono interagire con AWS l'esterno di Console di gestione AWS Il modo per concedere l'accesso programmatico dipende dal tipo di utente che accede. AWS

Per fornire agli utenti l'accesso programmatico, scegli una delle seguenti opzioni.

Quale utente necessita dell'accesso programmatico?	Per	Come
IAM	(Consigliato) Utilizza le credenziali della console come credenziali temporanee per firmare le richieste programmatiche a,, o. AWS CLI AWS SDKs AWS APIs	Segui le istruzioni per l'interfaccia che desideri utilizzare. • Per la AWS CLI, consulta Login for AWS local development nella Guida per l'AWS Command Line Interface utente. • Per AWS SDKs, consulta Login for AWS local development nella AWS SDKs and Tools Reference Guide.
Identità della forza lavoro (Utenti gestiti nel centro identità IAM)	Utilizza credenziali temporanee e per firmare le richieste programmatiche a AWS CLI, AWS SDKs, o. AWS APIs	Segui le istruzioni per l'interfaccia che desideri utilizzare. • Per la AWS CLI, vedere Configurazione dell'uso AWS IAM Identity Center nella AWS CLI Guida per l'utente.AWS Command Line Interface • Per AWS SDKs gli strumenti e AWS APIs, consulta l'autenticazione di IAM Identity Center nella Guida

Quale utente necessita dell'accesso programmatico?	Per	Come
		di riferimento AWS SDKs and Tools.
IAM	Utilizza credenziali temporanee e per firmare le richieste programmatiche a AWS CLI, AWS SDKs, o. AWS APIs	Seguendo le istruzioni riportate in Utilizzo delle credenziali temporanee con le AWS risorse nella Guida per l'utente IAM .
IAM	(Non consigliato) Utilizza credenziali a lungo termine per firmare richieste programmatiche a AWS CLI,, AWS SDKs o. AWS APIs	Segui le istruzioni per l'interfaccia che desideri utilizzare. • Per la AWS CLI, consulta Autenticazione tramite credenziali utente IAM nella Guida per l'utente.AWS Command Line Interface • Per gli strumenti AWS SDKs e gli strumenti, consulta Autenticazione tramite credenziali a lungo termine nella Guida di riferimento agli strumenti e agli AWS SDKs strumenti. • Per AWS APIs, consulta la sezione Gestione delle chiavi di accesso per gli utenti IAM nella Guida per l'utente IAM.

Fase 3: ottieni un codice di esempio pronto per l'uso

Questa guida include esempi che utilizzano l' AWS SDK for Java. Per eseguire il codice di esempio, segui le istruzioni di configurazione in [Guida introduttiva a AWS SDK per Java 2.0](#).

È possibile sviluppare AWS applicazioni in altri linguaggi di programmazione, come GoJavaScript, Python e Ruby. Per ulteriori informazioni, consulta [Tools to Building](#) on. AWS

Note

Puoi esplorare Amazon SQS senza scrivere codice con strumenti come AWS Command Line Interface (AWS CLI) o Windows. PowerShell Puoi trovare AWS CLI esempi nella [sezione Amazon SQS](#) del AWS CLI Command Reference. Puoi trovare PowerShell esempi di Windows nella sezione Amazon Simple Queue Service del [AWS Strumenti per PowerShell Cmdlet Reference](#).

Fasi successive

Ora sei pronto per [iniziare](#) a gestire le code e i messaggi di Amazon SQS utilizzando Console di gestione AWS.

Comprendere le nozioni di base della console Amazon SQS

Quando apri la console Amazon SQS, scegli Queues dal pannello di navigazione. La pagina Code fornisce informazioni su tutte le code nell'area attiva.

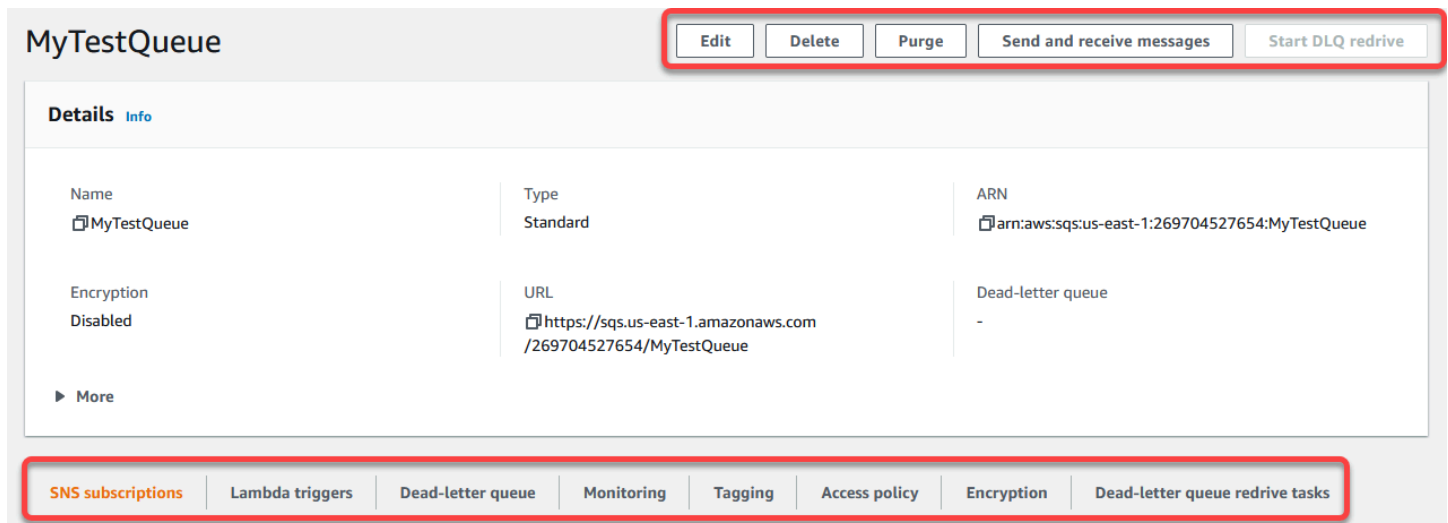
Ogni voce della coda fornisce informazioni essenziali sulla coda, inclusi il tipo e gli attributi chiave. [Le code standard](#), ottimizzate per la massima velocità di trasmissione e il miglior ordinamento dei messaggi, si distinguono dalle code [First-In-First-Out \(FIFO\)](#), che danno priorità all'ordine e all'unicità dei messaggi per le applicazioni che richiedono un rigoroso sequenziamento dei messaggi.

Queues (2)			Edit	Delete	Send and receive messages	Actions ▼	Create queue
Search queues by prefix		< 1 >					
	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

Elementi e azioni interattivi

Dalla pagina Code, sono disponibili diverse opzioni per la gestione delle code:

1. Azioni rapide: accanto al nome di ogni coda, un menu a discesa offre un accesso rapido ad azioni comuni come l'invio di messaggi, la visualizzazione o l'eliminazione di messaggi, la configurazione dei trigger e l'eliminazione della coda stessa.
2. Visualizzazione e configurazione dettagliate: facendo clic sul nome di una coda si apre la relativa pagina dei dettagli, in cui è possibile approfondire le impostazioni e le configurazioni della coda. Qui puoi regolare parametri come il periodo di conservazione dei messaggi, il timeout di visibilità e la dimensione massima dei messaggi per adattare la coda ai requisiti dell'applicazione.



Selezione della regione e tag delle risorse

Assicurati di essere nella posizione giusta Regione AWS per accedere e gestire le code in modo efficace. Inoltre, prendi in considerazione l'utilizzo dei tag delle risorse per organizzare e classificare le code, permettendo una migliore gestione delle risorse, l'allocazione dei costi e il controllo degli accessi all'interno del tuo ambiente condiviso. AWS

Sfruttando le caratteristiche e le funzionalità offerte all'interno della console Amazon SQS, puoi gestire in modo efficiente la tua infrastruttura di messaggistica, ottimizzare le prestazioni delle code e garantire la consegna affidabile dei messaggi per le tue applicazioni.

Tipi di code Amazon SQS

[Amazon SQS supporta due tipi di code: code standard e code FIFO.](#) Utilizza la tabella seguente per determinare la coda più adatta alle tue esigenze.

Code standard	Code FIFO
Throughput illimitato: le code standard supportano un numero molto elevato e quasi illimitato di chiamate API al secondo, per azione (SendMessage, ReceiveMessage o). DeleteMessage Questo throughput elevato le rende ideali per i casi d'uso che richiedono l'elaborazione rapida di grandi volumi di messaggi, come lo streaming di dati in tempo reale o applicazioni su larga scala. Sebbene le code standard si ridimensionino automaticamente in base alla domanda, è essenziale monitorare i modelli di utilizzo per garantire prestazioni ottimali, specialmente nelle regioni con carichi di lavoro più elevati. At-least-once consegna: at-least-once consegna garantita, il che significa che ogni messaggio viene recapitato almeno una volta, ma in alcuni casi un messaggio può essere recapitato più di una volta a causa di nuovi tentativi o ritardi nella rete. È necessario progettare l'applicazione in modo da gestire potenziali messaggi duplicati utilizzando operazioni idempotenti, che assicurano che l'elaborazione dello stesso messaggio più volte non influisca sullo stato del sistema. Ordinazione con la massima diligenza: offre la massima efficienza di ordinazione, il che significa che, sebbene Amazon SQS tenti di recapitare i messaggi nell'ordine in cui sono stati inviati, non lo garantisce. In alcuni casi, i messaggi possono arrivare fuori servizio, specialmente in condizioni di elevata velocità effettiva o di ripristino non riuscito. Per le	Velocità effettiva elevata: quando utilizzi il batch, le code FIFO elaborano fino a 3.000 messaggi al secondo per metodo API (, o). SendMessageBatch ReceiveMessageBatch DeleteMessageBatch Questo throughput si basa su 300 chiamate API al secondo, con ogni chiamata API che gestisce un batch di 10 messaggi. Abilitando la modalità a throughput elevato, puoi scalare fino a 30.000 transazioni al secondo (TPS) con ordinamenti rilassati all'interno dei gruppi di messaggi. Senza batch, le code FIFO supportano fino a 300 chiamate API al secondo per metodo API (, o). SendMessage ReceiveMessage DeleteMessage Se hai bisogno di una maggiore velocità di trasmissione, puoi richiedere un aumento della quota tramite il AWS Support Center. Per abilitare la modalità ad alta velocità effettiva, consulta Abilitazione di un throughput elevato per le code FIFO in Amazon SQS Elaborazione una sola volta: le code FIFO consegnano ogni messaggio una sola volta e lo mantengono disponibile fino a quando non lo elabori ed elimini. Utilizzando funzionalità come la deduplicazione basata sui contenuti, eviti la duplicazione dei messaggi, anche in caso di nuovi tentativi a causa di problemi di rete MessageDeduplicationId o timeout. First-in-first-out consegna: le code FIFO garantiscono la ricezione dei messaggi nell'ordine in cui vengono inviati all'interno di ciascun gruppo di messaggi. Distribuendo i messaggi

Code standard	Code FIFO
applicazioni in cui l'ordine di elaborazione dei messaggi è fondamentale, è necessario gestire la logica di riordino all'interno dell'applicazione o utilizzare le code FIFO per garantire un ordine rigoroso. Durabilità e ridondanza: le code standard garantiscono un'elevata durabilità archiviando più copie di ogni messaggio in più zone di disponibilità. AWS Ciò garantisce che i messaggi non vadano persi, anche in caso di guasti dell'infrastruttura. Timeout di visibilità: Amazon SQS consente di configurare un timeout di visibilità per controllare per quanto tempo un messaggio rimane nascosto dopo essere stato ricevuto, assicurando che altri consumatori non elaborino il messaggio fino a quando non è stato completamente gestito o il timeout è scaduto.	su più gruppi, puoi elaborarli in parallelo mantenendo l'ordine all'interno di ciascun gruppo.
 The diagram illustrates a standard queue where messages are not necessarily processed in the order they were sent. It shows a horizontal line representing the queue with several rectangular boxes containing numbers. The boxes are arranged in a non-sequential manner: 4, 3, 2, 3, 5, 1. This represents a lack of ordering guarantee.	 The diagram illustrates a FIFO (First In, First Out) queue where messages are processed in the exact order they were sent. It shows a horizontal line representing the queue with several rectangular boxes containing numbers. The boxes are arranged in a strictly sequential manner: 5, 4, 3, 2, 1. This represents a strict ordering guarantee.

Code standard	Code FIFO
Utilizza le code standard per inviare dati tra le applicazioni quando la velocità effettiva è fondamentale, ad esempio: • Separa le richieste degli utenti in tempo reale dal lavoro intensivo in background. Consenti agli utenti di caricare file multimediali rapidamente mentre esegui attività come il ridimensionamento o la codifica in background, garantendo tempi di risposta rapidi senza sovraccaricare il sistema. • Assegna le attività a più nodi di lavoro. Distribuisci un numero elevato di richieste di convalida delle carte di credito su più nodi di lavoro e gestisci i messaggi duplicati con operazioni idempotenti per evitare errori di elaborazione. • Messaggi in batch per elaborazioni future. Metti in coda più voci per le aggiunte in batch a un database. Poiché l'ordine dei messaggi non è garantito, progettate il sistema in modo da gestire l'out-of-order elaborazione, se necessario.	Utilizzate le code FIFO per inviare dati tra applicazioni quando l'ordine degli eventi è importante, ad esempio: • Assicurati che i comandi immessi dall'utente vengano eseguiti nell'ordine corretto. Questo è un caso d'uso fondamentale per le code FIFO, in cui l'ordine dei comandi è fondamentale. Ad esempio, se un utente esegue una sequenza di azioni in un'applicazione, le code FIFO assicurano che le azioni vengano elaborate nello stesso ordine in cui sono state inserite. • Visualizza il prezzo corretto del prodotto inviando le modifiche di prezzo nell'ordine corretto. Le code FIFO assicurano che arrivino più aggiornamenti al prezzo di un prodotto e vengano elaborati in sequenza. Senza FIFO, una riduzione di prezzo potrebbe essere elaborata dopo un aumento di prezzo, causando la visualizzazione di dati errati. • Impedisce a uno studente di iscriversi a un corso prima di registrare un account. Utilizzando le code FIFO, ti assicuri che il processo di registrazione avvenga nella sequenza corretta. Il sistema elabora prima la registrazione dell'account e poi l'iscrizione al corso, evitando che la richiesta di iscrizione venga eseguita prematuramente.

Implementazione di sistemi di richiesta e risposta in Amazon SQS

Quando implementi un sistema richiesta-risposta o di chiamata procedura remota (RPC), tieni presente le seguenti best practice:

- Crea code di risposta all'avvio: anziché creare code di risposta per messaggio, creale all'avvio, per produttore. Utilizza un attributo di messaggio ID di correlazione per mappare le risposte alle richieste in modo efficiente.
- Evita di condividere le code di risposta tra i produttori: assicurati che ogni produttore abbia la propria coda di risposta. La condivisione delle code di risposta può far sì che un produttore riceva messaggi di risposta destinati a un altro produttore.

Per ulteriori informazioni sull'implementazione del modello di richiesta-risposta utilizzando Temporary Queue Client, consulta [Modelli di messaggistica richiesta-risposta \(code virtuali\)](#).

Creazione di una coda Amazon SQS standard e invio di un messaggio

Puoi creare una [coda standard](#) e inviare messaggi utilizzando la console Amazon SQS. Questo argomento enfatizza anche le migliori pratiche, tra cui evitare informazioni sensibili nei nomi di coda e utilizzare la crittografia gestita lato server.

Creazione di una coda standard utilizzando la console Amazon SQS

Important

Il 17 agosto 2022, la crittografia lato server predefinita (SSE) è stata applicata a tutte le code Amazon SQS.

Non aggiungere informazioni di identificazione personale (PII) o altre informazioni riservate o sensibili nei nomi delle code. I nomi delle code sono accessibili a molti Amazon Web Services, inclusi fatturazione e CloudWatch log. I nomi delle code non sono destinati a essere utilizzati per dati privati o sensibili.

Creazione di una coda standard Amazon SQS

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>

2. Scegliere Crea coda.
3. Per Tipo, il tipo di coda Standard è impostato di default.

 Note

Il tipo di coda non può essere modificato dopo la creazione della coda.

4. Inserisci un Nome per la coda.
5. (Facoltativo) La console imposta i valori predefiniti per i [parametri di configurazione](#) della coda. In Configurazione, è possibile impostare nuovi valori per i seguenti parametri:
 - a. Per il timeout di visibilità, inserisci la durata e le unità. L'intervallo è compreso tra 0 secondi e 12 ore. Il valore di predefinito è 30 secondi.
 - b. Per Periodo di conservazione dei messaggi, inserisci la durata e le unità. L'intervallo valido è compreso tra 1 minuto e 14 giorni. Il valore predefinito è 4 giorni.
 - c. Per Ritardo di consegna, inserisci la durata e le unità. L'intervallo è compreso tra 0 secondi e 15 minuti. Il valore predefinito è 0 secondi.
 - d. Per Dimensione massima del messaggio, inserisci un valore. L'intervallo va da 1 KiB a 1024 KiB. Il valore predefinito è 1024 KiB.
 - e. Per Tempo di attesa per la ricezione del messaggio, inserisci un valore. L'intervallo è tra 0 e 20 secondi. Il valore predefinito è 0 secondi, che imposta uno [short polling](#). Qualsiasi valore diverso da zero imposta un long polling.
6. (Facoltativo) Definire una policy di accesso. La [policy di accesso](#) definisce gli account, gli utenti e i ruoli che possono accedere alla coda. La policy di accesso definisce anche le operazioni (ad esempio [SendMessage](#), [ReceiveMessage](#) o [DeleteMessage](#)) a cui gli utenti possono accedere. La policy predefinita consente solo al proprietario della coda di inviare e ricevere messaggi.

Per definire la policy di accesso, effettua una delle seguenti operazioni:

- Scegli Basic per configurare chi può inviare messaggi alla coda e chi può ricevere messaggi dalla coda. La console crea la policy in base alle tue scelte e visualizza la policy di accesso risultante nel pannello JSON di sola lettura.
- Scegli Avanzato per modificare direttamente la policy di accesso JSON. Ciò consente di specificare un set personalizzato di azioni che ogni principale (account, utente o ruolo) può eseguire.

7. Per la policy Redrive allow, scegli Abilitata. Seleziona una delle seguenti opzioni: Consenti tutto, Per coda o Nega tutto. Quando scegli Per coda, specifica un elenco di un massimo di 10 code di origine in base al nome della risorsa Amazon (ARN).
8. Amazon SQS fornisce la crittografia lato server gestita per impostazione predefinita. Per scegliere un tipo di chiave di crittografia o per disabilitare la crittografia lato server gestita da Amazon SQS, espandi Crittografia. Per ulteriori informazioni sui tipi di chiavi di crittografia, consulta [Configurazione della crittografia lato server per una coda utilizzando chiavi di crittografia gestite da SQL](#) e [Configurazione della crittografia lato server per una coda utilizzando la console Amazon SQS](#).

 Note

Con SSE abilitato, le richieste anonime SendMessage e ReceiveMessage alla coda crittografata verranno rifiutate. Le best practice di sicurezza di Amazon SQS consigliano di non utilizzare richieste anonime. Se desideri inviare richieste anonime a una coda Amazon SQS, assicurati di disabilitare SSE.

9. (Facoltativo) Per configurare una [coda DLQ](#) per la ricezione di messaggi non recapitabili, espandere Coda DLQ.
10. (Facoltativo) Per aggiungere [tag](#) alla coda, espandi Tag.
11. Scegliere Crea coda. Amazon SQS crea la coda e visualizza la pagina dei dettagli della coda.

Amazon SQS diffonde le informazioni sulla nuova coda in tutto il sistema. Poiché Amazon SQS è un sistema distribuito, potrebbe verificarsi un leggero ritardo prima che la console visualizzi la coda nella pagina Code.

Invio di un messaggio utilizzando una coda standard

Dopo aver creato la coda, puoi inviarle un messaggio.

1. Nel riquadro di navigazione sinistro scegliere Code. Nell'elenco delle code, seleziona la coda appena creata.
2. Scegliere Operazioni, quindi Invia e ricevi messaggi.

La console visualizza la pagina Invia e ricevi messaggi.

3. In Corpo del messaggio, inserisci il testo del messaggio.

4. Per una coda standard, puoi inserire un valore per Ritardo di consegna e scegliere le unità. Ad esempio, inserisci 60 e scegli i secondi. Per ulteriori informazioni, consulta [Timer di messaggi Amazon SQS](#).
5. Scegliere Invia messaggio.

Quando il messaggio viene inviato, la console ne visualizza la conferma. Scegli Visualizza dettagli per visualizzare le informazioni sul messaggio inviato.

Creazione di una coda FIFO Amazon SQS e invio di un messaggio

Puoi creare una coda FIFO Amazon SQS e inviare messaggi utilizzando la console. Questo argomento spiega come configurare i parametri di coda, tra cui il timeout di visibilità, la conservazione dei messaggi e la deduplicazione, seguendo al contempo le migliori pratiche di sicurezza, come evitare informazioni sensibili nei nomi delle code e abilitare la crittografia lato server. Illustra anche la definizione delle politiche di accesso, la configurazione delle code di lettere morte e l'invio di messaggi con attributi specifici FIFO come l'ID del gruppo di messaggi e l'ID di deduplicazione.

Creazione di una coda FIFO utilizzando la console Amazon SQS

È possibile utilizzare la console Amazon SQS per creare le [code FIFO](#). La console fornisce valori predefiniti per tutte le impostazioni ad eccezione del nome della coda.

Important

Il 17 agosto 2022, la crittografia lato server predefinita (SSE) è stata applicata a tutte le code Amazon SQS.

Non aggiungere informazioni di identificazione personale (PII) o altre informazioni riservate o sensibili nei nomi delle code. I nomi delle code sono accessibili a molti Amazon Web Services, inclusi fatturazione e CloudWatch log. I nomi delle code non sono destinati a essere utilizzati per dati privati o sensibili.

Creazione di una coda FIFO Amazon SQS

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Scegliere Crea coda.

3. Per Tipo, il tipo di coda Standard è impostato di default. Per creare una coda FIFO, scegliere FIFO.

 Note

Il tipo di coda non può essere modificato dopo la creazione della coda.

4. Inserisci un Nome per la coda.

Il nome di una coda FIFO deve terminare con il suffisso `.fifo`. Il suffisso viene conteggiato ai fini della quota di 80 caratteri dei nomi della coda. Per determinare se una coda è [FIFO](#), puoi verificare se il nome della coda termina con il suffisso.

5. (Facoltativo) La console imposta i valori predefiniti per i [parametri di configurazione](#) della coda. In Configurazione, è possibile impostare nuovi valori per i seguenti parametri:
 - a. Per il timeout di visibilità, inserisci la durata e le unità. L'intervallo è compreso tra 0 secondi e 12 ore. Il valore di predefinito è 30 secondi.
 - b. Per Periodo di conservazione dei messaggi, inserisci la durata e le unità. L'intervallo valido è compreso tra 1 minuto e 14 giorni. Il valore predefinito è 4 giorni.
 - c. Per Ritardo di consegna, inserisci la durata e le unità. L'intervallo è compreso tra 0 secondi e 15 minuti. Il valore predefinito è 0 secondi.
 - d. Per Dimensione massima del messaggio, inserisci un valore. L'intervallo va da 1 KiB a 1024 KiB. Il valore predefinito è 1024 KiB.
 - e. Per Tempo di attesa per la ricezione del messaggio, inserisci un valore. L'intervallo è tra 0 e 20 secondi. Il valore predefinito è 0 secondi, che imposta uno [short polling](#). Qualsiasi valore diverso da zero imposta un long polling.
 - f. Per una coda FIFO, scegli Deduplicazione basata sul contenuto per abilitare la deduplicazione basata sul contenuto. L'impostazione predefinita è disabilitata.
 - g. (Facoltativo) Affinché una coda FIFO consenta una velocità di trasmissione effettiva più elevata per l'invio e la ricezione di messaggi in coda, seleziona Abilita FIFO ad alta velocità di trasmissione effettiva.

La scelta di questa opzione modifica le opzioni correlate (ambito di deduplicazione e limite di velocità di trasmissione effettiva FIFO) con le impostazioni richieste per abilitare una velocità di trasmissione effettiva elevata per le code FIFO. Se si modifica una delle impostazioni necessarie per utilizzare FIFO ad alta velocità di trasmissione effettiva, si applica la velocità di trasmissione effettiva normale per la coda e la deduplicazione si verifica come specificato.

Per ulteriori informazioni, consultare [Throughput elevato per le code FIFO in Amazon SQS](#) e [Quote di messaggi Amazon SQS](#).

6. (Facoltativo) Definire una policy di accesso. La [policy di accesso](#) definisce gli account, gli utenti e i ruoli che possono accedere alla coda. La policy di accesso definisce anche le operazioni (ad esempio [SendMessage](#), [ReceiveMessage](#) o [DeleteMessage](#)) a cui gli utenti possono accedere. La policy predefinita consente solo al proprietario della coda di inviare e ricevere messaggi.

Per definire la policy di accesso, effettua una delle seguenti operazioni:

- Scegli Basic per configurare chi può inviare messaggi alla coda e chi può ricevere messaggi dalla coda. La console crea la policy in base alle tue scelte e visualizza la policy di accesso risultante nel pannello JSON di sola lettura.
 - Scegli Avanzato per modificare direttamente la policy di accesso JSON. Ciò consente di specificare un set personalizzato di azioni che ogni principale (account, utente o ruolo) può eseguire.
7. Per la policy Redrive allow, scegli Abilitata. Seleziona una delle seguenti opzioni: Consenti tutto, Per coda o Nega tutto. Quando scegli Per coda, specifica un elenco di un massimo di 10 code di origine in base al nome della risorsa Amazon (ARN).
 8. Amazon SQS fornisce la crittografia lato server gestita per impostazione predefinita. Per scegliere un tipo di chiave di crittografia o per disabilitare la crittografia lato server gestita da Amazon SQS, espandi Crittografia. Per ulteriori informazioni sui tipi di chiavi di crittografia, consulta [Configurazione della crittografia lato server per una coda utilizzando chiavi di crittografia gestite da SQL](#) e [Configurazione della crittografia lato server per una coda utilizzando la console Amazon SQS](#).

Note

Con SSE abilitato, le richieste anonime `SendMessage` e `ReceiveMessage` alla coda crittografata verranno rifiutate. Le best practice di sicurezza di Amazon SQS consigliano di non utilizzare richieste anonime. Se desideri inviare richieste anonime a una coda Amazon SQS, assicurati di disabilitare SSE.

9. (Facoltativo) Per configurare una [coda DLQ](#) per la ricezione di messaggi non recapitabili, espandere Coda DLQ.
10. (Facoltativo) Per aggiungere [tag](#) alla coda, espandi Tag.

11. Scegliere Crea coda. Amazon SQS crea la coda e visualizza la pagina dei dettagli della coda.

Amazon SQS diffonde le informazioni sulla nuova coda in tutto il sistema. Poiché Amazon SQS è un sistema distribuito, potrebbe verificarsi un leggero ritardo prima che la console visualizzi la coda nella pagina Code.

Dopo aver creato una coda, puoi [inviarle messaggi](#) e [riceverli ed eliminarli](#). È inoltre possibile [modificare](#) qualsiasi impostazione di configurazione della coda ad eccezione del tipo di coda.

Invio di un messaggio utilizzando una coda FIFO

Dopo aver creato la coda, puoi inviarle un messaggio.

1. Nel riquadro di navigazione sinistro scegliere Code. Nell'elenco delle code, seleziona la coda appena creata.
2. Scegliere Operazioni, quindi Invia e ricevi messaggi.

La console visualizza la pagina Invia e ricevi messaggi.

3. In Corpo del messaggio, inserisci il testo del messaggio.
4. Per una coda First-In-First-Out (FIFO), inserisci un ID del gruppo di messaggi. Per ulteriori informazioni, consulta [Logica di distribuzione delle code FIFO in Amazon SQS](#).
5. (Facoltativo) Per una coda FIFO, puoi inserire un ID di deduplicazione dei messaggi. Se hai abilitato la deduplicazione basata sul contenuto per la coda, l'ID di deduplicazione dei messaggi non è richiesto. Per ulteriori informazioni, consulta [Logica di distribuzione delle code FIFO in Amazon SQS](#).
6. Le code FIFO non supportano i timer sui singoli messaggi. Per ulteriori informazioni, consulta [Timer di messaggi Amazon SQS](#).
7. Scegliere Invia messaggio.

Quando il messaggio viene inviato, la console ne visualizza la conferma. Scegli Visualizza dettagli per visualizzare le informazioni sul messaggio inviato.

Attività comuni per iniziare a usare Amazon SQS

Dopo aver creato una coda e imparato a inviare, ricevere ed eliminare messaggi, potresti provare quanto segue:

- Attiva una [funzione Lambda](#) per elaborare automaticamente i messaggi in arrivo, abilitando flussi di lavoro basati sugli eventi senza la necessità di polling continui.
- [Configura le code](#), tra cui SSE e altre funzionalità.
- [Invia un messaggio con attributi](#)
- [Invio di un messaggio da un VPC.](#)
- Scopri la [funzionalità](#) e [l'architettura](#) di Amazon SQS.
- Scopri [linee guida e avvertenze](#) che ti aiuteranno a ottenere il massimo da Amazon SQS.
- [Esplora gli esempi di Amazon SQS per un AWS SDK, come la Developer Guide.AWS SDK for Java 2.x](#)
- Scopri di più sui [comandi Amazon SQS. AWS CLI](#)
- Scopri di più sulle azioni [API di Amazon SQS.](#)
- Scopri come interagire con Amazon SQS in modo programmatico. [Vedi Lavorare con APIs ed esplora il Development Center AWS :](#)
 - [Java](#)
 - [JavaScript](#)
 - [PHP](#)
 - [Python](#)
 - [Ruby](#)
 - [Windows e .NET](#)
- Scopri come monitorare [costi e risorse](#).
- Scopri come [proteggere i tuoi dati](#).
- Scopri di più sul flusso di [lavoro di Amazon SQS.](#)

Gestione di una coda Amazon SQS

Scopri come gestire le code Amazon SQS utilizzando la console, inclusa la modifica delle impostazioni delle code, la ricezione e l'eliminazione di messaggi, la conferma del vuoto della coda e l'eliminazione o lo svuotamento delle code. Comprendi le migliori pratiche per una gestione efficiente dei messaggi, come l'utilizzo di sondaggi lunghi, la gestione dei timeout di visibilità e la verifica delle metriche tramite dashboard di monitoraggio o il AWS CLI Segui i passaggi pratici per mantenere le code e gestire i messaggi in modo efficace, riducendo al minimo le interruzioni.

Modifica di una coda Amazon SQS tramite la console

Puoi utilizzare la console Amazon SQS per modificare i parametri di configurazione della coda (eccetto il tipo di coda) e modificare o rimuovere le funzionalità in base alle esigenze.

Per modificare una coda Amazon SQS (console)

1. Nella console Amazon SQS, apri la [pagina delle code](#).
2. Seleziona un argomento, quindi scegli Modifica.
3. (Facoltativo) In Configurazione, aggiorna i [parametri di configurazione](#) della coda.
4. (Facoltativo) Per aggiornare la politica di [accesso, in Politica](#) di accesso, modifica la politica JSON.
5. (Facoltativo) Per aggiornare una [policy redrive allow](#) di una coda DLQ, espandi la policy redrive allow.
6. (Facoltativo) Per aggiornare o rimuovere la [crittografia](#), espandi Crittografia.
7. (Facoltativo) Per aggiungere, aggiornare o rimuovere una [coda DLQ](#) (che consente di ricevere messaggi non recapitabili), espandi la coda DLQ.
8. (Facoltativo) Per aggiungere, aggiornare o rimuovere i [tag](#) per la coda, espandi Tag.
9. Scegli Save (Salva).

- La console visualizza la pagina Dettagli per la coda.

Ricezione ed eliminazione di un messaggio in Amazon SQS

Dopo aver inviato i messaggi a una coda Amazon SQS, puoi recuperarli ed eliminarli per elaborare il flusso di lavoro dell'applicazione. Questo processo garantisce una gestione sicura e affidabile dei

messaggi. Questo argomento illustra il recupero e l'eliminazione dei messaggi utilizzando la console Amazon SQS e spiega le impostazioni chiave per ottimizzare questa operazione. Di seguito sono riportati i concetti chiave per la ricezione e l'eliminazione dei messaggi:

1. Ricezione di messaggi

- Quando recuperi messaggi da una coda Amazon SQS, non puoi indirizzare messaggi specifici. Specificate invece il numero massimo di messaggi da recuperare in una singola richiesta (fino a 10).
- A causa della natura distribuita di Amazon SQS, il recupero da una coda con pochi messaggi può restituire una risposta vuota. Per mitigare questo problema:
 - Utilizza il polling lungo, che attende la disponibilità di un messaggio o il timeout del sondaggio. Questo approccio riduce i costi di sondaggio non necessari e migliora l'efficienza.
 - Emetti nuovamente la richiesta, se necessario.

2. Visibilità ed eliminazione dei messaggi

- I messaggi non vengono eliminati automaticamente dopo il recupero. Questa funzionalità consente di rielaborare i messaggi in caso di guasti delle applicazioni o interruzioni della rete.
- Dopo l'elaborazione, è necessario inviare esplicitamente una richiesta di eliminazione per rimuovere definitivamente il messaggio. Questa azione conferma la corretta gestione.
- I messaggi recuperati utilizzando la console Amazon SQS rimangono visibili per essere recuperati nuovamente. Modifica l'impostazione del timeout di visibilità per gli ambienti automatizzati per nascondere temporaneamente i messaggi agli altri consumatori durante l'elaborazione.

3. Timeout di visibilità

- Questa impostazione determina per quanto tempo un messaggio rimane nascosto dopo il recupero. Imposta un timeout appropriato per garantire che i messaggi vengano elaborati una sola volta e per evitare la duplicazione durante l'elaborazione distribuita.

Per ricevere ed eliminare un messaggio utilizzando la console

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Nella pagina Code, scegli la coda da cui desideri ricevere i messaggi, quindi seleziona Invia e ricevi messaggi.
4. Nella pagina Invia e ricevi messaggi, seleziona Sondaggio per i messaggi.

Amazon SQS visualizza una barra di avanzamento che indica la durata del polling. I messaggi recuperati verranno visualizzati nella sezione Messaggi e mostreranno:

- ID messaggio
- Data di invio
- Dimensione
- Conteggio delle ricevute

5. Per eliminare i messaggi, scegli quelli che desideri rimuovere e seleziona Elimina.

Conferma l'eliminazione nella finestra di dialogo Elimina messaggi selezionando Elimina.

[Per ulteriori dettagli sulle operazioni avanzate, tra cui il recupero e l'eliminazione dei messaggi basati su API, consulta la Amazon SQS API Reference Guide.](#)

Conferma che una coda Amazon SQS è vuota

Nella maggior parte dei casi, è possibile utilizzare un [long polling](#) per determinare se una coda è vuota. In rari casi, potresti ricevere risposte vuote anche quando una coda contiene ancora messaggi, specialmente se hai specificato un valore basso per il tempo di attesa per la ricezione dei messaggi quando hai creato la coda. Questa sezione spiega come confermare che una coda sia vuota.

Verifica che una coda sia vuota (console)

1. Impedisci a tutti i produttori di inviare messaggi.
2. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
3. Nel riquadro di navigazione, scegliere Code (Code).
4. Nella pagina Code, scegliere una coda.
5. Scegliere la scheda Monitoring (Monitoraggio).
6. In alto a destra nelle dashboard di monitoraggio, scegli la freccia rivolta verso il basso accanto al simbolo Aggiorna. Dal menu a discesa scegli Aggiornamento automatico. Lascia l'intervallo di aggiornamento a 1 minuto.
7. Osserva le seguenti dashboard:
 - Numero approssimativo di messaggi ritardati
 - Numero approssimativo di messaggi non visibili

- Numero approssimativo di messaggi visibili

Quando tutti mostrano valori 0 per diversi minuti, la coda è vuota.

Per confermare che una coda è vuota (AWS CLI, AWS API)

1. Impedisce a tutti i produttori di inviare messaggi.
2. Eseguire ripetutamente uno dei seguenti comandi:
 - AWS CLI: [get-queue-attributes](#)
 - AWS API: [GetQueueAttributes](#)
3. Osserva le metriche per gli attributi seguenti:
 - `ApproximateNumberOfMessagesDelayed`
 - `ApproximateNumberOfMessagesNotVisible`
 - `ApproximateNumberOfMessagesVisible`

Quando tutti mostrano valori 0 per diversi minuti, la coda è vuota.

Se ti affidi ai CloudWatch parametri di Amazon, assicurati di visualizzare più punti dati zero consecutivi prima di considerare la coda vuota. Per ulteriori informazioni sui CloudWatch parametri, consulta [CloudWatch Metriche disponibili per Amazon SQS](#)

Eliminazione di una coda Amazon SQS

Se non usi più una coda Amazon SQS e non prevedi di utilizzarla nelle prossime future, elimina la coda.

Tip

Se vuoi verificare che una coda sia vuota prima di eliminarla, consulta [Conferma che una coda Amazon SQS è vuota](#).

Puoi eliminare una coda anche quando non è vuota. Se desideri eliminare i messaggi in una coda, ma non la coda, è possibile [svuotare la coda](#).

Per eliminare una coda (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Nella pagina Code, scegliere la coda da eliminare.
4. Scegli Elimina.
5. Nella finestra di dialogo Elimina coda, conferma l'eliminazione inserendo **delete**.
6. Scegli Elimina.

Per eliminare una coda (AWS CLI e un'API)

Scegli il metodo appropriato per eliminare la coda in base alle tue esigenze:

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

Eliminazione dei messaggi da una coda utilizzando la console Amazon SQS

Per mantenere una coda Amazon SQS ma rimuovere tutti i relativi messaggi, puoi eliminare la coda. In questo modo verranno eliminati tutti i messaggi, inclusi quelli attualmente invisibili (in fuga). Il processo di eliminazione può richiedere fino a 60 secondi, quindi attendi tutti i 60 secondi indipendentemente dalle dimensioni della coda.

Important

Quando elimini una coda, non puoi recuperare i messaggi eliminati.

Eliminare una coda (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Nella pagina Code, scegliere la coda da eliminare.
4. In Operazioni, scegliere Eliminare.

5. Nella finestra di dialogo Elimina coda, conferma l'eliminazione inserendo **purge** e scegliendo Elimina.
 - Tutti i messaggi vengono rimossi dalla coda. Nella console viene visualizzato un banner di conferma.

Code standard Amazon SQS

Amazon SQS fornisce code standard come tipo di coda predefinito, supportando un numero quasi illimitato di chiamate API al secondo per azioni come [SendMessage](#), e. [ReceiveMessageDeleteMessage](#). Le code standard garantiscono la consegna dei at-least-once messaggi, ma a causa dell'architettura altamente distribuita, è possibile che venga recapitata più di una copia di un messaggio e occasionalmente i messaggi potrebbero arrivare fuori ordine. Nonostante ciò, le code standard fanno del loro meglio per mantenere l'ordine di invio dei messaggi.

Quando invii un messaggio utilizzando `SendMessage`, Amazon SQS memorizza in modo ridondante il messaggio in più zone di disponibilità (AZs) prima di confermarlo. Questa ridondanza garantisce che nessun errore di un singolo computer, rete o AZ possa rendere i messaggi inaccessibili.

Puoi creare e configurare code utilizzando la console Amazon SQS. Per istruzioni dettagliate, vedi [Creazione di una coda standard utilizzando la console Amazon SQS](#). Per esempi specifici di Java, consulta. [Esempi di SDK Java di Amazon SQS](#)

Casi d'uso per code standard

Le code di messaggi standard sono adatte a vari scenari, a condizione che l'applicazione sia in grado di gestire messaggi che potrebbero arrivare più di una volta o che potrebbero non funzionare correttamente. Gli esempi includono:

- Separazione delle richieste live degli utenti dal lavoro intensivo in background: gli utenti possono caricare contenuti multimediali mentre il sistema li ridimensiona o li codifica in background.
- Allocazione delle attività a più nodi di lavoro: ad esempio, gestione di un volume elevato di richieste di convalida delle carte di credito.
- Messaggi in batch per elaborazioni future: pianificazione dell'aggiunta di più voci a un database in un secondo momento.

Per informazioni sulle quote relative alle code standard, vedere. [Quote di coda standard di Amazon SQS](#)

Per le best practice relative all'utilizzo di code standard, consulta [Best practice di Amazon SQS](#).

Consegna Amazon at-least-once SQS

Amazon SQS archivia copie dei messaggi su più server per ridondanza e disponibilità elevata. In rare occasioni, uno dei server che memorizza una copia di un messaggio potrebbe non essere disponibile quando ricevi o elimini un messaggio.

In questo caso, la copia del messaggio non viene eliminata dal server che non è disponibile e potresti riceverla nuovamente quando ricevi messaggi. Progetta le tue applicazioni affinché siano idempotent (non devono essere condizionate negativamente quando elaborano lo stesso messaggio più di una volta).

Identificatori di code e messaggi Amazon SQS

Questo argomento descrive gli identificatori delle code standard e FIFO. Questi identificatori possono aiutarti a trovare e modificare code e messaggi specifici.

Identificatori per le code standard di Amazon SQS

Per ulteriori informazioni, consulta i seguenti argomenti nella [Documentazione di riferimento delle API di Amazon Simple Queue Service](#).

Nome e URL della coda

Quando crei una nuova coda, è necessario specificare un nome coda univoco per il tuo account e la tua regione AWS. Amazon SQS assegna a ogni coda che crei un identificatore chiamato URL coda che include il nome della coda e altri componenti Amazon SQS. Se desideri eseguire un'operazione su una coda, devi fornire il relativo URL coda.

Di seguito è riportato l'URL coda per una coda denominata MyQueue, di proprietà di un utente con il numero di account AWS 123456789012.

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

È possibile recuperare l'URL di una coda a livello di codice elencando le code e analizzando la stringa che segue il numero di account. Per ulteriori informazioni, consulta [ListQueues](#).

ID messaggio

Ogni messaggio riceve un ID messaggio assegnato dal sistema che Amazon SQS restituisce nella risposta [SendMessage](#). Questo identificatore è utile per l'identificazione di messaggi. La durata massima di un ID messaggio è di 100 caratteri.

Handle di ricezione

Ogni volta che ricevi un messaggio da una coda, ricevi un handle di ricezione per tale messaggio. Questo handle è associato all'operazione di ricezione del messaggio, non al messaggio stesso. Per eliminare il messaggio o per modificarne la visibilità, devi fornire l'handle di ricezione (non l'ID messaggio). Pertanto, devi sempre ricevere un messaggio prima di poterlo eliminare (non puoi inserire un messaggio nella coda e poi richiamarlo). La durata massima di un handle di ricezione è di 1024 caratteri.

Important

Se ricevi un messaggio più di una volta, ogni volta che lo ricevi ottieni un diverso handle di ricezione. Devi fornire l'handle di ricezione ricevuto più di recente quando richiedi di eliminare il messaggio (in caso contrario, il messaggio potrebbe non essere eliminato).

Di seguito è riportato un esempio di maniglia di ricezione suddivisa su tre righe.

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

Code FIFO di Amazon SQS

Le code FIFO (First-In-First-Out) hanno tutte le funzionalità delle [code standard](#), ma sono progettate per migliorare la messaggistica tra le applicazioni quando l'ordine delle operazioni e degli eventi è fondamentale o dove i duplicati non possono essere tollerati.

Le caratteristiche più importanti delle code FIFO sono l'invio in formato FIFO (First-In-First-Out) e l'elaborazione una sola volta:

- L'ordine in cui i messaggi vengono inviati e ricevuti viene rigorosamente rispettato e un messaggio viene consegnato una sola volta e non è disponibile finché un consumatore non lo elabora ed elimina.
- I duplicati non vengono introdotti nella coda.

Inoltre, le code FIFO supportano gruppi di messaggi che consentono più gruppi di messaggi ordinati all'interno di un'unica coda. Non è previsto alcun limite al numero di gruppi di messaggi all'interno di una coda FIFO.

Di seguito sono riportati alcuni esempi di situazioni in cui è possibile utilizzare le code FIFO:

1. Sistema di gestione degli ordini di e-commerce in cui l'ordine è fondamentale
2. Integrazione con sistemi di terze parti in cui gli eventi devono essere elaborati in ordine
3. Elaborazione degli input inseriti dall'utente nell'ordine inserito
4. Comunicazioni e reti: invio e ricezione di dati e informazioni nello stesso ordine
5. Sistemi informatici: garanzia che i comandi immessi dall'utente vengano eseguiti nell'ordine corretto
6. Istituti di istruzione: impedire a uno studente di iscriversi a un corso prima di registrare un account.
7. Sistema di biglietteria online: dove i biglietti vengono distribuiti in base al principio "primo arrivato, primo servito"

Note

Le code FIFO forniscono inoltre l'elaborazione exactly-once, ma dispongono di un numero limitato di transazioni al secondo (TPS): Puoi utilizzare la modalità velocità di trasmissione effettiva elevata di Amazon SQS con la coda FIFO per aumentare il limite di transazioni.

Per dettagli sull'utilizzo della modalità velocità di trasmissione effettiva elevata, consulta [Throughput elevato per le code FIFO in Amazon SQS](#). Per ulteriori informazioni sulle quote di velocità di trasmissione effettiva, vedere [the section called "Quote dei messaggi"](#).

Le code FIFO di Amazon SQS sono disponibili in tutte le regioni in cui è disponibile Amazon SQS.

Per ulteriori informazioni sull'utilizzo delle code FIFO con ordinamenti complessi, consulta [Soluzione delle sfide di ordinamento complesse con le code FIFO di Amazon SQS](#).

Per informazioni su come creare e configurare code utilizzando la console Amazon SQS, consulta [Creazione di una coda standard utilizzando la console Amazon SQS](#). Per esempi in Java, consulta [Esempi di SDK Java di Amazon SQS](#).

Per le migliori pratiche per lavorare con le code FIFO, consulta. [Best practice di Amazon SQS](#)

Termini chiave della coda FIFO di Amazon SQS

I seguenti termini chiave possono aiutarti a comprendere meglio le funzionalità delle code FIFO. Per ulteriori informazioni, consulta la [Guida di riferimento per l'API di Amazon Simple Storage Service](#).

Client

L'Amazon SQS Buffered Asynchronous Client attualmente non supporta le code FIFO.

ID deduplicazione messaggi

Un token utilizzato nelle code FIFO di Amazon SQS per identificare in modo univoco i messaggi e prevenire la duplicazione. Se più messaggi con lo stesso ID di deduplicazione vengono inviati entro un intervallo di deduplicazione di 5 minuti, vengono trattati come duplicati e viene consegnata una sola copia. Se non specifichi un ID di deduplicazione e la deduplicazione basata sul contenuto è abilitata, Amazon SQS genera un ID di deduplicazione eseguendo l'hashing del corpo del messaggio. Questo meccanismo garantisce la consegna esattamente una volta eliminando i messaggi duplicati entro il periodo di tempo specificato.

Note

Amazon SQS continua a tracciare l'ID di deduplicazione anche dopo la ricezione e l'eliminazione del messaggio.

ID gruppo di messaggi

Nelle code FIFO (First-In-First-Out), `MessageGroupId` è un attributo che organizza i messaggi in gruppi distinti. I messaggi all'interno dello stesso gruppo di messaggi vengono sempre elaborati uno alla volta, in ordine rigoroso, assicurando che non vengano elaborati due messaggi dello stesso gruppo contemporaneamente. Nelle code standard, l'utilizzo `MessageGroupId` consente [code equie](#). Se è richiesto un ordine rigoroso, utilizza una coda FIFO.

ID tentativo richiesta di ricezione

L'ID del tentativo di ricezione è un token univoco utilizzato per deduplicare [ReceiveMessage](#) chiamate in Amazon SQS.

Numero sequenza

Il numero grande e non consecutivo che Amazon SQS assegna a ciascun messaggio.

Servizi

Se l'applicazione utilizza più AWS servizi o una combinazione di AWS servizi esterni, è importante capire quali funzionalità del servizio non supportano le code FIFO.

Alcuni servizi AWS o servizi esterni che inviano notifiche ad Amazon SQS potrebbero non essere compatibili con le code FIFO, nonostante consentano di impostare una coda FIFO come destinazione.

Le seguenti funzionalità dei AWS servizi non sono attualmente compatibili con le code FIFO:

- [Notifiche di eventi Amazon S3](#)
- [Hook del ciclo di vita del dimensionamento automatico](#)
- [AWS IoT Azioni relative alle regole](#)
- [AWS Lambda Code DLQ](#)

Per informazioni sulla compatibilità di altri servizi con le code FIFO, consulta la documentazione del servizio.

Logica di distribuzione delle code FIFO in Amazon SQS

I seguenti concetti chiariscono come le code FIFO di Amazon SQS gestiscono l'invio e la ricezione di messaggi, in particolare quando si tratta di ordinamento e gruppo di messaggi. IDs

Invio di messaggi

Le code FIFO di Amazon SQS preservano l'ordine dei messaggi utilizzando una IDs deduplicazione e un gruppo di messaggi unici. IDs Questo argomento evidenzia l'importanza del gruppo di messaggi IDs per mantenere un ordine rigoroso all'interno dei gruppi e mette in evidenza le migliori pratiche per garantire una consegna affidabile e ordinata dei messaggi tra più produttori.

1. Conservazione degli ordini

- Quando più messaggi vengono inviati in successione a una coda FIFO con deduplicazione univoca dei messaggi, IDs Amazon SQS li archivia e ne conferma la trasmissione. Questi messaggi vengono quindi ricevuti ed elaborati nell'ordine esatto in cui sono stati trasmessi.

2. ID gruppo di messaggi

- Nelle code FIFO, i messaggi vengono ordinati in base all'ID del gruppo di messaggi. Se più produttori o thread inviano messaggi con lo stesso ID di gruppo di messaggi, Amazon SQS garantisce che vengano archiviati ed elaborati nell'ordine in cui arrivano.
- Best practice: per garantire un ordine rigoroso dei messaggi tra più produttori, assegna un ID di gruppo di messaggi univoco a tutti i messaggi di ciascun produttore.

3. Ordinazione per gruppo

- La logica di coda FIFO si applica in base all'ID del gruppo di messaggi:
 - Ogni ID del gruppo di messaggi rappresenta un gruppo di messaggi distinto e ordinato.
 - All'interno di un ID di gruppo di messaggi, tutti i messaggi vengono inviati e ricevuti in ordine rigoroso.
 - I messaggi con gruppi di messaggi diversi IDs possono arrivare o essere elaborati in modo errato l'uno rispetto all'altro.
- Requisito: è necessario associare un ID del gruppo di messaggi a ciascun messaggio. Se un messaggio viene inviato senza un ID di gruppo, l'azione ha esito negativo.
- Scenario a gruppo singolo: se desideri che tutti i messaggi vengano elaborati in ordine rigoroso, utilizza lo stesso ID di gruppo di messaggi per ogni messaggio.

Ricezione di messaggi

Le code FIFO di Amazon SQS gestiscono il recupero dei messaggi, inclusa l'elaborazione in batch, le garanzie degli ordini FIFO e le limitazioni alla richiesta di gruppi di messaggi specifici. IDs Questo argomento spiega come Amazon SQS recupera i messaggi all'interno e tra gruppi di messaggi IDs mantenendo regole rigorose di ordinamento e visibilità.

1. Recupero del batch

- Quando si ricevono messaggi da una coda FIFO con più gruppi di messaggi IDs, Amazon SQS:
 - Tenta di restituire il maggior numero possibile di messaggi con lo stesso ID di gruppo di messaggi in una singola chiamata.
 - Consente ad altri consumatori di elaborare IDs contemporaneamente messaggi provenienti da diversi gruppi di messaggi.
- Chiarimento importante
 - È possibile ricevere più messaggi dallo stesso ID di gruppo di messaggi in un unico batch (fino a 10 messaggi in una singola chiamata utilizzando il `MaxNumberOfMessages` parametro).
 - Tuttavia, non puoi ricevere messaggi aggiuntivi dallo stesso ID di gruppo di messaggi nelle richieste successive fino a quando:
 - I messaggi attualmente ricevuti vengono eliminati oppure
 - Diventano nuovamente visibili (ad esempio, dopo la scadenza del timeout di visibilità).

2. Garanzia d'ordine FIFO

- I messaggi recuperati in un batch mantengono il loro ordine FIFO all'interno del gruppo.
- Se sono disponibili meno di 10 messaggi per lo stesso ID di gruppo di messaggi, Amazon SQS può includere messaggi di altri gruppi IDs di messaggi nello stesso batch, ma ogni gruppo mantiene l'ordine FIFO.

3. Limitazioni per i consumatori

- Non è possibile richiedere esplicitamente di ricevere messaggi da un ID di gruppo di messaggi specifico.

Tentativi di riesecuzione multipli

Produttori e consumatori possono riprovare in sicurezza le azioni non riuscite nelle code FIFO di Amazon SQS senza interrompere l'ordine dei messaggi o introdurre duplicati. Questo argomento evidenzia come i timeout di deduplicazione IDs e visibilità garantiscono l'integrità dei messaggi durante i nuovi tentativi.

1. Il produttore riprova

- Se un' [SendMessage](#) azione fallisce, il produttore può riprovare a inviare il messaggio più volte con lo stesso ID di deduplicazione del messaggio.
- Se il produttore riceve almeno una conferma prima della scadenza dell'intervallo di deduplicazione, riprova a:

- Non inserire messaggi duplicati.
- Non alterate l'ordine dei messaggi.

2. I tentativi dei consumatori

- Se un'[ReceiveMessage](#)azione fallisce, il consumatore può riprovare tutte le volte che è necessario utilizzando lo stesso ID del tentativo di richiesta di ricezione.
- A condizione che il consumatore riceva almeno una conferma prima della scadenza del timeout di visibilità, riprova a:
 - Non interrompere l'ordine dei messaggi.

Note aggiuntive sul comportamento FIFO

Scopri come gestire i timeout di visibilità, abilitare l'elaborazione parallela con più gruppi di messaggi e garantire un'elaborazione sequenziale rigorosa in scenari a gruppo IDs singolo.

1. Gestione del timeout di visibilità

- Quando un messaggio viene recuperato ma non eliminato, rimane invisibile fino alla scadenza del timeout di visibilità.
- Non vengono restituiti altri messaggi dello stesso ID del gruppo di messaggi finché il primo messaggio non viene eliminato o diventa nuovamente visibile.

2. Concorrenza ed elaborazione parallela

- Le code FIFO consentono l'elaborazione parallela dei messaggi tra diversi gruppi di messaggi. IDs
- Per massimizzare la concorrenza, progetta il tuo sistema con più gruppi IDs di messaggi per flussi di lavoro indipendenti.

3. Scenari a gruppo singolo

- Per un'elaborazione sequenziale rigorosa di tutti i messaggi in una coda FIFO, utilizza un unico ID di gruppo di messaggi per tutti i messaggi in coda.

Esempi per una migliore comprensione

Di seguito sono riportati scenari pratici che illustrano il comportamento delle code FIFO in Amazon SQS.

1. Scenario 1: ID di gruppo singolo

- Un produttore invia cinque messaggi con lo stesso ID del gruppo di messaggi Gruppo A.
- Un consumatore riceve questi messaggi nell'ordine FIFO. Finché il consumatore non elimina questi messaggi o non scade il timeout di visibilità, non vengono ricevuti altri messaggi dal Gruppo A.

2. Scenario 2: gruppo multiplo IDs

- Un produttore invia cinque messaggi al Gruppo A e 5 al Gruppo B.
- Il Consumatore 1 elabora i messaggi del Gruppo A, mentre il Consumatore 2 elabora i messaggi del Gruppo B. Ciò consente l'elaborazione parallela con ordini rigorosi mantenuti all'interno di ciascun gruppo.

3. Scenario 3: recupero in batch

- Un produttore invia sette messaggi al Gruppo A e tre al Gruppo B.
- Un singolo consumatore recupera fino a 10 messaggi. Se la coda lo consente, potrebbe restituire:
 - Sette messaggi dal Gruppo A e tre dal Gruppo B (o meno se sono disponibili meno messaggi da un singolo gruppo).

Elaborazione una sola volta in Amazon SQS

A differenza delle code standard, le code FIFO non introducono messaggi duplicati. Le code FIFO consentono di evitare l'invio di un duplicato a una coda. Se ritenti l'operazione `SendMessage` entro l'intervallo di deduplicazione di 5 minuti, Amazon SQS non introduce duplicati nella coda.

Per configurare la deduplicazione, devi procedere in uno dei seguenti modi:

- Abilitare la deduplicazione basata sui contenuti. Questo istruisce Amazon SQS a utilizzare un hash SHA-256 per generare l'ID di deduplicazione messaggio utilizzando il corpo del messaggio, ma non gli attributi dello stesso. Per ulteriori informazioni, consulta la documentazione sulle operazioni [CreateQueue](#), [GetQueueAttributes](#) e [SetQueueAttributes](#) nella Guida di riferimento delle API Amazon Simple Queue Service.
- Offrire in modo esplicito l'ID di deduplicazione messaggio (o visualizzare il numero di sequenza) per il messaggio. Per ulteriori informazioni, consulta la documentazione sulle operazioni [SendMessage](#), [SendMessageBatch](#) e [ReceiveMessage](#) nella Guida di riferimento delle API Amazon Simple Queue Service.

Passaggio da una coda standard a una coda FIFO in Amazon SQS

Se l'applicazione esistente utilizza le code standard e si desidera sfruttare le funzionalità di ordinamento o di elaborazione esatte delle code FIFO, è necessario configurare correttamente sia la coda che l'applicazione.

Considerazioni chiave

- Creazione di una coda FIFO: non è possibile convertire una coda standard esistente in una coda FIFO. È necessario creare una nuova coda FIFO per l'applicazione o eliminare la coda standard esistente e ricrearla come coda FIFO.
- Parametro di ritardo: le code FIFO non supportano ritardi per messaggio, ma solo ritardi per coda. Se l'applicazione imposta il `DelaySeconds` parametro su ogni messaggio, è necessario modificarlo in modo da impostarlo sull'intera coda. `DelaySeconds`
- ID del gruppo di messaggi: fornisci un [ID del gruppo di messaggi](#) per ogni messaggio inviato. Questo ID consente l'elaborazione parallela dei messaggi mantenendo il rispettivo ordine. Utilizza una dimensione aziendale granulare per l'ID del gruppo di messaggi per una migliore scalabilità con le code FIFO. Maggiore è il gruppo di messaggi a IDs cui distribuisce i messaggi, maggiore è il numero di messaggi disponibili per l'utilizzo.
- Modalità High Throughput: utilizza la modalità [ad alta velocità](#) di trasmissione consigliata per le code FIFO per aumentare la velocità effettiva. Per ulteriori informazioni sulle quote di messaggistica, vedere. [Quote di messaggi Amazon SQS](#)

Lista di controllo per passare alle code FIFO

Prima di inviare messaggi a una coda FIFO, conferma quanto segue:

1. Configura le impostazioni del ritardo
 - Modifica la tua applicazione per rimuovere i ritardi per messaggio.
 - Imposta il `DelaySeconds` parametro sull'intera coda.
2. Imposta il gruppo di messaggi IDs
 - Organizza i messaggi in gruppi di messaggi specificando un ID del gruppo di messaggi basato su una dimensione aziendale.
 - Utilizza dimensioni aziendali più granulari per migliorare la scalabilità.
3. Gestisci la deduplicazione dei messaggi

- Se la tua applicazione non è in grado di inviare messaggi con corpi di messaggio identici, fornisci un ID di deduplicazione dei messaggi univoco per ogni messaggio.
- Se l'applicazione invia messaggi con corpi dei messaggi unici, abilita la deduplicazione basata sul contenuto.

4. Configura il consumatore

- In genere, non sono necessarie modifiche al codice per il consumatore.
- Se l'elaborazione dei messaggi richiede molto tempo e il timeout di visibilità è impostato su un valore elevato, valuta la possibilità di aggiungere un ID del tentativo di richiesta di ricezione a ciascuna `ReceiveMessage` azione. Ciò consente di ripetere i tentativi di ricezione in caso di errori di rete e impedisce che le code si interrompano a causa di tentativi di ricezione non riusciti.

Seguendo questi passaggi, potete assicurarvi che l'applicazione funzioni correttamente con le code FIFO, sfruttando appieno le relative funzionalità di ordinamento e di elaborazione Exactly-Once. Per informazioni più dettagliate, consulta il [riferimento alle API di Amazon Simple Queue Service](#).

Coda FIFO di Amazon SQS e comportamento di concorrenza Lambda

Utilizzando una coda FIFO (First-In-First-Out) con Lambda, puoi garantire l'elaborazione ordinata dei messaggi all'interno di ciascun gruppo di messaggi. La funzione Lambda non eseguirà più istanze per lo stesso gruppo di messaggi contemporaneamente, mantenendo così l'ordine. Tuttavia, può scalare fino a gestire più gruppi di messaggi in parallelo, garantendo un'elaborazione efficiente del carico di lavoro della coda. I seguenti punti descrivono il comportamento delle funzioni Lambda durante l'elaborazione dei messaggi da una coda FIFO di Amazon SQS rispetto al gruppo di messaggi: IDs

- Singola istanza per gruppo di messaggi: in qualsiasi momento, solo un'istanza Lambda elaborerà i messaggi da un ID di gruppo di messaggi specifico. Ciò garantisce che i messaggi all'interno dello stesso gruppo vengano elaborati in ordine, mantenendo l'integrità della sequenza FIFO.
- Elaborazione simultanea di diversi gruppi: Lambda può elaborare contemporaneamente messaggi provenienti da IDs gruppi di messaggi diversi utilizzando più istanze. Ciò significa che mentre un'istanza della funzione Lambda gestisce i messaggi da un ID di gruppo di messaggi, altre istanze possono gestire contemporaneamente i messaggi di un altro gruppo di messaggi IDs, sfruttando le funzionalità di concorrenza di Lambda per elaborare più gruppi in parallelo.

Raggruppamento dei messaggi in coda FIFO

Le code FIFO assicurano che i messaggi vengano elaborati nell'ordine esatto in cui vengono inviati. Utilizzano un ID di gruppo di messaggi per raggruppare i messaggi che devono essere elaborati in sequenza.

I messaggi all'interno dello stesso gruppo di messaggi vengono elaborati in ordine e per mantenere questo ordine viene elaborato un solo messaggio per ogni gruppo alla volta.

Concorrenza Lambda con le code FIFO

Dopo aver creato la coda, puoi inviarle un messaggio.

Quando configuri una funzione Lambda per elaborare i messaggi da una coda FIFO di Amazon SQS, Lambda rispetta le garanzie di ordinazione fornite dalla coda FIFO. I seguenti punti descrivono il comportamento delle funzioni Lambda in termini di concorrenza e scalabilità durante l'elaborazione di messaggi da una coda FIFO di Amazon SQS quando si utilizza un gruppo di messaggi. IDs

- Concorrenza all'interno dei gruppi di messaggi: solo un'istanza Lambda elabora i messaggi per un particolare ID di gruppo di messaggi alla volta. Ciò garantisce che i messaggi all'interno di un gruppo vengano gestiti in sequenza.
- Scalabilità e gruppi di messaggi multipli: sebbene Lambda sia in grado di eseguire la scalabilità verso l'alto per elaborare i messaggi contemporaneamente, questa scalabilità avviene tra diversi gruppi di messaggi. Se disponi di più gruppi di messaggi, Lambda può elaborare più gruppi in parallelo, con ogni gruppo gestito da un'istanza Lambda separata.

Per ulteriori informazioni, consulta [Scalabilità e concorrenza in Lambda](#) nella Guida dell'operatore AWS Lambda.

Esempio di caso d'uso

Supponiamo che la coda FIFO riceva messaggi con lo stesso ID di gruppo di messaggi e che la funzione Lambda abbia un limite di concorrenza elevato (fino a 1000).

Se un messaggio dall'ID di gruppo «A» è in fase di elaborazione e arriva un altro messaggio dall'ID di gruppo «A», il secondo messaggio non attiverà una nuova istanza Lambda finché il primo messaggio non sarà completamente elaborato.

Tuttavia, se arrivano messaggi dal gruppo IDs 'A' e 'B', entrambi i messaggi possono essere elaborati contemporaneamente da istanze Lambda separate.

Throughput elevato per le code FIFO in Amazon SQS

Le code FIFO ad alto throughput di Amazon SQS gestiscono in modo efficiente un'elevata velocità di trasmissione dei messaggi mantenendo al contempo un ordine rigoroso dei messaggi, garantendo affidabilità e scalabilità per le applicazioni che elaborano numerosi messaggi. Questa soluzione è ideale per scenari che richiedono sia un throughput elevato che la consegna di messaggi ordinati.

Le code FIFO ad alta velocità di Amazon SQS non sono necessarie in scenari in cui l'ordinamento rigoroso dei messaggi non è fondamentale e in cui il volume dei messaggi in arrivo è relativamente basso o sporadico. Ad esempio, se disponi di un'applicazione su piccola scala che elabora messaggi poco frequenti o non sequenziali, la complessità e i costi aggiuntivi associati alle code FIFO ad alta velocità potrebbero non essere giustificati. Inoltre, se la tua applicazione non richiede le funzionalità di throughput avanzate fornite dalle code FIFO ad alto throughput, optare per una coda Amazon SQS standard potrebbe essere più conveniente e più semplice da gestire.

Per migliorare la capacità di richiesta nelle code FIFO ad alta velocità di trasmissione, si consiglia di aumentare il numero di gruppi di messaggi. Per ulteriori informazioni sulle quote di messaggi a throughput elevato, consulta [Amazon SQS service quotas](#) nella Riferimenti generali di Amazon Web Services.

Per informazioni sulle quote per coda e sulle strategie di distribuzione dei dati, vedere e. [Quote di messaggi Amazon SQS Partizioni e distribuzione dei dati per una velocità di trasmissione effettiva elevata per le code FIFO SQS](#)

Casi d'uso per un throughput elevato per le code FIFO di Amazon SQS

I seguenti casi d'uso evidenziano le diverse applicazioni delle code FIFO ad alta velocità e ne dimostrano l'efficacia in diversi settori e scenari:

1. Elaborazione dei dati in tempo reale: le applicazioni che gestiscono flussi di dati in tempo reale, come l'elaborazione di eventi o l'inserimento di dati di telemetria, possono trarre vantaggio dalle code FIFO ad alta velocità per gestire l'afflusso continuo di messaggi preservandone l'ordine per un'analisi accurata.
2. Elaborazione degli ordini di e-commerce: nelle piattaforme di e-commerce in cui il mantenimento dell'ordine delle transazioni con i clienti è fondamentale, le code FIFO ad alta produttività

assicurano che gli ordini vengano elaborati in sequenza e senza ritardi, anche durante l'alta stagione degli acquisti.

3. Servizi finanziari: gli istituti finanziari che gestiscono dati di negoziazione o transazionali ad alta frequenza si affidano a code FIFO ad alta velocità per elaborare i dati di mercato e le transazioni con una latenza minima, rispettando al contempo i severi requisiti normativi per l'ordinazione dei messaggi.
4. Streaming multimediale: le piattaforme di streaming e i servizi di distribuzione multimediale utilizzano code FIFO ad alta velocità per gestire la distribuzione di file multimediali e contenuti in streaming, garantendo esperienze di riproduzione fluide per gli utenti e mantenendo il corretto ordine di distribuzione dei contenuti.

Partizioni e distribuzione dei dati per una velocità di trasmissione effettiva elevata per le code FIFO SQS

Amazon SQS memorizza i dati delle code FIFO nelle partizioni. Una partizione è un'allocazione di spazio di archiviazione per una coda che viene replicata automaticamente su più zone di disponibilità all'interno di una regione. AWS non si gestiscono le partizioni. Al contrario, Amazon SQS gestisce la gestione delle partizioni.

Per le code FIFO, Amazon SQS modifica il numero di partizioni in una coda nelle seguenti situazioni:

- Se la frequenza di richiesta corrente si avvicina o supera quella supportata dalle partizioni esistenti, vengono allocate partizioni aggiuntive finché la coda non raggiunge la quota regionale. Per informazioni sulle quote, consulta [Quote di messaggi Amazon SQS](#).
- Se le partizioni correnti sono poco utilizzate, il numero di partizioni può essere ridotto.

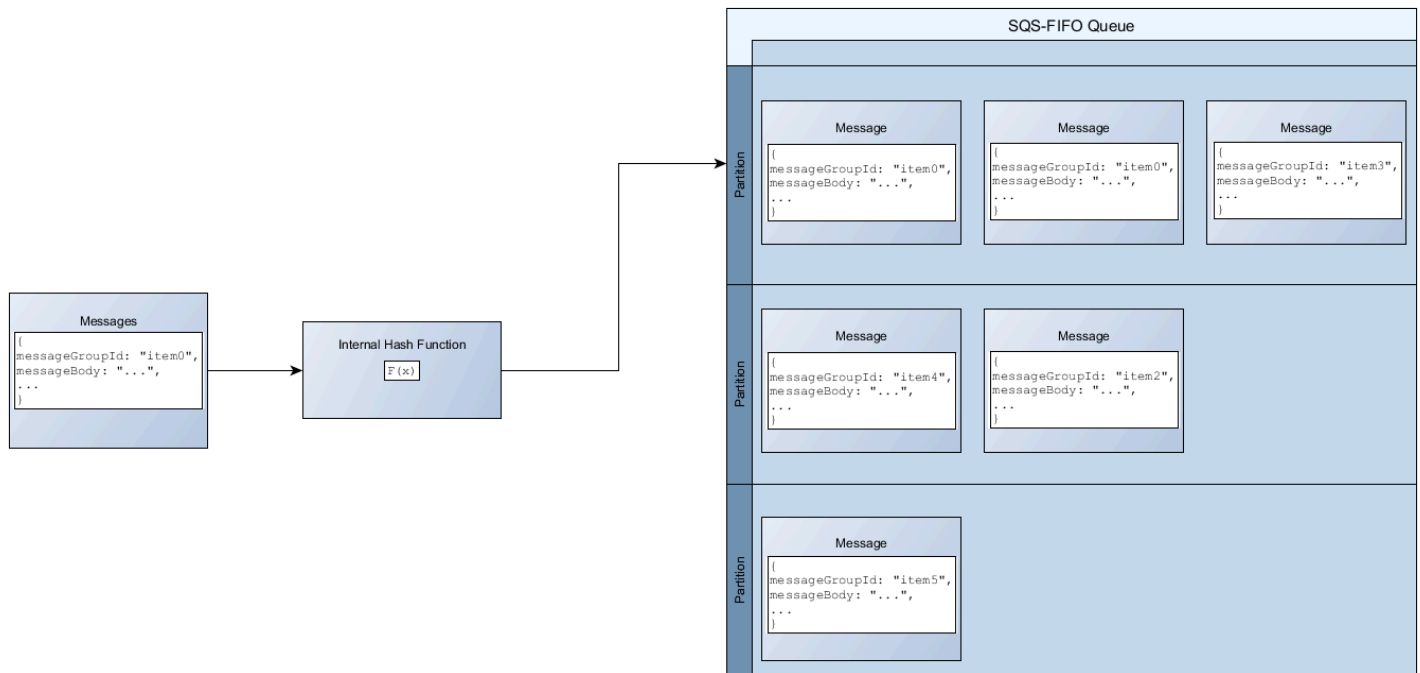
La gestione delle partizioni avviene automaticamente in background ed è trasparente per le tue applicazioni. La coda e i messaggi sono sempre disponibili.

Distribuzione dei dati per gruppo di messaggi IDs

Per aggiungere un messaggio a una coda FIFO, Amazon SQS utilizza il valore dell'ID del gruppo di messaggi di ogni messaggio come input per una funzione hash interna. Il valore di output dalla funzione hash determina la partizione in cui verrà memorizzato il messaggio.

Il seguente diagramma mostra una coda che si estende su più partizioni. L'ID del gruppo di messaggi della coda si basa sul numero dell'elemento. Amazon SQS utilizza la funzione hash per determinare

dove memorizzare un nuovo elemento, in questo caso in base al valore hash della stringa `item0`. Si noti che gli elementi vengono memorizzati nello stesso ordine in cui vengono aggiunti alla coda. La posizione di ciascun item è determinata dal valore hash dell'ID del gruppo di messaggi.



Note

Amazon SQS è ottimizzato per la distribuzione uniforme degli elementi nelle partizioni di una coda FIFO, indipendentemente dal numero di partizioni. AWS consiglia di utilizzare un gruppo di messaggi IDs che può avere un gran numero di valori distinti.

Ottimizzazione dell'utilizzo delle partizioni

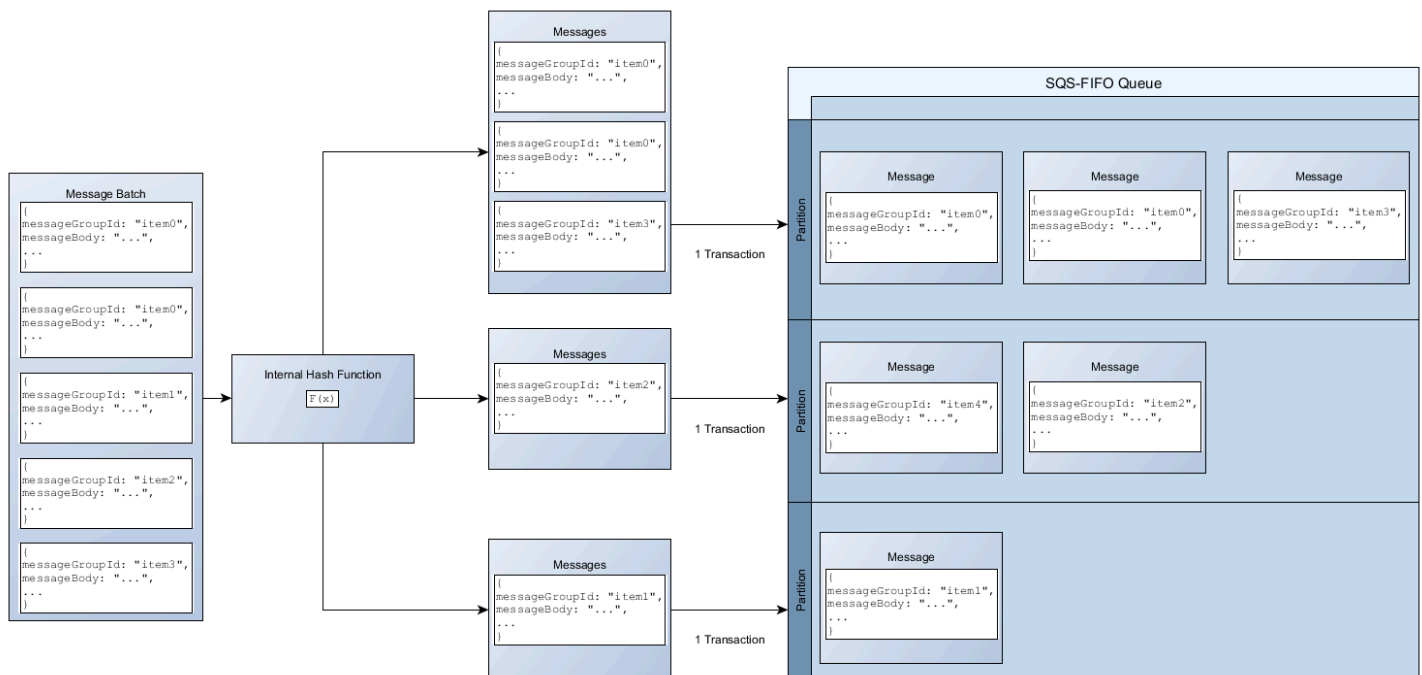
Ogni partizione supporta fino a 3.000 messaggi al secondo con batch o fino a 300 messaggi al secondo per operazioni di invio, ricezione ed eliminazione nelle regioni supportate. Per ulteriori informazioni sulle quote di messaggi a throughput elevato, consulta [Amazon SQS service quotas](#) nella Riferimenti generali di Amazon Web Services.

Quando si utilizza il batch APIs, ogni messaggio viene instradato in base al processo descritto in [Distribuzione dei dati per gruppo di messaggi IDs](#). I messaggi indirizzati alla stessa partizione vengono raggruppati ed elaborati in un'unica transazione.

Per ottimizzare l'utilizzo delle partizioni per l'SendMessageBatchAPI, AWS consiglia di raggruppare i messaggi in batch con lo stesso gruppo di messaggi, quando possibile. IDs

Per ottimizzare l'utilizzo della partizione per DeleteMessageBatch and ChangeMessageVisibilityBatch APIs, AWS consiglia di utilizzare le ReceiveMessage richieste con il MaxNumberOfMessages parametro impostato su 10 e di raggruppare in batch gli handle di ricezione restituiti da una singola richiesta. ReceiveMessage

Nell'esempio seguente, viene inviato un batch di messaggi con diversi gruppi di messaggi. IDs Il batch viene suddiviso in tre gruppi, ognuno dei quali viene conteggiato nella quota della partizione.



Note

Amazon SQS garantisce solo che i messaggi con la funzione hash interna dello stesso ID del gruppo di messaggi siano raggruppati all'interno di una richiesta batch. A seconda dell'output della funzione hash interna e del numero di partizioni, è IDs possibile raggruppare i messaggi con gruppi di messaggi diversi. Poiché la funzione hash o il numero di partizioni possono cambiare in qualsiasi momento, i messaggi raggruppati in un punto non possono essere raggruppati in un secondo momento.

Abilitazione di un throughput elevato per le code FIFO in Amazon SQS

Puoi abilitare una velocità di trasmissione effettiva elevata per qualsiasi coda FIFO nuova o esistente. La funzionalità include tre nuove opzioni per la creazione e la modifica delle code FIFO:

- **Abilita FIFO a velocità di trasmissione effettiva elevata:** da utilizzare per abilitare una velocità di trasmissione effettiva elevata per i messaggi nella coda FIFO attuale.
- **Ambito di deduplicazione:** specifica se la deduplicazione avviene a livello di coda o di gruppo di messaggi.
- **Limite di velocità effettiva FIFO:** specifica se la quota di velocità effettiva per i messaggi nella coda FIFO è impostata a livello di coda o di gruppo di messaggi.

Per abilitare una velocità di trasmissione effettiva elevata per una coda FIFO (console)

1. Inizia a [creare](#) o [modificare](#) una coda FIFO.
2. Quando si specificano le opzioni per la coda, scegliere **Abilita FIFO ad alta velocità di trasmissione effettiva**.

L'abilitazione di una velocità di trasmissione effettiva elevata per le code FIFO imposta le opzioni correlate come segue:

- **Ambito di deduplicazione** è impostato su Gruppo di messaggi, l'impostazione richiesta per l'utilizzo di una velocità di trasmissione effettiva elevata per le code FIFO.
- **Limite FIFO di velocità di trasmissione effettiva** è impostato su ID gruppo di messaggi, l'impostazione richiesta per l'utilizzo di una velocità di trasmissione effettiva elevata per le code FIFO.

Se si modifica una delle impostazioni necessarie per utilizzare le code FIFO ad alta velocità di trasmissione effettiva, si applica la velocità di trasmissione effettiva normale per la coda e la deduplicazione si verifica come specificato.

3. Continuare a specificare tutte le opzioni per la coda. Al termine, scegliere **Crea coda** o **Salva**.

Dopo aver creato o modificato la coda FIFO, puoi [inviarle messaggi](#) e [riceverli ed eliminarli](#), il tutto con un TPS più elevato. Per quote di velocità di trasmissione effettiva elevata, consulta **Velocità di trasmissione effettiva dei messaggi** in [Quote di messaggi Amazon SQS](#).

Identificatori di code e messaggi FIFO in Amazon SQS

Questa sezione descrive gli identificatori delle code FIFO. Questi identificatori possono aiutarti a trovare e modificare code e messaggi specifici.

Identificatori per le code FIFO in Amazon SQS

Per ulteriori informazioni, consulta i seguenti argomenti nella [Documentazione di riferimento delle API di Amazon Simple Queue Service](#).

Nome e URL della coda

Quando crei una nuova coda, è necessario specificare un nome coda univoco per il tuo account e la tua regione AWS. Amazon SQS assegna a ogni coda che crei un identificatore chiamato URL coda che include il nome della coda e altri componenti Amazon SQS. Se desideri eseguire un'operazione su una coda, devi fornire il relativo URL coda.

Il nome di una coda FIFO deve terminare con il suffisso `.fifo`. Il suffisso viene conteggiato ai fini della quota di 80 caratteri dei nomi della coda. Per determinare se una coda è [FIFO](#), puoi verificare se il nome della coda termina con il suffisso.

Di seguito è riportato l'URL della coda per una coda FIFO denominata di `MyQueue` proprietà di un utente con il numero di account AWS. `123456789012`

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

È possibile recuperare l'URL di una coda a livello di codice elencando le code e analizzando la stringa che segue il numero di account. Per ulteriori informazioni, consulta [ListQueues](#).

ID messaggio

Ogni messaggio riceve un ID messaggio assegnato dal sistema che Amazon SQS restituisce nella risposta [SendMessage](#). Questo identificatore è utile per l'identificazione di messaggi. La durata massima di un ID messaggio è di 100 caratteri.

Handle di ricezione

Ogni volta che ricevi un messaggio da una coda, ricevi un handle di ricezione per tale messaggio. Questo handle è associato all'operazione di ricezione del messaggio, non al messaggio stesso.

Per eliminare il messaggio o per modificarne la visibilità, devi fornire l'handle di ricezione (non l'ID messaggio). Pertanto, devi sempre ricevere un messaggio prima di poterlo eliminare (non puoi inserire un messaggio nella coda e poi richiamarlo). La durata massima di un handle di ricezione è di 1024 caratteri.

Important

Se ricevi un messaggio più di una volta, ogni volta che lo ricevi ottieni un diverso handle di ricezione. Devi fornire l'handle di ricezione ricevuto più di recente quando richiedi di eliminare il messaggio (in caso contrario, il messaggio potrebbe non essere eliminato).

Di seguito è riportato un esempio di handle di ricezione (suddiviso su tre linee).

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

Identificatori aggiuntivi per le code FIFO di Amazon SQS

Per ulteriori informazioni sui seguenti identificatori, consulta gli argomenti [Elaborazione una sola volta in Amazon SQS](#) e la [Documentazione di riferimento delle API di Amazon Simple Queue Service](#).

ID deduplicazione messaggi

Un token utilizzato nelle code FIFO di Amazon SQS per identificare in modo univoco i messaggi e prevenire la duplicazione. Se più messaggi con lo stesso ID di deduplicazione vengono inviati entro un intervallo di deduplicazione di 5 minuti, vengono trattati come duplicati e viene consegnata una sola copia. Se non specifichi un ID di deduplicazione e la deduplicazione basata sul contenuto è abilitata, Amazon SQS genera un ID di deduplicazione eseguendo l'hashing del corpo del messaggio. Questo meccanismo garantisce la consegna esattamente una volta eliminando i messaggi duplicati entro il periodo di tempo specificato.

ID gruppo di messaggi

MessageGroupIdÈ un attributo utilizzato solo nelle code FIFO (First-In-First-Out) di Amazon SQS per organizzare i messaggi in gruppi distinti. I messaggi all'interno dello stesso gruppo di messaggi vengono sempre elaborati uno alla volta, in ordine rigoroso, assicurando che non vengano elaborati due messaggi dello stesso gruppo contemporaneamente. Le code standard non utilizzano

MessageGroupId e non forniscono garanzie di ordinazione. Se è richiesto un ordine rigoroso, utilizza invece una coda FIFO.

Numero sequenza

Il numero grande e non consecutivo che Amazon SQS assegna a ciascun messaggio.

Quote di Amazon SQS

Questo argomento spiega le quote e le limitazioni per Amazon SQS FIFO e le code standard, spiegando in dettaglio come influiscono sulla creazione, configurazione e gestione dei messaggi delle code. Scopri vincoli come i limiti di conservazione dei messaggi, i limiti massimi per i messaggi in transito e le soglie di velocità effettiva, nonché le strategie per massimizzare l'efficienza attraverso il batching, l'ottimizzazione delle chiamate API e il polling a lungo termine. Questo argomento tratta anche le convenzioni di denominazione, le regole di etichettatura e i metodi per richiedere aumenti delle quote per soddisfare carichi di lavoro ad alta richiesta, garantendo una gestione efficace delle code e prestazioni ottimali.

Quote di coda FIFO di Amazon SQS

Quote di Amazon SQS

La tabella seguente elenca le quote correlate alle code FIFO.

Quota	Descrizione
Coda di ritardo	Il ritardo predefinito (minimo) per una coda è 0 secondi. Il valore massimo è 15 minuti.
Code elencate	1.000 code per ogni richiesta ListQueues .
Tempo di attesa del long polling	Il tempo massimo di attesa per il long polling è di 20 secondi.
Gruppi di messaggi	Non è previsto alcun limite al numero di gruppi di messaggi all'interno di una coda FIFO.
Messaggi per coda (backlog)	Il numero di messaggi che una coda Amazon SQS può archiviare è illimitato.
Messaggi per coda (in transito)	Le code FIFO supportano un massimo di 120.000 messaggi in volo (messaggi ricevuti da un consumatore ma non ancora eliminati). Se viene raggiunto questo limite, Amazon SQS non restituisce alcun errore, ma

Quota	Descrizione
	l'elaborazione potrebbe risentirne. Puoi richiedere un aumento oltre questo limite contattando AWS il Supporto .
Nome coda	Il nome di una coda FIFO deve terminare con il suffisso <code>.fifo</code> . Il suffisso viene conteggiato ai fini della quota di 80 caratteri dei nomi della coda. Per determinare se una coda è FIFO , puoi verificare se il nome della coda termina con il suffisso.
Tag coda	Non è consigliabile aggiungere più di 50 tag a una coda. I tag supportano i caratteri Unicode in formato UTF-8.
	Il tag Key è obbligatorio, ma il tag Value è facoltativo.
	I tag Key e tag Value fanno distinzione tra maiuscole e minuscole.
	I tag Key e Value possono includere caratteri alfanumerici Unicode in UTF-8 e spazi. I seguenti caratteri speciali sono consentiti: <code>_ . : / = + - @</code>
	Il tag Key o Value non deve includere il prefisso riservato <code>aws:</code> (non puoi eliminare chiavi o valori di tag con questo prefisso).
	La lunghezza massima del tag Key è 128 caratteri Unicode in UTF-8. Il tag Key non deve essere vuoto o nullo.
	La lunghezza massima del tag Value è 256 caratteri Unicode in UTF-8. Il tag Value può essere vuoto o nullo.
	Le azioni di tagging sono limitate a 30 TPS ciascuna. Account AWS Se la tua applicazione richiede una velocità di trasmissione effettiva più elevata, invia una richiesta .

Quote di coda standard di Amazon SQS

La tabella seguente elenca le quote correlate alle code standard.

Quota	Descrizione
Coda di ritardo	Il ritardo predefinito (minimo) per una coda è 0 secondi. Il valore massimo è 15 minuti.
Code elencate	1.000 code per ogni richiesta ListQueues .
Tempo di attesa del long polling	Il tempo massimo di attesa per il long polling è di 20 secondi.
Messaggi per coda (backlog)	Il numero di messaggi che una coda Amazon SQS può archiviare è illimitato.
Messaggi per coda (in transito)	Per la maggior parte delle code standard (a seconda del traffico in coda e del backlog di messaggi), possono esserci un massimo di circa 120.000 messaggi in transito (ricevuti da una coda da un consumatore, ma non ancora eliminati dalla coda). Se si raggiunge questa quota durante l'utilizzo del polling breve , Amazon SQS restituisce il messaggio di errore <code>OverLimit</code> . Se si utilizza un long polling , Amazon SQS non restituisce alcun messaggio di errore. Per evitare di raggiungere la quota, elimina i messaggi dalla coda dopo che sono stati elaborati. Puoi anche aumentare il numero di code utilizzate per elaborare i messaggi. Per richiedere un incremento della quota, invia una richiesta di supporto .
Nome coda	Un nome della coda può avere fino a 80 caratteri. I seguenti caratteri sono accettati: caratteri alfanumerici, trattini (-) e trattini bassi (_).

Quota	Descrizione
	 Note I nomi coda fanno distinzione tra maiuscole e minuscole (ad esempio, <code>Test-queue</code> e <code>test-queue</code> sono code diverse).
Tag coda	Non è consigliabile aggiungere più di 50 tag a una coda. I tag supportano i caratteri Unicode in formato UTF-8. Il tag <code>Key</code> è obbligatorio, ma il tag <code>Value</code> è facoltativo. I tag <code>Key</code> e tag <code>Value</code> fanno distinzione tra maiuscole e minuscole. I tag <code>Key</code> e <code>Value</code> possono includere caratteri alfanumerici Unicode in UTF-8 e spazi. I seguenti caratteri speciali sono consentiti: <code>_ . : / = + - @</code> Il tag <code>Key</code> o <code>Value</code> non deve includere il prefisso riservato <code>aws:</code> (non puoi eliminare chiavi o valori di tag con questo prefisso). La lunghezza massima del tag <code>Key</code> è 128 caratteri Unicode in UTF-8. Il tag <code>Key</code> non deve essere vuoto o nullo. La lunghezza massima del tag <code>Value</code> è 256 caratteri Unicode in UTF-8. Il tag <code>Value</code> può essere vuoto o nullo. Le azioni di tagging sono limitate a 30 TPS ciascuna. Account AWS Se la tua applicazione richiede una velocità di trasmissione effettiva più elevata, invia una richiesta.

Quote di messaggi Amazon SQS

La tabella seguente elenca le quote relative ai messaggi.

Quota	Descrizione
ID messaggio con batch	Un ID messaggio con batch può contenere fino a 80 caratteri. I seguenti caratteri sono accettati: caratteri alfanumerici, trattini (-) e trattini bassi (_).
Attributi di messaggio	Un messaggio può contenere fino a 10 attributi di metadati.
Batch di messaggi	Una singola richiesta di batch di messaggi può includere un massimo di 10 messaggi. Per ulteriori informazioni, consulta Configurazione di Amazon SQSBuffer ed AsyncClient nella sezione Operazioni in batch per Amazon SQS .
Contenuto del messaggio	Un messaggio può includere solo in formato JSON, XML e testo non formattato. I seguenti caratteri Unicode sono consentiti: #x9 #xA #xD da #x20 a #xD7FF da #xE000 a #xFFFD da #x10000 a #x10FFFF I caratteri non inclusi in questo elenco vengono rifiutati. Per maggiori informazioni consulta le specifiche W3C per i caratteri.
ID gruppo di messaggi	MessageGroupId è obbligatorio per le code FIFO. Se non fornisci un messaggio MessageGroupId quando invii un messaggio a una coda FIFO, l'azione ha esito negativo. Nelle code standard, l'utilizzo MessageGroupId abilita le code eque. Si consiglia di includere un messaggio MessageGroupId in tutti i messaggi quando si utilizzano code eque. La lunghezza massima di MessageGroupId è 128 caratteri. Valori validi: caratteri alfanumerici e punteggiatura (!"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~) .

Quota	Descrizione
Conservazione dei messaggi	Per impostazione predefinita, un messaggio viene conservato per 4 giorni. Il minimo è di 60 secondi (1 minuto). Il massimo è 1.209.600 secondi (14 giorni).
Throughput dei messaggi	Code standard Le code standard supportano un numero molto elevato e quasi illimitato di chiamate API al secondo, per azione (SendMessage, ReceiveMessage, o DeleteMessage). Questo throughput elevato le rende ideali per i casi d'uso che richiedono l'elaborazione rapida di grandi volumi di messaggi, come lo streaming di dati in tempo reale o applicazioni su larga scala. Sebbene le code standard si ridimensionino automaticamente in base alla domanda, è essenziale monitorare i modelli di utilizzo per garantire prestazioni ottimali, specialmente nelle regioni con carichi di lavoro più elevati.

Quota	Descrizione
	<u>Code FIFO</u> • Ogni partizione in una coda FIFO è limitata a 300 transazioni al secondo, per azione API (SendMessage, ReceiveMessage, DeleteMessage). Questo limite si applica specificamente alla modalità di trasmissione non elevata. Passando alla modalità ad alto rendimento, è possibile superare questo limite predefinito. Per abilitare la modalità ad alto rendimento, vedere Abilitazione di un throughput elevato per le code FIFO in Amazon SQS. • Se utilizzi il batch, le code FIFO con velocità di trasmissione non elevata supportano fino a 3.000 messaggi al secondo, per azione API (SendMessage, ReceiveMessage, DeleteMessage). I 3.000 messaggi rappresentano 300 chiamate API, ognuna con un batch di 10 messaggi. <u>Throughput elevato per le code FIFO</u> I limiti FIFO di Amazon SQS si basano sul numero di richieste API, non sui limiti dei messaggi. Per la modalità ad alto throughput, questi limiti di richiesta API sono i seguenti: Limiti di throughput delle transazioni (chiamate API non in batch) Questi limiti definiscono la frequenza con cui ciascuna operazione API (ad esempio SendMessageReceiveMessage, o DeleteMessage) può essere eseguita in modo indipendente, garantendo prestazioni di sistema efficienti entro le transazioni al secondo consentite (TPS). I seguenti limiti si basano su chiamate API non in batch:

Quota	Descrizione
	• Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda): fino a 70.000 transazioni al secondo (TPS). • Stati Uniti orientali (Ohio) ed Europa (Francoforte): fino a 19.000 TPS. • Asia Pacifico (Mumbai), Asia Pacifico (Singapore), Asia Pacifico (Sydney) e Asia Pacifico (Tokyo): fino a 9.000 TPS. • Europa (Londra) e Sud America (San Paolo): fino a 4.500 TPS. • Tutti gli altri Regioni AWS: throughput predefinito di 2.400 TPS. Massimizzazione della produttività con il batching Elabora più messaggi in un'unica chiamata API, il che aumenta notevolmente l'efficienza. Invece di gestire ogni messaggio singolarmente, il batch consente di inviare, ricevere o eliminare fino a 10 messaggi in una singola richiesta API. Ciò riduce il numero totale di chiamate API, consentendoti di elaborare più messaggi al secondo rispettando i limiti di transazione (TPS) per la regione, massimizzando la velocità effettiva e le prestazioni del sistema. Per ulteriori informazioni, consulta Aumento del throughput utilizzando la scalabilità orizzontale e l'action batching con Amazon SQS. I seguenti limiti si basano su chiamate API in batch: • Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda): fino a 700.000 messaggi al secondo (10 volte il limite non batch di 70.000 TPS).

Quota	Descrizione
	• Stati Uniti orientali (Ohio) ed Europa (Francoforte): fino a 190.000 messaggi al secondo. • Asia Pacifico (Mumbai), Asia Pacifico (Singapore), Asia Pacifico (Sydney) e Asia Pacifico (Tokyo): fino a 90.000 messaggi al secondo. • Europa (Londra) e Sud America (San Paolo): fino a 45.000 messaggi al secondo. • Tutti gli altri Regioni AWS: fino a 24.000 messaggi al secondo. Ottimizzazione del throughput oltre al batch Sebbene il batch possa aumentare notevolmente la produttività, è importante prendere in considerazione altre strategie per ottimizzare le prestazioni FIFO: • Distribuzione dei messaggi su più gruppi di messaggi IDs: poiché i messaggi all'interno di un singolo gruppo vengono elaborati in sequenza, la distribuzione del carico di lavoro su più gruppi di messaggi consente un migliore parallelismo e una maggiore velocità di trasmissione complessiva. Per ulteriori informazioni, consulta Partizioni e distribuzione dei dati per una velocità di trasmissione effettiva elevata per le code FIFO SQS. • Uso efficiente delle chiamate API: riduci al minimo le chiamate API non necessarie, come le frequenti modifiche di visibilità o le eliminazioni ripetute dei messaggi, per ottimizzare l'uso del TPS disponibile e migliorare l'efficienza. • Utilizza sondaggi lunghi: utilizza il polling lungo WaitTimeSeconds impostando le richieste di ricezione per ridurre le risposte vuote quando non

Quota	Descrizione
	sono disponibili messaggi, ridurre le chiamate API non necessarie e sfruttare meglio la quota TPS. • Richiesta di aumenti della velocità effettiva: se l'applicazione richiede un throughput superiore ai limiti predefiniti, richiedi un aumento utilizzando la console Service Quotas. Ciò può essere necessario per carichi di lavoro molto richiesti o in regioni con limiti predefiniti inferiori. Per abilitare la modalità ad alto rendimento, vedere. Abilitazione di un throughput elevato per le code FIFO in Amazon SQS
Timer messaggio	Il ritardo predefinito (minimo) per un messaggio è 0 secondi. Il valore massimo è 15 minuti.
Dimensione dei messaggi	La dimensione minima del messaggio è pari a 1 byte (1 carattere). Il massimo è 1.048.576 byte (1 MiB). Per inviare messaggi di dimensioni superiori a 1 MiB, puoi utilizzare Amazon SQS Extended Client Library per Java e Amazon SQS Extended Client Library for Python. Questa libreria consente di inviare un messaggio Amazon SQS che contiene un riferimento a un payload del messaggio in Amazon S3. La dimensione massima di payload è pari a 2 GB.  Note Questa libreria estesa funziona solo per client sincroni.
Timeout visibilità del messaggio	Il timeout visibilità predefinito per una coda è di 30 secondi. Il valore minimo è 0 secondi. La durata massima è 12 ore.

Quota	Descrizione
Informazioni sulle policy	La quota massima è di 8.192 byte, 20 istruzioni, 50 principal o 10 condizioni. Per ulteriori informazioni, consulta Quote delle policy di Amazon SQS .

Quote delle policy di Amazon SQS

La tabella seguente elenca le quote relative alle policy.

Nome	Massimo
Byte	8,192
Condizioni	10
Principali	50
Istruzioni	20
Operazioni per dichiarazione	7

Caratteristiche e funzionalità di Amazon SQS

Questo argomento fornisce funzionalità di uso comune in Amazon SQS per gestire le code di messaggi, ottimizzare le prestazioni, garantire la consegna affidabile dei messaggi e gestire l'elaborazione dei messaggi in modo efficiente.

Utilizzo di code di lettere non scritte in Amazon SQS

Amazon SQS supporta dead-letter queues (DLQs), a cui le code di origine possono indirizzare i messaggi che non vengono elaborati correttamente. DLQs sono utili per il debug dell'applicazione perché consentono di isolare i messaggi non utilizzati per determinare il motivo per cui l'elaborazione non è riuscita. Per prestazioni ottimali, è consigliabile mantenere la coda di origine e il DLQ all'interno della stessa regione. Account AWS Una volta che i messaggi si trovano in una coda di lettere non scritte, puoi:

- Esaminare i log per trovare eccezioni che potrebbero aver causato il trasferimento dei messaggi a una coda DLQ.
- Analizza il contenuto dei messaggi spostati nella coda delle lettere non scritte per diagnosticare i problemi delle applicazioni.
- Determinare se hai concesso al tuo consumatore tempo sufficiente per elaborare i messaggi.
- [Sposta i messaggi fuori dalla coda delle lettere morte utilizzando il redrive della coda delle lettere morte.](#)

È innanzitutto necessario creare una nuova coda prima di configurarla come coda di lettere morte. Per informazioni su come configurare code DLQ utilizzando la console Amazon SQS, consulta [Configura una coda di lettere non scritte utilizzando la console Amazon SQS](#). Per informazioni sulle code di lettere non scritte, ad esempio su come configurare un avviso per qualsiasi messaggio spostato in una coda di lettere non scritte, consulta [Creazione di allarmi per code di lettere scadenti con Amazon CloudWatch](#)

Note

Non utilizzare una coda DLQ con una coda FIFO se non desideri interrompere l'ordine esatto di messaggi oppure operazioni. Ad esempio, non utilizzare una coda DLQ con istruzioni in

un EDL (Edit Decision List) per una suite di editing video, dove la modifica dell'ordine delle modifiche modifica il contesto delle modifiche successive.

Utilizzo delle politiche per le code di lettere morte

Utilizza una politica di `redrive` per specificare il `maxReceiveCount`. `maxReceiveCount` è il numero di volte in cui un consumatore può ricevere un messaggio da una coda di origine prima che venga spostato in una coda di lettere non scritte. Ad esempio, se `maxReceiveCount` è impostato su un valore basso come 1, la mancata ricezione di un messaggio causerebbe lo spostamento del messaggio nella coda delle lettere morte. Per garantire la resilienza del sistema contro gli errori, imposta un valore `maxReceiveCount` sufficientemente alto da consentire un numero sufficiente di tentativi.

Per le code standard con una politica di `redrive` dove `maxReceiveCount` è maggiore di 3, se un messaggio viene ricevuto 3 o più volte senza essere eliminato, SQS lo sposta in fondo alla coda. La `ApproximateAgeOfOldestMessage` metrica riflette quindi l'età del messaggio successivo che non ha superato questa soglia.

La policy `redrive allow` specifica quali code di origine possono accedere a una coda DLQ. Puoi scegliere se consentire tutte le code di origine, consentire code di origine specifiche o negare a tutte le code di origine l'uso della coda di lettere morte. L'impostazione predefinita consente a tutte le code di origine di utilizzare la coda di lettere morte. Se scegli di consentire code specifiche utilizzando l'opzione `byQueue`, puoi specificare fino a 10 code di origine utilizzando la coda di origine Amazon Resource Name (ARN). Se si specifica `denyAll`, la coda non può essere utilizzata come coda DLQ.

Comprensione dei periodi di conservazione dei messaggi per le code di lettere scadute

Per le code standard, la scadenza di un messaggio si basa sempre sul timestamp della coda originale. Quando un messaggio viene spostato in una coda DLQ, il timestamp della coda rimane invariato. La `ApproximateAgeOfOldestMessage` metrica indica quando il messaggio è stato spostato nella coda delle lettere non scritte, non quando è stato originariamente inviato. Ad esempio, supponiamo che un messaggio trascorra 1 giorno nella coda originale prima di essere spostato in una coda DLQ. Se il periodo di conservazione della coda DLQ è di 4 giorni, il messaggio viene eliminato dalla coda DLQ dopo 3 giorni e `ApproximateAgeOfOldestMessage` è di tre giorni. Pertanto, è consigliabile impostare sempre il periodo di conservazione di una coda DLQ in modo che sia più lungo del periodo di conservazione della coda originale.

Per le code FIFO, il timestamp della coda si reimposta quando il messaggio viene spostato in una coda DLQ. La metrica `ApproximateAgeOfOldestMessage` indica quando il messaggio è stato spostato nella coda DLQ. Nello stesso esempio precedente, il messaggio viene eliminato dalla coda delle lettere morte dopo quattro giorni e sono trascorsi quattro giorni.

`ApproximateAgeOfOldestMessage`

Configura una coda di lettere non scritte utilizzando la console Amazon SQS

Una coda di lettere morte (DLQ) è una coda che riceve messaggi che non sono stati elaborati correttamente da un'altra coda, nota come coda di origine. Amazon SQS non crea automaticamente la coda DLQ. Occorre innanzitutto creare una coda prima di designarla come coda DLQ. [Quando si configura un DLQ, il tipo di coda deve corrispondere al tipo di coda di origine: una coda FIFO può utilizzare solo un DLQ FIFO e una coda standard può utilizzare solo un DLQ standard.](#) È possibile configurare una coda di lettere morte quando si crea o si modifica una coda. Per ulteriori dettagli, consultare [Utilizzo di code di lettere non scritte in Amazon SQS](#).

Configurazione di una coda DLQ per una coda esistente (console).

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Seleziona la coda di origine (la coda che invierà i messaggi non riusciti alla coda delle lettere non valide), quindi scegli Modifica.
4. Scorri fino alla sezione Coda di lettere non scritte e attiva l'opzione Attiva.
5. Nelle impostazioni della coda Dead-letter, scegli l'Amazon Resource Name (ARN) di una coda esistente che desideri utilizzare come coda di lettere non scritte.
6. Imposta il valore Maximum received, che definisce quante volte un messaggio può essere ricevuto prima di essere inviato alla coda di lettere non scritte (intervallo valido: da 1 a 1.000).
7. Scegli Save (Salva).

Scopri come configurare un redrive della coda di lettere non scritte in Amazon SQS

Utilizza il redrive della coda di lettere non utilizzate per spostare i messaggi non utilizzati da una coda di lettere non scritte a un'altra destinazione per l'elaborazione. Per impostazione predefinita, il redrive della coda DLQ sposta i messaggi da una coda DLQ a una coda di origine. Tuttavia, puoi

anche configurare qualsiasi altra coda come destinazione di redrive se entrambe le code sono di tipo uguale. Ad esempio, se la coda DLQ è una coda FIFO, anche la coda di destinazione di redrive deve essere una coda FIFO. Inoltre, puoi configurare la velocità di redrive per impostare la velocità con cui Amazon SQS sposta i messaggi.

Note

Quando un messaggio viene spostato da una coda FIFO a una DLQ FIFO, l'ID di deduplicazione del messaggio originale verrà sostituito con l'ID del messaggio originale. Ciò serve a garantire che la deduplicazione DLQ non impedisca l'archiviazione di due messaggi indipendenti che condividono un ID di deduplicazione.

Le code di lettere morte reindirizzano i messaggi nell'ordine in cui vengono ricevuti, a partire dal messaggio più vecchio. Tuttavia, la coda di destinazione inserisce i messaggi reindirizzati, così come i nuovi messaggi degli altri produttori, in base all'ordine in cui li riceve. Ad esempio, se un produttore invia messaggi a una coda FIFO di origine e riceve contemporaneamente messaggi reindirizzati da una coda a lettere morte, i messaggi reindirizzati si intrecceranno con i nuovi messaggi del produttore.

Note

L'attività di redrive reimposta il periodo di conservazione. Tutti i messaggi reindirizzati sono considerati nuovi messaggi con un nuovo messaggioID e vengono assegnati ai messaggi reindirizzati. messageID enqueueTime

Configurazione di un redrive di una coda di lettere non scritte per una coda standard esistente utilizzando l'API Amazon SQS

Puoi configurare un redrive di una coda di lettere morte utilizzando le azioni, e API:
`StartMessageMoveTask` `ListMessageMoveTasks` `CancelMessageMoveTask`

Azione API	Description
StartMessageMoveTask	Avvia un'attività asincrona per spostare i messaggi da una coda di origine specificata a una coda di destinazione specificata.

Azione API	Description
<u>ListMessageMoveTasks</u>	Ottiene le attività di spostamento dei messaggi più recenti (fino a 10) in una coda di origine specifica.
<u>CancelMessageMoveTask</u>	Annulla un'operazione di spostamento dei messaggi specificata. Lo spostamento di un messaggio può essere annullato solo quando lo stato corrente è IN ESECUZIONE.

Configurazione di un redrive di una coda di lettere non scritte per una coda standard esistente utilizzando la console Amazon SQS

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Scegli il nome della coda che hai configurato come [coda DLQ](#).
4. Scegli Avvia redrive DLQ.
5. In Configurazione Redrive, per Destinazione del messaggio, esegui una delle seguenti operazioni:
 - Per reindirizzare i messaggi nella relativa coda di origine, scegli Reindirizza verso le code di origine.
 - Per reindirizzare i messaggi su un'altra coda, scegli Reindirizza verso una destinazione personalizzata. Quindi, immettere il nome della risorsa Amazon (ARN) di una coda di destinazione esistente.
6. In Impostazioni per la regolazione della velocità, scegli una delle seguenti opzioni:
 - Ottimizzato per il sistema: reindirizza i messaggi della coda DLQ al numero massimo di messaggi al secondo.
 - Velocità massima personalizzata: reindirizza i messaggi in coda DLQ con una frequenza massima personalizzata di messaggi al secondo. La frequenza massima consentita è di 500 messaggi al secondo.

- Si consiglia di iniziare con un valore basso per la velocità massima personalizzata e verificare che la coda di origine non sia sovraccarica di messaggi. Da lì, aumentare gradualmente il valore di velocità massima personalizzata, continuando a monitorare lo stato della coda di origine.
7. Al termine della configurazione del redrive della coda DLQ, scegli Messaggi di redrive.

Important

Amazon SQS non supporta il filtraggio e la modifica dei messaggi mentre li reindirizza dalla coda DLQ.

Un'attività di redrive di una coda DLQ può essere eseguita per un massimo di 36 ore. Amazon SQS supporta un massimo di 100 attività di redrive attive per account.

8. Se desideri annullare l'operazione di redrive dei messaggi, nella pagina Dettagli della coda, scegli Annulla redrive DLQ. Quando si annulla un redrive di messaggio in corso, tutti i messaggi che sono già stati spostati correttamente nella coda di destinazione dello spostamento rimarranno nella coda di destinazione.

Configurazione delle autorizzazioni per il redrive della coda DLQ

Puoi concedere agli utenti l'accesso a specifiche operazioni relative alla coda DLQ aggiungendo autorizzazioni alla tua policy. Le autorizzazioni minime richieste per reindirizzare una coda DLQ sono le seguenti:

Autorizzazioni minime	Metodi API richiesti
Per avviare il reindirizzamento di un messaggio	• Aggiungi <code>sqs:StartMessageMoveTask</code>, <code>sqs:ReceiveMessage</code>, <code>sqs:DeleteMessage</code> e <code>sqs:GetQueueAttributes</code> della coda DLQ. Se la coda DLQ o la coda dell'origine sono crittografate (nota anche come coda SSE), è necessaria anche <code>kms:Decrypt</code> per la chiave KMS utilizzata per crittografare i messaggi. • Aggiungere <code>sqs:SendMessage</code> della coda di destinazione. Se la coda di destinazione è crittografata, sono inoltre obbligatori <code>kms:GenerateDataKey</code> e <code>kms:Decrypt</code>.

Autorizzazioni minime	Metodi API richiesti
Per annullare il redrive di un messaggio in corso	Aggiungere <code>sqs:CancelMessageMoveTask</code> , <code>sqs:ReceiveMessage</code> , <code>sqs>DeleteMessage</code> e <code>sqs:GetQueueAttributes</code> della coda DLQ. Se la coda DLQ è crittografata (nota anche come coda SSE) è necessario anche <code>kms:Decrypt</code> .
Per mostrare lo stato di spostamento di un messaggio	Aggiungere <code>sqs:ListMessageMoveTasks</code> e <code>sqs:GetQueueAttributes</code> della coda DLQ.

Per configurare le autorizzazioni per una coppia di coda crittografata (una coda di origine con una coda DLQ)

Utilizza i seguenti passaggi per configurare le autorizzazioni minime per un redrive DLQ (Dead-letter Queue):

1. Aprire la console IAM all'indirizzo <https://console.aws.amazon.com/iam/>.
2. Nel riquadro di navigazione selezionare Policy.
3. Crea una nuova [politica](#) e aggiungi le seguenti autorizzazioni. Allega la policy all'[utente](#) o al [ruolo](#) IAM che eseguirà l'operazione di redrive.
 - Autorizzazioni per il DLQ (coda di origine):
 - `sqs:StartMessageMoveTask`
 - `sqs:CancelMessageMoveTask`
 - `sqs:ListMessageMoveTasks`
 - `sqs:ReceiveMessage`
 - `sqs>DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `sqs:ListDeadLetterSourceQueues`
 - Specificare l'ARN della risorsa del DLQ (coda di origine) (ad esempio, «arn:aws:sqs::: *<DLQ_region> <DLQ_accountId> <DLQ_name>* »).

- Autorizzazioni per la coda di destinazione:
 - `sqs:SendMessage`
 - Specificare la coda Resource ARN di destinazione (ad esempio, «arn:aws:sqs: »).
`<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>`
- Autorizzazioni per le chiavi KMS:
 - `kms:Decrypt`(Necessario per decrittografare i messaggi nel DLQ).
 - `kms:GenerateDataKey`(Necessario per crittografare i messaggi nella coda di destinazione.)
 - Resource ARNs:
 - L'ARN della chiave KMS utilizzata per crittografare i messaggi nella DLQ (coda di origine) (ad esempio, «arn:aws:kms: ::key/ »). `<region> <accountId> <SourceQueueKeyId>`
 - L'ARN della chiave KMS utilizzata per crittografare i messaggi nella coda di destinazione (ad esempio, «arn:aws:kms: :key/ »). `<region> <accountId> <DestinationQueueKeyId>`

La policy di accesso deve essere simile alla seguente:

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>"
    ]
  }
]
}

```

Per configurare le autorizzazioni per una coppia di coda non crittografata (una coda di origine con una coda DLQ)

Segui questi passaggi per configurare le autorizzazioni minime richieste per la gestione di una coda di lettere mortali (DLQ) standard non crittografata. Le autorizzazioni minime richieste sono per ricevere, eliminare e recuperare gli attributi dalla coda DLQ e inviare attributi alla coda di origine.

1. Aprire la console IAM all'indirizzo <https://console.aws.amazon.com/iam/>.
2. Nel riquadro di navigazione selezionare Policy.
3. Crea una nuova [politica](#) e aggiungi le seguenti autorizzazioni. Allega la policy all'[utente](#) o al [ruolo](#) IAM che eseguirà l'operazione di redrive.

- Autorizzazioni per il DLQ (coda di origine):
 - sqs:StartMessageMoveTask
 - sqs:CancelMessageMoveTask
 - sqs:ListMessageMoveTasks
 - sqs:ReceiveMessage
 - sqs>DeleteMessage
 - sqs:ListDeadLetterSourceQueues
 - Specificare l'ARN della risorsa del DLQ (coda di origine) (ad esempio, «arn:aws:sqs::: »).
`<DLQ_region> <DLQ_accountId> <DLQ_name>`
- Autorizzazioni per la coda di destinazione:
 - sqs:SendMessage
 - Specificare la coda Resource ARN di destinazione (ad esempio, «arn:aws:sqs: »).
`<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>`

La policy di accesso deve essere simile alla seguente:

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"
            }
        }
    }
]
}

```

Utilizzo del redrive della coda di lettere morte con controllo degli accessi agli endpoint VPC

Quando si limita l'accesso alla coda a un VPCs utilizzo specifico della `aws:sourceVpc` condizione, è necessario fare un'eccezione per i AWS servizi per abilitare la funzionalità di reindirizzamento della coda di lettere morte (DLQ). Questo perché il servizio Amazon SQS opera all'esterno del tuo VPC durante lo spostamento dei messaggi.

Per consentire le operazioni di redrive DLQ, aggiungi la `aws:CalledViaLast` condizione alla tua politica di coda. Ciò consente ad Amazon SQS di effettuare chiamate API per tuo conto, mantenendo al contempo le restrizioni VPC per l'accesso diretto.

Per consentire sia l'accesso limitato al VPC che il redrive DLQ:

1. Utilizza la `aws:CalledViaLast` condizione nella tua politica di coda.
2. Applica la policy sia alla coda di origine che al DLQ
3. Mantieni le restrizioni VPC per l'accesso diretto da altre fonti

Ecco un esempio di policy che implementa questi requisiti:

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"
        }
      }
    }
  ]
}
```

- Sostituisci i valori segnaposto con i tuoi valori effettivi

- Questa politica utilizza un'istruzione «deny» con condizioni, che è più sicura rispetto all'utilizzo di istruzioni «allow»
- L'`StringNotEqualsIfExists` operatore gestisce i casi in cui la chiave di condizione potrebbe non essere presente nel contesto della richiesta.

In alternativa, puoi utilizzare la chiave `aws:ViaAWSService` condition per consentire l'accesso basato sui servizi mantenendo le restrizioni del VPC. Questa chiave di condizione indica se la richiesta proviene da un servizio. AWS Ecco un esempio di politica che utilizza `aws:ViaAWSService` invece di `aws:CalledViaLast`:

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"  
    }  
  }  
}  
]  
}
```

L' `BoolIfExists` operatore con `aws:ViaAWSService` condizione garantisce che le richieste siano consentite quando provengono dai servizi, mantenendo al contempo le restrizioni VPC per l'accesso diretto. Questo può essere più semplice da comprendere e gestire, in quanto verifica direttamente se la richiesta viene effettuata da un AWS servizio anziché verificare quale servizio ha effettuato l'ultima chiamata.

Per ulteriori informazioni sulle chiavi di condizione utilizzate in IAM e nelle policy relative alle risorse, consulta IAM JSON policy elements: Condition.

CloudTrail requisiti di aggiornamento e autorizzazione per il redrive della coda di lettere morte di Amazon SQS

L'8 giugno 2023, Amazon SQS ha introdotto il redrive DLQ (dead-letter queue) per AWS SDK e (CLI). AWS Command Line Interface Questa funzionalità è un'aggiunta alla redrive DLQ già supportata per la console. AWS Se in precedenza hai utilizzato la AWS console per reindirizzare i messaggi in coda contenenti lettere morte, potrebbero interessarti le seguenti modifiche:

CloudTrail ridenominazione degli eventi

Il 15 ottobre 2023, i nomi degli CloudTrail eventi per il redrive della coda di lettere non scritte cambieranno sulla console Amazon SQS. Se hai impostato allarmi per questi CloudTrail eventi, devi aggiornarli ora. Di seguito sono riportati i nuovi nomi CloudTrail degli eventi per DLQ redrive:

Nome evento precedente	Nuovo nome evento
CreateMoveTask	StartMessageMoveTask
CancelMoveTask	CancelMessageMoveTask

Autorizzazioni aggiornate

Incluso nella versione SDK e CLI, Amazon SQS ha anche aggiornato le autorizzazioni di coda per il redrive DLQ per aderire alle best practice di sicurezza. Utilizza i seguenti tipi di autorizzazione alla coda per reindirizzare i messaggi dal tuo DLQs

1. Autorizzazioni basate sulle azioni (aggiornamento per le azioni dell'API DLQ)
2. Autorizzazioni della policy di Amazon SQS gestita
3. Policy di autorizzazione che utilizza caratteri jolly sqs:*

Important

Per utilizzare il redrive DLQ per SDK o CLI, è necessario disporre di una policy di autorizzazione di redrive DLQ che corrisponda a una delle opzioni precedenti.

Se le autorizzazioni di coda per il redrive DLQ non corrispondono a una delle opzioni precedenti, devi aggiornare le autorizzazioni entro il 31 agosto 2023. Entro il 31 agosto 2023, il tuo account sarà in grado di reindirizzare i messaggi utilizzando le autorizzazioni configurate tramite la console AWS solo nelle regioni in cui hai precedentemente utilizzato il redrive DLQ. Ad esempio, supponiamo di avere un "Account A" sia in us-east-1 che in eu-west-1. L' «Account A» è stato utilizzato per reindirizzare i messaggi sulla AWS console in us-east-1 prima dell'8 giugno 2023, ma non in eu-west-1. Tra l'8 giugno 2023 e il 31 agosto 2023, se le autorizzazioni dei criteri dell' «Account A» non corrispondono a una delle opzioni precedenti, possono essere utilizzate solo per reindirizzare i messaggi sulla AWS console in us-east-1 e non in eu-west-1.

Important

Se le tue autorizzazioni di redrive DLQ non corrispondono a una di queste opzioni dopo il 31 agosto 2023, il tuo account non sarà più in grado di reindirizzare i messaggi DLQ utilizzando la console AWS .

Tuttavia, se hai utilizzato la funzione di redrive DLQ sulla AWS Console nel mese di agosto 2023, hai un'estensione fino al 15 ottobre 2023 per adottare le nuove autorizzazioni in base a una di queste opzioni.

Per ulteriori informazioni, consulta [the section called "Identificazione delle policy interessate"](#).

Di seguito sono riportati alcuni esempi di autorizzazioni di coda per ogni opzione di redrive DLQ. Quando si utilizzano [code crittografate lato server \(SSE\)](#), è richiesta l'autorizzazione della chiave corrispondente. AWS KMS

Basato sull'azione

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>"
    }
  ]
}
```

Policy gestita

Le seguenti policy gestite contengono le autorizzazioni aggiornate richieste:

- Amazon SQSFull Access: include le seguenti attività di reindirizzamento delle code senza lettera morta: avvio, annullamento ed elenco.
- Amazon SQSRead OnlyAccess: fornisce l'accesso in sola lettura e include l'attività di redrive della coda di lettere morte degli elenchi.

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

Policy di autorizzazione che utilizza caratteri jolly sqs*

JSON

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

Identificazione delle policy interessate

Se utilizzi le politiche gestite dai clienti (CMPs), puoi utilizzare AWS CloudTrail e IAM per identificare le politiche interessate dall'aggiornamento delle autorizzazioni di coda.

 Note

Se si utilizza `AmazonSQSFullAccess` e `AmazonSQSReadOnlyAccess`, non sono necessarie ulteriori azioni.

1. Accedi alla console. AWS CloudTrail
2. Nella pagina Cronologia degli eventi, in Cerca attributi, utilizza il menu a discesa per selezionare Nome evento. Quindi, cerca `CreateMoveTask`.
3. Scegli un evento per aprire la pagina dei Dettagli. Nella sezione Record degli eventi, recuperare `UserName` o `RoleName` dall'ARN `userIdentity`.
4. Accedi alla console IAM.
 - Per gli utenti, scegli Utenti. Selezionare l'utente con i dati `UserName` identificati nella fase precedente.
 - Per i ruoli, scegli Ruoli. Selezionare l'utente con i dati `RoleName` identificati nella fase precedente.
5. Nella pagina Dettagli, nella sezione Autorizzazioni, esamina tutte le politiche con il prefisso `sqs :` prefisso in `Action` o esamina le politiche in cui è definita la coda Amazon SQS in `Resource`.

Creazione di allarmi per code di lettere scadenti con Amazon CloudWatch

Imposta un CloudWatch allarme per monitorare i messaggi in una coda di lettere non scritte utilizzando la metrica. [ApproximateNumberOfMessagesVisible](#) Per istruzioni dettagliate, vedi [Creazione di CloudWatch allarmi per i parametri di Amazon SQS](#). Quando si attiva l'allarme, che indica che i messaggi sono stati spostati nella coda delle lettere non scritte, puoi sondare la coda per esaminarli e recuperarli.

Metadati dei messaggi per Amazon SQS

Utilizza gli attributi dei messaggi per aggiungere metadati personalizzati ai messaggi Amazon SQS per le tue applicazioni. Utilizza gli attributi del sistema di messaggistica per archiviare i metadati per l'integrazione con altri Servizi AWS, ad esempio. AWS X-Ray

Attributi messaggio di Amazon SQS

Amazon SQS consente di includere metadati strutturati (come timestamp, dati geospaziali, firme e identificatori) nei messaggi utilizzando gli attributi dei messaggi. Ogni messaggio può avere fino a 10 attributi. Gli attributi di messaggio sono facoltativi e separati dal corpo del messaggio (sebbene vengano inviati insieme a esso). Il tuo consumatore può utilizzare gli attributi di messaggio per gestire un messaggio in un determinato modo senza la necessità di elaborare prima il corpo del messaggio. Per ulteriori informazioni sull'invio di messaggi con attributi tramite la console Amazon SQS, consulta [Invio di un messaggio con attributi tramite Amazon SQS](#).

Note

Non confondere gli attributi dei messaggi con gli attributi del sistema di messaggistica: mentre puoi utilizzare gli attributi dei messaggi per allegare metadati personalizzati ai messaggi Amazon SQS per le tue applicazioni, puoi [utilizzare gli attributi del sistema di messaggistica](#) per archiviare metadati per AWS altri servizi, come AWS X-Ray

Argomenti

- [Componenti attributo del messaggio](#)
- [Tipi di dati degli attributi di messaggio](#)
- [Calcolo del digest dei MD5 messaggi per gli attributi dei messaggi](#)

Componenti attributo del messaggio

Important

Tutti i componenti di un attributo di messaggio sono inclusi nella restrizione della dimensione dei messaggi di 1 MiB.

Name, Type, Value e il corpo del messaggio non devono essere vuoti o nulli.

Ogni attributo di messaggio è costituito dai seguenti elementi:

- Nome: il nome dell'attributo del messaggio può contenere i seguenti caratteri: A-Z, a-z, 0-9, sottolineatura (_), trattino (-) e punto (.). Le restrizioni si applicano come segue:
 - Può contenere fino a 256 caratteri

- Non può iniziare con `AWS.` o `Amazon.` (o qualsiasi variazione in maiuscole/minuscole)
- Fa distinzione tra lettere maiuscole e minuscole
- Deve essere univoco tra tutti i nomi di attributo per il messaggio
- Non deve iniziare o finire con un punto
- Non deve avere punti in una sequenza
- Tipo: il tipo di dati dell'attributo del messaggio. I tipi supportati includono `String`, `Number` e `Binary`. È anche possibile aggiungere informazioni personalizzate per qualsiasi tipo di dati. Il tipo di dati ha le stesse limitazioni del corpo del messaggio (per ulteriori informazioni, consulta [SendMessage](#) nella Guida di riferimento di Amazon Simple Queue Service API). Inoltre, si applicano le limitazioni seguenti:
 - Può contenere fino a 256 caratteri
 - Fa distinzione tra lettere maiuscole e minuscole
- Valore: il valore dell'attributo di messaggio. Per i tipi di dati `String`, i valori dell'attributo hanno le stesse restrizioni del corpo del messaggio.

Tipi di dati degli attributi di messaggio

I tipi di dati degli attributi di messaggio indicano a Amazon SQS il modo in cui gestire i valori degli attributi dei messaggi corrispondenti. Ad esempio, se il tipo è `Number`, Amazon SQS convalida i valori numerici.

Amazon SQS supporta i tipi di dati logici `String`, `Number` e `Binary` con etichette opzionali di tipi di dati personalizzati nel formato *`.custom-data-type`*

- Stringa: gli attributi `String` possono memorizzare testo Unicode utilizzando qualsiasi carattere XML valido.
- Numero: gli attributi di `Number` possono memorizzare valori numerici positivi o negativi. Un numero può avere fino a 38 cifre di precisione e può essere compreso tra 10^{-128} e 10^{+126} .

Note

Amazon SQS rimuove zero iniziali e finali.

- Binary: gli attributi di tipo binario possono archiviare qualsiasi tipo di dati binari, ad esempio dati compressi, dati crittografati o immagini.

- Personalizzato: per creare un tipo di dato personalizzato, aggiungi un'etichetta di tipo personalizzato a qualsiasi tipo di dato. Esempio:
 - `Number.byte`, `Number.short`, `Number.int` e `Number.float` possono aiutare a distinguere tra i tipi di numero.
 - `Binary.gif` e `Binary.png` possono aiutare a distinguere tra tipi di file.

Note

Amazon SQS non interpreta, convalida oppure utilizza i dati aggiunti.
L'etichetta di tipo personalizzato ha le stesse restrizioni del corpo del messaggio.

Calcolo del digest dei MD5 messaggi per gli attributi dei messaggi

Se utilizzi il AWS SDK per Java, puoi saltare questa sezione. La `MessageMD5ChecksumHandler` classe dell'SDK for Java MD5 supporta i digest dei messaggi per gli attributi dei messaggi di Amazon SQS.

Se utilizzi l'API Query o una di quelle AWS SDKs che non supportano i MD5 message digest per gli attributi dei messaggi di Amazon SQS, devi utilizzare le seguenti linee guida per eseguire MD5 il calcolo del message digest.

Note

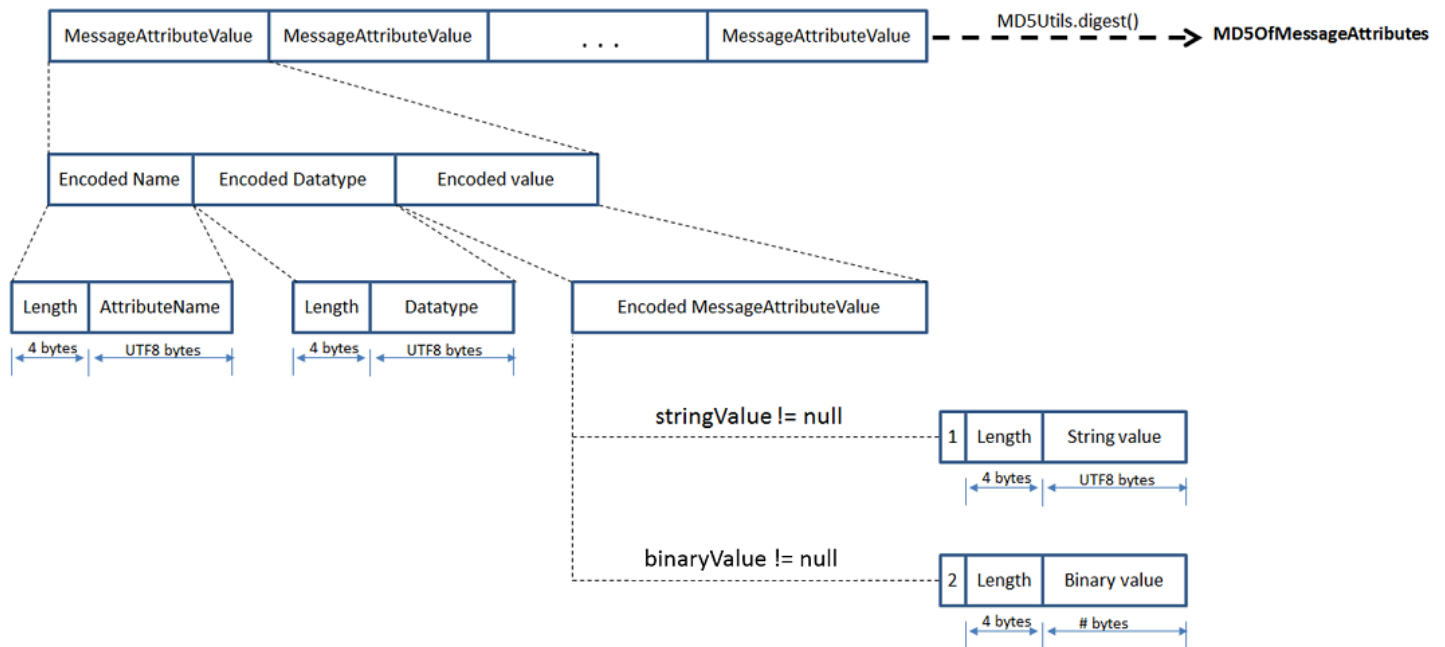
Includi sempre i suffissi dei tipi di dati personalizzati nel calcolo del message-digest. MD5

Panoramica di

Di seguito è riportata una panoramica dell'algoritmo di calcolo del MD5 message digest:

1. Ordinare tutti gli attributi dei messaggi in base al nome in ordine crescente.
2. Codificare le singole parti di ogni attributo (`Name`, `Type` e `Value`) in un buffer.
3. Calcolare il digest del messaggio dell'intero buffer.

Il diagramma seguente mostra la codifica del MD5 message digest per un singolo attributo del messaggio:



Per codificare un singolo attributo del messaggio Amazon SQS

1. Codificare il nome: lunghezza del nome (4 byte) e byte UTF-8 del nome.
2. Codificare il tipo di dati: lunghezza del nome (4 byte) e byte UTF-8 del tipo di dati.
3. Codificare il tipo di trasporto (String o Binary) del valore [1 byte].

Note

I tipi di dati logici `String` e `Number` usano il tipo di trasporto `String`.
Il tipo di dati logici `Binary` utilizza il tipo di trasporto `Binary`.

- a. Per il tipo di trasporto `String`, codificare 1.
- b. Per il tipo di trasporto `Binary`, codificare 2.
4. Codificare il valore attributo.
 - a. Per il tipo di trasporto `String`, codifica il valore di attributo: la lunghezza (4 byte) e i byte UTF-8 del valore.
 - b. Per il tipo di trasporto `Binary`, codifica il valore di attributo: la lunghezza (4 byte) e i byte non elaborati del valore.

Attributi del sistema di messaggistica Amazon SQS

Mentre puoi utilizzare gli [attributi di messaggio](#) per collegare metadati personalizzati ai messaggi Amazon SQS per le applicazioni, puoi utilizzare gli attributi del sistema di messaggi per archiviare i metadati per altri servizi AWS, ad esempio AWS X-Ray. Per ulteriori informazioni, consulta il parametro di richiesta `MessageSystemAttribute` delle operazioni API [SendMessage](#) e [SendMessageBatch](#), l'attributo `AWSTraceHeader` dell'operazione API [ReceiveMessage](#) e il tipo di dati [MessageSystemAttributeValue](#) nella Guida di riferimento di Amazon Simple Queue Service API.

Gli attributi del sistema di messaggi sono strutturati esattamente come gli attributi di messaggio, con le seguenti eccezioni:

- Al momento, l'unico attributo del sistema di messaggi supportato è `AWSTraceHeader`. Il suo tipo deve essere `String` e il suo valore deve essere una stringa di intestazione di AWS X-Ray traccia formattata correttamente.
- Le dimensioni di un attributo di sistema di messaggi non vengono conteggiate ai fini della dimensione totale di un messaggio.

Risorse necessarie per l'elaborazione di messaggi Amazon SQS

Amazon SQS fornisce stime del numero approssimativo di messaggi ritardati, visibili e non visibili in una coda per aiutarti a valutare le risorse necessarie per l'elaborazione. Per ulteriori informazioni sulla visibilità, consulta [Timeout visibilità Amazon SQS](#).

Note

Per alcuni parametri, il risultato è approssimativo a causa dell'architettura distribuita di Amazon SQS. Nella maggior parte dei casi, il conteggio deve essere vicino al numero effettivo di messaggi in coda.

La tabella seguente elenca il nome attributo da usare con l'operazione [GetQueueAttributes](#):

Operazione	Nome attributo
Ottenere il numero approssimativo di messaggi disponibili per il recupero dalla coda.	<code>ApproximateNumberOfMessagesVisible</code>
Ottenere il numero approssimativo di messaggi nella coda che vengono differiti e non sono disponibili per la lettura immediata. Ciò può accadere quando la coda è configurata come coda di ritardo o quando un messaggio è stato inviato con un parametro di ritardo.	<code>ApproximateNumberOfMessagesDelayed</code>
Ottenere il numero approssimativo di messaggi che sono in transito. I messaggi sono considerati in transito se sono stati inviati a un client ma non sono ancora stati eliminati o non hanno ancora raggiunto il termine della loro finestra di visibilità.	<code>ApproximateNumberOfMessagesNotVisible</code>

Impaginazione della coda di elenchi di Amazon SQS

I metodi [listQueues](#) e [listDeadLetterQueues](#) API supportano controlli di impaginazione opzionali. Per impostazione predefinita, questi metodi API restituiscono fino a 1000 code nel messaggio di risposta. È possibile impostare il parametro `MaxResults` in modo che restituisca un numero inferiore di risultati in ogni risposta.

Imposta il parametro `MaxResults` nella richiesta `listQueues` o `listDeadLetterQueues` per specificare il numero massimo di risultati da restituire nella risposta. Se non imposti `MaxResults`, la risposta include un massimo di 1.000 risultati e il valore `NextToken` nella risposta è nullo.

Se sono disponibili risultati aggiuntivi da visualizzare e imposti `MaxResults`, la risposta include un valore per `NextToken`. Utilizza `NextToken` come parametro nella richiesta successiva a `listQueues` per ricevere la pagina successiva dei risultati. Se il valore di `NextToken` nella risposta è nullo, non ci sono ulteriori risultati da visualizzare.

Tag per l'allocazione dei costi Amazon SQS

Per organizzare e identificare le code Amazon SQS per l'allocazione dei costi, puoi aggiungere tag metadati che identificano lo scopo, il proprietario o l'ambiente di una coda. Questo è particolarmente utile quando si dispone di numerose code. Per configurare i tag utilizzando la console di Amazon SQS, consulta [the section called “Configurazione di tag per una coda”](#)

Puoi utilizzare i tag di allocazione dei costi per organizzare la AWS fattura in modo che rifletta la tua struttura dei costi. A tale scopo, registrati per ricevere nella Account AWS fattura le chiavi e i valori dei tag. Per ulteriori informazioni, consulta [Impostazione di un report di allocazione dei costi mensili](#) nella Guida per l'utente di AWS Billing .

Ogni tag è composto da una coppia chiave-valore definita dall'utente. Ad esempio, è possibile identificare facilmente le code di produzione e testing se tagghi le tue code nel modo seguente:

Queue	Chiave	Valore
MyQueueA	QueueType	Production
MyQueueB	QueueType	Testing

Note

Quando usi i tag di coda, tieni a mente le seguenti linee guida:

- Non è consigliabile aggiungere più di 50 tag a una coda. I tag supportano i caratteri Unicode in formato UTF-8.
- I tag non hanno alcun significato semantico. Amazon SQS interpreta i tag come stringhe di caratteri.
- I tag rispettano la distinzione tra maiuscole e minuscole.
- Un nuovo tag con una chiave identica a quella di un tag esistente sovrascrive il tag esistente.
- Le azioni di tagging sono limitate a 30 TPS ciascuna. Account AWS Se la tua applicazione richiede una velocità di trasmissione effettiva più elevata, [invia una richiesta](#).

Per l'elenco completo dei tipi di risorse, consulta [Quote di coda standard di Amazon SQS](#).

Polling brevi e lunghi di Amazon SQS

Amazon SQS offre opzioni di polling brevi e lunghe per ricevere messaggi da una coda. Considera i requisiti della tua applicazione in termini di reattività ed efficienza dei costi quando scegli tra queste due opzioni di polling:

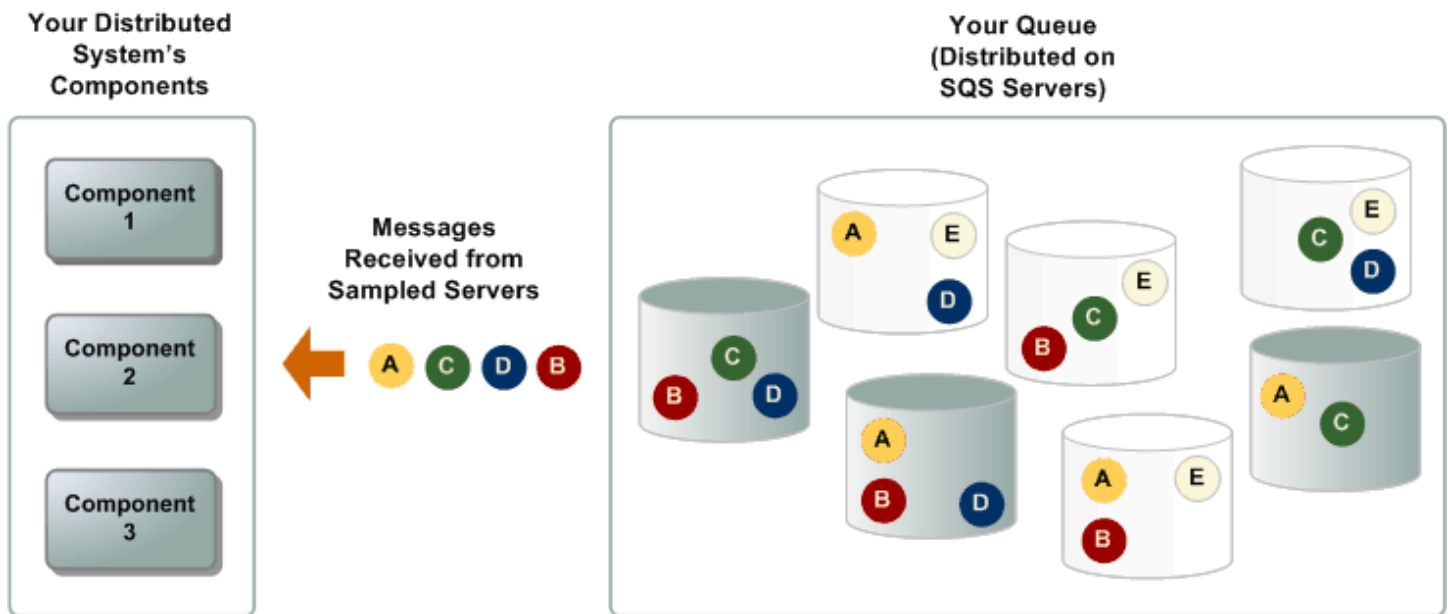
- Sondaggio breve (impostazione predefinita): la [ReceiveMessage](#)richiesta interroga un sottoinsieme di server (in base a una distribuzione casuale ponderata) per trovare i messaggi disponibili e invia una risposta immediata, anche se non viene trovato alcun messaggio.
- Sondaggio prolungato: [ReceiveMessage](#)interroga tutti i server alla ricerca di messaggi, inviando una risposta una volta che è disponibile almeno un messaggio, fino al massimo specificato. Una risposta vuota viene inviata solo se scade il tempo di attesa per il polling. Questa opzione può ridurre il numero di risposte vuote e potenzialmente ridurre i costi.

Nelle sezioni seguenti vengono illustrati i dettagli del polling breve e del polling lungo.

Utilizzo dei messaggi con lo short polling

Quando consumi messaggi da una coda (FIFO o standard) utilizzando un polling breve, Amazon SQS campiona un sottoinsieme dei suoi server (in base a una distribuzione casuale ponderata) e restituisce messaggi solo da quei server. Pertanto, una determinata richiesta [ReceiveMessage](#) potrebbe non restituire tutti i messaggi. Tuttavia, se hai meno di 1.000 messaggi in coda, una richiesta successiva restituirà i tuoi messaggi. Se continui a utilizzare messaggi dalle code, Amazon SQS campiona tutti i propri server e riceverai tutti i messaggi.

Il diagramma qui di seguito mostra il comportamento di short-polling dei messaggi restituiti da una coda standard dopo che uno dei componenti del sistema invia una richiesta di ricezione. Amazon SQS campiona diversi server (in grigio) e restituisce i messaggi A, C, D e B da questi server. Il messaggio E non viene restituito in questa richiesta, ma viene restituito in una richiesta successiva.



Utilizzo di messaggi con long polling

Quando il tempo di attesa per l'azione dell'API [ReceiveMessage](#) è superiore a 0, viene attivato un polling lungo. Il tempo massimo di attesa per il polling lungo è di 20 secondi. Il polling prolungato aiuta a ridurre i costi di utilizzo di Amazon SQS riducendo il numero di risposte vuote (quando non ci sono messaggi disponibili per `ReceiveMessage` una richiesta) e di false risposte vuote (quando i messaggi sono disponibili ma non sono inclusi in una risposta). Per informazioni su come abilitare il polling lungo per una coda nuova o esistente utilizzando la console Amazon SQS, consulta [Configurazione dei parametri della coda tramite la console Amazon SQS](#). Per le best practice, consulta [Configurazione di sondaggi lunghi in Amazon SQS](#).

I sondaggi prolungati offrono i seguenti vantaggi:

- Elimina le risposte vuote consentendo a Amazon SQS di attendere fino a che un messaggio non è disponibile nella coda prima di inviare una risposta. A meno che la connessione non scada, la risposta alla richiesta `ReceiveMessage` contiene almeno uno dei messaggi disponibili, fino al numero massimo di messaggi specificato nell'operazione `ReceiveMessage`. In rari casi, potresti ricevere risposte vuote anche quando una coda contiene ancora messaggi, specialmente se hai specificato un valore basso per il parametro [ReceiveMessageWaitTimeSeconds](#).
- Riduci le false risposte vuote interrogando tutti i server Amazon SQS, anziché un sottoinsieme di essi.
- Restituisci i messaggi non appena diventano disponibili.

Per informazioni su come confermare che una coda è vuota, consulta [Conferma che una coda Amazon SQS è vuota](#).

Differenze tra short e long polling

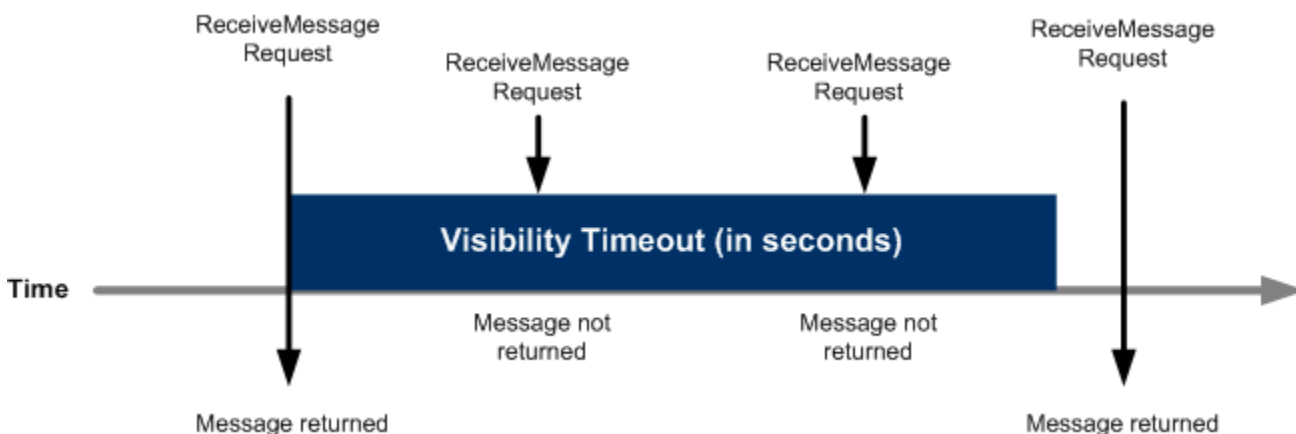
Lo short polling si verifica quando il parametro [WaitTimeSeconds](#) di una richiesta [ReceiveMessage](#) è impostato su 0 in uno dei seguenti due modi:

- La chiamata `ReceiveMessage` imposta `WaitTimeSeconds` su 0.
- La chiamata `ReceiveMessage` non imposta `WaitTimeSeconds` e l'attributo coda [ReceiveMessageWaitTimeSeconds](#) è impostato su 0.

Timeout visibilità Amazon SQS

Quando ricevi un messaggio da una coda Amazon SQS, questo rimane in coda ma diventa temporaneamente invisibile agli altri consumatori. Questa invisibilità è controllata dal timeout di visibilità, che garantisce che altri consumatori non possano elaborare lo stesso messaggio mentre lo stai elaborando. Amazon SQS offre due opzioni per eliminare i messaggi dopo l'elaborazione:

- Eliminazione manuale: i messaggi vengono eliminati in modo esplicito utilizzando l'azione [DeleteMessage](#)
- Eliminazione automatica: supportata in alcuni casi AWS SDKs, i messaggi vengono eliminati automaticamente al termine dell'elaborazione, semplificando i flussi di lavoro.



Casi d'uso del timeout di visibilità

Gestione delle attività di lunga durata: utilizza il timeout di visibilità per gestire attività che richiedono tempi di elaborazione prolungati. Imposta un timeout di visibilità appropriato per i messaggi che richiedono tempi di elaborazione prolungati. Ciò garantisce che altri utenti non ricevano lo stesso messaggio durante l'elaborazione, evitando operazioni duplicate e mantenendo l'efficienza del sistema.

Implementazione di meccanismi di ripetizione dei tentativi: estendi il timeout di visibilità a livello di codice per le attività che non vengono completate entro il timeout iniziale. Se un'attività non viene completata entro il timeout di visibilità iniziale, puoi estendere il timeout a livello di codice. Ciò consente al sistema di rielaborare il messaggio senza che diventi visibile agli altri utenti, migliorando la tolleranza agli errori e l'affidabilità. Combinalo con Dead-Letter Queues (DLQs) per gestire gli errori persistenti.

Sistemi distribuiti di coordinate: utilizzate il timeout di visibilità di SQS per coordinare le attività tra sistemi distribuiti. Imposta dei timeout di visibilità in linea con i tempi di elaborazione previsti per i diversi componenti. Ciò aiuta a mantenere la coerenza e previene le condizioni di gara in architetture complesse e distribuite.

Ottimizza l'utilizzo delle risorse: regola i timeout di visibilità di SQS per ottimizzare l'utilizzo delle risorse nella tua applicazione. Impostando i timeout appropriati, è possibile garantire che i messaggi vengano elaborati in modo efficiente senza occupare inutilmente le risorse. Ciò porta a migliori prestazioni complessive del sistema e a un miglior rapporto qualità-prezzo.

Impostazione e regolazione del timeout di visibilità

Il timeout di visibilità inizia non appena ti viene recapitato un messaggio. Durante questo periodo, dovresti elaborare ed eliminare il messaggio. Se non lo elimini prima della scadenza del timeout, il messaggio diventa nuovamente visibile nella coda e può essere recuperato da un altro consumatore. Il timeout di visibilità predefinito per una coda è di 30 secondi, ma puoi modificarlo in modo che corrisponda al tempo necessario all'applicazione per elaborare ed eliminare un messaggio. È inoltre possibile impostare un timeout di visibilità specifico per i singoli messaggi senza modificare l'impostazione generale della coda. Utilizza l'[ChangeMessageVisibility](#) azione per estendere o abbreviare il timeout a livello di codice in base alle esigenze.

Messaggi e quote in corso

In Amazon SQS, i messaggi in arrivo sono messaggi ricevuti da un consumatore ma non ancora eliminati. Per le code standard, c'è un limite di circa 120.000 messaggi in volo, a seconda del traffico in coda e dell'arretrato di messaggi. Se raggiungi questo limite, Amazon SQS restituisce un `OverLimit` errore, indicando che non è possibile ricevere messaggi aggiuntivi fino a quando alcuni messaggi in volo non vengono eliminati. Per le code FIFO, i limiti dipendono dai gruppi di messaggi attivi.

- Quando si utilizza il polling breve: se questo limite viene raggiunto durante l'utilizzo del polling breve, Amazon SQS restituirà `OverLimit` un errore che indica che non è possibile ricevere messaggi aggiuntivi fino a quando alcuni messaggi in transito non vengono eliminati.
- Quando si utilizza un polling lungo: se utilizzi un polling lungo, Amazon SQS non restituisce un errore quando viene raggiunto il limite di messaggi in corso. Al contrario, non restituirà nuovi messaggi finché il numero di messaggi in transito non scende al di sotto del limite.

Per gestire efficacemente i messaggi in volo:

1. Eliminazione rapida: elimina i messaggi (manualmente o automaticamente) dopo l'elaborazione per ridurre il numero di messaggi in volo.
2. Monitora con CloudWatch: imposta allarmi per un numero elevato di passeggeri in volo per evitare di raggiungere il limite.
3. Distribuisci il carico: se stai elaborando un volume elevato di messaggi, utilizza code o utenti aggiuntivi per bilanciare il carico ed evitare colli di bottiglia.
4. Richiedi un aumento della quota: invia una richiesta al [AWS Supporto](#) se sono richiesti limiti più elevati.

Comprensione del timeout di visibilità nelle code standard e FIFO

Sia nelle code standard che in quelle FIFO (First-In-First-Out), il timeout di visibilità aiuta a impedire che più consumatori elaborino lo stesso messaggio contemporaneamente. Tuttavia, a causa del modello di at-least-once consegna di Amazon SQS, non esiste alcuna garanzia assoluta che un messaggio non venga recapitato più di una volta durante il periodo di timeout di visibilità.

- Code standard: il timeout di visibilità nelle code standard impedisce a più consumatori di elaborare lo stesso messaggio contemporaneamente. Tuttavia, a causa del modello di at-least-once

consegna, Amazon SQS non garantisce che un messaggio non venga recapitato più di una volta entro il periodo di timeout di visibilità.

- **Code FIFO:** per le code FIFO, i messaggi con lo stesso ID di gruppo di messaggi vengono elaborati in una sequenza rigorosa. Quando un messaggio con un ID di gruppo di messaggi è in corso, i messaggi successivi in quel gruppo non vengono resi disponibili fino a quando il messaggio in corso non viene eliminato o non scade il timeout di visibilità. Tuttavia, ciò non «blocca» il gruppo a tempo indeterminato: ogni messaggio viene elaborato in sequenza e solo quando ogni messaggio viene eliminato o diventa nuovamente visibile il messaggio successivo di quel gruppo sarà disponibile per i consumatori. Questo approccio garantisce un'elaborazione ordinata all'interno del gruppo senza impedire inutilmente al gruppo di recapitare i messaggi.

Gestione degli errori

Se non elabori ed elimini un messaggio prima della scadenza del timeout di visibilità, a causa di errori dell'applicazione, arresti anomali o problemi di connettività, il messaggio diventa nuovamente visibile nella coda. Può quindi essere recuperato dallo stesso consumatore o da un altro consumatore per un altro tentativo di elaborazione. Ciò garantisce che i messaggi non vadano persi anche se l'elaborazione iniziale fallisce. Tuttavia, l'impostazione di un timeout di visibilità troppo elevato può ritardare la ricomparsa dei messaggi non elaborati, rallentando potenzialmente i nuovi tentativi. È fondamentale impostare un timeout di visibilità appropriato in base al tempo di elaborazione previsto per una gestione tempestiva dei messaggi.

Modifica e interruzione del timeout di visibilità

È possibile modificare o terminare il timeout di visibilità utilizzando l'azione `ChangeMessageVisibility`:

- **Modifica del timeout:** regola il timeout di visibilità in modo dinamico utilizzando [ChangeMessageVisibility](#). Ciò consente di estendere o ridurre la durata dei timeout per soddisfare le esigenze di elaborazione.
- **Cessazione del timeout:** se decidi di non elaborare un messaggio ricevuto, interrompi il timeout di visibilità impostando il valore su 0 secondi dopo l'azione `ChangeMessageVisibility`. Ciò rende immediatamente il messaggio disponibile per l'elaborazione da parte di altri consumatori.

Best practice

Utilizza le seguenti best practice per gestire i timeout di visibilità in Amazon SQS, tra cui l'impostazione, la regolazione e l'estensione dei timeout, nonché la gestione dei messaggi non elaborati utilizzando Dead-Letter Queues (). DLQs

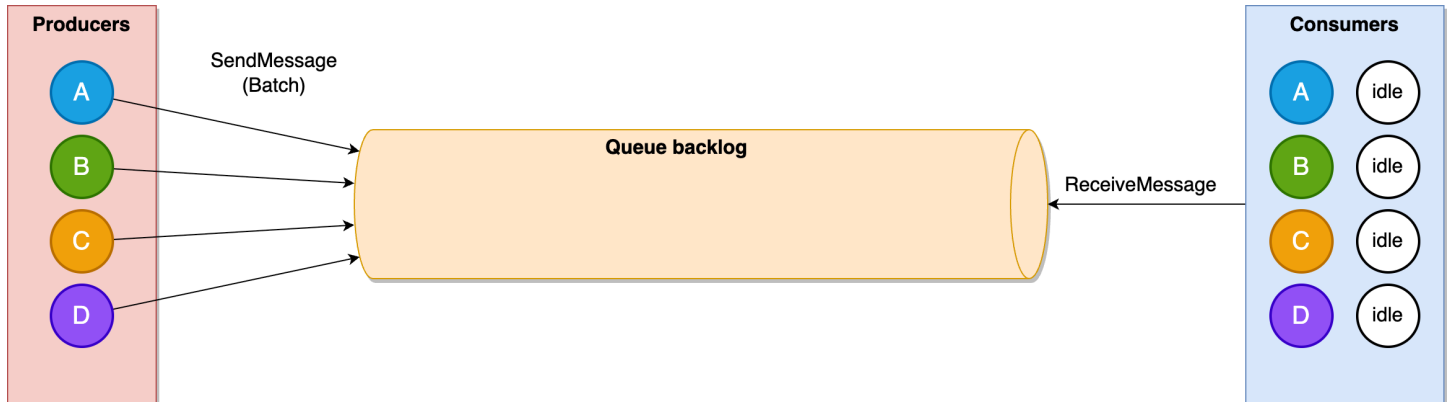
- **Impostazione e regolazione del timeout.** Inizia impostando il timeout di visibilità in modo che corrisponda al tempo massimo normalmente necessario all'applicazione per elaborare ed eliminare un messaggio. Se non sei sicuro del tempo esatto di elaborazione, inizia con un timeout più breve (ad esempio, 2 minuti) ed estendilo se necessario. Implementa un meccanismo heartbeat per estendere periodicamente il timeout di visibilità, assicurando che il messaggio rimanga invisibile fino al completamento dell'elaborazione. Ciò riduce al minimo i ritardi nella rielaborazione dei messaggi non gestiti e impedisce una visibilità prematura.
- **Estensione del timeout e gestione del limite di 12 ore.** Se il tempo di elaborazione varia o può superare il timeout inizialmente impostato, utilizza `ChangeMessageVisibility` per estendere il timeout di visibilità durante l'elaborazione del messaggio. Tieni presente che il timeout di visibilità ha un limite massimo di 12 ore dalla prima ricezione del messaggio. L'estensione del timeout non reimposta questo limite di 12 ore. Se l'elaborazione richiede più tempo rispetto a questo limite, valuta la possibilità di utilizzare AWS Step Functions o suddividere l'attività in fasi più piccole.
- **Gestione dei messaggi non elaborati.** Per gestire i messaggi che non superano più tentativi di elaborazione, configura una Dead-Letter Queue (DLQ). Ciò garantisce che i messaggi che non possono essere elaborati dopo diversi tentativi vengano acquisiti separatamente per un'ulteriore analisi o gestione, evitando che circolino ripetutamente nella coda principale.

Le code per le fiere di Amazon SQS

Amazon SQS fair queues mitiga automaticamente l'impatto del rumore di prossimità nelle code multi-tenant che contengono messaggi provenienti da più entità logiche, come clienti, applicazioni client o tipi di messaggi. In questi ambienti di coda condivisi, una metrica prestazionale fondamentale è il tempo di permanenza, che misura il tempo totale di permanenza dei messaggi in coda dall'arrivo all'elaborazione. Quando un tenant crea un arretrato nella coda pubblicando più messaggi di quanti il sistema sia in grado di gestire, fair queues riduce al minimo l'impatto sul tempo di permanenza per gli altri tenant.

Stato stazionario

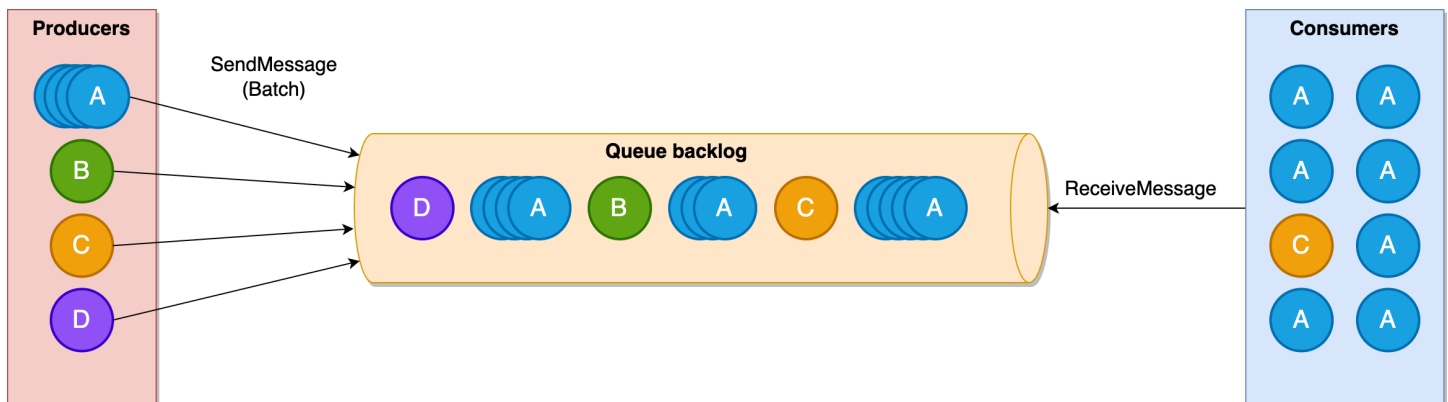
Il diagramma seguente illustra una coda multi-tenant contenente messaggi provenienti da quattro tenant distinti (etichettati A, B, C e D). La coda funziona in uno stato stazionario e non vi è alcun arretrato di messaggi in quanto i consumatori ricevono i messaggi non appena compaiono nella coda. Tutti gli inquilini hanno tempi di permanenza bassi. Non tutta la capacità dei consumatori è pienamente utilizzata in questo stato stazionario.



Impatto rumoroso sui vicini

L'impatto di Noisy Neighbor si verifica quando un tenant in una coda multi-tenant crea un backlog, aumentando il tempo di permanenza dei messaggi per tutti gli altri tenant. Un tenant può diventare un vicino rumoroso inviando un volume di messaggi maggiore rispetto agli altri tenant o quando i consumatori impiegano più tempo a elaborare i messaggi di quel particolare tenant.

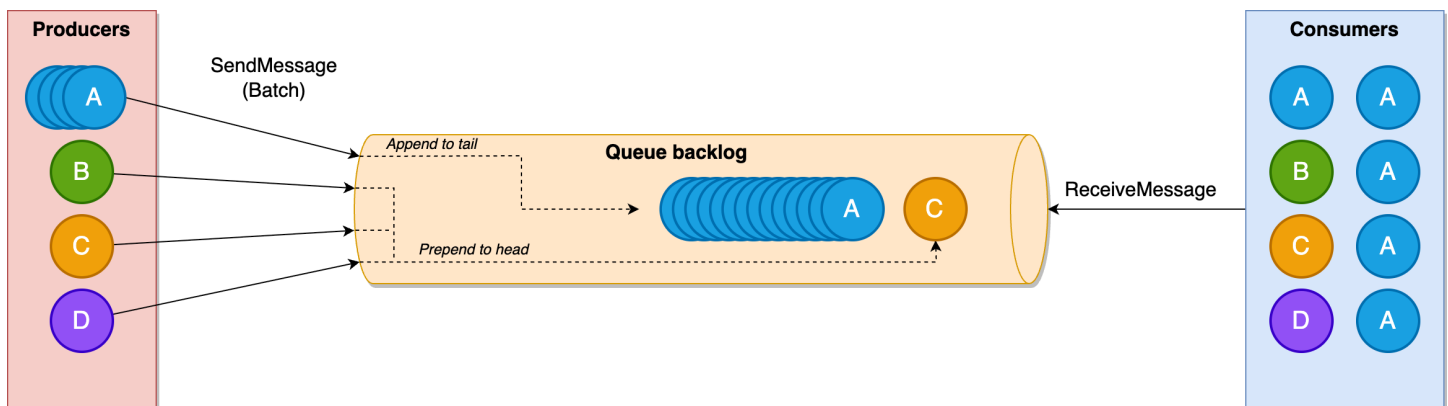
Questo diagramma illustra come l'aumento del traffico proveniente dal Tenant A crei un arretrato nella coda. I consumatori sono impegnati a elaborare i messaggi provenienti solo dal Tenant A, mentre i messaggi degli altri tenant attendono nel backlog, con conseguenti tempi di permanenza più lunghi per tutti i tenant.



Attenuazione con code eque

Amazon SQS rileva i vicini rumorosi monitorando la distribuzione dei messaggi tra i tenant durante l'elaborazione (lo stato «in volo»). Quando un tenant ha un numero sproporzionato di messaggi in volo rispetto agli altri, Amazon SQS identifica quel tenant come un vicino rumoroso e dà priorità alla consegna dei messaggi per gli altri tenant. Questo approccio riduce l'impatto del tempo di permanenza sugli altri tenant.

Questo diagramma illustra come Amazon SQS fair queues risolve il problema dei vicini rumorosi. Quando un tenant (Tenant A) diventa rumoroso, Amazon SQS dà la priorità alla restituzione dei messaggi da altri tenant (B, C e D). Questa prioritizzazione aiuta a mantenere bassi i tempi di permanenza per i tenant silenziosi (Tenant B, C e D), mentre il tempo di permanenza per i messaggi del Tenant A è elevato fino a esaurire il backlog della coda senza influire sugli altri tenant.



Note

Amazon SQS non limita il tasso di consumo per tenant. Consente ai consumatori di ricevere messaggi da inquilini vicini rumorosi quando il numero di utenti è disponibile e la coda non ha altri messaggi da restituire. Come le code standard di Amazon SQS, le code fair consentono un throughput praticamente illimitato e non ci sono limiti al numero di tenant che è possibile avere in coda.

Differenza rispetto alle code FIFO

Le code FIFO mantengono un ordine rigoroso limitando il numero di messaggi in volo da ciascun inquilino. In questo modo si evitano i rumorosi vicini, dall'altro limita la velocità di trasmissione per ogni inquilino. Le code eque sono progettate per scenari multi-tenant in cui la produttività elevata, il tempo di permanenza ridotto e l'equa allocazione delle risorse sono priorità. Le code eque consentono a più consumatori di elaborare contemporaneamente i messaggi dello stesso inquilino, aiutando tutti gli inquilini a mantenere tempi di permanenza costanti.

Utilizzo di code eque

I produttori di messaggi possono aggiungere un identificatore del tenant impostando un `MessageGroupId` su un messaggio in uscita:

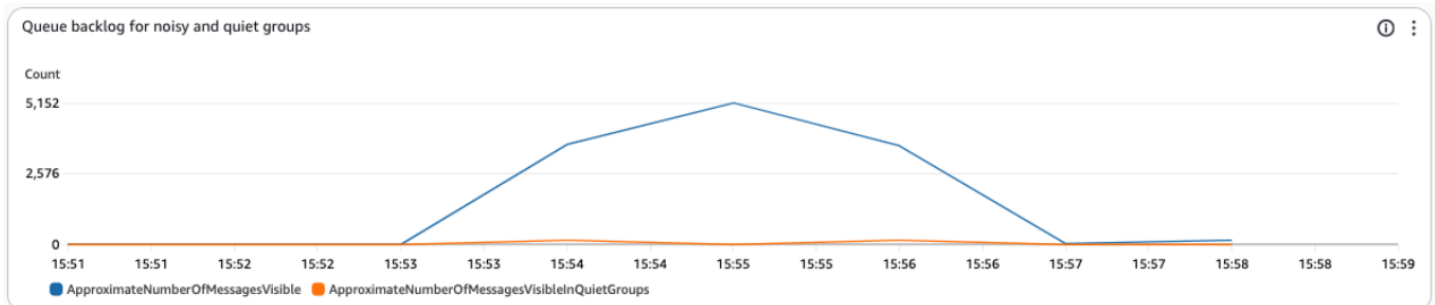
```
// Send message with tenant identifier
SendMessageRequest request = new SendMessageRequest()
    .withQueueUrl(queueUrl)
    .withMessageBody(messageBody)
    .withMessageGroupId("tenant-123"); // Tenant identifier
sqs.sendMessage(request);
```

La funzionalità di equità verrà applicata automaticamente in tutte le code standard di Amazon SQS per i messaggi con la proprietà `MessageGroupId`. Non richiede alcuna modifica del codice consumer, non ha alcun impatto sulla latenza delle API e non presenta limitazioni di throughput.

Metriche relative alle code eque CloudWatch

Amazon SQS fornisce CloudWatch parametri aggiuntivi per aiutarti a monitorare la mitigazione dell'impatto dei rumori vicini. Ad esempio, puoi confrontare `ApproximateNumberOfMessagesVisibleInQuietGroups` le metriche con le metriche standard a livello di coda. Durante i picchi di traffico per uno specifico tenant, le metriche generali a livello di coda potrebbero rivelare un aumento degli arretrati o una vecchiaia dei messaggi più vecchia. Tuttavia, esaminando isolatamente i gruppi silenziosi, è possibile constatare che la maggior parte dei gruppi di messaggi o degli inquilini non rumorosi non ne risente.

Di seguito è riportato un esempio in cui la metrica standard del backlog delle code (`ApproximateNumberOfMessagesVisible`) aumenta a causa di un tenant rumoroso, mentre il backlog per i tenant non rumorosi () rimane basso. `ApproximateNumberOfMessagesVisibleInQuietGroups`



Per un elenco completo dei parametri di Amazon SQS e delle relative descrizioni, consulta CloudWatch [CloudWatch Metrics for Amazon SQS](#).

Code di ritardo Amazon SQS

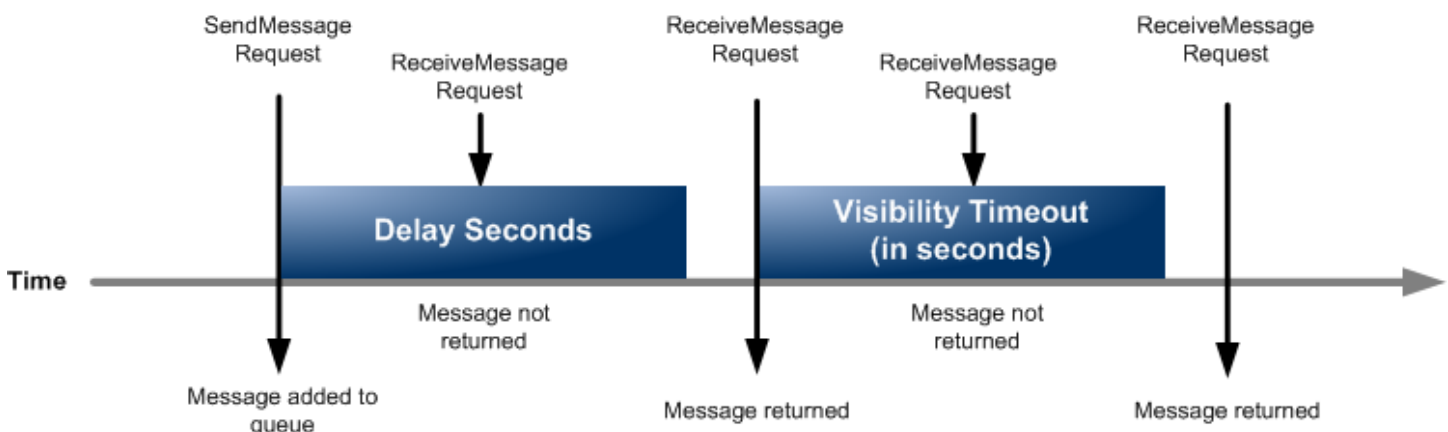
Le code di ritardo consentono di posticipare la consegna di nuovi messaggi ai consumatori per alcuni secondi, ad esempio quando l'applicazione consumer richiede più tempo per elaborare i messaggi. Se crei una coda di ritardo, qualsiasi messaggio inviato a tale coda rimane invisibile ai consumatori per la durata del periodo di ritardo. Il ritardo predefinito (minimo) per una coda è 0 secondi. Il valore massimo è 15 minuti. Per ulteriori informazioni sulla configurazione delle code di ritardo mediante la console, consultare [Configurazione dei parametri della coda tramite la console Amazon SQS](#).

Note

Per le code standard, l'impostazione di ritardo per coda non è retroattiva: se modifichi l'impostazione, non influisce sul ritardo dei messaggi già nella coda.

Per le code FIFO, l'impostazione di ritardo per coda è retroattiva: se modifichi l'impostazione, influisce sul ritardo dei messaggi già nella coda.

Le code di ritardo sono simili al [timeout visibilità](#) perché entrambe le caratteristiche rendono non disponibili i messaggi ai consumatori per un determinato periodo di tempo. La differenza tra le code di ritardo e i timeout visibilità è che per le code di ritardo un messaggio viene nascosto quando viene aggiunto per la prima volta alla coda, mentre per i timeout visibilità un messaggio viene nascosto solo dopo che un messaggio viene consumato dalla coda. Il seguente diagramma mostra il rapporto tra le code di ritardo e i timeout visibilità.



Opzioni di pianificazione estese

Sebbene le code di ritardo e i timer dei messaggi di Amazon SQS consentano di pianificare la consegna dei messaggi fino a 15 minuti in futuro, potresti aver bisogno di funzionalità di

pianificazione più flessibili. In questi casi, prendi in considerazione l'utilizzo di [EventBridge Scheduler](#), che ti consente di pianificare miliardi di azioni API una tantum o ricorrenti senza limiti di tempo. EventBridge Scheduler è la soluzione consigliata per casi d'uso avanzati di pianificazione dei messaggi.

Per impostare i secondi di ritardo su singoli messaggi, anziché su un'intera coda, utilizza [i timer dei messaggi](#) per consentire ad Amazon SQS di utilizzare il valore `DelaySeconds` del timer dei messaggi anziché il valore della coda di ritardo. `DelaySeconds` [EventBridge Scheduler supporta anche la pianificazione di](#) singoli messaggi.

Code temporanee di Amazon SQS

Le code temporanee consentono di risparmiare tempo di sviluppo e costi di implementazione quando si utilizzano schemi di messaggi comuni come request-response. È possibile utilizzare il [Temporary Queue Client](#) per creare code temporanee ad alta velocità, convenienti e gestite dalle applicazioni.

Il client mappa automaticamente più code temporanee, code gestite da applicazioni create su richiesta per un particolare processo, su una singola coda Amazon SQS. In questo modo l'applicazione può effettuare meno chiamate API e incrementare il throughput quando il traffico per ogni coda temporanea è basso. Quando una coda temporanea non è più in uso, il client la pulisce automaticamente, anche se alcuni processi che utilizzano il client non sono chiusi correttamente.

Di seguito sono elencati i vantaggi delle code temporanee:

- Agiscono da canali di comunicazione leggeri per specifici thread o processi.
- Possono essere creati ed eliminati senza incorrere in costi aggiuntivi.
- Sono compatibili a livello di API con le code statiche (normali) Amazon SQS. Ciò significa che il codice esistente che invia e riceve messaggi può inviare e ricevere messaggi dalle code virtuali.

Code virtuali

Le code virtuali sono strutture di dati locali create da Temporary Queue Client. Le code virtuali consentono di combinare più destinazioni a traffico ridotto in un'unica coda Amazon SQS. Per le migliori pratiche, consulta [Evitare di riutilizzare lo stesso ID del gruppo di messaggi con le code virtuali](#).

 Note

- Quando si crea una coda virtuale vengono create solo le strutture di dati temporanee in cui i consumatori ricevono i messaggi. Dato che una coda virtuale non effettua chiamate API a Amazon SQS, le code virtuali non prevedono costi.
- Le quote TPS si applicano a tutte le code virtuali in una singola coda host. Per ulteriori informazioni, consulta [Quote di messaggi Amazon SQS](#).

La classe wrapper `AmazonSQSVirtualQueuesClient` aggiunge il supporto per gli attributi relativi alle code virtuali. Per creare una coda virtuale, è necessario chiamare l'operazione API `CreateQueue` utilizzando l'attributo `HostQueueURL`. Questo attributo specifica la coda esistente che ospita le code virtuali.

L'URL di una coda virtuale è nel formato seguente.

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

Quando un produttore chiama l'operazione API `SendMessageBatch` o `SendMessage` sull'URL di una coda virtuale, `Temporary Queue Client` esegue queste operazioni:

1. Estrae il nome della coda virtuale.
2. Collega il nome della coda virtuale come ulteriore attributo del messaggio.
3. Invia il messaggio alla coda host.

Mentre il produttore invia messaggi, un thread in background esegue il polling della coda host e invia i messaggi ricevuti alle code virtuali in base ai corrispondenti attributi di messaggio.

Mentre il consumatore chiama l'operazione API `ReceiveMessage` sull'URL di una coda virtuale, `Temporary Queue Client` blocca la chiamata in locale finché il thread in background non invia un messaggio alla coda virtuale. Si tratta di una procedura simile al prefetching del messaggio nel [Buffered Asynchronous Client](#): una singola operazione API può fornire messaggi fino a 10 code virtuali). L'eliminazione di una coda virtuale rimuove tutte le risorse lato client, senza chiamare Amazon SQS.

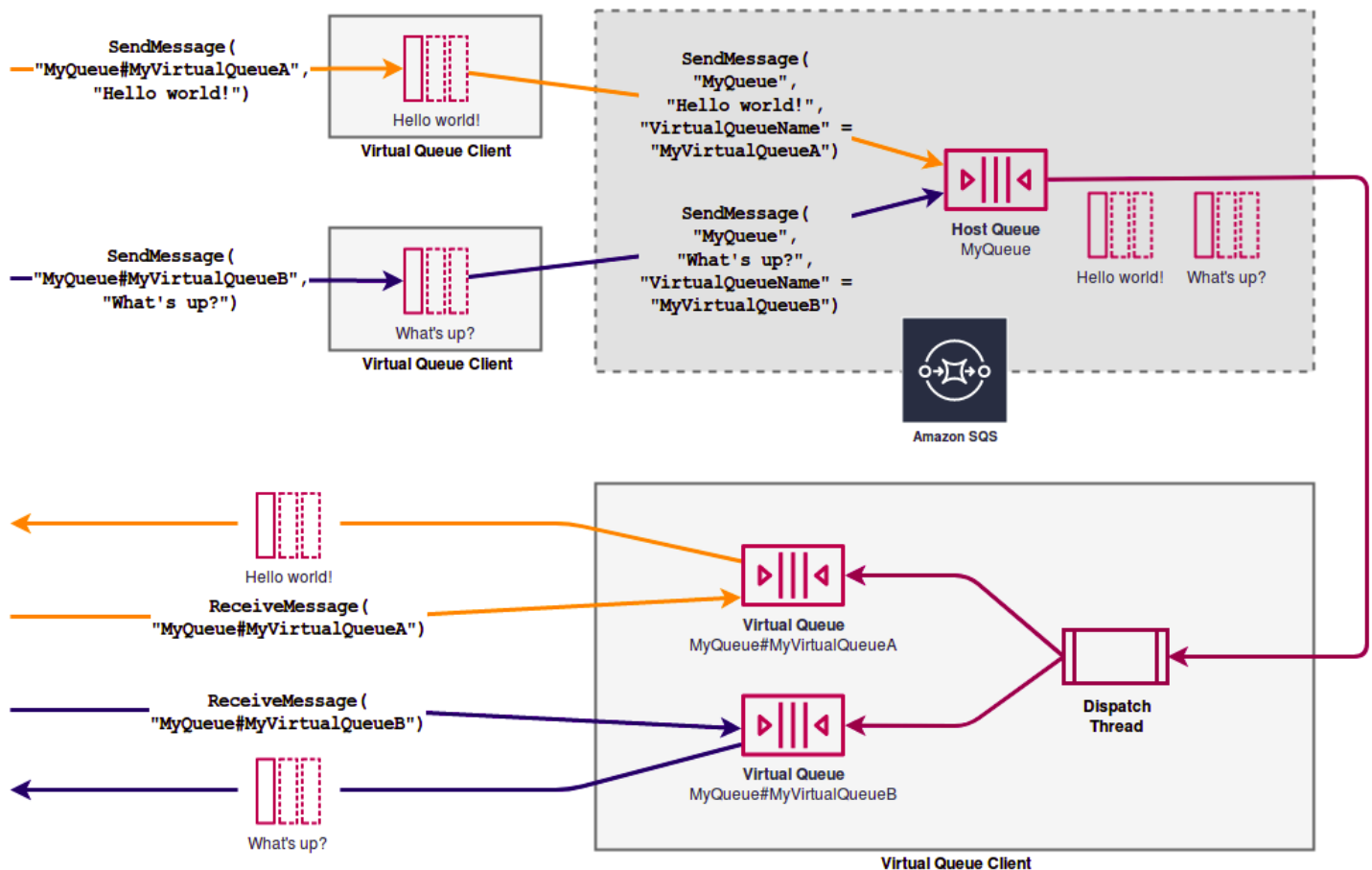
La classe `AmazonSQSTemporaryQueuesClient` trasforma automaticamente tutte le code che crea in code temporanee. Inoltre, crea automaticamente le code host con gli stessi attributi della coda,

su richiesta. Questi nomi di code condividono un prefisso comune configurabile (per impostazione predefinita `__RequesterClientQueues__`) che li identifica come code temporanee. In questo modo il client può agire da sostituzione drop-in che ottimizza la coda esistente che crea ed elimina le code. Il client include anche le interfacce `AmazonSQSRequester` e `AmazonSQSResponder` che consentono la comunicazione bidirezionale tra le code.

Modelli di messaggistica richiesta-risposta (code virtuali)

Il caso d'uso più comune per le code temporanee è il modello di messaggistica richiesta-risposta, in cui un richiedente crea una coda temporanea per la ricezione di ciascun messaggio di risposta. Per evitare di creare una coda Amazon SQS per ogni messaggio di risposta, Temporary Queue Client consente di creare ed eliminare più code temporanee senza effettuare chiamate API Amazon SQS. Per ulteriori informazioni, vedere Implementazione di sistemi di richiesta-risposta.

Il diagramma seguente mostra una configurazione comune che usa questo modello.



Scenario di esempio: elaborazione di una richiesta di accesso

Il seguente scenario di esempio mostra come usare le interfacce `AmazonSQSResponder` e `AmazonSQSRequester` per elaborare la richiesta di accesso di un utente.

Sul lato client

```
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();
    }
}
```

L'invio di una richiesta di accesso esegue queste operazioni:

1. Crea una coda temporanea.
2. Collega l'URL della coda temporanea al messaggio come attributo.
3. Invia il messaggio.

4. Riceve una risposta dalla coda temporanea.
5. Elimina la coda temporanea.
6. Restituisce la risposta.

Sul lato server

L'esempio seguente presuppone che, al momento della creazione, venga creato un thread per eseguire il polling della coda e chiamare il metodo `handleLoginRequest()` per ogni messaggio. Inoltre, presuppone l'uso del metodo `doLogin()`.

```
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));
    }
}
```


Pulizia delle query

Per garantire che Amazon SQS recuperi tutte le risorse in memoria utilizzate dalle code virtuali, quando l'applicazione non ha più bisogno di Temporary Queue Client, deve chiamare il metodo `shutdown()`. Puoi anche utilizzare il metodo `shutdown()` dell'interfaccia `AmazonSQSRequester`.

Temporary Queue Client, inoltre, fornisce un modo per eliminare le code host orfane. Per ogni coda che riceve una chiamata API in un periodo di tempo (per impostazione predefinita, cinque minuti), il client utilizza l'operazione `API TagQueue` per applicare tag a una coda che rimane in uso.

Note

Qualsiasi operazione API eseguita su una coda la contrassegna come non inattiva, compresa un'operazione `ReceiveMessage` che non restituisce messaggi.

Il thread in background usa le operazioni API `ListQueues` e `ListTags` per controllare tutte le code con il prefisso configurato, eliminando quelle che non sono state contrassegnate con tag per almeno cinque minuti. In questo modo, se un client non si chiude correttamente, gli altri client attivi vengono puliti dopo di esso. Per ridurre le duplicazioni di lavoro, tutti i client con lo stesso prefisso comunicano attraverso una coda di lavoro interna condivisa, il cui nome corrisponde al prefisso.

Timer di messaggi Amazon SQS

I timer dei messaggi consentono di impostare un periodo di invisibilità iniziale per un messaggio quando viene aggiunto a una coda. Ad esempio, se invii un messaggio con un timer di 45 secondi, questo rimane nascosto ai consumatori per i primi 45 secondi. Il ritardo predefinito (minimo) per un messaggio è 0 secondi. Il valore massimo è 15 minuti. Per ulteriori informazioni sull'invio di messaggi con timer tramite la console, consulta [Invio di un messaggio utilizzando una coda standard](#).

Note

Le code FIFO non supportano i timer sui singoli messaggi.

[Per impostare un periodo di ritardo su un'intera coda, anziché su singoli messaggi, utilizza le code di ritardo.](#) Un'impostazione di timer messaggio per un singolo messaggio sostituisce il valore `DelaySeconds` che si applica a una coda di ritardo Amazon SQS.

Opzioni di pianificazione estese

Sebbene le code di ritardo e i timer dei messaggi di Amazon SQS consentano di pianificare la consegna dei messaggi fino a 15 minuti in futuro, potresti aver bisogno di funzionalità di pianificazione più flessibili. In questi casi, prendi in considerazione l'utilizzo di [EventBridge Scheduler](#), che ti consente di pianificare miliardi di azioni API una tantum o ricorrenti senza limiti di tempo. EventBridge Scheduler è la soluzione consigliata per casi d'uso avanzati di pianificazione dei messaggi.

Accesso ad Amazon EventBridge Pipes tramite la console Amazon SQS

Amazon EventBridge Pipes collega le sorgenti alle destinazioni. Le pipe sono destinate point-to-point alle integrazioni tra sorgenti e destinazioni supportate, con supporto per trasformazioni e arricchimenti avanzati. EventBridge Pipes offrono un modo altamente scalabile per connettere la coda Amazon SQS AWS a servizi come Step Functions, Amazon SQS e API Gateway, nonché ad applicazioni SaaS (Software as a Service) di terze parti come Salesforce.

Per configurare una pipe, si sceglie l'origine, si aggiungono filtri facoltativi, si definisce l'arricchimento facoltativo e si sceglie la destinazione per i dati dell'evento.

Nella pagina dei dettagli di una coda Amazon SQS, puoi visualizzare le pipe che utilizzano quella coda come origine. Da lì, puoi anche:

- Avvia la console per visualizzare i dettagli delle tubazioni. EventBridge
- Avvia la EventBridge console per creare una nuova pipe con la coda come sorgente.

Per ulteriori informazioni sulla configurazione di una coda Amazon SQS come sorgente pipe, consulta Amazon [SQS queue as a source nella Amazon User Guide](#). EventBridge [Per ulteriori informazioni sulle EventBridge tubazioni in generale, consulta Pipes. EventBridge](#)

Per accedere alle EventBridge pipe per una determinata coda Amazon SQS

1. Nella console Amazon SQS, apri la [pagina delle code](#).
2. Seleziona una coda.
3. Nella pagina dei dettagli della coda, scegli la scheda Pipes. EventBridge

La scheda EventBridge Pipes include un elenco di tutte le pipe attualmente configurate per utilizzare la coda selezionata come sorgente, tra cui:

- nome della pipe
- stato corrente
- destinazione della pipe
- quando la pipe è stata modificata l'ultima volta

4. Visualizza altri dettagli sulla pipe o crea una nuova pipe, se lo desideri:

- Accesso a dettagli aggiuntivi su una pipe:

Scegliere il nome della pipe.

Verrà avviata la pagina dei dettagli di Pipe della EventBridge console.

- Creazione di una nuova pipe:

Scegli Connetti la coda Amazon SQS alla pipe.

Verrà avviata la pagina Create pipe della EventBridge console, con la coda Amazon SQS specificata come origine della pipe. Per ulteriori informazioni, consulta [Creating an EventBridge pipe](#) nella Amazon EventBridge User Guide.

 Important

Un messaggio su una coda Amazon SQS viene letto da una singola pipe e quindi eliminato dalla coda dopo l'elaborazione, indipendentemente dal fatto che il messaggio corrisponda o meno al filtro configurato per quella pipe. Quando configuri più pipe fai attenzione in modo che utilizzino la stessa coda come sorgente.

Gestione di messaggi Amazon SQS di grandi dimensioni con Extended Client Library e Amazon Simple Storage Service

Usa [Amazon SQS Extended Client Library per Java](#) e [Amazon SQS Extended Client Library per Python per](#) inviare messaggi di grandi dimensioni, in particolare per payload compresi tra 256 KB e 2 GB. Queste librerie archiviano il payload dei messaggi in un bucket Amazon S3 e inviano un riferimento all'oggetto archiviato nella coda Amazon SQS.

 Note

Le librerie Amazon SQS Extended Client sono compatibili con le code standard e FIFO.

Gestione di messaggi Amazon SQS di grandi dimensioni con Java e Amazon S3

Usa la [Amazon SQS Extended Client Library per Java](#) con Amazon S3 per gestire messaggi Amazon SQS di grandi dimensioni, in particolare per payload che vanno da 256 KB a 2 GB. La libreria archivia il payload del messaggio in un bucket Amazon S3 e invia un messaggio contenente un riferimento all'oggetto memorizzato nella coda Amazon SQS.

Con Amazon SQS Extended Client Library per Java, puoi:

- Specificare se i messaggi vengono sempre archiviati in Amazon S3 o solo quando le dimensioni di un messaggio superano i 256 KB.
- Inviare un messaggio che fa riferimento a un solo oggetto messaggio archiviato in un bucket S3
- Recuperare l'oggetto messaggio da un bucket Amazon S3
- Eliminare l'oggetto messaggio da un bucket Amazon S3

Prerequisiti

L'esempio seguente utilizza l'SDK AWS Java. Per installare e configurare l'SDK, consulta [Configurare l' AWS SDK for Java](#) nella AWS SDK per Java Developer Guide.

Prima di eseguire il codice di esempio, configura le tue AWS credenziali. Per ulteriori informazioni, consulta [Configurare AWS le credenziali e la regione per lo sviluppo nella Guida per gli AWS SDK per Java](#) sviluppatori.

Gli [SDK per Java](#) e la libreria client ampia Amazon SQS per Java richiedono J2SE Development Kit 8.0 o versioni successive.

 Note

Puoi utilizzare la libreria client ampia Amazon SQS per Java per gestire i messaggi Amazon SQS utilizzando Amazon S3 solo con AWS SDK per Java. Non puoi farlo con AWS CLI, la console Amazon SQS, l'API HTTP di Amazon SQS o qualsiasi altra. AWS SDKs

AWS SDK for Java 2.x Esempio: utilizzo di Amazon S3 per gestire messaggi Amazon SQS di grandi dimensioni

Il seguente esempio di SDK for SDK for Java 2.x utilizza la Extended Client Library for Java per lavorare con messaggi di grandi dimensioni. Nel costruttore, il codice seguente:

- Crea un bucket Amazon S3 con un nome casuale
- Crea una coda SQS che inizia con MyQueue
- Racchiude un client Amazon S3 Java SDK standard in un'istanza di `AmazonSQSExtendedClient`

Nel `sendAnReceiveMessage` metodo, l'esempio invia un messaggio casuale che viene archiviato in un bucket Amazon S3 perché supera i 256 KB (la dimensione massima standard del messaggio). Infine, il metodo recupera il messaggio e ne visualizza le informazioni sulla console.

È possibile visualizzare l'esempio completo in 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(messageReceiptHandle).build());
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());
    }
}

```

Puoi [usare Apache Maven](#) per configurare e creare Amazon SQS Extended Client per il tuo progetto Java o per creare l'SDK stesso. Specificate i singoli moduli dall'SDK che utilizzate nella vostra applicazione.

```

<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>
</dependencies>

```

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

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

Gestione di messaggi Amazon SQS di grandi dimensioni con Python e Amazon S3

Usa la Amazon SQS [Amazon SQS Extended Client Library for Python](#) con Amazon S3 per gestire messaggi Amazon SQS di grandi dimensioni, in particolare per payload compresi tra 256 KB e 2 GB. La libreria archivia il payload del messaggio in un bucket Amazon S3 e invia un messaggio contenente un riferimento all'oggetto memorizzato nella coda Amazon SQS.

Con la Amazon SQS Extended Client Library per Python, puoi:

- Specificare se i payload sono sempre archiviati in Amazon S3 o archiviati solo in Amazon S3 quando una dimensione del payload supera i 256 KB
- Invia un messaggio che fa riferimento a un singolo oggetto di messaggio archiviato in un bucket Amazon S3
- Recupera l'oggetto payload corrispondente da un bucket Amazon S3
- Eliminare l'oggetto payload corrispondente da un bucket Amazon S3

Prerequisiti

Di seguito sono riportati i prerequisiti per l'utilizzo della Amazon SQS Extended Client Library for Python:

- Un AWS account con le credenziali necessarie. Per creare un AWS account, vai alla [AWS home page](#), quindi scegli Crea un AWS account. Segui le istruzioni. Per informazioni sulle credenziali, consulta [Credenziali](#).
- Un AWS SDK: l'esempio in questa pagina utilizza AWS Python SDK Boto3. Per installare e configurare l'SDK, consulta la documentazione dell'[AWS SDK per Python](#) nella Guida per sviluppatori SDK AWS for Python
- Python 3.x (o successivo) e. `pip`
- [La libreria Amazon SQS Extended Client per Python, disponibile da PyPI](#)

Note

Puoi utilizzare Amazon SQS Extended Client Library for Python per gestire i messaggi Amazon SQS utilizzando Amazon S3 solo con l'SDK per Python. AWS Non puoi farlo con la AWS CLI, la console Amazon SQS, l'API HTTP di Amazon SQS o qualsiasi altra. AWS SDKs

Configurazione dello storage dei messaggi

Amazon SQS Extended Client utilizza i seguenti attributi dei messaggi per configurare le opzioni di storage dei messaggi di Amazon S3:

- `large_payload_support`: il nome del bucket Amazon S3 per archiviare messaggi di grandi dimensioni.
- `always_through_s3`: Se `True`, allora tutti i messaggi vengono archiviati in Amazon S3. Se `False`, i messaggi di dimensioni inferiori a 256 KB non verranno serializzati nel bucket s3. Il valore predefinito è `False`.
- `use_legacy_attribute`: Se `True`, tutti i messaggi pubblicati utilizzano l'attributo Legacy reserved message (`SQSLargePayloadSize`) anziché l'attuale attributo riservato del messaggio (`ExtendedPayloadSize`).

Gestione di messaggi Amazon SQS di grandi dimensioni con Extended Client Library for Python

L'esempio seguente crea un bucket Amazon S3 con un nome casuale. Quindi crea una coda Amazon SQS denominata MyQueue e invia un messaggio che viene archiviato in un bucket S3 e contiene più di 256 KB nella coda. Infine, il codice utilizza il messaggio, restituisce informazioni su di esso e poi elimina il messaggio, la coda e il 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
```

Configurazione delle code Amazon SQS tramite la console Amazon SQS

Usa la console Amazon SQS per configurare e gestire le code e le funzionalità di Amazon SQS. Puoi anche:

- Abilita la crittografia lato server per una maggiore sicurezza.
- Associa una coda di lettere non scritte per gestire i messaggi non elaborati.
- Imposta un trigger per richiamare una funzione Lambda per l'elaborazione basata sugli eventi.

Controllo degli accessi basato su attributi per Amazon SQS

Che cos'è ABAC?

Il controllo degli accessi basato sugli attributi (ABAC) è un processo di autorizzazione che definisce le autorizzazioni in base ai tag allegati a utenti e risorse. AWS ABAC fornisce un controllo degli accessi granulare e flessibile basato su attributi e valori, riduce i rischi per la sicurezza legati alle policy riconfigurate basate sui ruoli e centralizza il controllo e la gestione delle policy di accesso. Per ulteriori informazioni su ABAC, consulta [Che cos'è ABAC per AWS](#) nella Guida per l'utente di IAM.

Amazon SQS supporta ABAC consentendoti di controllare l'accesso alle code Amazon SQS in base ai tag e agli alias associati a una coda Amazon SQS. Le chiavi di condizione tag e alias che abilitano ABAC in Amazon SQS autorizzano le entità IAM a utilizzare le code Amazon SQS senza modificare le policy o gestire le concessioni. Per ulteriori informazioni sulle chiavi di condizione ABAC, consulta [Condition keys for Amazon SQS](#) nel Service Authorization Reference.

Con ABAC, puoi utilizzare i tag per configurare le autorizzazioni e le policy di accesso IAM per le tue code Amazon SQS, il che ti aiuta a dimensionare la gestione delle autorizzazioni. Puoi creare un'unica policy di autorizzazioni in IAM utilizzando i tag che aggiungi a ciascun ruolo aziendale, senza dover aggiornare la policy ogni volta che aggiungi una nuova risorsa. Puoi anche allegare tag ai presidi IAM per creare una policy ABAC. Puoi progettare policy ABAC per consentire le operazioni di Amazon SQS quando il tag sul ruolo utente IAM che effettua la chiamata corrisponde al tag di coda di Amazon SQS. Per ulteriori informazioni sul tagging in AWS, consulta [AWS Tagging Strategies and Tag per l'allocazione dei costi Amazon SQS](#)

 Note

ABAC per Amazon SQS è attualmente disponibile in AWS tutte le regioni commerciali in cui è disponibile Amazon SQS, con le seguenti eccezioni:

- Asia Pacifico (Hyderabad)
- Asia Pacifico (Melbourne)
- Europa (Spagna)
- Europa (Zurigo)

Qual è il vantaggio di utilizzare ABAC in Amazon SQS?

Ecco alcuni vantaggi dell'utilizzo di ABAC in Amazon SQS:

- ABAC per Amazon SQS richiede un numero minore di policy di autorizzazione. Non è necessario creare policy diverse per diverse mansioni lavorative. Puoi utilizzare tag di risorse e richieste che si applicano a più di una coda, il che riduce il sovraccarico operativo.
- Usa ABAC per dimensionare rapidamente i team. Le autorizzazioni per le nuove risorse vengono concesse automaticamente in base ai tag quando le risorse vengono etichettate in modo appropriato durante la loro creazione.
- Utilizza le autorizzazioni sul principale IAM per limitare l'accesso alle risorse. Puoi creare tag per il principale IAM e utilizzarli per limitare l'accesso ad azioni specifiche che corrispondono ai tag sul principale IAM. Ciò ti aiuta ad automatizzare il processo di concessione delle autorizzazioni per le richieste.
- Tieni traccia di chi accede alle tue risorse. Puoi determinare l'identità di una sessione esaminando gli attributi utente in AWS CloudTrail.

Argomenti

- [Etichettatura per il controllo degli accessi in Amazon SQS](#)
- [Creazione di utenti IAM e code Amazon SQS](#)
- [Test del controllo degli accessi basato sugli attributi in Amazon SQS](#)

Etichettatura per il controllo degli accessi in Amazon SQS

Di seguito è riportato un esempio di utilizzo dei tag per il controllo degli accessi in Amazon SQS. La policy IAM limita un utente IAM a tutte le azioni Amazon SQS per tutte le code che includono un tag di risorsa con l'ambiente della chiave e la produzione di valore. Per ulteriori informazioni, consulta [Controllo degli accessi basato sugli attributi con tag e Organizations](#). AWS

JSON

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

Creazione di utenti IAM e code Amazon SQS

Gli esempi seguenti spiegano come creare una policy ABAC per controllare l'accesso ad Amazon SQS utilizzando Console di gestione AWS and. CloudFormation

Utilizzando il Console di gestione AWS

Crea un utente IAM

1. Accedi Console di gestione AWS e apri la console IAM all'indirizzo <https://console.aws.amazon.com/iam/>.
2. Seleziona Utenti dal pannello di navigazione sinistro.
3. Scegli Aggiungi utenti e inserisci un nome nella casella di testo Nome utente.

4. Seleziona Chiave di accesso - Accesso programmatico e scegli Next:Permissions.
5. Scegli Successivo: Tag.
6. Aggiungi un tag con la chiave environment e il valore del tag beta.
7. Scegli Next:Review, quindi Crea utente.
8. Copiare e archiviare l'ID chiave di accesso e la chiave di accesso segreta in una posizione sicura.

Aggiungi le autorizzazioni degli utenti IAM

1. Seleziona l'utente IAM creato.
2. Scegli Aggiungi policy in linea.
3. Nella scheda JSON incolla la policy seguente:
4. Scegliere Esamina policy.
5. Scegli Crea policy.

Usando AWS CloudFormation

Utilizza il seguente CloudFormation modello di esempio per creare un utente IAM con una policy in linea allegata e una coda Amazon SQS:

```
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"
```

Test del controllo degli accessi basato sugli attributi in Amazon SQS

Gli esempi seguenti mostrano come testare il controllo degli accessi basato su attributi in Amazon SQS.

Crea una coda con la chiave del tag impostata su environment e il valore del tag impostato su prod

Esegui questo comando AWS CLI per testare la creazione della coda con la chiave tag impostata su environment e il valore del tag impostato su prod. Se non disponi della AWS CLI, puoi [scaricarla e configurarla](#) per la tua macchina.

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

Ricevi un errore AccessDenied dall'endpoint Amazon SQS:

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

Questo perché il valore del tag sull'utente IAM non corrisponde al tag passato nella chiamata [CreateQueue](#) API. Ricorda che abbiamo applicato un tag all'utente IAM con la chiave impostata su environment e il valore impostato su beta.

Crea una coda con la chiave del tag impostata su environment e il valore del tag impostato su beta

Esegui questo comando CLI per testare la creazione della coda con la chiave tag impostata su environment e il valore del tag impostato su beta.

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

Riceverai un messaggio di conferma dell'avvenuta creazione della coda, simile a quello riportato di seguito.

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

Invio di un messaggio a una coda

Esegui questo comando CLI per testare l'invio di un messaggio a una coda.

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

La risposta indica che il messaggio è stato recapitato correttamente alla coda di Amazon SQS. L'autorizzazione utente IAM ti consente di inviare un messaggio a una coda con un tag beta. La risposta include MD5ofMessageBody e MessageId contenente il messaggio.

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

Configurazione dei parametri della coda tramite la console Amazon SQS

Quando si [crea](#) o si [modifica](#) una coda, è possibile configurare i seguenti parametri:

- Timeout di visibilità: il periodo di tempo in cui un messaggio ricevuto da una coda (da un consumatore) non sarà visibile agli altri utenti di messaggi. Per ulteriori informazioni, consulta [Timeout visibilità](#).

Note

L'utilizzo della console per configurare il timeout di visibilità configura il valore di timeout per tutti i messaggi in coda. Per configurare il timeout per uno o più messaggi, è necessario utilizzare uno dei AWS SDKs

- **Periodo di conservazione dei messaggi:** la quantità di tempo in cui Amazon SQS conserva i messaggi che rimangono in coda. Per impostazione predefinita, una coda conserva i messaggi per quattro giorni. È possibile configurare una coda in modo che conservi i messaggi fino a un massimo di 14 giorni. Per altre informazioni, consulta [Periodo di conservazione dei messaggi](#).
- **Ritardo di consegna:** il periodo di tempo che Amazon SQS ritarderà prima di recapitare un messaggio aggiunto alla coda. Per ulteriori informazioni, consulta [Ritardo nella consegna](#).
- **Dimensione massima dei messaggi:** la dimensione massima dei messaggi per questa coda. Per ulteriori informazioni consulta [Dimensione massima dei messaggi](#).
- **Tempo di attesa per la ricezione dei messaggi:** il tempo massimo in cui Amazon SQS attende che i messaggi diventino disponibili dopo che la coda riceve una richiesta di ricezione. Per ulteriori informazioni, consulta [Polling brevi e lunghi di Amazon SQS](#).
- **Abilita la deduplicazione basata sui contenuti:** Amazon SQS può creare automaticamente la deduplicazione in IDs base al corpo del messaggio. Per ulteriori informazioni, consulta [Code FIFO di Amazon SQS](#).
- **Abilita FIFO a velocità di trasmissione effettiva elevata:** da utilizzare per abilitare una velocità di trasmissione effettiva elevata per i messaggi in coda. La scelta di questa opzione modifica le opzioni correlate ([ambito di deduplicazione](#) e [limite di velocità di trasmissione effettiva FIFO](#)) con le impostazioni richieste per abilitare una velocità di trasmissione effettiva elevata per le code FIFO. Per ulteriori informazioni, consultare [Throughput elevato per le code FIFO in Amazon SQS](#) e [Quote di messaggi Amazon SQS](#).
- **Policy redrive allow:** definisce quali code di origine possono utilizzare questa coda come coda DLQ. Per ulteriori informazioni, consulta [Utilizzo di code di lettere non scritte in Amazon SQS](#).

Configurazione dei parametri della coda per una coda esistente (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code). Scegli una coda e seleziona Modifica.
3. Scorri fino alla sezione Configurazione.
4. Per il timeout di visibilità, inserisci la durata e le unità. L'intervallo è compreso tra 0 secondi e 12 ore. Il valore di predefinito è 30 secondi.
5. Per Periodo di conservazione dei messaggi, inserisci la durata e le unità. L'intervallo valido è compreso tra 1 minuto e 14 giorni. Il valore predefinito è 4 giorni.

6. Per una coda standard, inserisci un valore per il tempo di attesa per la ricezione dei messaggi. L'intervallo è compreso tra 0 e 20 secondi. Il valore predefinito è 0 secondi, che imposta uno [short polling](#). Qualsiasi valore diverso da zero imposta un long polling.
7. Per Ritardo di consegna, inserisci la durata e le unità. L'intervallo è compreso tra 30 secondi e 15 minuti. Il valore predefinito è 0 secondi.
8. Per Dimensione massima del messaggio, inserisci un valore. L'intervallo va da 1 KiB a 1024 KiB. Il valore predefinito è 1024 KiB.
9. Per una coda FIFO, scegli Abilita la deduplicazione basata sul contenuto per abilitare la deduplicazione basata sul contenuto. L'impostazione predefinita è disabilitata.
10. (Facoltativo) Affinché una coda FIFO consenta una velocità di trasmissione effettiva più elevata per l'invio e la ricezione di messaggi in coda, seleziona Abilita FIFO ad alta velocità di trasmissione effettiva.

La scelta di questa opzione modifica le opzioni correlate (ambito di deduplicazione e limite di velocità di trasmissione effettiva FIFO) con le impostazioni richieste per abilitare una velocità di trasmissione effettiva elevata per le code FIFO. Se si modifica una delle impostazioni necessarie per utilizzare FIFO ad alta velocità di trasmissione effettiva, si applica la velocità di trasmissione effettiva normale per la coda e la deduplicazione si verifica come specificato. Per ulteriori informazioni, consultare [Throughput elevato per le code FIFO in Amazon SQS](#) e [Quote di messaggi Amazon SQS](#).

11. Per la policy Redrive allow, scegli Abilitata. Seleziona una delle seguenti opzioni: Consenti tutto (impostazione predefinita), Per coda o Nega tutto. Quando scegli Per coda, specifica un elenco di un massimo di 10 code di origine in base al nome della risorsa Amazon (ARN).
12. Al termine della configurazione dei parametri della coda, scegli Salva.

Configurazione di una policy di accesso in Amazon SQS

Quando [modifichi](#) una coda, puoi configurarne la politica di accesso per controllare chi può interagire con essa.

- La politica di accesso definisce quali account, utenti e ruoli dispongono delle autorizzazioni per accedere alla coda.
- Specifica le azioni consentite, ad esempio [SendMessage](#), [ReceiveMessage](#) o [DeleteMessage](#).
- Per impostazione predefinita, solo il proprietario della coda è autorizzato a inviare e ricevere messaggi.

Per configurare la politica di accesso per una coda esistente (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Scegli una coda e seleziona Modifica.
4. Scorri fino alla sezione Policy di accesso.
5. Modifica le dichiarazioni sulla politica di accesso nella casella di input. Per ulteriori informazioni sulle dichiarazioni relative alle policy di accesso, vedi [Identity and Access Management in Amazon SQS](#).
6. Al termine della configurazione della policy di accesso, scegli Salva.

Configurazione della crittografia lato server per una coda utilizzando chiavi di crittografia gestite da SQL

Oltre all'opzione di crittografia lato server gestita (SSE) [predefinita](#) di Amazon SQS, Amazon SQS managed SSE (SSE-SQS) consente di creare una crittografia lato server gestita personalizzata che utilizza chiavi di crittografia gestite da SQS per proteggere i dati sensibili inviati tramite code di messaggi. Con SSE-SQS, non è necessario creare e gestire chiavi di crittografia o modificare il codice per crittografare i dati. SSE-SQS consente di trasmettere i dati in modo sicuro e consente di soddisfare i rigorosi requisiti normativi e di conformità alla crittografia senza costi aggiuntivi.

Per proteggere i dati a riposo, SSE-SQS utilizza la crittografia standard avanzata a 256 bit (AES-256). SSE esegue la crittografia dei messaggi non appena vengono ricevuti da Amazon SQS. Amazon SQS archivia i messaggi in forma crittografata e li decrittografa solo quando li invia a un consumatore autorizzato.

Note

- L'opzione SSE predefinita è efficace solo quando si crea una coda senza specificare gli attributi di crittografia.
- Amazon SQS consente di disattivare la crittografia di tutte le code. Pertanto, la disattivazione di KMS-SSE non abiliterà automaticamente SQS-SSE. Se desideri abilitare SQS-SSE dopo aver disattivato KMS-SSE, devi aggiungere una modifica dell'attributo nella richiesta.

Per configurare la crittografia SSE-SQS per una coda (console)

Note

Qualsiasi nuova coda creata utilizzando l'endpoint HTTP (non TLS) non abiliterà la crittografia SSE-SQS per impostazione predefinita. È una best practice di sicurezza creare code Amazon SQS utilizzando endpoint HTTPS o [Signature Version 4](#).

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Scegli una coda, quindi seleziona Modifica.
4. Espandi crittografia.
5. In Crittografia lato server, scegli Abilita .

Note

Con SSE abilitato, le richieste anonime SendMessage e ReceiveMessage alla coda crittografata verranno rifiutate. Le best practice di sicurezza di Amazon SQS consigliano di non utilizzare richieste anonime. Se desideri inviare richieste anonime a una coda Amazon SQS, assicurati di disabilitare SSE.

6. Seleziona la chiave Amazon SQS (SSE-SQS). L'utilizzo di questa opzione non prevede costi aggiuntivi.
7. Scegli Save (Salva).

Configurazione della crittografia lato server per una coda utilizzando la console Amazon SQS

Per proteggere i dati nei messaggi di una coda, Amazon SQS dispone di crittografia lato server (SSE) abilitata di default per tutte le code appena create. Amazon SQS si integra con Amazon Web Services Key Management Service (Amazon Web Services KMS) per gestire le [chiavi KMS](#) per la crittografia lato server (SSE). Per ulteriori informazioni sull'uso di SSE, consultare [Crittografia inattiva in Amazon SQS](#).

La chiave KMS assegnata alla coda deve avere una policy della chiave che includa le autorizzazioni per tutti i principali autorizzati a utilizzare la coda. Per ulteriori informazioni, consulta [Gestione delle chiavi](#).

Se non sei il proprietario della chiave KMS, oppure se effettui l'accesso con un account che non dispone delle autorizzazioni `kms:ListAliases` e `kms:DescribeKey`, non puoi visualizzare le informazioni sulla chiave KMS nella console Amazon SNS. Chiedi al proprietario della chiave KMS di concederti queste autorizzazioni. Per ulteriori informazioni, consulta [Gestione delle chiavi](#).

Quando si [crea](#) o si [modifica](#) una coda, è possibile configurare SSE-KMS.

Per configurare SSE-KMS per una coda esistente (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Scegli una coda, quindi seleziona Modifica.
4. Espandi crittografia.
5. In Crittografia lato server, scegli Abilita .

 Note

Con SSE abilitato, le richieste anonime `SendMessage` e `ReceiveMessage` alla coda crittografata verranno rifiutate. Le best practice di sicurezza di Amazon SQS consigliano di non utilizzare richieste anonime. Se desideri inviare richieste anonime a una coda Amazon SQS, assicurati di disabilitare SSE.

6. Seleziona l'opzione Chiave del servizio di gestione delle chiavi AWS (SSE-KMS).

La console visualizza la descrizione, l'account e l'ARN della chiave KMS della chiave KMS.

7. Specifica l'ID della chiave KMS per la coda. Per ulteriori informazioni, consulta [Termini chiave](#).
 - a. Scegli l'opzione Scegli un alias chiave KMS.
 - b. La chiave predefinita è la chiave KMS gestita di Amazon Web Services per Amazon SQS. Per utilizzare questa chiave, selezionala dall'elenco delle chiavi KMS.
 - c. Per utilizzare una chiave KMS personalizzata dal tuo account Amazon Web Services, selezionala dall'elenco delle chiavi KMS. Per istruzioni sulla creazione di chiavi KMS personalizzate, consulta [Creazione delle chiavi](#) nella Guida per sviluppatori di Amazon Web Services Key Management Service.

- d. Per utilizzare una chiave KMS personalizzata non presente nell'elenco o una chiave KMS personalizzata di un altro account Amazon Web Services, scegli Inserisci l'alias della chiave KMS e inserisci il nome della risorsa Amazon (ARN) della chiave KMS.
8. (Facoltativo) Per il periodo di riutilizzo della chiave dati, specifica un valore compreso tra 1 minuto e 24 ore. Il valore predefinito è 5 minuti. Per ulteriori informazioni, consulta [Informazioni sul periodo di riutilizzo della chiave di dati](#).
9. Al termine della configurazione di SSE-KMS, scegli Salva.

Configurazione dei tag di allocazione dei costi per una coda utilizzando la console Amazon SQS

Per organizzare e identificare le code di Amazon SQS, puoi aggiungere tag di allocazione dei costi. Per ulteriori informazioni, consulta [Tag per l'allocazione dei costi Amazon SQS](#).

- La scheda Tagging nella pagina Dettagli mostra i tag della coda.
- È possibile aggiungere o modificare tag durante la [creazione o la modifica di una coda](#).

Per configurare i tag per una coda esistente (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Scegli una coda e seleziona Modifica.
4. Scorri fino alla sezione Tag.
5. Aggiungere, modificare o rimuovere i tag della coda:
 - a. Per aggiungere un tag, scegli Aggiungi nuovo tag, inserire una Chiave e un Valore, quindi scegli Applica modifiche.
 - b. Per aggiornare un tag, modificane la chiave e il valore.
 - c. Per rimuovere un tag, scegli Rimuovi tag accanto a una coppia chiave-valore.
6. Al termine della configurazione dei tag, scegli Salva.

Sottoscrizione di una coda a un argomento di Amazon SNS utilizzando la console Amazon SQS

Puoi sottoscrivere una o più code Amazon SQS a un argomento di Amazon SNS. Quando pubblichi un messaggio su un argomento, Amazon SNS invia il messaggio a ogni coda sottoscritta. Amazon SQS gestisce l'abbonamento e gestisce le autorizzazioni richieste. Per ulteriori informazioni su Amazon SNS, consulta [Che cos'è Amazon SNS?](#) nella Guida per gli sviluppatori di Amazon Simple Notification Service.

Quando sottoscrivi una coda Amazon SQS a un argomento Amazon SNS, Amazon SNS utilizza HTTPS per inoltrare messaggi ad Amazon SQS. Per ulteriori informazioni sull'uso di Amazon SNS con code Amazon SQS crittografate, consulta [AWS Configura le autorizzazioni KMS per i servizi](#).

Important

Amazon SQS supporta un massimo di 20 istruzioni per ogni policy di accesso. La sottoscrizione a un argomento Amazon SNS aggiunge una di queste istruzioni. Il superamento di tale importo comporterà la mancata consegna dell'abbonamento all'argomento.

Per iscrivere una coda a un argomento di Amazon SNS (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Dall'elenco di code, scegliere la coda per iscriversi all'argomento Amazon SNS.
4. Dal menu Operazioni scegliere Subscribe to topic (Sottoscrivi argomento).
5. Dal menu Specificare un argomento Amazon SNS disponibile per questa coda, scegli l'argomento Amazon SNS per la tua coda.

Se l'argomento SNS non è elencato, scegli Inserisci l'argomento Amazon SNS ARN, quindi inserisci il nome Amazon Resource Name (ARN) dell'argomento.

6. Scegli Save (Salva).
7. Per verificare l'abbonamento, pubblica un messaggio sull'argomento e visualizza il messaggio in coda. Per ulteriori informazioni, consulta [Pubblicazione dei messaggi Amazon SNS](#) nella Guida per gli sviluppatori di Amazon Simple Notification Service.

Abbonamenti per più account

Se la tua coda Amazon SQS e l'argomento Amazon SNS sono diversi Account AWS, sono necessarie autorizzazioni aggiuntive.

Titolare dell'argomento (account A)

Modifica la politica di accesso dell'argomento Amazon SNS per consentire alle code Account AWS Amazon SQS di iscriversi. Esempio di dichiarazione politica:

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

Questa politica consente 111122223333 all'account di iscriversi aMyTopic.

Proprietario della coda (Account B)

Modifica la politica di accesso della coda Amazon SQS per consentire all'argomento Amazon SNS di inviare messaggi. Esempio di dichiarazione politica:

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

Questa politica consente MyTopic di inviare messaggi aMyQueue.

Abbonamenti in più regioni

Per iscriverti a un argomento di Amazon SNS su un argomento diverso Regione AWS, assicurati che:

- La politica di accesso dell'argomento Amazon SNS consente abbonamenti tra regioni.

- La policy di accesso della coda Amazon SQS consente all'argomento Amazon SNS di inviare messaggi tra regioni.

Per ulteriori informazioni, consulta la [sezione Invio di messaggi Amazon SNS a una coda AWS Lambda o funzione Amazon SQS in un'altra regione nella](#) Amazon Simple Notification Service Developer Guide.

Configurazione di una coda Amazon SQS per attivare una funzione AWS Lambda

Puoi utilizzare una funzione Lambda per elaborare i messaggi da una coda Amazon SQS. Lambda analizza la coda e richiama la funzione in modo sincrono, passando un batch di messaggi come evento.

Configurazione del timeout di visibilità

[Imposta il timeout di visibilità della coda su almeno sei volte il timeout della funzione.](#) In questo modo Lambda ha abbastanza tempo per riprovare se una funzione viene limitata durante l'elaborazione di un batch precedente.

Utilizzo di una coda di lettere morte (DLQ)

Specificate una coda di lettere morte per acquisire i messaggi che la funzione Lambda non riesce a elaborare.

Gestione di più code e funzioni

Una funzione Lambda può elaborare più code creando un'origine eventi separata per ogni coda. Puoi anche associare più funzioni Lambda alla stessa coda.

Autorizzazioni per le code crittografate

Se associ una coda crittografata a una funzione Lambda, ma Lambda non esegue il polling dei messaggi, aggiungi l'autorizzazione `kms:Decrypt` al tuo ruolo di esecuzione Lambda.

Restrizioni

La coda e la funzione Lambda devono essere uguali. Regione AWS

Una [coda crittografata](#) che utilizza la chiave predefinita (chiave KMS AWS gestita per Amazon SQS) non può richiamare una funzione Lambda in un'altra. Account AWS

Per i dettagli sull'implementazione, consulta [Using AWS Lambda with Amazon SQS](#) nella AWS Lambda Developer Guide.

Prerequisiti

Per configurare i trigger delle funzioni Lambda, devi soddisfare i seguenti requisiti:

- Se usi un utente, il ruolo Amazon SQS deve includere le seguenti autorizzazioni:
 - `lambda:CreateEventSourceMapping`
 - `lambda:ListEventSourceMappings`
 - `lambda:ListFunctions`
- Il ruolo di esecuzione Lambda deve includere le seguenti autorizzazioni:
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `sqs:ReceiveMessage`
- Se associ una coda crittografata a una funzione Lambda, aggiungi l'autorizzazione `kms:Decrypt` al ruolo di esecuzione Lambda.

Per ulteriori informazioni, consulta [Panoramica sulla gestione degli accessi in Amazon SQS](#).

Per configurare una coda per attivare una funzione Lambda (console)

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Nella pagina Code, scegliere la coda da configurare.
4. Nella pagina della coda, scegli la scheda Trigger Lambda.
5. Nella pagina Trigger Lambda, scegli un trigger Lambda.

Se l'elenco non include il trigger Lambda di cui hai bisogno, scegli Configura il trigger della funzione Lambda. Inserisci il nome della risorsa Amazon (ARN) della funzione Lambda o scegli una risorsa esistente. Quindi scegli Save (Salva).

6. Scegli Save (Salva). La console salva la configurazione e visualizza la pagina Dettagli per la coda.

Nella pagina Dettagli, la scheda Trigger Lambda mostra la funzione Lambda e il relativo stato. Per associare la funzione Lambda alla coda è richiesto circa 1 minuto.

7. Per verificare i risultati della configurazione, è possibile [inviare un messaggio alla coda](#), quindi visualizzare la funzione Lambda attivata nella console Lambda.

Automazione delle notifiche dai AWS servizi ad Amazon SQS tramite Amazon EventBridge

Amazon ti EventBridge consente di automatizzare Servizi AWS e rispondere a eventi, come problemi delle applicazioni o modifiche delle risorse, quasi in tempo reale.

- Puoi creare regole per filtrare eventi specifici e definire azioni automatizzate quando una regola corrisponde a un evento.
- EventBridge supporta più destinazioni, tra cui le code Amazon SQS standard e FIFO, che ricevono eventi in formato JSON.

Per ulteriori informazioni, consulta [Amazon EventBridge targets](#) nella [Amazon EventBridge User Guide](#).

Invio di un messaggio con attributi tramite Amazon SQS

Per le code standard e FIFO, puoi includere nei messaggi metadati strutturati, tra cui timestamp, dati geospaziali, firme e identificatori. Per ulteriori informazioni, consulta [Attributi messaggio di Amazon SQS](#).

Per inviare un messaggio con attributi a una coda utilizzando la console Amazon SQS

1. Apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Nel riquadro di navigazione, scegliere Code (Code).
3. Nella pagina Code, scegliere una coda.
4. Scegli Invia e ricevi messaggi.
5. Inserire i parametri dell'attributo del messaggio.
 - a. Nella casella di testo nome, inserire un nome univoco di massimo 256 caratteri.
 - b. Per il tipo di attributo, scegliete Stringa, Numero o Binario.
 - c. (Facoltativo) Immettere un tipo di dati personalizzato. Ad esempio, puoi aggiungere **byte**, **int** o **float** come tipo di dati personalizzato per Numero.

- d. Nella casella di testo del valore, inserire il valore dell'attributo messaggio.

The screenshot shows a dialog box titled "Message attributes - Optional" with an "Info" link. Inside, there is a form for adding a new attribute. It consists of four input fields: "Enter name", a dropdown menu currently set to "String", "Custom type", and "Enter value". Below these fields is a button labeled "Add new attribute".

6. Per aggiungere un altro attributo, scegliere Aggiungi nuovo attributo.

This screenshot shows the same "Message attributes" dialog box, but now it contains two attribute entries. Each entry has its own set of four input fields: "Enter name", a dropdown menu set to "String", "Custom type", and "Enter value". To the right of the second entry is a "Remove" button. At the bottom left of the dialog is the "Add new attribute" button.

7. È possibile modificare i valori dell'attributo prima di inviare il messaggio.
8. Per eliminare un attributo, scegliere Rimuovi. Per eliminare il primo attributo, chiudere Attributi del messaggio.
9. Una volta terminato di aggiungere gli attributi al messaggio, scegli Invia messaggio. Quando il messaggio viene inviato, la console ne visualizza la conferma. Per visualizzare informazioni sugli attributi del messaggio inviato, scegli Visualizza dettagli. Scegli Fine per chiudere la finestra di dialogo dei dettagli del messaggio.

Best practice di Amazon SQS

Amazon SQS gestisce ed elabora le code di messaggi, consentendo a diverse parti di un'applicazione di scambiare messaggi in modo affidabile e su larga scala. Questo argomento illustra le migliori pratiche operative chiave, tra cui l'uso di sondaggi lunghi per ridurre le risposte vuote, l'implementazione di code di lettere scadenti per gestire gli errori di elaborazione e l'ottimizzazione delle autorizzazioni di coda per motivi di sicurezza.

Argomenti

- [Gestione degli errori e messaggi problematici di Amazon SQS](#)
- [Deduplicazione e raggruppamento dei messaggi Amazon SQS](#)
- [Elaborazione e tempistica dei messaggi di Amazon SQS](#)

Gestione degli errori e messaggi problematici di Amazon SQS

Questo argomento fornisce istruzioni dettagliate sulla gestione e la mitigazione degli errori in Amazon SQS, comprese le tecniche per gestire gli errori di richiesta, l'acquisizione di messaggi problematici e la configurazione della conservazione delle code di lettera morta per garantire l'affidabilità dei messaggi.

Argomenti

- [Gestione degli errori di richiesta in Amazon SQS](#)
- [Acquisizione di messaggi problematici in Amazon SQS](#)
- [Configurazione della conservazione della coda di lettere non scritte in Amazon SQS](#)

Gestione degli errori di richiesta in Amazon SQS

Per gestire gli errori di richiesta, utilizza una delle seguenti strategie:

- Se utilizzi un AWS SDK, hai già a disposizione la logica automatica di riprova e backoff. Per ulteriori informazioni, consulta [Ripetizione dei tentativi in caso di errore e backoff esponenziale in AWS](#) nella Riferimenti generali di Amazon Web Services.
- Se non utilizzi le funzionalità AWS SDK per riprovare e disattivare, concedi una pausa (ad esempio, 200 ms) prima di riprovare l'[ReceiveMessage](#) azione dopo non aver ricevuto alcun

messaggio, un timeout o un messaggio di errore da Amazon SQS. Per il successivo uso di `ReceiveMessage` che offre gli stessi risultati, lascia un po' più di tempo (ad esempio, 400 ms).

Acquisizione di messaggi problematici in Amazon SQS

[Per acquisire tutti i messaggi che non possono essere elaborati e per raccogliere CloudWatch metriche accurate, configura una coda di lettere non scritte.](#)

- La policy di reindirizzamento reindirizza i messaggi a una coda DLQ dopo che la coda origine non riesce a elaborare un messaggio per un determinato numero di volte.
- L'utilizzo di una coda DLQ riduce il numero di messaggi e la possibilità di esporti a messaggi poison pill (messaggi che possono essere ricevuti ma non elaborati).
- L'inclusione di un messaggio relativo alla pillola avvelenata in una coda può alterare la [ApproximateAgeOfOldestMessage](#) CloudWatch metrica indicando un'età errata del messaggio relativo alla pillola avvelenata. La configurazione di una coda DLQ consente di evitare falsi allarmi durante l'utilizzo di questo parametro.

Configurazione della conservazione della coda di lettere non scritte in Amazon SQS

Per le code standard, la scadenza di un messaggio si basa sempre sul timestamp della coda originale. Quando un messaggio viene spostato in una coda DLQ, il timestamp della coda rimane invariato. La metrica `ApproximateAgeOfOldestMessage` indica quando il messaggio è stato spostato nella coda DLQ, non quando è stato originariamente inviato. Ad esempio, supponiamo che un messaggio trascorra 1 giorno nella coda originale prima di essere spostato in una coda DLQ. Se il periodo di conservazione della coda DLQ è di 4 giorni, il messaggio viene eliminato dalla coda DLQ dopo 3 giorni e `ApproximateAgeOfOldestMessage` è di tre giorni. Pertanto, è consigliabile impostare sempre il periodo di conservazione di una coda DLQ in modo che sia più lungo del periodo di conservazione della coda originale.

Per le code FIFO, il timestamp della coda si reimposta quando il messaggio viene spostato in una coda DLQ. La metrica `ApproximateAgeOfOldestMessage` indica quando il messaggio è stato spostato nella coda DLQ. Nello stesso esempio precedente, il messaggio viene eliminato dalla coda DLQ dopo 4 giorni e `ApproximateAgeOfOldestMessage` è 4 giorni.

Deduplicazione e raggruppamento dei messaggi Amazon SQS

Questo argomento fornisce le best practice per garantire un'elaborazione coerente dei messaggi in Amazon SQS. Spiega come usare:

- [MessageDeduplicationId](#) per evitare la duplicazione dei messaggi nelle code FIFO.
- [MessageGroupId](#) per gestire l'ordine dei messaggi all'interno di gruppi di messaggi distinti.

Argomenti

- [Evitare l'elaborazione incoerente dei messaggi in Amazon SQS](#)
- [Utilizzo dell'ID di deduplicazione dei messaggi in Amazon SQS](#)
- [Utilizzo dell'ID del gruppo di messaggi con Amazon SQS FIFO Queues](#)
- [Utilizzo dell'ID tentativo richiesta di ricezione Amazon SQS](#)

Evitare l'elaborazione incoerente dei messaggi in Amazon SQS

Poiché Amazon SQS è un sistema distribuito, è possibile per un consumatore non ricevere un messaggio anche quando Amazon SQS contrassegna il messaggio come recapitato mentre restituisce correttamente da una chiamata al metodo API [ReceiveMessage](#). In questo caso, Amazon SQS registra il messaggio come consegnato almeno una volta, anche se il consumatore non lo ha mai ricevuto. Poiché non vengono effettuati ulteriori tentativi di recapito dei messaggi in queste condizioni, si sconsiglia di impostare il numero massimo di ricevute su 1 per una [dead letter queue](#).

Utilizzo dell'ID di deduplicazione dei messaggi in Amazon SQS

[MessageDeduplicationId](#) è un token utilizzato solo nelle code FIFO di Amazon SQS per impedire la consegna di messaggi duplicati. Garantisce che entro una finestra di deduplicazione di 5 minuti, venga elaborata e recapitata solo un'istanza di messaggio con lo stesso ID di deduplicazione.

Se Amazon SQS ha già accettato un messaggio con un ID di deduplicazione specifico, tutti i messaggi successivi con lo stesso ID verranno riconosciuti ma non consegnati ai consumatori.

Note

Amazon SQS continua a tracciare l'ID di deduplicazione anche dopo la ricezione e l'eliminazione del messaggio.

Argomenti

- [Quando fornire un ID di deduplicazione dei messaggi in Amazon SQS](#)
- [Abilitazione della deduplicazione per un sistema a singolo produttore/consumatore in Amazon SQS](#)
- [Scenari di ripristino delle interruzioni in Amazon SQS](#)
- [Configurazione dei timeout di visibilità in Amazon SQS](#)

Quando fornire un ID di deduplicazione dei messaggi in Amazon SQS

Un produttore deve specificare un ID di deduplicazione dei messaggi nei seguenti scenari:

- Quando si inviano corpi di messaggi identici, questi devono essere trattati come unici.
- Quando si inviano messaggi con lo stesso contenuto ma attributi di messaggio diversi, assicuratevi che ogni messaggio venga elaborato separatamente.
- Quando si inviano messaggi con contenuti diversi (ad esempio, un contatore di tentativi nel corpo del messaggio) ma è necessario che Amazon SQS li riconosca come duplicati.

Abilitazione della deduplicazione per un sistema a singolo produttore/consumatore in Amazon SQS

Se hai un solo produttore e un solo consumatore e i messaggi sono unici perché includono un ID del messaggio specifico dell'applicazione nel corpo, segui queste procedure consigliate:

- Abilita la funzionalità di deduplicazione basata sui contenuti per la coda (ciascun messaggio ha un corpo specifico). Il produttore può omettere l'ID di deduplicazione messaggio.
- Quando la deduplicazione basata sul contenuto è abilitata per una coda FIFO di Amazon SQS e viene inviato un messaggio con un ID di deduplicazione, l'ID di deduplicazione sostituisce l'ID di deduplicazione basato sul [SendMessage](#) contenuto generato.
- Anche se il consumatore non è tenuto a fornire un ID del tentativo di richiesta di ricezione per ogni richiesta, è una best practice perché consente un'esecuzione più veloce delle sequenze ritentate dopo esito negativo.
- Puoi riprovare a inviare o ricevere richieste perché non interferiscono con l'ordinazione di messaggi nelle code FIFO.

Scenari di ripristino delle interruzioni in Amazon SQS

Il processo di deduplicazione nelle code FIFO è legata al fattore tempo. Durante la progettazione dell'applicazione, assicuratevi che sia il produttore che il consumatore possano riprendersi dalle interruzioni del client o della rete senza introdurre duplicati o errori di elaborazione.

Considerazioni sul produttore

- Amazon SQS applica una finestra di deduplicazione di 5 minuti.
- Se un produttore riprova una [SendMessage](#) richiesta dopo 5 minuti, Amazon SQS la tratta come un nuovo messaggio, creando potenzialmente dei duplicati.

Considerazioni relative ai consumatori

- Se un consumatore non riesce a elaborare un messaggio prima della scadenza del timeout di visibilità, un altro consumatore potrebbe riceverlo ed elaborarlo, con conseguente elaborazione duplicata.
- Regola il timeout di visibilità in base al tempo di elaborazione dell'applicazione.
- Utilizza l'[ChangeMessageVisibility](#) API per prolungare il timeout mentre un messaggio è ancora in fase di elaborazione.
- Se un messaggio non viene elaborato ripetutamente, indirizzalo a una [coda di lettere morte \(DLQ\)](#) invece di consentirne la rielaborazione all'infinito.
- Il produttore deve essere a conoscenza dell'intervallo di deduplicazione della coda. Amazon SQS dispone di un intervallo di deduplicazione minimo di 5 minuti. La riesecuzione di richieste `SendMessage` dopo la scadenza dell'intervallo di deduplicazione può introdurre messaggi duplicati in coda. Ad esempio, un dispositivo mobile in un'auto invia messaggi il cui ordine è importante. Se l'automobile perde connettività cellulare per un periodo di tempo prima di ricevere una conferma, ritentare la richiesta dopo il ritorno della connettività del cellulare può portare alla creazione di un duplicato.
- Il consumatore deve avere un timeout visibilità, che riduce al minimo il rischio di impossibilità di elaborare i messaggi di timeout visibilità prima della scadenza. È possibile estendere la visibilità timeout mentre i messaggi vengono elaborati richiamando l'azione `ChangeMessageVisibility`. Tuttavia, se il timeout visibilità scade, un altro consumatore può immediatamente iniziare a elaborare i messaggi, causando l'elaborazione di un messaggio più volte. Per evitare questo scenario, configura una [coda DLQ](#).

Configurazione dei timeout di visibilità in Amazon SQS

Per garantire un'elaborazione affidabile dei messaggi, imposta il timeout di visibilità in modo che sia più lungo del timeout di lettura dell' AWS SDK. Questo vale quando si utilizza l'[ReceiveMessageAPI](#) sia con polling breve che con polling lungo. Un timeout di visibilità più lungo impedisce che i messaggi diventino disponibili per altri consumatori prima del completamento della richiesta originale, riducendo il rischio di un'elaborazione duplicata.

Utilizzo dell'ID del gruppo di messaggi con Amazon SQS FIFO Queues

Nelle code FIFO (First-In-First-Out), [MessageGroupId](#) è un attributo che organizza i messaggi in gruppi distinti. I messaggi all'interno dello stesso gruppo di messaggi vengono sempre elaborati uno alla volta, in ordine rigoroso, assicurando che non vengano elaborati due messaggi dello stesso gruppo contemporaneamente. Nelle code standard, l'utilizzo MessageGroupId consente [code equie](#). Se è richiesto un ordine rigoroso, utilizza una coda FIFO.

Argomenti

- [Interconnessione di più gruppi di messaggi ordinati in Amazon SQS](#)
- [Prevenzione dell'elaborazione duplicata in un sistema con più produttori/consumatori in Amazon SQS](#)
- [Evita grandi arretrati di messaggi con lo stesso ID di gruppo di messaggi in Amazon SQS](#)
- [Evita di riutilizzare lo stesso ID di gruppo di messaggi con code virtuali in Amazon SQS](#)

Interconnessione di più gruppi di messaggi ordinati in Amazon SQS

Per inserire più gruppi di messaggi ordinati all'interno di una singola coda FIFO, [MessageGroupId](#) assegna un gruppo a ciascun gruppo (ad esempio, dati di sessione per utenti diversi). Ciò consente a più utenti di leggere dalla coda contemporaneamente, garantendo al contempo che i messaggi all'interno dello stesso gruppo vengano elaborati in ordine.

Quando un messaggio con uno specifico messaggio MessageGroupId viene elaborato ed è invisibile, nessun altro consumatore può elaborare i messaggi dello stesso gruppo fino alla scadenza del timeout di visibilità o all'eliminazione del messaggio.

Prevenzione dell'elaborazione duplicata in un sistema con più produttori/consumatori in Amazon SQS

In un sistema ad alta velocità e bassa latenza in cui l'ordine dei messaggi non è una priorità, i produttori possono assegnare un messaggio univoco a ciascun messaggio. [MessageGroupId](#) Ciò garantisce che le code FIFO di Amazon SQS eliminino i duplicati, anche in una configurazione con più produttori/più consumatori. Sebbene questo approccio impedisca la duplicazione dei messaggi, non garantisce l'ordinamento dei messaggi poiché ogni messaggio viene trattato come un gruppo indipendente.

In qualsiasi sistema con più produttori e consumatori, c'è sempre il rischio di una doppia consegna. Se un consumatore non riesce a elaborare un messaggio prima della scadenza del timeout di visibilità, Amazon SQS lo rende nuovamente disponibile, permettendo potenzialmente a un altro consumatore di riceverlo. Per mitigare questo problema, assicurati di impostare correttamente il riconoscimento dei messaggi e il timeout di visibilità in base al tempo di elaborazione.

Evita grandi arretrati di messaggi con lo stesso ID di gruppo di messaggi in Amazon SQS

Le code FIFO supportano un massimo di 120.000 messaggi in volo (messaggi ricevuti da un consumatore ma non ancora eliminati). Se viene raggiunto questo limite, Amazon SQS non restituisce alcun errore, ma l'elaborazione potrebbe risentirne. Puoi richiedere un aumento oltre questo limite contattando [AWS il Supporto](#).

Le code FIFO analizzano i primi 120.000 messaggi per determinare i gruppi di messaggi disponibili. Se si accumula un backlog di grandi dimensioni in un singolo gruppo di messaggi, i messaggi di altri gruppi inviati successivamente rimarranno bloccati fino all'elaborazione del backlog.

Note

Un backlog di messaggi può verificarsi quando un consumatore non riesce ripetutamente a elaborare un messaggio. Ciò potrebbe essere dovuto a problemi relativi al contenuto dei messaggi o a errori sul lato consumatore. Per evitare ritardi nell'elaborazione dei messaggi, configura una [coda di lettere morte](#) per spostare i messaggi non elaborati dopo più tentativi falliti. Ciò garantisce la possibilità di elaborare altri messaggi nello stesso gruppo di messaggi, evitando rallentamenti del sistema.

Evita di riutilizzare lo stesso ID di gruppo di messaggi con code virtuali in Amazon SQS

Quando utilizzi code virtuali con una coda di host condivisa, evita di riutilizzarle su code virtuali [MessageGroupId](#) diverse. Se più code virtuali condividono la stessa coda host e contengono messaggi con la stessa coda `MessageGroupId`, tali messaggi possono bloccarsi a vicenda, impedendo un'elaborazione efficiente. Per garantire un'elaborazione fluida dei messaggi, assegna `MessageGroupId` valori univoci ai messaggi in diverse code virtuali.

Utilizzo dell'ID tentativo richiesta di ricezione Amazon SQS

L'ID del tentativo di ricezione è un token univoco utilizzato per deduplicare [ReceiveMessage](#) le chiamate in Amazon SQS. Durante un'interruzione della rete o un problema di connettività tra l'applicazione e Amazon SQS, è consigliabile:

- Fornisci un ID del tentativo di richiesta di ricezione quando effettui una `ReceiveMessage` chiamata.
- Riprova a utilizzare lo stesso ID del tentativo di richiesta di ricezione se l'operazione non riesce.

Elaborazione e tempistica dei messaggi di Amazon SQS

Questo argomento fornisce una guida completa sull'ottimizzazione della velocità e dell'efficienza della gestione dei messaggi in Amazon SQS, comprese le strategie per l'elaborazione tempestiva dei messaggi, la selezione della migliore modalità di polling e la configurazione di polling lunghi per migliorare le prestazioni.

Argomenti

- [Elaborazione tempestiva dei messaggi in Amazon SQS](#)
- [Configurazione di sondaggi lunghi in Amazon SQS](#)
- [Utilizzo della modalità di polling appropriata in Amazon SQS](#)

Elaborazione tempestiva dei messaggi in Amazon SQS

Le impostazioni del timeout visibilità dipendono da quanto tempo richiede la tua applicazione per elaborare ed eliminare un messaggio. Ad esempio, se la tua applicazione richiede 10 secondi per elaborare un messaggio e imposti il timeout visibilità su 15 minuti, è necessario attendere un tempo relativamente lungo per tentare di elaborare il messaggio di nuovo se il precedente tentativo di elaborazione ha esito negativo. In alternativa, se la tua applicazione richiede 10 secondi per

elaborare un messaggio, ma imposti il timeout visibilità su solo 2 secondi, un duplicato del messaggio verrà ricevuto da un altro consumatore mentre il consumatore originale sta ancora lavorando sul messaggio.

Per garantire il tempo sufficiente per elaborare un messaggio, utilizza una delle seguenti strategie:

- Se sai (o puoi ragionevolmente stimare) la quantità di tempo necessaria per elaborare un messaggio, estendi il timeout visibilità del messaggio per il tempo massimo richiesto per elaborare ed eliminare il messaggio. Per ulteriori informazioni, consulta [Configurazione del timeout di visibilità](#).
- Se non sai quanto tempo sia necessario per elaborare un messaggio, crea un heartbeat per il processo del consumatore: specifica il timeout visibilità iniziale (ad esempio, 2 minuti) e poi, se il consumatore lavora ancora sul messaggio, continua a estendere il timeout visibilità di 2 minuti ogni minuto.

Important

Il timeout massimo di visibilità è di 12 ore dal momento in cui Amazon SQS riceve la richiesta `ReceiveMessage`. L'estensione del timeout di visibilità non reimposta il massimo di 12 ore.

Inoltre, potresti non essere in grado di impostare il timeout per un singolo messaggio per tutte le 12 ore (ad esempio 43.200 secondi) trascorse dalla richiesta `ReceiveMessage` che avvia il timer. Ad esempio, se si riceve un messaggio e si imposta immediatamente il limite massimo di 12 ore inviando una chiamata `ChangeMessageVisibility` della durata di 43.200 secondi, è probabile che la chiamata abbia esito negativo. Tuttavia, l'utilizzo di un valore di 43.195 secondi funzionerà a meno che non vi sia un ritardo significativo tra la richiesta del messaggio tramite `ReceiveMessage` e l'aggiornamento del timeout di visibilità. Se il consumatore ha bisogno di più di 12 ore, prendere in considerazione l'utilizzo di Step Functions.

Configurazione di sondaggi lunghi in Amazon SQS

Quando il tempo di attesa per l'azione dell'API [ReceiveMessage](#) è superiore a 0, viene attivato un polling lungo. Il tempo massimo di attesa per il polling lungo è di 20 secondi. Il polling prolungato aiuta a ridurre i costi di utilizzo di Amazon SQS riducendo il numero di risposte vuote (quando non ci sono messaggi disponibili per `ReceiveMessage` una richiesta) e di false risposte vuote (quando

i messaggi sono disponibili ma non sono inclusi in una risposta). Per ulteriori informazioni, consulta [Polling brevi e lunghi di Amazon SQS](#).

Per garantire l'elaborazione ottimale del messaggio, utilizza le seguenti strategie:

- Nella maggior parte dei casi, è possibile impostare il tempo di attesa `ReceiveMessage` su 20 secondi. Se 20 secondi è troppo lungo per la tua applicazione, imposta un tempo di attesa `ReceiveMessage` inferiore (minimo 1 secondo). Se non utilizzi un AWS SDK per accedere ad Amazon SQS o se configuri AWS un SDK per avere un tempo di attesa più breve, potresti dover modificare il tuo client Amazon SQS per consentire richieste più lunghe o utilizzare un tempo di attesa più breve per lunghi polling.
- Se implementi il long polling per più code, utilizza un thread per ogni coda invece di un singolo thread per tutte le code. L'utilizzo di un singolo thread per ogni coda consente alla tua applicazione di elaborare i messaggi in ognuna delle code non appena sono disponibili, mentre l'utilizzo di un singolo thread per il polling di più code potrebbe causare la non disponibilità della tua applicazione per elaborare i messaggi disponibili in altre code, mentre l'applicazione attende (fino a 20 secondi) la coda che non ha messaggi disponibili.

Important

Per evitare errori HTTP, assicurarsi che il timeout della risposta HTTP per le richieste `ReceiveMessage` sia più lungo del parametro `WaitTimeSeconds`. Per ulteriori informazioni, consulta [ReceiveMessage](#).

Utilizzo della modalità di polling appropriata in Amazon SQS

- Il long polling consente di utilizzare messaggi dalla coda Amazon SQS non appena diventano disponibili.
 - Per ridurre il costo di utilizzo di Amazon SQS e ridurre il numero di ricezioni vuote in una coda vuota (risposte all'operazione `ReceiveMessage` che non restituisce messaggi), abilita il long polling. Per ulteriori informazioni, consulta [Long Polling Amazon SQS](#).
 - Per incrementare l'efficienza durante il polling per più thread con più ricezioni, riduci il numero di thread.
- Il polling lungo è preferibile al polling breve nella maggior parte dei casi.

- Lo short polling restituisce risposte immediatamente, anche se la coda Amazon SQS di cui è stato eseguito il polling è vuota.
- Per soddisfare i requisiti di un'applicazione che richiede risposte immediate per la richiesta `ReceiveMessage`, utilizza il polling a breve.
- Il polling breve viene fatturato allo stesso costo del polling lungo.

Esempi di SDK Java di Amazon SQS

Ti AWS SDK per Java consente di creare applicazioni Java che interagiscono con Amazon SQS e altri. Servizi AWS

- Per installare e configurare l'SDK, consulta la [Guida introduttiva](#) nella Guida dello sviluppatore di AWS SDK for Java 2.x .
- Per le operazioni di coda di base, come la creazione di una coda o l'invio di un messaggio, consulta [Working with Amazon SQS](#) Message Queues nella Developer Guide.AWS SDK for Java 2.x
- Questa guida include anche esempi di funzionalità aggiuntive di Amazon SQS, come:
 - [Utilizzo della crittografia lato server con le code Amazon SQS](#)
 - [Configurazione dei tag per una coda Amazon SQS](#)
 - [Invio degli attributi dei messaggi a una coda Amazon SQS](#)

Utilizzo della crittografia lato server con le code Amazon SQS

Utilizza AWS SDK per Java per aggiungere la crittografia lato server (SSE) a una coda Amazon SQS. Ogni coda utilizza una chiave AWS Key Management Service (AWS KMS) KMS per generare le chiavi di crittografia dei dati. Questo esempio utilizza la chiave KMS gestita da AWS per Amazon SQS.

Per ulteriori informazioni su come utilizzare la chiave KMS predefinita, consulta [Crittografia inattiva in Amazon SQS](#).

Aggiunta di SSE a una coda esistente

Per abilitare la crittografia lato server per una coda esistente, utilizza il metodo [SetQueueAttributes](#) per impostare l'attributo `KmsMasterKeyId`.

Il seguente esempio di codice imposta AWS KMS key come chiave KMS AWS gestita per Amazon SQS. L'esempio imposta inoltre il [periodo di riutilizzo di AWS KMS key](#) su 140 secondi.

Prima di eseguire il codice di esempio, assicurati di aver impostato le AWS credenziali. Per ulteriori informazioni, consulta [Configurare AWS le credenziali e la regione per lo sviluppo nella Guida per gli AWS SDK for Java 2.x sviluppatori](#).

```
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();
    }
}
```

Disabilitazione di SSE per una coda

Per disabilitare la crittografia lato server per una coda esistente, imposta l'attributo `KmsMasterKeyId` su una stringa vuota tramite l'operazione `SetQueueAttributes`.

Important

`null` non è un valore valido per `KmsMasterKeyId`.

Creazione di una coda con SSE

Per abilitare SSE quando crei la coda, aggiungi l'attributo `KmsMasterKeyId` al metodo API [CreateQueue](#).

L'esempio seguente spiega come creare una coda con SSE abilitato. La coda utilizza la chiave KMS gestita da AWS per Amazon SQS. L'esempio imposta inoltre il [periodo di riutilizzo di AWS KMS key](#) su 160 secondi.

Prima di eseguire il codice di esempio, assicurati di aver impostato le AWS credenziali. Per ulteriori informazioni, consulta [Configurare AWS le credenziali e la regione per lo sviluppo nella Guida per gli AWS SDK for Java 2.x sviluppatori](#).

```
// 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);
```

Recupero degli attributi SSE

Per informazioni sul recupero degli attributi di coda, consulta [Examples](#) nella Documentazione di riferimento delle API di Amazon Simple Queue Service.

Per recuperare l'ID della chiave KMS o il periodo di riutilizzo della chiave dati per una particolare coda, esegui il metodo [GetQueueAttributes](#) e recupera i valori `KmsMasterKeyId` e `KmsDataKeyReusePeriodSeconds`.

Configurazione dei tag per una coda Amazon SQS

Puoi aggiungere i tag di allocazione dei costi alle tue code Amazon SQS per organizzarle e identificarle più facilmente. Gli esempi seguenti mostrano come gestire i tag con AWS SDK per Java. Per ulteriori informazioni, consulta [Tag per l'allocazione dei costi Amazon SQS](#).

Prima di eseguire il codice di esempio, assicurati di aver impostato le AWS credenziali. Per ulteriori informazioni, consulta [Configurare AWS le credenziali e la regione per lo sviluppo nella Guida per gli AWS SDK for Java 2.x sviluppatori](#).

Come elencare i tag

Per elencare i tag di una coda, utilizza il metodo `ListQueueTags`.

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

Aggiunta o aggiornamento dei tag

Per aggiungere o aggiornare i valori dei tag per una coda, utilizza il metodo `TagQueue`.

```
// 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);
```

Rimozione dei tag

Per rimuovere uno o più tag dalla coda, utilizza il metodo `UntagQueue`. L'esempio seguente rimuove il tag `Accounting ID`.


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

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

Invio degli attributi dei messaggi a una coda Amazon SQS

Puoi includere metadati strutturati (come time stamp, dati geospaziali, firme e identificatori) con i messaggi tramite gli attributi dei messaggi. Per ulteriori informazioni, consulta [Attributi messaggio di Amazon SQS](#).

Prima di eseguire il codice di esempio, assicurati di aver impostato le AWS credenziali. Per ulteriori informazioni, consulta [Configurare AWS le credenziali e la regione per lo sviluppo nella Guida per gli AWS SDK for Java 2.x sviluppatori](#).

Definizione degli attributi

Per definire un attributo per un messaggio, aggiungi il codice seguente che utilizza il tipo di dati [MessageAttributeValue](#). Per ulteriori informazioni, consultare [Componenti attributo del messaggio](#) e [Tipi di dati degli attributi di messaggio](#).

Calcola AWS SDK per Java automaticamente i checksum del corpo del messaggio e degli attributi del messaggio e li confronta con i dati restituiti da Amazon SQS. Per ulteriori informazioni, consulta la [Guida per gli sviluppatori AWS SDK for Java 2.x](#) e [Calcolo del digest dei MD5 messaggi per gli attributi dei messaggi](#) per altri linguaggi di programmazione.

String

Questo esempio definisce un attributo String denominato Name con il valore Jane.

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

```
.withStringValue("Jane"));
```

Number

Questo esempio definisce un attributo `Number` denominato `AccurateWeight` con il valore `230.000000000000000001`.

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

Binary

Questo esempio definisce un attributo `Binary` denominato `ByteArray` con il valore di un array non inizializzato di 10 byte.

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

String (custom)

Questo esempio definisce l'attributo personalizzato `String.EmployeeId` denominato `EmployeeId` con il valore `ABC123456`.

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

Number (custom)

Questo esempio definisce l'attributo personalizzato `Number.AccountId` denominato `AccountId` con il valore `000123456`.

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

 Note

Poiché il tipo di dati di base è `Number`, il metodo [ReceiveMessage](#) restituisce 123456.

Binary (custom)

Questo esempio definisce l'attributo personalizzato `Binary.JPEG` denominato `ApplicationIcon` con il valore di un array non inizializzato di 10 byte.

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

Invio di un messaggio con attributi

Questo esempio aggiunge gli attributi a `SendMessageRequest` prima dell'invio del messaggio.

```
// 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

Se invii un messaggio a una coda First-In-First-Out (FIFO), assicurati che il `sendMessage` metodo venga eseguito dopo aver fornito l'ID del gruppo di messaggi.

Se utilizzi l'operazione [SendMessageBatch](#) invece di [SendMessage](#), devi specificare gli attributi per ogni singolo messaggio presente nel batch.

Utilizzo APIs con Amazon SQS

Questo argomento fornisce informazioni sulla creazione di endpoint Amazon SQS, sull'esecuzione di richieste API di query utilizzando i metodi GET e POST e sull'utilizzo di azioni API in batch. Per informazioni dettagliate sulle [azioni](#) di Amazon SQS, inclusi parametri, errori, esempi e [tipi di dati](#), consulta [Guida di riferimento delle API Amazon Simple Queue Service](#).

Per accedere ad Amazon SQS utilizzando una varietà di linguaggi di programmazione, puoi anche utilizzare [AWS SDKs](#), che contengono le seguenti funzionalità automatiche:

- Firma crittografica delle richieste di servizio
- Nuovi tentativi di richiesta
- Gestione delle risposte di errore

Per ulteriori informazioni, consulta [the section called “Lavorare con AWS SDKs”](#).

Per informazioni sullo strumento a riga di comando, consulta le sezioni Amazon SQS nella [Guida di riferimento AWS CLI](#) e la [AWS Strumenti per PowerShell Guida di riferimento Cmlet](#).

Amazon SQS APIs con protocollo JSON AWS

[Amazon SQS utilizza il protocollo AWS JSON come meccanismo di trasporto per tutti gli Amazon SQS nelle versioni SDK APIs specificate.](#) AWS Il protocollo JSON offre un throughput più elevato, una latenza inferiore e una comunicazione più rapida. application-to-application AWS Il protocollo JSON è più efficiente nelle richieste e nelle risposte rispetto al protocollo serialization/deserialization di query. AWS Se preferisci comunque utilizzare il protocollo di AWS query con SQS APIs, consulta [Quali lingue sono supportate per il protocollo AWS JSON utilizzato in Amazon APIs SQS?](#) le versioni AWS SDK che supportano il protocollo di query Amazon AWS SQS.

Amazon SQS utilizza il protocollo AWS JSON per comunicare tra i client AWS SDK (ad esempio, Java, Python, Golang) e JavaScript il server Amazon SQS. Una richiesta HTTP di un'operazione API Amazon SQS accetta input in formato JSON. L'operazione Amazon SQS viene eseguita e la risposta di esecuzione viene inviata al client SDK in formato JSON. Rispetto alle AWS query, AWS JSON è più semplice, veloce ed efficiente per il trasporto di dati tra client e server.

- AWS Il protocollo JSON funge da mediatore tra il client e il server Amazon SQS.
- Il server non comprende il linguaggio di programmazione in cui viene creata l'operazione Amazon SQS, ma comprende il AWS protocollo JSON.

- Il protocollo AWS JSON utilizza la serializzazione (conversione dell'oggetto in formato JSON) e la deserializzazione (conversione del formato JSON in oggetto) tra client e server Amazon SQS.

Per ulteriori informazioni sul protocollo AWS JSON con Amazon SQS, consulta [Protocollo Amazon SQS JSON AWS FAQs](#)

AWS [Il protocollo JSON è disponibile nella versione SDK specificata AWS](#) . Per esaminare la versione e le date di rilascio dell'SDK nelle diverse varianti linguistiche, consulta la [matrice di supporto delle versioni di AWS SDKs and Tools](#) nella and Tools Reference AWS SDKs Guide

Effettuazione di richieste API di query utilizzando il protocollo AWS JSON in Amazon SQS

Questo argomento spiega come costruire un endpoint Amazon SQS, effettuare richieste POST e interpretare le risposte.

Note

AWS Il protocollo JSON è supportato per la maggior parte delle varianti linguistiche. Per un elenco delle varianti di linguaggi supportate, consulta [Quali lingue sono supportate per il protocollo AWS JSON utilizzato in Amazon APIs SQS?](#).

Costruzione di un endpoint

Per utilizzare le code Amazon SQS, è necessario creare un endpoint. Per informazioni sugli endpoint Amazon SQS, consulta le seguenti pagine in Riferimenti generali di Amazon Web Services:

- [Endpoint regionali](#)
- [Endpoint e quote di Amazon Simple Queue Service](#)

Ogni endpoint Amazon SQS è completamente indipendente. Ad esempio, se due code sono denominate MyQueue e una ha l'endpoint `sqs.us-east-2.amazonaws.com`, mentre l'altra ha l'endpoint `sqs.eu-west-2.amazonaws.com`, le due code non condividono alcun dato tra loro.

Di seguito è riportato un esempio di un endpoint che invia una richiesta per creare una coda.

```
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

I nomi delle code e la coda fanno distinzione tra maiuscole e URLs minuscole.

La struttura di **AUTHPARAMS** dipende dalla modalità di firma della richiesta API. Per ulteriori informazioni, consulta [Signing AWS API Requests](#) nel Amazon Web Services General Reference.

Effettuare una richiesta POST

Una richiesta Amazon SQS POST invia i parametri di query come modulo nel corpo di una richiesta HTTP.

Di seguito è riportato un esempio di intestazione HTTP con X-Amz-Target impostato su AmazonSQS.<operationName> e di intestazione HTTP con Content-Type impostato su 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"
}
```

Questa richiesta HTTP POST invia un messaggio a una coda Amazon SQS.

Note

Entrambe le intestazioni HTTP X-Amz-Target e Content-Type sono obbligatorie. Il client HTTP potrebbe aggiungere altri elementi alla richiesta HTTP, a seconda della versione HTTP del client.

Interpretazione delle risposte dell'API JSON di Amazon SQS

Quando invii una richiesta ad Amazon SQS, viene restituita una risposta JSON con i risultati. La struttura della risposta dipende dall'azione API utilizzata.

Per comprendere i dettagli di queste risposte, consulta:

- L'azione API specifica nelle [azioni API](#) nel riferimento all'API di Amazon Simple Queue Service
- La [Protocollo Amazon SQS JSON AWS FAQs](#)

Struttura di una risposta JSON corretta

Se la richiesta ha esito positivo, l'elemento di risposta principale è x-amzn-RequestId, che contiene l'Universal Unique Identifier (UUID) della richiesta e altri campi di risposta aggiunti. Ad esempio, l'elemento [CreateQueue](#) contiene il campo QueueUrl che, a sua volta, contiene l'URL della coda creata.

```
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"
}
```

Struttura di una risposta di errore JSON

Se una richiesta genera un errore, Amazon SQS restituisce la risposta principale, inclusi l'intestazione HTTP e il corpo.

Nell'intestazione HTTP, `x-amzn-RequestId` contiene l'UUID della richiesta. `x-amzn-query-error` contiene due informazioni: il tipo di errore e se si tratta di un errore del produttore o del consumatore.

Nel corpo della risposta, `"__type"` indica altri dettagli sull'errore e `Message` indica la condizione di errore in un formato leggibile.

Di seguito è riportato un esempio di risposta di errore in formato JSON:

```
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."
}
```

Protocollo Amazon SQS JSON AWS FAQs

Questo argomento copre le domande frequenti sull'uso del protocollo AWS JSON con Amazon SQS.

Cos'è il protocollo AWS JSON e in che cosa differisce dalle richieste e dalle risposte API Amazon SQS esistenti?

JSON è uno dei metodi di cablaggio più utilizzati e accettati per la comunicazione tra sistemi eterogenei. Amazon SQS utilizza JSON come mezzo per comunicare tra un client AWS SDK (ad esempio, Java, Python, Golang) JavaScript e il server Amazon SQS. Una richiesta HTTP di un'operazione API Amazon SQS accetta input in formato JSON. L'operazione Amazon SQS viene eseguita e la risposta di esecuzione viene inviata al client SDK in formato JSON. Rispetto alle query AWS, JSON è più efficiente in termini di trasporto dei dati tra client e server.

- Il protocollo Amazon SQS AWS JSON funge da mediatore tra il client e il server Amazon SQS.
- Il server non comprende il linguaggio di programmazione in cui viene creata l'operazione Amazon SQS, ma comprende il AWS protocollo JSON.
- Il protocollo Amazon SQS AWS JSON utilizza la serializzazione (conversione dell'oggetto in formato JSON) e la deserializzazione (conversione del formato JSON in oggetto) tra il client e il server Amazon SQS.

Come posso iniziare a usare i protocolli AWS JSON per Amazon SQS?

Per iniziare a usare l'ultima versione dell' AWS SDK e ottenere una messaggistica più veloce per Amazon SQS, aggiorna AWS il tuo SDK alla versione specificata o a qualsiasi versione successiva. Per ulteriori informazioni sui client SDK, consulta la colonna Guida nella tabella seguente.

Di seguito è riportato un elenco di versioni SDK in diverse varianti linguistiche per il protocollo AWS JSON da utilizzare con Amazon SQS: APIs

Lingua	Archivio client SDK	Versione del client SDK richiesta	Guida
C++	aws/ aws-sdk-cpp	1.11,98	AWS SDK per 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 per Go V2
Java 1.x	aws/ aws-sdk-java	1,12.585	AWS SDK per Java
Java 2.x	seghe/2 aws-sdk-j ava-v	2,21,19	AWS SDK per Java
JavaScript v2.x	aws/ aws-sdk-js	JavaScript su AWS	
JavaScript v3.x	aws/ 3 aws-sdk-js-v	v3.447.0	JavaScript su AWS

Lingua	Archivio client SDK	Versione del client SDK richiesta	Guida
.NET	seghe/aws-sdk-net	3,7681,0	AWS SDK per.NET
PHP	aws/ aws-sdk-php	3,285,2	AWS SDK per PHP
Python-boto3	boto/boto3	1.28.82	AWS SDK per Python (Boto3)
Python-botocore	boto/botocore	1.31.82	AWS SDK per Python (Boto3)
awscli	CLI AWS	1.29.82	Interfaccia a riga di comando AWS
Ruby	seghe/aws-sdk-ruby	1.67,0	AWS SDK per Ruby

Quali sono i rischi dell'abilitazione del protocollo JSON per i miei carichi di lavoro Amazon SQS?

Se utilizzi un'implementazione personalizzata di AWS SDK o una combinazione di client personalizzati e AWS SDK per interagire con Amazon SQS che AWS genera risposte basate su Query (ovvero basate su XML), potrebbe essere incompatibile con il protocollo JSON. AWS In caso di problemi, contatta l' AWS assistenza.

Cosa succede se uso già la versione AWS SDK più recente, ma la mia soluzione open source non supporta JSON?

È necessario modificare la versione dell'SDK con la versione precedente a quella in uso. [Come posso iniziare a usare i protocolli AWS JSON per Amazon SQS?](#) Per ulteriori informazioni, vedere. AWS Le versioni SDK elencate in [Come posso iniziare a usare i protocolli AWS JSON per Amazon SQS?](#) utilizzano il protocollo Wire JSON per Amazon SQS. APIs Se modifichi il tuo AWS SDK alla versione precedente, Amazon APIs SQS AWS utilizzerà la query.

Quali lingue sono supportate per il protocollo AWS JSON utilizzato in Amazon APIs SQS?

Amazon SQS supporta tutte le varianti linguistiche in cui AWS SDKs sono generalmente disponibili (GA). Al momento non supportiamo Kotlin, Rust o Swift. Per saperne di più sulle altre varianti linguistiche, consulta [Strumenti per costruire su AWS](#).

Quali regioni sono supportate per il protocollo AWS JSON utilizzato in Amazon SQS APIs

Amazon SQS supporta il protocollo AWS JSON in tutte le [AWS regioni](#) in cui è disponibile Amazon SQS.

Quali miglioramenti della latenza posso aspettarmi eseguendo l'aggiornamento alle versioni AWS SDK specificate per Amazon SQS utilizzando il protocollo JSON? AWS

AWS Il protocollo JSON è più efficiente nella serializzazione e deserializzazione di richieste e risposte rispetto al protocollo di query. AWS In base ai test AWS delle prestazioni per un payload di messaggi di 5 KB, il protocollo JSON per Amazon SQS end-to-end riduce la latenza di elaborazione dei messaggi fino al 23% e riduce l'utilizzo della CPU e della memoria lato client dell'applicazione.

Il protocollo di interrogazione sarà obsoleto AWS ?

AWS il protocollo di interrogazione continuerà a essere supportato. Puoi continuare a utilizzare il protocollo di AWS query purché la tua versione dell' AWS SDK sia impostata su una versione precedente diversa da quella elencata in [Come iniziare a usare i protocolli AWS JSON per Amazon SQS](#).

Dove posso trovare ulteriori informazioni sul AWS protocollo JSON?

Puoi trovare ulteriori informazioni sul protocollo JSON nel [AWS protocollo JSON 1.0](#) nella documentazione Smithy. Per ulteriori informazioni sulle richieste API di Amazon SQS che utilizzano il protocollo AWS JSON, consulta [Effettuazione di richieste API di query utilizzando il protocollo AWS JSON in Amazon SQS](#).

Effettuazione di richieste API di AWS interrogazione utilizzando il protocollo di query in Amazon SQS

Questo argomento spiega come costruire un endpoint Amazon SQS, effettuare richieste GET e POST e interpretare le risposte.

Costruzione di un endpoint

Per lavorare con le code Amazon SQS, è necessario creare un endpoint. Per informazioni sugli endpoint Amazon SQS, consulta le seguenti pagine in Riferimenti generali di Amazon Web Services:

- [Endpoint regionali](#)
- [Endpoint e quote di Amazon Simple Queue Service](#)

Ogni endpoint Amazon SQS è completamente indipendente. Ad esempio, se due code sono denominate MyQueue e una ha l'endpoint `sqs.us-east-2.amazonaws.com`, mentre l'altra ha l'endpoint `sqs.eu-west-2.amazonaws.com`, le due code non condividono alcun dato tra loro.

Di seguito è riportato un esempio di un endpoint che invia una richiesta per creare una coda.

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

Note

I nomi delle code e la coda fanno distinzione tra maiuscole e minuscole. URLs

La struttura di **AUTHPARAMS** dipende dalla modalità di firma della richiesta API. Per ulteriori informazioni, consulta [Signing AWS API Requests](#) nel Amazon Web Services General Reference.

Effettuare una richiesta GET

Una richiesta Amazon SQS GET è strutturata come un URL che contiene le seguenti informazioni:

- Endpoint: la risorsa su cui agisce la richiesta (il [nome della coda e l'URL](#)), ad esempio: `https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`
- Operazione: l'[operazione](#) che desideri eseguire sull'endpoint. Un punto di domanda (?) separa l'endpoint dall'azione, ad esempio: `?Action=SendMessage&MessageBody=Your%20Message%20Text`
- Parametri: gli eventuali parametri della richiesta. Ogni parametro è separato da una e commerciale (&) ad esempio: `&Version=2012-11-05&AUTHPARAMS`

Il seguente è un esempio di una richiesta GET per inviare un messaggio a una coda Amazon SQS.

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

Note

I nomi delle code e la coda fanno distinzione tra maiuscole e URLs minuscole. Poiché le richieste GET lo sono URLs, è necessario codificare in URL tutti i valori dei parametri. Poiché gli spazi non sono consentiti URLs, ogni spazio è codificato come URL. %20 Il resto dell'esempio non è stato codificato nell'URL per renderne più facile la lettura.

Effettuare una richiesta POST

Una richiesta Amazon SQS POST invia i parametri di query come modulo nel corpo di una richiesta HTTP.

Di seguito è riportato un esempio di intestazione HTTP con Content-Type impostato su `application/x-www-form-urlencoded`.

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

L'intestazione è seguita da una richiesta GET [form-urlencoded](#) che invia un messaggio a una coda Amazon SQS. Ogni parametro è separato da una e commerciale (&).

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

Note

Solo l'intestazione HTTP Content-Type è obbligatoria. La richiesta **AUTHPARAMS** è la stessa della richiesta GET.

Il client HTTP potrebbe aggiungere altri elementi alla richiesta HTTP, a seconda della versione HTTP del client.

Interpretazione delle risposte dell'API XML di Amazon SQS

Quando invii una richiesta ad Amazon SQS, restituisce una risposta XML contenente i risultati della richiesta. Per comprendere la struttura e i dettagli di queste risposte, consulta le [azioni API](#) specifiche nel riferimento alle API di Amazon Simple Queue Service.

Struttura di una risposta XML corretta

Se la richiesta è andata a buon fine, l'elemento principale della risposta prende il nome dell'azione, con la dicitura Response aggiunta (ad esempio, **ActionNameResponse**).

Questo elemento contiene i seguenti elementi figli:

- **ActionNameResult**: contiene un elemento specifico dell'operazione. Ad esempio, l'elemento **CreateQueueResult** contiene l'elemento **QueueUrl** che, a sua volta, contiene l'URL della coda creata.
- **ResponseMetadata**: contiene **RequestId** che, a sua volta, contiene l'UUID della richiesta.

Di seguito è riportato un esempio di risposta corretta in formato XML:

```
<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>
```

Struttura di una risposta di errore XML

Se una richiesta ha esito negativo, Amazon SQS restituisce sempre il principale elemento di risposta `ErrorResponse`. Questo elemento contiene un elemento `Error` e un elemento `RequestId`.

L'elemento `Error` contiene i seguenti elementi figli:

- **Type**: specifica se l'errore è stato un errore di produttore o consumatore.
- **Code**: specifica il tipo di errore.
- **Message**: specifica la condizione dell'errore in un formato leggibile.
- **Detail**: (Facoltativo) Specifica ulteriori dettagli sull'errore.

L'elemento `RequestId` contiene l'UUID della richiesta.

Di seguito è riportato un esempio di risposta di errore in formato XML:

```
<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>
```

Autenticazione delle richieste per Amazon SQS

L'autenticazione è un processo per l'identificazione e la verifica di chi invia una richiesta. Durante la prima fase di autenticazione, AWS verifica l'identità del produttore e se il produttore è [registrato per](#)

[utilizzare AWS](#) (per ulteriori informazioni, consulta [Fase 1: creare un utente Account AWS e IAM](#)).

Successivamente, si AWS attiene alla seguente procedura:

1. Il produttore (mittente) ottiene le credenziali necessarie.
2. Il produttore invia al consumatore (ricevitore) una richiesta con la credenziale.
3. Il consumatore utilizza la credenziale per verificare se il produttore ha inviato la richiesta.
4. Si verifica una delle seguenti situazioni:
 - Se l'autenticazione va a buon fine, il consumatore elabora la richiesta.
 - Se l'autenticazione ha esito negativo, il consumatore rifiuta la richiesta e restituisce un errore.

Processo di autenticazione di base con HMAC-SHA

Quando accedi ad Amazon SQS utilizzando l'API della query, devi fornire le seguenti voci affinché la richiesta possa essere autenticata:

- L'ID della chiave di AWS accesso che identifica la tua Account AWS, che viene AWS utilizzato per cercare la tua chiave di accesso segreta.
 - La firma della richiesta HMAC-SHA, calcolata utilizzando la chiave di accesso segreta (un segreto condiviso noto solo a te e, AWS per ulteriori informazioni, consulta). [RFC2104](#) L'[AWS SDK](#) gestisce il processo di firma; tuttavia, se invii una richiesta di query su HTTP o HTTPS, devi includere una firma in ogni richiesta di query.
1. Ottenere una chiave di firma Signature Version 4. Per ulteriori informazioni, consulta [Ottenere una chiave di firma con Java](#).

Note

Amazon SQS supporta la versione 4 di Signature, che offre sicurezza e prestazioni SHA256 migliorate rispetto alle versioni precedenti. Se crei nuove applicazioni che utilizzano Amazon SQS, è consigliabile usare Signature Version 4.

2. Le firme di richiesta devono essere codificate in base64. Il seguente codice Java di esempio procede in questo modo:

```
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);  
    }  
}
```

- Il timestamp (o scadenza) della richiesta. Il timestamp utilizzato nella richiesta deve essere un `dateTime` oggetto, con [la data completa, incluso ore, minuti e secondi](#). Ad esempio: `2007-01-31T23:59:59Z` Anche se non è obbligatorio, ti consigliamo di fornire l'oggetto nell'orario UTC (fuso orario di Greenwich).

Note

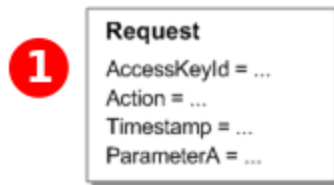
Assicurati che l'orario del tuo server sia impostato correttamente. Se specifichi un timestamp (anziché una scadenza), la richiesta scade automaticamente 15 minuti dopo l'ora specificata (AWS non elabora le richieste con timestamp precedenti di più di 15 minuti all'ora corrente sui server). AWS

Se usi .NET, non devi inviare timestamp troppo specifici (poiché diverse interpretazioni della precisione del tempo aggiuntivo devono essere ignorate). In questo caso, è necessario costruire manualmente oggetti `dateTime` con una precisione non superiore a un millisecondo.

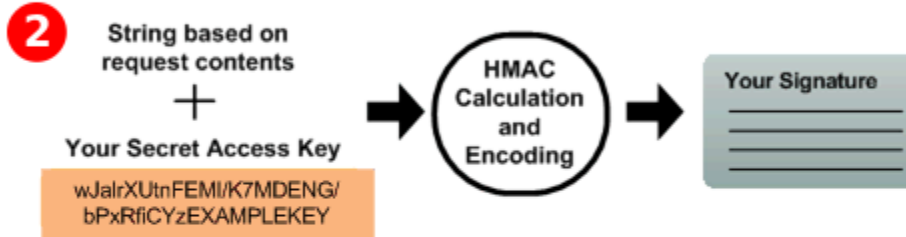
Parte 1: la richiesta dall'utente

Di seguito è riportato il processo da seguire per autenticare le AWS richieste utilizzando una firma di richiesta HMAC-SHA.

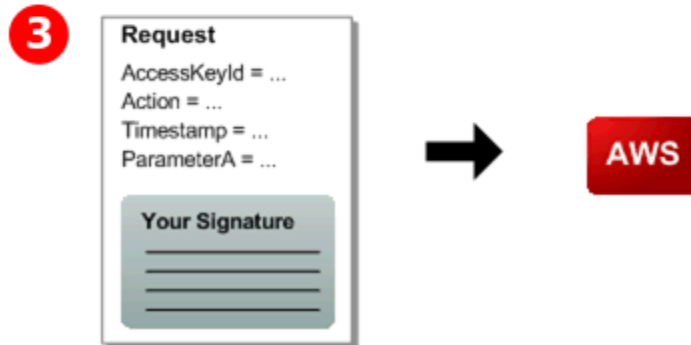
Create a request:



Create an
HMAC-SHA
signature:



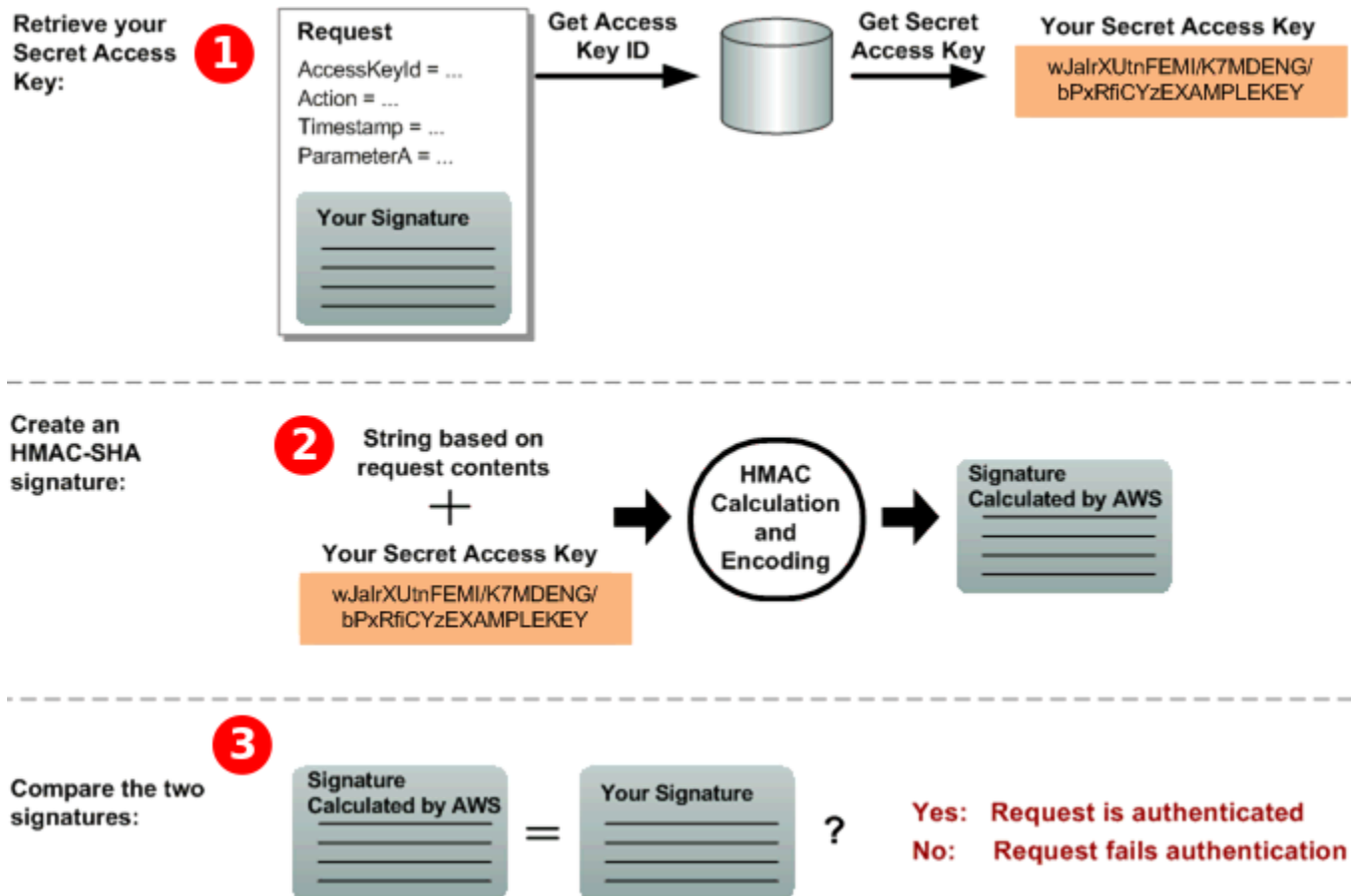
Send the request
and signature to
AWS:



1. Costruisci una richiesta a. AWS
2. Calcola una firma (HMAC-SHA) con il codice di autenticazione del messaggio di hash con chiave utilizzando la tua Secret Access Key.
3. Includi la firma e l'ID della chiave di accesso nella richiesta, quindi invia la richiesta a AWS.

Parte 2: La risposta di AWS

AWS inizia il seguente processo in risposta.



1. AWS utilizza l'ID della chiave di accesso per cercare la chiave di accesso segreta.
2. AWS genera una firma dai dati della richiesta e dalla chiave di accesso segreta, utilizzando lo stesso algoritmo utilizzato per calcolare la firma inviata nella richiesta.
3. Si verifica una delle seguenti situazioni:
 - Se la firma AWS generata corrisponde a quella inviata nella richiesta, AWS considera la richiesta autentica.
 - Se il confronto fallisce, la richiesta viene scartata e AWS restituisce un errore.

Operazioni in batch per Amazon SQS

Amazon SQS offre azioni in batch per aiutarti a ridurre i costi e manipolare fino a 10 messaggi con una sola azione. Queste azioni in batch includono:

- [SendMessageBatch](#)
- [DeleteMessageBatch](#)

- [ChangeMessageVisibilityBatch](#)

Utilizzando le azioni in batch, è possibile eseguire più operazioni in una singola chiamata API, il che aiuta a ottimizzare le prestazioni e ridurre i costi. Puoi sfruttare la funzionalità batch utilizzando l'API di interrogazione o qualsiasi AWS SDK che supporti le azioni batch di Amazon SQS.

Dettagli importanti

- Limite di dimensione dei messaggi: la dimensione totale di tutti i messaggi inviati in una singola `SendMessageBatch` chiamata non può superare 1.048.576 byte (1 MiB)
- Autorizzazioni: non è possibile impostare esplicitamente le autorizzazioni per, o, `SendMessageBatch` `DeleteMessageBatch` `ChangeMessageVisibilityBatch`. Invece, `imposta` o `ChangeMessageVisibility` imposta le autorizzazioni per `SendMessage` le versioni batch corrispondenti delle azioni. `DeleteMessage`
- Supporto console: la console Amazon SQS non supporta azioni in batch. È necessario utilizzare l'API di interrogazione o un AWS SDK per eseguire operazioni in batch.

Raggruppamento delle azioni con messaggio

Per ottimizzare ulteriormente i costi e l'efficienza, prendi in considerazione le seguenti best practice per il raggruppamento in batch delle azioni relative ai messaggi:

- Azioni API in batch: utilizza le azioni dell'[API batch di Amazon SQS](#) per inviare, ricevere ed eliminare messaggi e per modificare il timeout di visibilità dei messaggi per più messaggi con una singola azione. Ciò riduce il numero di chiamate API e i costi associati.
- Buffering lato client e long polling: [combina il buffering lato client con il batching delle richieste utilizzando il polling lungo insieme al client asincrono bufferizzato incluso in](#). AWS SDK per Java Questo approccio aiuta a ridurre al minimo il numero di richieste e ottimizza la gestione di grandi volumi di messaggi.

Note

L'Amazon SQS Buffered Asynchronous Client attualmente non supporta le code FIFO.

Abilitazione del buffering lato client e del batching delle richieste con Amazon SQS

[AWS SDK per Java](#) include `AmazonSQSBufferedAsyncClient` che accede ad Amazon SQS. Questo client consente un semplice raggruppamento delle richieste utilizzando il buffering lato client. Le chiamate effettuate dal client vengono prima memorizzate nel buffer e quindi inviate come richiesta batch ad Amazon SQS.

Il buffering lato client consente di memorizzare e inviare fino a 10 richieste in batch, riducendo il costo di utilizzo di Amazon SQS e il numero di richieste inviate. `AmazonSQSBufferedAsyncClient` effettua il buffering delle chiamate sincrone e asincrone. Le richieste in batch e il supporto per il [polling lungo](#) possono inoltre contribuire ad aumentare il throughput. Per ulteriori informazioni, consulta [Aumento del throughput utilizzando la scalabilità orizzontale e l'action batching con Amazon SQS](#).

Poiché `AmazonSQSBufferedAsyncClient` implementa la stessa interfaccia di `AmazonSQSAsyncClient`, la migrazione da `AmazonSQSAsyncClient` a `AmazonSQSBufferedAsyncClient` di solito richiede solo modifiche minime al codice esistente.

Note

L'Amazon SQS Buffered Asynchronous Client attualmente non supporta le code FIFO.

Usare Amazon SQSBuffered AsyncClient

Prima di iniziare, completa i passaggi descritti in [Configurazione di Amazon SQS](#).

AWS SDK per Java 1.x

Per AWS SDK for Java 1.x, puoi crearne uno `AmazonSQSBufferedAsyncClient` nuovo basato sul seguente esempio:

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

Una volta creato il nuovo `AmazonSQSBufferedAsyncClient`, è possibile utilizzarlo per l'invio di più richieste ad Amazon SQS; (proprio come con `AmazonSQSAsyncClient`), ad esempio:

```
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);
```

Configurazione di Amazon SQSBuffered AsyncClient

`AmazonSQSBufferedAsyncClient` è preconfigurato con impostazioni che funzionano per la maggior parte dei casi d'uso. È possibile configurare ulteriormente `AmazonSQSBufferedAsyncClient`, ad esempio:

1. Crea un'istanza della classe `QueueBufferConfig` con i parametri di configurazione obbligatori.
2. Fornisci l'istanza al costruttore `AmazonSQSBufferedAsyncClient`.

```
// 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 parametri di configurazione

Parametro	Valore predefinito	Description
longPoll	true	Quando longPoll è impostato su true, AmazonSQSBufferedAsyncClient tenta di utilizzare il polling lungo durante l'utilizzo dei messaggi.
longPollWaitTimeoutSeconds	20 s	Il tempo massimo, in secondi, che una chiamata di ReceiveMessage si blocca sul server in attesa che i messaggi compaiano nella coda prima di restituire un risultato di ricezione vuoto.  Note Questa impostazione non ha alcun effetto se il polling lungo è disattivato.
maxBatchOpenMs	200 ms	Il tempo massimo, in millisecondi, che una chiamata in uscita attende altre chiamate dello stesso tipo con cui raggrupparsi. Maggiore è l'impostazione, minore è il numero di batch

Parametro	Valore predefinito	Description
		necessari per eseguire la stessa quantità di lavoro (tuttavia, la prima chiamata in un batch deve dedicare più tempo all'attesa). Quando questo parametro è impostato su 0, le richieste inviate non attendono altre richieste, disabilitando effettivamente il batching.
maxBatchSize	10 richieste per batch	Il numero massimo di messaggi raggruppati in una singola richiesta. Maggiore è l'impostazione, minore è il numero di batch che sono necessari per eseguire la stessa quantità di richieste.  Note 10 richieste per batch è il valore massimo consentito per Amazon SQS.

Parametro	Valore predefinito	Description
maxBatchSizeBytes	1 MiB	Le dimensioni massime del batch di un messaggio, in byte, che il client tenta di inviare ad Amazon SQS .  Note 1 MiB è il valore massimo consentito per Amazon SQS.

Parametro	Valore predefinito	Description
<code>maxDoneReceiveBatches</code>	10 batch	Il numero massimo di pre-fetching e archiviazioni di batch AmazonSQS <code>BufferedAsyncClient</code> sul lato client. Maggiore è l'impostazione, più richieste di ricezione possono essere soddisfatte senza la necessità di effettuare una chiamata ad Amazon SQS (tuttavia, più messaggi sono pre-recuperati, più tempo rimangono nel buffer, il che significa che il loro timeout visibilità scade).  Note 0 indica che il precaricamento di tutti i messaggi è disabilitato e che i messaggi vengono consumati solo su richiesta.

Parametro	Valore predefinito	Description
<code>maxInflightOutboundBatches</code>	5 batch	Il numero massimo di batch in uscita attivi che possono essere elaborati contemporaneamente. Maggiore è l'impostazione, più rapidamente possono essere inviati i batch in uscita (in base alle altre quote, ad esempio CPU o larghezza di banda) e più thread possono essere utilizzati da <code>AmazonSQS.BufferedAsyncClient</code>.

Parametro	Valore predefinito	Description
<code>maxInflightReceive Batches</code>	10 batch	Il numero massimo di batch di ricezione attivi che possono essere elaborati contemporaneamente. Maggiore è l'impostazione e più messaggi possono essere ricevuti (in base alle quote, ad esempio CPU o larghezza di banda) e più thread possono essere utilizzati da <code>AmazonSQS BufferedAsyncClient</code>.  Note 0 indica che la prelettura di tutti i messaggi è disabilitata e che i messaggi vengono consumati solo su richiesta.

Parametro	Valore predefinito	Description
<code>visibilityTimeoutSeconds</code>	-1	Quando questo parametro è impostato su un valore diverso da zero, questo timeout visibilità sostituisce il timeout visibilità impostato sulla coda dalla quale vengono utilizzati i messaggi.  Note -1 indica che per la coda è selezionata l'impostazione predefinita. Non è possibile impostare il timeout visibilità su 0.

AWS SDK per Java 2.x

Per AWS SDK for Java 2.x, puoi crearne uno `SqsAsyncBatchManager` nuovo basato sul seguente esempio:

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

Una volta creato il nuovo `SqsAsyncBatchManager`, è possibile utilizzarlo per l'invio di più richieste ad Amazon SQS; (proprio come con `SqsAsyncClient`), ad esempio:

```

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);

```

Configurazione SqsAsyncBatchManager

SqsAsyncBatchManager è preconfigurato con impostazioni che funzionano per la maggior parte dei casi d'uso. È possibile configurare ulteriormente SqsAsyncBatchManager, ad esempio:

Creazione di una configurazione personalizzata tramite `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();

```

Parametri di **BatchOverrideConfiguration**

Parametro	Valore predefinito	Description
maxBatchSize	10 richieste per batch	Il numero massimo di messaggi raggruppati in una singola richiesta. Maggiore è l'impostazione, minore è il numero di batch che sono necessari per eseguire la stessa quantità di richieste.  Note Il valore massimo consentito per Amazon SQS è di 10 richieste per batch.
sendRequestFrequency	200 ms	Il tempo massimo, in millisecondi, che una chiamata in uscita attende altre chiamate dello stesso tipo con cui raggrupparsi. Maggiore è l'impostazione, minore è il numero di batch necessari per eseguire la

Parametro	Valore predefinito	Description
		stessa quantità di lavoro (tuttavia, la prima chiamata in un batch deve dedicare più tempo all'attesa). Quando questo parametro è impostato su 0, le richieste inviate non attendono altre richieste, disabilitando effettivamente il batching.
<code>receiveMessageVisibilityTimeout</code>	-1	Quando questo parametro è impostato su un valore diverso da zero, questo timeout visibilità sostituisce il timeout visibilità impostato sulla coda dalla quale vengono utilizzati i messaggi.  Note 1 indica che per la coda è selezionata l'impostazione predefinita. Non è possibile impostare il timeout visibilità su 0.

Parametro	Valore predefinito	Description
<code>receiveMessageMinWaitDuration</code>	50 ms	La quantità minima di tempo (in millisecondi) di attesa di una <code>receiveMessage</code> chiamata prima che i messaggi disponibili vengano recuperati. Più alta è l'impostazione, minore è il numero di batch necessari per eseguire lo stesso numero di richieste.

Aumento del throughput utilizzando la scalabilità orizzontale e l'action batching con Amazon SQS

Amazon SQS supporta la messaggistica ad alto throughput. Per dettagli sui limiti di velocità effettiva, consulta [Quote di messaggi Amazon SQS](#).

Per massimizzare la produttività:

- [Scala](#) produttori e consumatori orizzontalmente aggiungendo più istanze di ciascuno.
- Utilizza l'[action batching](#) per inviare o ricevere più messaggi in un'unica richiesta, riducendo il sovraccarico delle chiamate API.

Dimensionamento orizzontale

Poiché accedi ad Amazon SQS attraverso un protocollo di richieste/risposte HTTP, la latenza della richiesta (l'intervallo di tempo compreso tra l'avvio di una richiesta e la ricezione di una risposta) limita il throughput che puoi ottenere da un singolo thread su un'unica connessione. Ad esempio, se la latenza da un client basato su Amazon EC2 ad Amazon SQS nella stessa regione è in media di 20 ms, la velocità di trasmissione effettiva massima da un singolo thread su un'unica connessione è in media di 50 TPS.

Per dimensionamento orizzontale si intende aumentare il numero di produttori di messaggi (che generano richieste [SendMessage](#)) e di consumatori di messaggi (che generano richieste [ReceiveMessage](#) e [DeleteMessage](#)) per aumentare il throughput complessivo della coda. È possibile dimensionare orizzontalmente in tre modi:

- Aumentare il numero di thread per client
- Aggiungi altri client
- Aumentare il numero di thread per client e aggiungere altri client

Aggiungendo più client dovresti ottenere essenzialmente guadagni lineari nel throughput della coda. Ad esempio, se raddoppi il numero di clienti, puoi ottenere il doppio del throughput.

Raggruppare le operazioni

Il Raggruppamento esegue più lavoro durante ogni round trip al servizio (ad esempio, quando invii più messaggi con una singola richiesta `SendMessageBatch`). Le operazioni Amazon SQS in batch sono [SendMessageBatch](#), [DeleteMessageBatch](#) e [ChangeMessageVisibilityBatch](#). Per usufruire del batching senza modificare produttori e consumatori, utilizza [Amazon SQS Buffered Asynchronous Client](#).

Note

Poiché [ReceiveMessage](#) può elaborare 10 messaggi alla volta, non c'è nessuna azione `ReceiveMessageBatch`.

Il raggruppamento distribuisce la latenza dell'operazione in batch su più messaggi in una richiesta batch, invece di accettare l'intera latenza per un solo messaggio (per esempio, una richiesta [SendMessage](#)). Poiché ogni round trip trasporta più lavoro, le richieste in batch fanno un utilizzo più efficiente di thread e connessioni, migliorano così il throughput.

Puoi abbinare il batching al dimensionamento orizzontale per offrire un throughput con un numero minore di thread, connessioni e richieste rispetto a quelli necessari per richieste di messaggi individuali. Puoi utilizzare le operazioni in batch di Amazon SQS per inviare, ricevere o eliminare fino a 10 messaggi alla volta. Poiché Amazon SQS addebita un costo per richiesta, il raggruppamento è in grado di ridurre in modo significativo i costi.

Il raggruppamento può introdurre alcune complessità per la tua applicazione (ad esempio, l'applicazione deve accumulare i messaggi prima di inviarli oppure talvolta è necessario attendere di più per una risposta). Tuttavia, il raggruppamento può essere ancora efficace nei seguenti casi:

- La tua applicazione genera una notevole quantità di messaggi in un breve periodo di tempo, pertanto il ritardo non è mai molto lungo.

- Il consumatore di un messaggio recupera i messaggi da una coda a sua discrezione, a differenza dei tipici produttori di messaggi che devono inviare messaggi in risposta a eventi che non controllano.

Important

Una richiesta di batch potrebbe avere esito positivo anche se i singoli messaggi nel batch hanno avuto esito negativo. Dopo una richiesta di batch, verifica sempre la presenza di errori di messaggio individuali e, se necessario, prova nuovamente l'operazione.

Esempio di utilizzo di Java Working per richieste con operazioni singole e in batch

Prerequisiti

Aggiungi i pacchetti `aws-java-sdk-sqs.jar`, `aws-java-sdk-ec2.jar` e `commons-logging.jar` al percorso di classe build Java. L'esempio seguente mostra queste dipendenze in un file `pom.xml` di progetto Maven.

```
<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

Il seguente esempio di codice Java implementa un semplice modello produttore-consumatore. Il thread principale genera dinamicamente diversi thread produttore e consumatore che elaborano messaggi di 1 KB per un periodo di tempo specificato. Questo esempio include produttori e consumatori che effettuano richieste con operazioni singole e altri che effettuano richieste in batch.

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

import com.amazonaws.AmazonClientException;
import com.amazonaws.ClientConfiguration;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Start a specified number of producer and consumer threads, and produce-consume
 * for the least of the specified duration and 1 hour. Some messages can be left
```

```
* in the queue because producers and consumers might not be in exact balance.
*/
public class SimpleProducerConsumer {

    // The maximum runtime of the program.
    private final static int MAX_RUNTIME_MINUTES = 60;
    private final static Log log = LogFactory.getLog(SimpleProducerConsumer.class);

    public static void main(String[] args) throws InterruptedException {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the number of producers: ");
        final int producerCount = input.nextInt();

        System.out.print("Enter the number of consumers: ");
        final int consumerCount = input.nextInt();

        System.out.print("Enter the number of messages per batch: ");
        final int batchSize = input.nextInt();

        System.out.print("Enter the message size in bytes: ");
        final int messageSizeByte = input.nextInt();

        System.out.print("Enter the run time in minutes: ");
        final int runTimeMinutes = input.nextInt();

        /*
        * Create a new instance of the builder with all defaults (credentials
        * and region) set automatically. For more information, see Creating
        * Service Clients in the AWS SDK for Java Developer Guide.
        */
        final ClientConfiguration clientConfiguration = new ClientConfiguration()
            .withMaxConnections(producerCount + consumerCount);

        final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
            .withClientConfiguration(clientConfiguration)
            .build();

        final String queueUrl = sqsClient
            .getQueueUrl(new GetQueueUrlRequest(queueName)).getQueueUrl();
```

```
// The flag used to stop producer, consumer, and monitor threads.
final AtomicBoolean stop = new AtomicBoolean(false);

// Start the producers.
final AtomicInteger producedCount = new AtomicInteger();
final Thread[] producers = new Thread[producerCount];
for (int i = 0; i < producerCount; i++) {
    if (batchSize == 1) {
        producers[i] = new Producer(sqsClient, queueUrl, messageSizeByte,
            producedCount, stop);
    } else {
        producers[i] = new BatchProducer(sqsClient, queueUrl, batchSize,
            messageSizeByte, producedCount,
            stop);
    }
    producers[i].start();
}

// Start the consumers.
final AtomicInteger consumedCount = new AtomicInteger();
final Thread[] consumers = new Thread[consumerCount];
for (int i = 0; i < consumerCount; i++) {
    if (batchSize == 1) {
        consumers[i] = new Consumer(sqsClient, queueUrl, consumedCount,
            stop);
    } else {
        consumers[i] = new BatchConsumer(sqsClient, queueUrl, batchSize,
            consumedCount, stop);
    }
    consumers[i].start();
}

// Start the monitor thread.
final Thread monitor = new Monitor(producedCount, consumedCount, stop);
monitor.start();

// Wait for the specified amount of time then stop.
Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runtimeMinutes,
    MAX_RUNTIME_MINUTES)));
stop.set(true);

// Join all threads.
for (int i = 0; i < producerCount; i++) {
```

```

        producers[i].join();
    }

    for (int i = 0; i < consumerCount; i++) {
        consumers[i].join();
    }

    monitor.interrupt();
    monitor.join();
}

private static String makeRandomString(int sizeByte) {
    final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
    new Random().nextBytes(bs);
    bs[0] = (byte) ((bs[0] | 64) & 127);
    return new BigInteger(bs).toString(32);
}

/**
 * The producer thread uses {@code SendMessage}
 * to send messages until it is stopped.
 */
private static class Producer extends Thread {
    final AmazonSQS sqsClient;
    final String queueUrl;
    final AtomicInteger producedCount;
    final AtomicBoolean stop;
    final String theMessage;

    Producer(AmazonSQS sqsQueueBuffer, String queueUrl, int messageSizeByte,
            AtomicInteger producedCount, AtomicBoolean stop) {
        this.sqsClient = sqsQueueBuffer;
        this.queueUrl = queueUrl;
        this.producedCount = producedCount;
        this.stop = stop;
        this.theMessage = makeRandomString(messageSizeByte);
    }

    /**
     * The producedCount object tracks the number of messages produced by
     * all producer threads. If there is an error, the program exits the
     * run() method.
     */
    public void run() {

```

```

        try {
            while (!stop.get()) {
                sqsClient.sendMessage(new SendMessageRequest(queueUrl,
                    theMessage));
                producedCount.incrementAndGet();
            }
        } catch (AmazonClientException e) {
            /*
             * By default, AmazonSQSClient retries calls 3 times before
             * failing. If this unlikely condition occurs, stop.
             */
            log.error("Producer: " + e.getMessage());
            System.exit(1);
        }
    }
}

/**
 * The producer thread uses {@code SendMessageBatch}
 * to send messages until it is stopped.
 */
private static class BatchProducer extends Thread {
    final AmazonSQS sqsClient;
    final String queueUrl;
    final int batchSize;
    final AtomicInteger producedCount;
    final AtomicBoolean stop;
    final String theMessage;

    BatchProducer(AmazonSQS sqsQueueBuffer, String queueUrl, int batchSize,
        int messageSizeByte, AtomicInteger producedCount,
        AtomicBoolean stop) {
        this.sqsClient = sqsQueueBuffer;
        this.queueUrl = queueUrl;
        this.batchSize = batchSize;
        this.producedCount = producedCount;
        this.stop = stop;
        this.theMessage = makeRandomString(messageSizeByte);
    }

    public void run() {
        try {
            while (!stop.get()) {
                final SendMessageBatchRequest batchRequest =

```



```

        new SendMessageBatchRequest().withQueueUrl(queueUrl);

    final List<SendMessageBatchRequestEntry> entries =
        new ArrayList<SendMessageBatchRequestEntry>();
    for (int i = 0; i < batchSize; i++)
        entries.add(new SendMessageBatchRequestEntry()
            .withId(Integer.toString(i))
            .withMessageBody(theMessage));
    batchRequest.setEntries(entries);

    final SendMessageBatchResult batchResult =
        sqsClient.sendMessageBatch(batchRequest);
    producedCount.addAndGet(batchResult.getSuccessful().size());

    /*
     * Because SendMessageBatch can return successfully, but
     * individual batch items fail, retry the failed batch items.
     */
    if (!batchResult.getFailed().isEmpty()) {
        log.warn("Producer: retrying sending "
            + batchResult.getFailed().size() + " messages");
        for (int i = 0, n = batchResult.getFailed().size();
            i < n; i++) {
            sqsClient.sendMessage(new
                SendMessageRequest(queueUrl, theMessage));
            producedCount.incrementAndGet();
        }
    }
}
} catch (AmazonClientException e) {
    /*
     * By default, AmazonSQSClient retries calls 3 times before
     * failing. If this unlikely condition occurs, stop.
     */
    log.error("BatchProducer: " + e.getMessage());
    System.exit(1);
}
}

/**
 * The consumer thread uses {@code ReceiveMessage} and {@code DeleteMessage}
 * to consume messages until it is stopped.
 */

```

```
private static class Consumer extends Thread {
    final AmazonSQS sqsClient;
    final String queueUrl;
    final AtomicInteger consumedCount;
    final AtomicBoolean stop;

    Consumer(AmazonSQS sqsClient, String queueUrl, AtomicInteger consumedCount,
            AtomicBoolean stop) {
        this.sqsClient = sqsClient;
        this.queueUrl = queueUrl;
        this.consumedCount = consumedCount;
        this.stop = stop;
    }

    /*
     * Each consumer thread receives and deletes messages until the main
     * thread stops the consumer thread. The consumedCount object tracks the
     * number of messages that are consumed by all consumer threads, and the
     * count is logged periodically.
     */
    public void run() {
        try {
            while (!stop.get()) {
                try {
                    final ReceiveMessageResult result = sqsClient
                        .receiveMessage(new
                            ReceiveMessageRequest(queueUrl));

                    if (!result.getMessages().isEmpty()) {
                        final Message m = result.getMessages().get(0);
                        sqsClient.deleteMessage(new
                            DeleteMessageRequest(queueUrl,
                                m.getReceiptHandle()));
                        consumedCount.incrementAndGet();
                    }
                } catch (AmazonClientException e) {
                    log.error(e.getMessage());
                }
            }
        } catch (AmazonClientException e) {
            /*
             * By default, AmazonSQSClient retries calls 3 times before
             * failing. If this unlikely condition occurs, stop.
             */
        }
    }
}
```

```

        log.error("Consumer: " + e.getMessage());
        System.exit(1);
    }
}

/**
 * The consumer thread uses {@code ReceiveMessage} and {@code
 * DeleteMessageBatch} to consume messages until it is stopped.
 */
private static class BatchConsumer extends Thread {
    final AmazonSQS sqsClient;
    final String queueUrl;
    final int batchSize;
    final AtomicInteger consumedCount;
    final AtomicBoolean stop;

    BatchConsumer(AmazonSQS sqsClient, String queueUrl, int batchSize,
        AtomicInteger consumedCount, AtomicBoolean stop) {
        this.sqsClient = sqsClient;
        this.queueUrl = queueUrl;
        this.batchSize = batchSize;
        this.consumedCount = consumedCount;
        this.stop = stop;
    }

    public void run() {
        try {
            while (!stop.get()) {
                final ReceiveMessageResult result = sqsClient
                    .receiveMessage(new ReceiveMessageRequest(queueUrl)
                        .withMaxNumberOfMessages(batchSize));

                if (!result.getMessages().isEmpty()) {
                    final List<Message> messages = result.getMessages();
                    final DeleteMessageBatchRequest batchRequest =
                        new DeleteMessageBatchRequest()
                            .withQueueUrl(queueUrl);

                    final List<DeleteMessageBatchRequestEntry> entries =
                        new ArrayList<DeleteMessageBatchRequestEntry>();
                    for (int i = 0, n = messages.size(); i < n; i++)
                        entries.add(new DeleteMessageBatchRequestEntry()
                            .withId(Integer.toString(i))

```

```

        .withReceiptHandle(messages.get(i)
            .getReceiptHandle()));
    batchRequest.setEntries(entries);

    final DeleteMessageBatchResult batchResult = sqsClient
        .deleteMessageBatch(batchRequest);
    consumedCount.addAndGet(batchResult.getSuccessful().size());

    /*
     * Because DeleteMessageBatch can return successfully,
     * but individual batch items fail, retry the failed
     * batch items.
     */
    if (!batchResult.getFailed().isEmpty()) {
        final int n = batchResult.getFailed().size();
        log.warn("Producer: retrying deleting " + n
            + " messages");
        for (BatchResultErrorEntry e : batchResult
            .getFailed()) {

            sqsClient.deleteMessage(
                new DeleteMessageRequest(queueUrl,
                    messages.get(Integer
                        .parseInt(e.getId()))
                        .getReceiptHandle()));

            consumedCount.incrementAndGet();
        }
    }
}
} catch (AmazonClientException e) {
    /*
     * By default, AmazonSQSClient retries calls 3 times before
     * failing. If this unlikely condition occurs, stop.
     */
    log.error("BatchConsumer: " + e.getMessage());
    System.exit(1);
}
}
}

/**
 * This thread prints every second the number of messages produced and

```

```
    * consumed so far.
    */
private static class Monitor extends Thread {
    private final AtomicInteger producedCount;
    private final AtomicInteger consumedCount;
    private final AtomicBoolean stop;

    Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
            AtomicBoolean stop) {
        this.producedCount = producedCount;
        this.consumedCount = consumedCount;
        this.stop = stop;
    }

    public void run() {
        try {
            while (!stop.get()) {
                Thread.sleep(1000);
                log.info("produced messages = " + producedCount.get()
                        + ", consumed messages = " + consumedCount.get());
            }
        } catch (InterruptedException e) {
            // Allow the thread to exit.
        }
    }
}
```

Monitoraggio delle metriche di volume dall'esecuzione di esempio

Amazon SQS genera automaticamente i parametri di volume per i messaggi inviati, ricevuti ed eliminati. Puoi accedere a tali parametri e ad altri tramite la scheda Monitoring (Monitoraggio) per la coda o nella [console CloudWatch](#).

Note

Le metriche possono diventare disponibili fino a 15 minuti dopo l'avvio della coda.

Utilizzo di Amazon SQS con un SDK AWS

AWS i kit di sviluppo software (SDKs) sono disponibili per molti linguaggi di programmazione più diffusi. Ogni SDK fornisce un'API, esempi di codice, e documentazione che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio preferito.

Documentazione sugli SDK	Esempi di codice
AWS SDK per C++	AWS SDK per C++ esempi di codice
AWS CLI	AWS CLI esempi di codice
AWS SDK per Go	AWS SDK per Go esempi di codice
AWS SDK per Java	AWS SDK per Java esempi di codice
AWS SDK per JavaScript	AWS SDK per JavaScript esempi di codice
AWS SDK per Kotlin	AWS SDK per Kotlin esempi di codice
AWS SDK per .NET	AWS SDK per .NET esempi di codice
AWS SDK per PHP	AWS SDK per PHP esempi di codice
AWS Strumenti per PowerShell	AWS Strumenti per PowerShell esempi di codice
AWS SDK per Python (Boto3)	AWS SDK per Python (Boto3) esempi di codice
AWS SDK per Ruby	AWS SDK per Ruby esempi di codice
AWS SDK per Rust	AWS SDK per Rust esempi di codice
AWS SDK per SAP ABAP	AWS SDK per SAP ABAP esempi di codice
AWS SDK per Swift	AWS SDK per Swift esempi di codice

 Esempio di disponibilità

Non riesci a trovare quello che ti serve? Richiedi un esempio di codice utilizzando il link [Provide feedback \(Fornisci un feedback\)](#) nella parte inferiore di questa pagina.

Utilizzo di JMS con Amazon SQS

Amazon SQS Java Messaging Library è un'interfaccia Java Message Service (JMS) per Amazon SQS che consente di sfruttare Amazon SQS in applicazioni che già utilizzano JMS. L'interfaccia consente di usare Amazon SQS come provider JMS, riducendo al minimo le modifiche al codice. Insieme a AWS SDK per Java, la raccolta di messaggistica Amazon SQS Java ti consente di creare connessioni e sessioni JMS, nonché produttori e consumatori in grado di inviare e ricevere messaggi da e verso code Amazon SQS.

La libreria supporta l'invio e la ricezione di messaggi in una coda (il point-to-point modello JMS) secondo la specifica [JMS 1.1](#). La libreria supporta l'invio di testo, byte o messaggi oggetto in modo sincrono a code Amazon SQS. La libreria supporta anche la ricezione di oggetti in modo sincrono o asincrono.

[Per informazioni sulle funzionalità della libreria di messaggistica Java di Amazon SQS che supportano la specifica JMS 1.1, consulta Amazon SQS supportava le implementazioni di JMS 1.1 Amazon SQS. FAQs](#)

Prerequisiti per lavorare con JMS e Amazon SQS

Prima di iniziare, devi disporre dei seguenti requisiti preliminari:

- SDK per Java

Ci sono due modi per includere l'SDK per Java nel tuo progetto:

- Scarica e installa l'SDK per Java.
- Utilizza Maven per ottenere la raccolta di messaggistica Amazon SQS Java.

Note

SDK per Java è incluso come una dipendenza.

Gli [SDK per Java](#) e la libreria client ampia Amazon SQS per Java richiedono J2SE Development Kit 8.0 o versioni successive.

Per informazioni su come scaricare il kit SDK AWS per Java, consulta [SDK per Java](#).

- Raccolta di messaggistica Amazon SQS Java

Se non utilizzi Maven, devi aggiungere il file del pacchetto `amazon-sqs-java-messaging-lib.jar` al percorso di classe Java. Per informazioni su come scaricare la libreria, consulta [raccolta di messaggistica Amazon SQS Java](#).

Note

La raccolta di messaggistica Amazon SQS Java include il supporto per [Maven](#) e [Spring Framework](#).

Per esempi di codice che utilizzano Maven, Spring Framework e la raccolta di messaggistica Amazon SQS Java, consulta [Esempi Java funzionanti per l'utilizzo di JMS con le code standard di Amazon SQS](#).

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
  <version>1.0.4</version>
  <type>jar</type>
</dependency>
```

- Coda Amazon SQS

Crea una coda utilizzando Console di gestione AWS per Amazon SQS, l'API o `CreateQueue` il client Amazon SQS wrappato incluso nella Amazon SQS Java Messaging Library.

- Per ulteriori informazioni su come creare una coda con Amazon SQS utilizzando Console di gestione AWS oppure l'API `CreateQueue`, consulta [Creazione di una coda](#).
- Per informazioni su come utilizzare la raccolta di messaggistica Amazon SQS Java, consulta [Utilizzo della libreria di messaggistica Java di Amazon SQS](#).

Utilizzo della libreria di messaggistica Java di Amazon SQS

Per iniziare a utilizzare il Servizio messaggi Java (JMS) con Amazon SQS, utilizza gli esempi di codice in questa sezione. Le seguenti sezioni illustrano come creare una connessione e una sessione JMS e come inviare e ricevere un messaggio.

L'oggetto client Amazon SQS incluso nella raccolta di messaggistica Amazon SQS Java verifica se esiste una coda Amazon SQS. Se la coda non esiste, il client la crea.

Creazione di una connessione JMS

Prima di iniziare, consulta i prerequisiti in: [Prerequisiti per lavorare con JMS e Amazon SQS](#)

1. Creare una connection factory e chiamare il metodo `createConnection` rispetto allo stesso.

```
// 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();
```

La classe `SQSConnection` estende `javax.jms.Connection`. Insieme ai metodi di connessione JMS standard, `SQSConnection` offre ulteriori metodi, ad esempio `getAmazonSQSClient` e `getWrappedAmazonSQSClient`. Entrambi i metodi consentono di eseguire operazioni di amministrazione non incluse nella specifica JMS, ad esempio la creazione di nuove code. Tuttavia, il metodo `getWrappedAmazonSQSClient` fornisce inoltre una versione integrata del client Amazon SQS utilizzato dalla connessione corrente. Il wrapper trasforma ogni eccezione dal client in una `JMSException`, che può essere utilizzata più facilmente dal codice esistente che si aspetta occorrenze `JMSException`.

2. Puoi utilizzare gli oggetti client restituiti da `getAmazonSQSClient` e `getWrappedAmazonSQSClient` per eseguire operazioni amministrative non incluse nella specifica JMS (ad esempio, è possibile creare una coda Amazon SQS).

Se disponi di codice esistente che si aspetta eccezioni JMS, devi utilizzare `getWrappedAmazonSQSClient`:

- Se utilizzi `getWrappedAmazonSQSClient`, l'oggetto client restituito trasforma tutte le eccezioni in eccezioni JMS.
- Se utilizzi `getAmazonSQSClient`, le eccezioni sono tutte eccezioni Amazon SQS.

Creazione di una coda Amazon SQS

L'oggetto client integrato verifica se esiste una coda Amazon SQS.

Se la coda non esiste, il client la crea. Se la coda esiste, la funzione non restituisce nulla. Per ulteriori informazioni, consulta la sezione "Creare la coda se necessario" nell'esempio [TextMessageSender.java](#).

Per creare una coda standard

```
// 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");
}
```

Per creare una coda FIFO

```
// 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

Il nome di una coda FIFO deve terminare con il suffisso `.fifo`. Per ulteriori informazioni sull'attributo `ContentBasedDeduplication`, consulta [Elaborazione una sola volta in Amazon SQS](#).

Invio di messaggi in modo sincrono

1. Quando la connessione e la coda Amazon SQS sottostante sono pronte, creare una sessione JMS nontransacted con la modalità `AUTO_ACKNOWLEDGE`.

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. Per inviare un messaggio di testo alla coda, creare una coda di identità JMS e un produttore del messaggio.

```
// 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. Creare un messaggio di testo e inviarlo alla coda.

- Per inviare un messaggio a una coda standard, non è necessario impostare parametri aggiuntivi.

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

- Per inviare un messaggio a una coda FIFO, è necessario impostare l'ID del gruppo messaggi. Puoi anche impostare un ID di deduplicazione messaggio. Per ulteriori informazioni, consulta [Termini chiave della coda FIFO di Amazon SQS](#).

```
// 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"));
```

Ricezione di messaggi in modo sincrono

1. Per ricevere messaggi, creare un consumatore per la stessa coda e chiamare il metodo `start`.

Puoi chiamare il metodo `start` sulla connessione in qualsiasi momento. Tuttavia, il consumatore non inizia a ricevere messaggi finché non lo chiami.

```
// Create a consumer for the 'MyQueue'
MessageConsumer consumer = session.createConsumer(queue);
// Start receiving incoming messages
connection.start();
```

2. Chiamare il metodo `receive` sul consumatore con un timeout impostati su 1 secondo, quindi stampare i contenuti del messaggio ricevuto.

- Dopo aver ricevuto un messaggio da una coda standard, puoi accedere al contenuto del messaggio.

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

- Dopo aver ricevuto un messaggio da una coda FIFO puoi accedere al contenuto del messaggio e ad altri attributi di messaggio specifici FIFO, ad esempio l'ID gruppo di messaggi, l'ID di deduplicazione messaggio e il numero di sequenza. Per ulteriori informazioni, consulta [Termini chiave della coda FIFO di Amazon SQS](#).

```
// 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. Chiudere la connessione e la sessione.

```
// Close the connection (and the session).
connection.close();
```

L'esito si presenta in maniera analoga all'immagine riportata di seguito.

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Note

Puoi utilizzare Spring Framework per inizializzare questi oggetti. Per ulteriori informazioni, consulta `SpringExampleConfiguration.xml`, `SpringExample.java` e le altre classi di supporto `ExampleConfiguration.java` e `ExampleCommon.java` nella sezione [Esempi Java funzionanti per l'utilizzo di JMS con le code standard di Amazon SQS](#).

Per completare esempi di invio e ricezione di oggetti, consulta [TextMessageSender.java](#) e [SyncMessageReceiver.java](#).

Ricezione di messaggi in modo asincrono

In questo esempio in [Utilizzo della libreria di messaggistica Java di Amazon SQS](#), viene inviato un messaggio a MyQueue e ricevuto in modo sincrono.

L'esempio seguente mostra come ricevere i messaggi in modo asincrono tramite un listener.

1. Implementare l'interfaccia `MessageListener`.

```
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();
        }
    }
}
```

Il metodo `onMessage` dell'interfaccia `MessageListener` viene chiamato quando si riceve un messaggio. In questa implementazione listener, il testo memorizzato nel messaggio è stampato.

2. Invece di chiamare esplicitamente il metodo `receive` sul consumatore, impostare il listener del messaggio del consumatore a un'istanza dell'implementazione `MyListener`. Il thread principale attende per un secondo.

```
// 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);
```

Gli altri passaggi sono identici a quelli dell'esempio [Utilizzo della libreria di messaggistica Java di Amazon SQS](#). Per un esempio completo di consumatore asincrono, consulta `AsyncMessageReceiver.java` [Esempi Java funzionanti per l'utilizzo di JMS con le code standard di Amazon SQS](#).

L'output per questo esempio appare simile al seguente:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2  
Received: Hello World!
```

Utilizzo della modalità di riconoscimento client

L'esempio in [Utilizzo della libreria di messaggistica Java di Amazon SQS](#) utilizza la modalità `AUTO_ACKNOWLEDGE` in cui ogni messaggio ricevuto viene confermato automaticamente (e quindi eliminato dalla coda Amazon SQS sottostante).

1. Per riconoscere i messaggi esplicitamente dopo l'elaborazione, è necessario creare la sessione con la modalità `CLIENT_ACKNOWLEDGE`.

```
// Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.  
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

2. Quando il messaggio viene ricevuto, visualizzarlo e quindi dichiararlo esplicitamente.

```
// 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

In questo modo, quando un messaggio viene confermato, tutti i messaggi ricevuti prima di esso sono implicitamente riconosciuti. Ad esempio, se 10 messaggi vengono ricevuti e solo il decimo messaggio viene confermato (nell'ordine in cui i messaggi vengono ricevuti), anche tutti i nove messaggi precedenti vengono riconosciuti.

Gli altri passaggi sono identici a quelli dell'esempio [Utilizzo della libreria di messaggistica Java di Amazon SQS](#). Per un esempio completo di un consumatore sincrono con modalità di riconoscimento client, consulta `SyncMessageReceiverClientAcknowledge.java` [Esempi Java funzionanti per l'utilizzo di JMS con le code standard di Amazon SQS](#).

L'output per questo esempio appare simile al seguente:


```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5  
Received: Hello World!  
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

Utilizzo della modalità di riconoscimento non ordinata

Quando utilizzi la modalità `CLIENT_ACKNOWLEDGE`, tutti i messaggi ricevuti prima di un messaggio esplicitamente riconosciuto sono riconosciuti automaticamente. Per ulteriori informazioni, consulta [Utilizzo della modalità di riconoscimento client](#).

La raccolta di messaggistica Amazon SQS Java fornisce un'altra modalità di conferma. Quando utilizzi la modalità `UNORDERED_ACKNOWLEDGE`, tutti i messaggi ricevuti devono essere individualmente ed esplicitamente riconosciuti dal client, indipendentemente dall'ordine di ricezione. Per eseguire questa operazione, è necessario creare una sessione con la modalità `UNORDERED_ACKNOWLEDGE`.

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.  
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

Gli altri passaggi sono identici a quelli dell'esempio [Utilizzo della modalità di riconoscimento client](#). Per un esempio completo di consumatore sincrono con la modalità `UNORDERED_ACKNOWLEDGE`, consulta `SyncMessageReceiverUnorderedAcknowledge.java`.

In questo esempio, l'output appare simile al seguente:

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7  
Received: Hello World!  
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Utilizzo di Java Message Service con altri client Amazon SQS

L'utilizzo del client Amazon SQS Java Message Service (JMS) con l'AWS SDK limita la dimensione dei messaggi Amazon SQS a 256 KB. Tuttavia, è possibile creare un provider JMS utilizzando qualsiasi client Amazon SQS. Ad esempio, puoi utilizzare il client JMS con la libreria client ampia di Amazon SQS per Java per inviare un messaggio Amazon SQS che contiene un riferimento a un payload del messaggio (fino a 2 GB) in Amazon S3. Per ulteriori informazioni, consulta [Gestione di messaggi Amazon SQS di grandi dimensioni con Java e Amazon S3](#).

Il seguente esempio di codice Java crea il provider JMS per la Extended Client Library.

Consultate i prerequisiti [Prerequisiti per lavorare con JMS e Amazon SQS](#) prima di testare questo esempio.

```
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);
```

Il seguente codice Java di esempio crea la connection factory:

```
// 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();
```

Esempi Java funzionanti per l'utilizzo di JMS con le code standard di Amazon SQS

Negli esempi di codice seguenti viene illustrato come utilizzare Servizio messaggi Java (JMS) con le code standard Amazon SQS. Per ulteriori informazioni sull'utilizzo delle code FIFO, consulta [Per creare una coda FIFO](#), [Invio di messaggi in modo sincrono](#) e [Ricezione di messaggi in modo sincrono](#). (La ricezione sincrona dei messaggi è la stessa per le code standard e FIFO. Tuttavia, i messaggi nelle code FIFO contengono più attributi).

[Prerequisiti per lavorare con JMS e Amazon SQS](#) Prima di testare i seguenti esempi, consulta i prerequisiti.

ExampleConfiguration.java

Il seguente esempio di codice Java SDK v 1.x imposta il nome di coda predefinito, la regione e le credenziali da utilizzare con gli altri esempi 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.
 */

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

Il seguente codice Java crea un produttore del messaggio di testo.

```
/*  
 * 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

Il seguente codice Java crea un consumatore del messaggio di testo sincrono.

```
/*
 * 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 (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}

```

AsyncMessageReceiver.java

Il seguente codice Java crea un consumatore del messaggio di testo asincrono.

```

/*
 * 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.getJMSMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSEException e) {
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}

```

SyncMessageReceiverClientAcknowledge.java

Il seguente esempio di codice Java crea un consumatore asincrono con modalità di riconoscimento client.

```

/*
 * 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

Il seguente esempio di codice Java crea un consumatore asincrono con modalità di riconoscimento non ordinato.

```

/*
 * 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

L'esempio di codice XML riportato di seguito è un file di configurazione bean per [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

Il seguente codice di esempio Java utilizza il file di configurazione bean per inizializzare i tuoi oggetti.

```

/*
 * 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

Il seguente codice di esempio Java verifica se una coda Amazon SQS esiste e quindi ne crea una se non esiste. Include anche il codice di registrazione di esempio.

```

/*
 * 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() );
    }
}
```

Amazon SQS supportava le implementazioni di JMS 1.1

La raccolta di messaggistica Amazon SQS Java supporta le seguenti [implementazioni JMS 1.1](#). Per ulteriori informazioni sulle caratteristiche e le funzionalità supportate dalla raccolta di messaggistica Amazon SQS Java, consulta le [domande frequenti su Amazon SQS](#).

Interfacce comuni supportate

- Connection
- ConnectionFactory
- Destination
- Session
- MessageConsumer
- MessageProducer

Tipi di messaggi supportati

- BytesMessage
- ObjectMessage
- TextMessage

Modalità di riconoscimento del messaggio supportate

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

Note

La modalità `UNORDERED_ACKNOWLEDGE` non fa parte della specifica JMS 1.1. Grazie a questa modalità, Amazon SQS può consentire a un client JMS di dichiarare esplicitamente un messaggio.

Intestazioni definite da JMS e proprietà riservate

Per l'invio di messaggi

Quando invii messaggi, puoi impostare le seguenti intestazioni e proprietà per ogni messaggio:

- `JMSXGroupID` (necessaria per le code FIFO, non consentita per le code standard)
- `JMS_SQS_DeduplicationId` (opzionale per le code FIFO, non consentita per le code standard)

Dopo che invii messaggi, Amazon SQS imposta le seguenti intestazioni e proprietà per ogni messaggio:

- `JMSMessageID`
- `JMS_SQS_SequenceNumber` (solo per code FIFO)

Per la ricezione di messaggi

Quando ricevi messaggi, Amazon SQS imposta le seguenti intestazioni e proprietà per ogni messaggio:

- `JMSDestination`
- `JMSMessageID`

- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID` (solo per code FIFO)
- `JMS_SQS_DeduplicationId` (solo per code FIFO)
- `JMS_SQS_SequenceNumber` (solo per code FIFO)

Tutorial per Amazon SQS

Questo argomento fornisce tutorial per aiutarti a esplorare le caratteristiche e le funzionalità di Amazon SQS.

Tutorial

- [Creazione di una coda Amazon SQS utilizzando CloudFormation](#)
- [Tutorial: Invio di un messaggio a una coda Amazon SQS da Amazon Virtual Private Cloud](#)

Creazione di una coda Amazon SQS utilizzando CloudFormation

Usa la CloudFormation console insieme a un modello JSON o YAML per creare una coda Amazon SQS. Per maggiori dettagli, consulta [Lavorare con i CloudFormation modelli e la AWS::SQS::Queue](#) risorsa nella Guida per l'utente AWS CloudFormation

Da utilizzare CloudFormation per creare una coda Amazon SQS.

1. Copia il seguente codice JSON in un file denominato `MyQueue.json`. Per creare una coda standard, ometti le proprietà `FifoQueue` e `ContentBasedDeduplication`. Per ulteriori informazioni sulla deduplicazione basata sui contenuti, consulta [Elaborazione una sola volta in Amazon SQS](#).

Note

Il nome di una coda FIFO deve terminare con il suffisso `.fifo`.

```
{
  "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. Accedi alla [console CloudFormation](#), quindi scegli Create Stack (Crea stack).
3. Nel pannello Specify Template (Specifica modello) scegliere Upload a template file (Carica file modello), scegliere il file `MyQueue.json`, quindi scegliere Next (Successivo).
4. Nella pagina Specify Details (Specifica dettagli), digita `MyQueue` per Stack Name (Nome stack), quindi scegli Next (Avanti).
5. Nella pagina Opzioni, scegli Next (Avanti).
6. Nella pagina Revisione scegli Create (Crea).

CloudFormation inizia a creare lo **MyQueue** stack e visualizza lo stato `CREATE_IN_PROGRESS`. Al termine del processo, CloudFormation mostra lo stato `CREATE_COMPLETE`.

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. (Facoltativo) Per visualizzare il nome, l'URL e l'ARN della coda, scegli il nome dello stack e quindi nella pagina successiva espandi la sezione Outputs.

Tutorial: Invio di un messaggio a una coda Amazon SQS da Amazon Virtual Private Cloud

Questo tutorial mostra come inviare messaggi a una coda Amazon SQS su una rete privata sicura. La rete include:

- Un VPC contenente un'istanza Amazon EC2.
- Un endpoint VPC di interfaccia, che consente all'istanza Amazon EC2 di connettersi ad Amazon SQS senza utilizzare la rete Internet pubblica.

Anche in una rete completamente privata, puoi connetterti all'istanza Amazon EC2 e inviare messaggi alla coda Amazon SQS. Per ulteriori informazioni, consulta [Endpoint di Amazon Virtual Private Cloud per Amazon SQS](#).

Important

- Puoi utilizzare Amazon Virtual Private Cloud solo con endpoint Amazon SQS HTTPS.
- Quando configuri Amazon SQS per inviare messaggi da Amazon VPC, devi abilitare il DNS privato e specificare gli endpoint nel formato `sqs.us-east-2.amazonaws.com` o per l'endpoint dual-stack. `sqs.us-east-2.api.aws`
- Amazon SQS supporta anche gli endpoint FIPS tramite l' PrivateLink utilizzo del servizio endpoint. `com.amazonaws.region.sqs-fips` Puoi connetterti agli endpoint FIPS nel formato. `sqs-fips.region.amazonaws.com`
- Quando si utilizza l'endpoint dual-stack in Amazon Virtual Private Cloud, le richieste verranno inviate utilizzando e. IPv4 IPv6
- Il DNS privato non supporta endpoint precedenti come `queue.amazonaws.com` o `us-east-2.queue.amazonaws.com`.

Fase 1: Creazione di una coppia di chiavi di Amazon EC2

Una coppia di chiavi consente di effettuare la connessione a un'istanza Amazon EC2. È costituita da una chiave pubblica per la crittografia delle informazioni di accesso e una chiave privata per decrittografare tali informazioni.

1. Accedi alla [console Amazon EC2](#).
2. Nel menu di navigazione, in Network & Security (Rete e sicurezza), scegliere Key Pairs (Coppie di chiavi).
3. Scegli Crea coppia di chiavi.
4. Nella finestra di dialogo Create Key Pair (Crea coppia di chiavi), per Key pair name (Nome coppia di chiavi), immettere `SQS-VPCE-Tutorial-Key-Pair`, quindi scegliere Create (Crea).
5. Il browser scarica automaticamente il file della chiave privata `SQS-VPCE-Tutorial-Key-Pair.pem`.

Important

Salvare il file in un percorso sicuro. EC2 non genera un secondo file `.pem` per la stessa coppia di chiavi.

6. Per consentire a un client SSH di connettersi all'istanza EC2, impostare le autorizzazioni per il file della chiave privata in modo che solo l'utente specificato abbia le relative autorizzazioni in lettura. Ad esempio:

```
chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
```

Fase 2: Creare risorse AWS

Per configurare l'infrastruttura necessaria, è necessario utilizzare un CloudFormation modello, che è un modello per la creazione di uno stack composto da AWS risorse, come istanze Amazon EC2 e code Amazon SQS.

Lo stack per questo tutorial include le risorse seguenti:

- Un VPC e le risorse di rete associate, tra cui una sottorete, un gruppo di sicurezza, un gateway Internet e una tabella di routing

- Un'istanza Amazon EC2 avviata nella sottorete VPC
- Una coda Amazon SQS

1. Scarica il CloudFormation modello denominato [SQS-VPCE-Tutorial-CloudFormation.yaml](#) da GitHub
2. Accedere alla [console CloudFormation](#).
3. Scegli Create Stack (Crea stack).
4. Nella pagina Select Template (Seleziona modello), scegliere Upload a template to Amazon S3 (Carica un modello in Amazon S3), selezionare il file `SQS-VPCE-SQS-Tutorial-CloudFormation.yaml`, quindi scegliere Next (Successivo).
5. Nella pagina Specify Details (Specifica dettagli), procedi come segue:
 - a. In Nome stack, immetti `SQS-VPCE-Tutorial-Stack`.
 - b. Per KeyName, scegliete `SQS-VPCE-Tutorial-Key-Pair`.
 - c. Scegli Next (Successivo).
6. Nella pagina Opzioni, scegli Avanti.
7. Nella pagina Revisione, nella sezione Funzionalità, scegli Riconosco che potrebbe creare risorse IAM con nomi personalizzati. AWS CloudFormation , quindi scegli Crea.

CloudFormation inizia a creare lo stack e visualizza lo stato `CREATE_IN_PROGRESS`. Al termine del processo, CloudFormation mostra lo stato `CREATE_COMPLETE`.

Fase 3: Verifica del fatto che l'istanza EC2 non è accessibile pubblicamente

Il CloudFormation modello avvia un'istanza EC2 denominata `SQS-VPCE-Tutorial-EC2-Instance` nel tuo VPC. Tale istanza EC2 non consente il traffico in uscita e non è in grado di inviare messaggi a Amazon SQS. Per verificare ciò, devi connetterti all'istanza, provare a effettuare una connessione a un endpoint pubblico e tentare di inviare un messaggio a Amazon SQS.

1. Accedi alla [console Amazon EC2](#).
2. Nel menu di navigazione, in Istanze, scegli Istanze.
3. Seleziona `SQS-VPCE-. Tutorial-EC2Instance`
4. Copia il nome host in Public DNS, ad esempio `ec2-203-0-113-0.us-west-2.compute.amazonaws.com`.

5. Dalla directory contenente [la coppia di chiavi creata in precedenza](#), effettuare la connessione all'istanza con il seguente comando. Ad esempio:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. Provare a effettuare la connessione a un qualsiasi endpoint pubblico. Ad esempio:

```
ping amazon.com
```

Il tentativo di connessione avrà esito negativo, come previsto.

7. Accedere alla [console Amazon SQS](#).
8. Dall'elenco delle code, seleziona la coda creata dal tuo modello, ad esempio VPCE-sqs-tutorial-stack- -1 IJK. CloudFormation CFQueue ABCDEFGH2
9. Nella tabella Dettagli, copia l'URL, ad esempio. `https://sqs.us-east-2.amazonaws.com/123456789012/`
10. Dall'istanza EC2, provare a pubblicare un messaggio nella coda utilizzando il seguente comando. Ad esempio:

```
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."
```

Il tentativo di invio avrà esito negativo, come previsto.

Important

Successivamente, una volta creato un endpoint VPC per Amazon SQS, il tentativo di invio andrà a buon fine.

Fase 4: Creazione di un endpoint Amazon VPC per Amazon SQS

Per collegare il tuo VPC a Amazon SQS, definisci un endpoint VPC dell'interfaccia. Dopo avere aggiunto l'endpoint, sarà possibile utilizzare l'API Amazon SQS dall'istanza EC2 nel VPC. Ciò consente di inviare messaggi a una coda all'interno della AWS rete senza attraversare la rete Internet pubblica.

 Note

L'istanza EC2 non ha ancora accesso ad altri AWS servizi ed endpoint su Internet.

1. Accedere alla [console Amazon VPC](#).
2. Nel menu di navigazione, scegliere Endpoints (Endpoint).
3. Scegliere Create Endpoint (Crea endpoint).
4. Nella pagina Crea endpoint, per Nome servizio, scegliere il nome del servizio per Amazon SQS.

 Note

I nomi dei servizi variano in base alla regione corrente AWS . Ad esempio, se ti trovi negli Stati Uniti orientali (Ohio), il nome del servizio è com.amazonaws. **us-east-2**.sqs.

5. Per VPC, scegliere SQS-VPCE-Tutorial-VPC.
6. Per Subnets (Sottoreti), scegliere la sottorete il cui Subnet ID (ID sottorete) include SQS-VPCE-Tutorial-Subnet.
7. Per Security group (Gruppo di sicurezza), scegliere Select security groups (Seleziona gruppi di sicurezza), quindi scegliere il gruppo di sicurezza il cui Group Name (Nome gruppo) include SQS VPCE Tutorial Security Group (Gruppo di sicurezza tutorial SQS VPCE).
8. Seleziona Crea endpoint.

Viene creato l'endpoint VPC e ne viene visualizzato il relativo ID. Ad esempio, vpce-0ab1cdef2ghi3j456k.

9. Scegli Chiudi.

La console Amazon VPC apre la pagina Endpoints.

Amazon VPC inizia a creare l'endpoint e visualizza lo stato in attesa. Al termine del processo, Amazon VPC visualizza lo stato disponibile.

Fase 5: Invio di un messaggio alla coda Amazon SQS

Il VPC include un endpoint per Amazon SQS. Puoi connetterti all'istanza EC2 e inviare messaggi alla coda.

1. Effettuare nuovamente la connessione all'istanza EC2. Ad esempio:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

2. Provare a pubblicare nuovamente un messaggio nella coda utilizzando il seguente comando. Ad esempio:

```
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."
```

Il tentativo di invio ha esito positivo e vengono visualizzati il MD5 riepilogo del corpo del messaggio e dell'ID del messaggio, ad esempio:

```
{
  "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
  "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

Per informazioni sulla ricezione e l'eliminazione del messaggio dalla coda creata dal CloudFormation modello (ad esempio, VPCE-sqs-Tutorial-Stack - -1 IJK), vedere. CFQueue ABCDEFGH2 [Ricezione ed eliminazione di un messaggio in Amazon SQS](#)

Per informazioni sull'eliminazione delle risorse, consulta quanto segue:

- [Eliminazione di un endpoint VPC](#) nella Guida per l'utente di Amazon VPC
- [Eliminazione di una coda Amazon SQS](#)
- [Termina l'istanza nella Guida](#) per l'utente di Amazon EC2
- [Eliminazione di un VPC](#) nella Guida per l'utente di Amazon VPC
- [Eliminazione di uno stack sulla CloudFormation console nella Guida per l'utente AWS CloudFormation](#)
- [Eliminazione della coppia di chiavi nella Guida](#) per l'utente di Amazon EC2

Esempi di codice per l'utilizzo di Amazon SQS AWS SDKs

I seguenti esempi di codice mostrano come usare Amazon SQS con un kit di sviluppo AWS software (SDK).

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Per un elenco completo di guide ed esempi di codice per sviluppatori AWS SDK, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi di codice

- [Esempi di base per l'utilizzo di Amazon SQS AWS SDKs](#)
 - [Hello Amazon SQS](#)
 - [Azioni per l'utilizzo di Amazon SQS AWS SDKs](#)
 - [Utilizzare AddPermission con una CLI](#)
 - [Utilizzo ChangeMessageVisibility con un AWS SDK o una CLI](#)
 - [Utilizzare ChangeMessageVisibilityBatch con una CLI](#)
 - [Utilizzo CreateQueue con un AWS SDK o una CLI](#)
 - [Utilizzo DeleteMessage con un AWS SDK o una CLI](#)
 - [Utilizzo DeleteMessageBatch con un AWS SDK o una CLI](#)
 - [Utilizzo DeleteQueue con un AWS SDK o una CLI](#)
 - [Utilizzo GetQueueAttributes con un AWS SDK o una CLI](#)
 - [Utilizzo GetQueueUrl con un AWS SDK o una CLI](#)
 - [Utilizzare ListDeadLetterSourceQueues con una CLI](#)
 - [Utilizzo ListQueues con un AWS SDK o una CLI](#)
 - [Utilizzare PurgeQueue con una CLI](#)
 - [Utilizzo ReceiveMessage con un AWS SDK o una CLI](#)
 - [Utilizzare RemovePermission con una CLI](#)

- [Utilizzo SendMessage con un AWS SDK o una CLI](#)
- [Utilizzo SendMessageBatch con un AWS SDK o una CLI](#)
- [Utilizzo SetQueueAttributes con un AWS SDK o una CLI](#)
- [Scenari per l'utilizzo di Amazon SQS AWS SDKs](#)
 - [Creare un'applicazione web che invia e recupera i messaggi utilizzando Amazon SQS](#)
 - [Creazione di un'applicazione di messaggistica con Step Functions](#)
 - [Creazione di un'applicazione Amazon Textract explorer](#)
 - [Crea e pubblica su un argomento FIFO Amazon SNS utilizzando un SDK AWS](#)
 - [Rileva persone e oggetti in un video con Amazon Rekognition utilizzando un SDK AWS](#)
 - [Gestisci messaggi Amazon SQS di grandi dimensioni utilizzando Amazon S3 con un SDK AWS](#)
 - [Ricevi ed elabora le notifiche degli eventi di Amazon S3 utilizzando un SDK AWS](#)
 - [Pubblica messaggi Amazon SNS nelle code Amazon SQS utilizzando un SDK AWS](#)
 - [Invia e ricevi batch di messaggi con Amazon SQS utilizzando un SDK AWS](#)
 - [Utilizza il AWS Message Processing Framework for .NET per pubblicare e ricevere messaggi Amazon SQS](#)
 - [Utilizzare la raccolta di messaggistica Amazon SQS Java per utilizzare l'interfaccia Servizio messaggi Java \(JMS\) per Amazon SQS](#)
 - [Lavora con i tag di coda e Amazon SQS utilizzando un SDK AWS](#)
- [Esempi serverless per Amazon SQS](#)
 - [Invocazione di una funzione Lambda da un trigger Amazon SQS](#)
 - [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS](#)

Esempi di base per l'utilizzo di Amazon SQS AWS SDKs

I seguenti esempi di codice mostrano come utilizzare le nozioni di base di Amazon Simple Queue Service con AWS SDKs

Esempi

- [Hello Amazon SQS](#)
- [Azioni per l'utilizzo di Amazon SQS AWS SDKs](#)
 - [Utilizzare AddPermission con una CLI](#)
 - [Utilizzo ChangeMessageVisibility con un AWS SDK o una CLI](#)

- [Utilizzare ChangeMessageVisibilityBatch con una CLI](#)
- [Utilizzo CreateQueue con un AWS SDK o una CLI](#)
- [Utilizzo DeleteMessage con un AWS SDK o una CLI](#)
- [Utilizzo DeleteMessageBatch con un AWS SDK o una CLI](#)
- [Utilizzo DeleteQueue con un AWS SDK o una CLI](#)
- [Utilizzo GetQueueAttributes con un AWS SDK o una CLI](#)
- [Utilizzo GetQueueUrl con un AWS SDK o una CLI](#)
- [Utilizzare ListDeadLetterSourceQueues con una CLI](#)
- [Utilizzo ListQueues con un AWS SDK o una CLI](#)
- [Utilizzare PurgeQueue con una CLI](#)
- [Utilizzo ReceiveMessage con un AWS SDK o una CLI](#)
- [Utilizzare RemovePermission con una CLI](#)
- [Utilizzo SendMessage con un AWS SDK o una CLI](#)
- [Utilizzo SendMessageBatch con un AWS SDK o una CLI](#)
- [Utilizzo SetQueueAttributes con un AWS SDK o una CLI](#)

Hello Amazon SQS

Gli esempi di codice seguenti mostrano come iniziare a utilizzare Amazon SQS.

.NET

SDK per .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
        }
    }
}
```

- Per i dettagli sull'API, consulta la [ListQueues](#) sezione AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Codice per il CMake file 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})
```

Codice per il file di origine 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;
}

```

- Per i dettagli sull'API, consulta la [ListQueues](#) sezione AWS SDK per C++ API Reference.

Go

SDK per Go V2

 Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
    }
}
}

```

- Per i dettagli sull'API, consulta la [ListQueues](#) sezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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);
        }
    }
}
```

- Per i dettagli sull'API, consulta la [ListQueues](#) sezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Inizializza un client Amazon SQS ed elenca le code.

```
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"));
};
```

- Per i dettagli sull'API, consulta la [ListQueues](#) sezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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")
            }
    }
}
```

- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK for Kotlin API reference.

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Il file `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")
]
)

```

Il codice sorgente Swift 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)
        }
    }
}
```

- Per i dettagli sull'API, consulta la [ListQueues](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Azioni per l'utilizzo di Amazon SQS AWS SDKs

I seguenti esempi di codice mostrano come eseguire singole azioni Amazon SQS con. AWS SDKs Ogni esempio include un collegamento a GitHub, dove puoi trovare le istruzioni per la configurazione e l'esecuzione del codice.

Questi estratti chiamano l'API Amazon SQS e sono estratti di codice da programmi più grandi che devono essere eseguiti in modo contestuale. È possibile visualizzare le azioni nel contesto in [Scenari per l'utilizzo di Amazon SQS AWS SDKs](#).

Gli esempi seguenti includono solo le azioni più comunemente utilizzate. Per un elenco completo, consulta la [documentazione di riferimento dell'API Amazon Simple Queue Service](#).

Esempi

- [Utilizzare AddPermission con una CLI](#)
- [Utilizzo ChangeMessageVisibility con un AWS SDK o una CLI](#)
- [Utilizzare ChangeMessageVisibilityBatch con una CLI](#)
- [Utilizzo CreateQueue con un AWS SDK o una CLI](#)
- [Utilizzo DeleteMessage con un AWS SDK o una CLI](#)

- [Utilizzo DeleteMessageBatch con un AWS SDK o una CLI](#)
- [Utilizzo DeleteQueue con un AWS SDK o una CLI](#)
- [Utilizzo GetQueueAttributes con un AWS SDK o una CLI](#)
- [Utilizzo GetQueueUrl con un AWS SDK o una CLI](#)
- [Utilizzare ListDeadLetterSourceQueues con una CLI](#)
- [Utilizzo ListQueues con un AWS SDK o una CLI](#)
- [Utilizzare PurgeQueue con una CLI](#)
- [Utilizzo ReceiveMessage con un AWS SDK o una CLI](#)
- [Utilizzare RemovePermission con una CLI](#)
- [Utilizzo SendMessage con un AWS SDK o una CLI](#)
- [Utilizzo SendMessageBatch con un AWS SDK o una CLI](#)
- [Utilizzo SetQueueAttributes con un AWS SDK o una CLI](#)

Utilizzare **AddPermission** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare AddPermission.

CLI

AWS CLI

Come aggiungere le autorizzazioni a una coda

Questo esempio consente all' AWS account specificato di inviare messaggi alla coda specificata.

Comando:

```
aws sqs add-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessageFromMyQueue --aws-account-ids 12345EXAMPLE --actions SendMessage
```

Output:

```
None .
```

- Per i dettagli sull'API, vedere [AddPermission](#) in AWS CLI Command Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: Questo esempio consente all'utente specificato Account AWS di inviare messaggi dalla coda specificata.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE  
-Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-  
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [AddPermission](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: Questo esempio consente all'utente specificato Account AWS di inviare messaggi dalla coda specificata.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE  
-Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-  
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [AddPermission](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **ChangeMessageVisibility** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare ChangeMessageVisibility.

C++

SDK per C++

 Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    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();
}
```

- Per i dettagli sull'API, consulta la [ChangeMessageVisibility](#) sezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come modificare la visibilità del timeout di un messaggio

Questo esempio modifica la visibilità del timeout del messaggio specificato impostandola su 10 ore (10 ore * 60 minuti * 60 secondi).

Comando:

```
aws sqs change-message-visibility --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --visibility-timeout 36000
```

Output:

```
None.
```

- Per i dettagli sull'API, consulta [ChangeMessageVisibility AWS CLI](#) Command Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve un messaggio Amazon SQS e ne modifica la visibilità del timeout.

```
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;
};
```

- Per i dettagli sull'API, consulta la [ChangeMessageVisibility](#) sezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve un messaggio Amazon SQS e ne modifica la visibilità del timeout.

```
// 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");
}
}
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, consulta la [ChangeMessageVisibility](#) sezione AWS SDK per JavaScript API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio modifica il timeout della visibilità del messaggio con l'handle di ricezione specificato nella coda specificata portandolo a 10 ore (10 ore * 60 minuti * 60 secondi = 36000 secondi).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Per i dettagli sull'API, vedere [ChangeMessageVisibility](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio modifica il timeout della visibilità del messaggio con l'handle di ricezione specificato nella coda specificata portandolo a 10 ore (10 ore * 60 minuti * 60 secondi = 36000 secondi).


```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Per i dettagli sull'API, vedere [ChangeMessageVisibility](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Ruby

SDK per Ruby

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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

```

- Per i dettagli sull'API, consulta la [ChangeMessageVisibility](#) sezione AWS SDK per Ruby API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **ChangeMessageVisibilityBatch** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare **ChangeMessageVisibilityBatch**.

CLI

AWS CLI

Come modificare la visibilità del timeout di più messaggi come batch

Questo esempio modifica la visibilità del timeout dei 2 messaggi specificati impostandola su 10 ore (10 ore * 60 minuti * 60 secondi).

Comando:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

File di input (change-message-visibility-batch.json):

```
[
  {
    "Id": "FirstMessage",
    "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
    "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
    "ReceiptHandle": "AQEBkTUH...HifSnw==",
    "VisibilityTimeout": 36000
  }
]
```

Output:

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

- Per i dettagli sull'API, consulta AWS CLI Command [ChangeMessageVisibilityBatch](#) Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio modifica il timeout della visibilità di 2 messaggi con gli handle di ricezione specificati nella coda specificata. Il timeout della visibilità del primo messaggio

viene impostato su 10 ore (10 ore * 60 minuti * 60 secondi = 36000 secondi). Il timeout della visibilità del secondo messaggio viene impostata su 5 ore (5 ore * 60 minuti * 60 secondi = 18000 secondi).

```
$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-SQSMMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
    $changeVisibilityRequest2
```

Output:

```
Failed    Successful
-----
{}        {Request2, Request1}
```

- Per i dettagli sull'API, vedere [ChangeMessageVisibilityBatch](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio modifica il timeout della visibilità di 2 messaggi con gli handle di ricezione specificati nella coda specificata. Il timeout della visibilità del primo messaggio viene impostato su 10 ore (10 ore * 60 minuti * 60 secondi = 36000 secondi). Il timeout della visibilità del secondo messaggio viene impostata su 5 ore (5 ore * 60 minuti * 60 secondi = 18000 secondi).

```
$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-SQSMessagesVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
    $changeVisibilityRequest2

```

Output:

```

Failed      Successful
-----
{}          {Request2, Request1}

```

- Per i dettagli sull'API, vedere [ChangeMessageVisibilityBatch](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **CreateQueue** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare CreateQueue.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Pubblicazione di messaggi nelle code](#)
- [Inviare e ricevere batch di messaggi](#)
- [Utilizzare la raccolta di messaggistica Amazon SQS Java per utilizzare l'interfaccia JMS](#)

.NET

SDK per .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una coda con un nome specifico.

```
/// <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;
}

```

Crea una coda Amazon SQS e invia un messaggio a tale coda.

```

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;
    }
}

```

- Per i dettagli sull'API, consulta la [CreateQueue](#) sezione AWS SDK per .NET API Reference.

C++

SDK per C++

 Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    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();
    }

```

- Per i dettagli sull'API, consulta la [CreateQueue](#) sezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come creare una coda

Questo esempio crea una coda con il nome specificato, imposta il periodo di conservazione dei messaggi su 3 giorni (3 giorni * 24 ore * 60 minuti * 60 secondi) e imposta la coda DLQ della coda sulla coda specificata con un conteggio massimo di ricezione pari a 1.000 messaggi.

Comando:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

File di input (create-queue.json):

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-east-1:80398EXAMPLE:MyDeadLetterQueue\", \"maxReceiveCount\": \"1000\"}\",
  \"MessageRetentionPeriod\": \"259200\"
}
```

Output:

```
{
  \"QueueUrl\": \"https://queue.amazonaws.com/80398EXAMPLE/MyQueue\"
}
```

- Per i dettagli sull'API, consulta [CreateQueue AWS CLI](#) Command Reference.

Go

SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sull'API, consulta la [CreateQueue](#) sezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void listQueues(SqsClient sqsClient) {

        System.out.println("\nList Queues");
        String prefix = "que";

        try {
            ListQueuesRequest listQueuesRequest =
                ListQueuesRequest.builder().queueNamePrefix(prefix).build();
            ListQueuesResponse listQueuesResponse =
                sqsClient.listQueues(listQueuesRequest);
            for (String url : listQueuesResponse.queueUrls()) {
                System.out.println(url);
            }
        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void listQueuesFilter(SqsClient sqsClient, String queueUrl) {
        // List queues with filters
        String namePrefix = "queue";
        ListQueuesRequest filterListRequest = ListQueuesRequest.builder()
            .queueNamePrefix(namePrefix)
            .build();

        ListQueuesResponse listQueuesFilteredResponse =
            sqsClient.listQueues(filterListRequest);
        System.out.println("Queue URLs with prefix: " + namePrefix);
        for (String url : listQueuesFilteredResponse.queueUrls()) {
            System.out.println(url);
        }

        System.out.println("\nSend message");
        try {
            sqsClient.sendMessage(SendMessageRequest.builder()
```

```
        .queueUrl(queueUrl)
        .messageBody("Hello world!")
        .delaySeconds(10)
        .build());

    } catch (SqsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void sendBatchMessages(SqsClient sqsClient, String queueUrl) {

    System.out.println("\nSend multiple messages");
    try {
        SendMessageBatchRequest sendMessageBatchRequest =
SendMessageBatchRequest.builder()
            .queueUrl(queueUrl)

.entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello
from msg 1").build(),

SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
2").delaySeconds(10)

                .build())
            .build();
        sqsClient.sendMessageBatch(sendMessageBatchRequest);

    } catch (SqsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static List<Message> receiveMessages(SqsClient sqsClient, String
queueUrl) {

    System.out.println("\nReceive messages");
    try {
        ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
            .queueUrl(queueUrl)
            .maxNumberOfMessages(5)
            .build();
```



```
        return sqsClient.receiveMessage(receiveMessageRequest).messages();

    } catch (SqsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

public static void changeMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {

    System.out.println("\nChange Message Visibility");
    try {

        for (Message message : messages) {
            ChangeMessageVisibilityRequest req =
ChangeMessageVisibilityRequest.builder()
                .queueUrl(queueUrl)
                .receiptHandle(message.receiptHandle())
                .visibilityTimeout(100)
                .build();
            sqsClient.changeMessageVisibility(req);
        }

    } catch (SqsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {
    System.out.println("\nDelete Messages");

    try {
        for (Message message : messages) {
            DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                .queueUrl(queueUrl)
                .receiptHandle(message.receiptHandle())
                .build();
            sqsClient.deleteMessage(deleteMessageRequest);
        }
    }
```

```
        } catch (SqsException e) {  
            System.err.println(e.awsErrorDetails().errorMessage());  
            System.exit(1);  
        }  
    }  
}
```

- Per i dettagli sull'API, consulta la [CreateQueue](#) sezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una coda standard 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;  
};
```

Crea una coda Amazon SQS con polling lungo.

```
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;
};
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, consulta la [CreateQueue](#) sezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una coda standard 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);
  }
});
```

Crea una coda Amazon SQS che rimane in attesa dell'arrivo di un messaggio.

```
// 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);
  }
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, consulta la [CreateQueue](#) sezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 getUrlResponse.queueUrl.toString()
    }
}
```

- Per i dettagli sull'API, [CreateQueue](#) consulta AWS SDK for Kotlin API reference.

PowerShell

Strumenti per V4 PowerShell

Esempio 1: questo esempio crea una coda con il nome specificato.

```
New-SQSQueue -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [CreateQueue](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio crea una coda con il nome specificato.

```
New-SQSQueue -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [CreateQueue](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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

```

- Per i dettagli sull'API, consulta [CreateQueue AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK per Ruby

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

# 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__
```

- Per i dettagli sull'API, consulta la [CreateQueue](#) sezione AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una coda standard 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_sqsqueue_name_exists.
```

```
MESSAGE 'A queue with this name already exists.' TYPE 'E'.
ENDTRY.
```

Crea una coda Amazon SQS che rimane in attesa dell'arrivo di un messaggio.

```
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.
```

- Per i dettagli sulle API, [CreateQueue](#) consulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sull'API, consulta la [CreateQueue](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **DeleteMessage** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare DeleteMessage.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- [Inviare e ricevere batch di messaggi](#)

.NET

SDK per .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve un messaggio da una coda Amazon SQS e poi elimina il messaggio.

```
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 queue 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;
    }

```

```
}
```

- Per i dettagli sull'API, consulta la [DeleteMessage](#) sezione AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
}
```

- Per i dettagli sull'API, consulta la [DeleteMessage](#) sezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come eliminare un messaggio

Questo esempio elimina il messaggio specificato.

Comando:

```
aws sqs delete-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

Output:

```
None .
```

- Per i dettagli sull'API, consulta [DeleteMessage AWS CLI](#) Command Reference.

Java

SDK per Java 2.x

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
```

- Per i dettagli sull'API, consulta la [DeleteMessage](#) sezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve ed elimina messaggi Amazon SQS.

```
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,
    })),
    }),
);
}
};
```

- Per i dettagli sull'API, consulta la [DeleteMessage](#) sezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve ed elimina messaggi 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 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);
            }
        });
    }
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, consulta la [DeleteMessage](#) sezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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!")  
    }  
}
```

- Per i dettagli sull'API, [DeleteMessage](#) consulta AWS SDK for Kotlin API reference.

PowerShell

Strumenti per V4 PowerShell

Esempio 1: questo esempio elimina il messaggio con l'handle di ricezione specificato dalla coda specificata.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/  
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Per i dettagli sull'API, vedere [DeleteMessage](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina il messaggio con l'handle di ricezione specificato dalla coda specificata.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/  
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Per i dettagli sull'API, vedere [DeleteMessage](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, consulta [DeleteMessage AWS](#) SDK for Python (Boto3) API Reference.

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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.
```

- Per i dettagli sulle API, [DeleteMessage](#) consulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **DeleteMessageBatch** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare DeleteMessageBatch.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Elaborare le notifiche di eventi S3](#)
- [Pubblicazione di messaggi nelle code](#)
- [Inviare e ricevere batch di messaggi](#)

.NET

SDK per .NET

Note

C'è altro su [GitHub](#). Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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();
}
```

- Per i dettagli sull'API, consulta la [DeleteMessageBatch](#) sezione AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).


```

    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::DeleteMessageBatchRequest request;
    request.SetQueueUrl(queueURLS[i]);
    int id = 1; // Ids must be unique within a batch delete request.
    for (const Aws::String &receiptHandle: receiptHandles) {
        Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
        entry.SetId(std::to_string(id));
        ++id;
        entry.SetReceiptHandle(receiptHandle);
        request.AddEntries(entry);
    }

    Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
        sqsClient.DeleteMessageBatch(request);

    if (outcome.IsSuccess()) {
        std::cout << "The batch deletion of messages was successful."
                  << std::endl;
    }
    else {
        std::cerr << "Error with SQS::DeleteMessageBatch. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }
}

```

- Per i dettagli sull'API, consulta la [DeleteMessageBatch](#) sezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come eliminare più messaggi in batch

Questo esempio elimina i messaggi specificati.

Comando:

```
aws sqs delete-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-batch.json
```

File di input (delete-message-batch.json):

```
[
  {
    "Id": "FirstMessage",
    "ReceiptHandle": "AQEB1mg1...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
    "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

Output:

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- Per i dettagli sull'API, consulta AWS CLI Command [DeleteMessageBatch](#) Reference.

Go

SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sull'API, consulta la [DeleteMessageBatch](#) sezione AWS SDK per Go API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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,
                })),
            }),
        );
    }
};
```

- Per i dettagli sull'API, consulta la [DeleteMessageBatch](#) sezione AWS SDK per JavaScript API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio elimina 2 messaggi con gli handle di ricezione specificati dalla coda specificata.

```
$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
```

Output:

Failed	Successful
-----	-----
{}	{Request1, Request2}

- Per i dettagli sull'API, vedere [DeleteMessageBatch](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina 2 messaggi con gli handle di ricezione specificati dalla coda specificata.

```
$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
```

Output:

```
Failed    Successful
-----
{}        {Request1, Request2}
```

- Per i dettagli sull'API, vedere [DeleteMessageBatch](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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

```

- Per i dettagli sull'API, consulta [DeleteMessageBatch AWS SDK for Python \(Boto3\) API Reference](#).

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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_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'.
ENDTRY.
```

- Per i dettagli sulle API, [DeleteMessageBatch](#) consulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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.")
    }
}

```

- Per i dettagli sull'API, consulta la [DeleteMessageBatch](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **DeleteQueue** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare DeleteQueue.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Pubblicazione di messaggi nelle code](#)
- [Inviare e ricevere batch di messaggi](#)
- [Utilizzare la raccolta di messaggistica Amazon SQS Java per utilizzare l'interfaccia JMS](#)

.NET

SDK per .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Per eliminare una coda utilizzando il relativo URL.

```
/// <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;
}
```

- Per i dettagli sull'API, consulta la [DeleteQueue](#) sezione AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
}
```

```
}
```

- Per i dettagli sull'API, consulta la [DeleteQueue](#) sezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come eliminare una coda

Questo esempio elimina la coda specificata.

Comando:

```
aws sqs delete-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

Output:

```
None.
```

- Per i dettagli sull'API, consulta [DeleteQueue AWS CLI](#) Command Reference.

Go

SDK per Go V2

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sull'API, consulta la [DeleteQueue](#) sezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
    }
}
```

- Per i dettagli sull'API, consulta la [DeleteQueue](#) sezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elimina una coda Amazon SQS.

```
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;
};
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, consulta la [DeleteQueue](#) sezione AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elimina una coda 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 = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, consulta la [DeleteQueue](#) sezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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!")
    }
}
```

- Per i dettagli sull'API, [DeleteQueue](#) consulta AWS SDK for Kotlin API reference.

PowerShell

Strumenti per V4 PowerShell

Esempio 1: questo esempio elimina la coda specificata.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [DeleteQueue](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina la coda specificata.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [DeleteQueue](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, consulta [DeleteQueue AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK per Ruby

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
```

- Per i dettagli sull'API, consulta la [DeleteQueue](#) sezione AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
TRY.  
    lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).  
    MESSAGE 'SQS queue deleted' TYPE 'I'.  
ENDTRY.
```

- Per i dettagli sulle API, [DeleteQueue](#) consulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
    }
```

- Per i dettagli sull'API, consulta la [DeleteQueue](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **GetQueueAttributes** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare GetQueueAttributes.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Elaborare le notifiche di eventi S3](#)
- [Pubblicazione di messaggi nelle code](#)

.NET

SDK per .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <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;
}

```

- Per i dettagli sull'API, [GetQueueAttributes](#) consulta AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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;
    }
}

```

- Per i dettagli sull'API, [GetQueueAttributes](#) consulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come ottenere gli attributi di una coda

Questo esempio ottiene tutti gli attributi della coda specificata.

Comando:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

Output:

```
{
  "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"
}
}
```

Questo esempio ottiene solo la dimensione massima dei messaggi e gli attributi di timeout della visibilità per la coda specificata.

Comando:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-names MaximumMessageSize VisibilityTimeout
```

Output:

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- Per i dettagli sull'API, consulta [GetQueueAttributes AWS CLI Command Reference](#).

Go

SDK per Go V2

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sull'API, [GetQueueAttributes](#) consulta AWS SDK per Go API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
};
```

- Per i dettagli sull'API, [GetQueueAttributes](#) consulta AWS SDK per JavaScript API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio elenca tutti gli attributi della coda specificata.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
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]...}
```

Esempio 2: questo esempio elenca separatamente solo gli attributi specificati della coda specificata.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
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]}
```

- Per i dettagli sull'API, vedere [GetQueueAttributes](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elenca tutti gli attributi della coda specificata.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```


Output:

```

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]...}

```

Esempio 2: questo esempio elenca separatamente solo gli attributi specificati della coda specificata.

```

Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

```

Output:

```

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]}

```

- Per i dettagli sull'API, vedere [GetQueueAttributes](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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
```

- Per i dettagli sull'API, consulta [GetQueueAttributes AWS SDK for Python \(Boto3\) API Reference](#).

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
    }
}
```

- Per i dettagli sull'API, consulta la [GetQueueAttributes](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **GetQueueUrl** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetQueueUrl`.

.NET

SDK per .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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.");
        }
    }
}

```

- Per i dettagli sull'API, [GetQueueUrl](#) consulta AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
}

```

- Per i dettagli sull'API, [GetQueueUrl](#) consulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come ottenere l'URL di una coda

Questo esempio ottiene l'URL della coda specificata.

Comando:

```
aws sqs get-queue-url --queue-name MyQueue
```

Output:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- Per i dettagli sull'API, consulta [GetQueueUrl AWS CLI Command Reference](#).

Java

SDK per Java 2.x

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
GetQueueUrlResponse getQueueUrlResponse = sqsClient
    .getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
    return getQueueUrlResponse.queueUrl();
```

- Per i dettagli sull'API, [GetQueueUrl](#) consulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Ottiene l'URL di una coda Amazon SQS.


```
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;
};
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, [GetQueueUrl](#) consulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Ottiene l'URL di una coda 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",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  }
```

```
    } else {  
        console.log("Success", data.QueueUrl);  
    }  
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, [GetQueueUrl](#) consulta AWS SDK per JavaScript API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio elenca l'URL della coda con il nome specificato.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [GetQueueUrl](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elenca l'URL della coda con il nome specificato.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [GetQueueUrl](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, consulta [GetQueueUrl AWS SDK for Python \(Boto3\) API Reference](#).

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

TRY.
    oo_result = lo_sqs->getqueueurl( iv_queueuname = iv_queue_name ).
oo_result is returned for testing purposes. "
    MESSAGE 'Queue URL retrieved.' TYPE 'I'.
CATCH /aws1/cx_sqsqueuedoesnotexist.
    MESSAGE 'The requested queue does not exist.' TYPE 'E'.
ENDTRY.

```

- Per i dettagli sulle API, [GetQueueUrl](#) consulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **ListDeadLetterSourceQueues** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListDeadLetterSourceQueues`.

CLI

AWS CLI

Come elencare le code DLQ di origine

Questo esempio elenca le code associate alla coda DLQ di origine specificata.

Comando:

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Output:

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- Per i dettagli sull'API, consulta [ListDeadLetterSourceQueues AWS CLI Command Reference](#).

PowerShell

Strumenti per PowerShell V4

Esempio 1: Questo esempio elenca tutte le URLs code che si basano sulla coda specificata come coda di lettere morte.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue  
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Per i dettagli sull'API, vedere [ListDeadLetterSourceQueues](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: Questo esempio elenca tutte le URLs code che si basano sulla coda specificata come coda di lettere morte.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue  
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Per i dettagli sull'API, vedere [ListDeadLetterSourceQueues](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **ListQueues** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare ListQueues.

C++

SDK per C++

Note

C'è altro su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
}
```

- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come elencare le code

Questo esempio elenca tutte le code.

Comando:

```
aws sqs list-queues
```

Output:

```
{
```

```
"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"  
]  
}
```

Questo esempio elenca solo le code che iniziano con “My”.

Comando:

```
aws sqs list-queues --queue-name-prefix My
```

Output:

```
{  
  "QueueUrls": [  
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",  
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",  
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"  
  ]  
}
```

- Per i dettagli sull'API, consulta [ListQueues AWS CLI Command Reference](#).

Go

SDK per Go V2

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
        }
    }
}
```

- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
}
```

- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca le code Amazon SQS.

```
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;
};
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Elenca le code 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 = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
        }
    }
}
```

- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK for Kotlin API reference.

PowerShell

Strumenti per V4 PowerShell

Esempio 1: questo esempio elenca tutte le code.

```
Get-SQSQueue
```

Output:

```
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
```

Esempio 2: questo esempio elenca qualsiasi coda che inizia con il nome specificato.

```
Get-SQSQueue -QueueNamePrefix My
```

Output:

```
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
```

- Per i dettagli sull'API, vedere [ListQueues](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elenca tutte le code.

```
Get-SQSQueue
```

Output:

```
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
```

Esempio 2: questo esempio elenca qualsiasi coda che inizia con il nome specificato.

```
Get-SQSQueue -QueueNamePrefix My
```

Output:

```
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
```

- Per i dettagli sull'API, vedere [ListQueues](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, consulta [ListQueues AWS](#) SDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, [ListQueues](#) consulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Recupera la prima coda Amazon SQS elencata nella Regione.

```
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())
}
```

- Per i dettagli sulle API, consulta la [ListQueues](#) guida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
TRY.
    oo_result = lo_sqs->listqueues( ).          " oo_result is returned for
testing purposes. "
    MESSAGE 'Retrieved list of queues.' TYPE 'I'.
ENDTRY.
```

- Per i dettagli sulle API, [ListQueues](#) consulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
    }
}
```

- Per i dettagli sull'API, consulta la [ListQueues](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **PurgeQueue** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare PurgeQueue.

CLI

AWS CLI

Come rimuovere una coda

Questo esempio elimina tutti i messaggi nella coda specificata.

Comando:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue
```

Output:

```
None .
```

- Per i dettagli sull'API, consulta [PurgeQueue AWS CLI](#) Command Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio elimina tutti i messaggi dalla coda specificata.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [PurgeQueue](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina tutti i messaggi dalla coda specificata.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [PurgeQueue](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **ReceiveMessage** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare ReceiveMessage.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Gestire messaggi di grandi dimensioni con S3](#)
- [Elaborare le notifiche di eventi S3](#)
- [Pubblicazione di messaggi nelle code](#)
- [Inviare e ricevere batch di messaggi](#)

.NET

SDK per .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve messaggi da una coda utilizzando il relativo URL.

```
/// <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;
}

```

Riceve un messaggio da una coda Amazon SQS e poi elimina il messaggio.

```

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;
}
}

```

- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    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 =
sqClient.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();
}

```

- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come ricevere un messaggio

Questo esempio riceve fino a 10 messaggi disponibili e restituisce tutti gli attributi disponibili.

Comando:

```
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
```

Output:

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

Questo esempio riceve il prossimo messaggio disponibile, restituendo solo gli `SentTimestamp` attributi `SenderId` and e l'attributo `PostalCode` message.

Comando:

```
aws sqs receive-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names SenderId SentTimestamp --message-attribute-names PostalCode
```

Output:

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEB6nR4...HzlvZQ==",
      "MD5ofBody": "1000f835...a35411fa",
      "MD5ofMessageAttributes": "b8e89563...e088e74f",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "SenderId": "AIDAIAZKMSNQ7EXAMPLE",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        }
      }
    }
  ]
}
```

- Per i dettagli sull'API, consulta [ReceiveMessage AWS CLI](#) Command Reference.

Go

SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
}
```

- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
```

- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve un messaggio da una coda Amazon SQS.

```
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,  
        }))),  
    );  
  }  
}
```

```
    }},  
    );  
  }  
};
```

Riceve un messaggio da una coda Amazon SQS utilizzando il supporto del long polling.

```
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;  
};
```

- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve un messaggio da una coda Amazon SQS utilizzando il supporto del long polling.

```
// 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);
  }
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
        }
    }
}
```

- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK for Kotlin API reference.

PowerShell

Strumenti per V4 PowerShell

Esempio 1: questo esempio elenca le informazioni per un massimo di 10 messaggi da ricevere per la coda specificata. Le informazioni conterranno i valori degli attributi del messaggio specificati, se esistenti.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
  StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
Attributes          : {[SenderId, AIDAIAZKMSNQ7EXAMPLE], [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==
```

- Per i dettagli sull'API, vedere [ReceiveMessage](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elenca le informazioni per un massimo di 10 messaggi da ricevere per la coda specificata. Le informazioni conterranno i valori degli attributi del messaggio specificati, se esistenti.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
  StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
Attributes          : {[SenderId, AIDAIAZKMSNQ7EXAMPLE], [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==
```

- Per i dettagli sull'API, vedere [ReceiveMessage](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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

```

- Per i dettagli sull'API, consulta [ReceiveMessage AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK per Ruby

Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# Receives messages in a queue in Amazon Simple Queue Service (Amazon SQS).
#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param max_number_of_messages [Integer] The maximum number of messages
#   to receive. This number must be 10 or less. The default is 10.
# @example

```

```
# receive_messages(  
#   Aws::SQS::Client.new(region: 'us-west-2'),  
#   'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',  
#   10  
# )  
def receive_messages(sqs_client, queue_url, max_number_of_messages = 10)  
  if max_number_of_messages > 10  
    puts 'Maximum number of messages to receive must be 10 or less. ' \  
      'Stopping program.'  
    return  
  end  
  
  response = sqs_client.receive_message(  
    queue_url: queue_url,  
    max_number_of_messages: max_number_of_messages  
  )  
  
  if response.messages.count.zero?  
    puts 'No messages to receive, or all messages have already ' \  
      'been previously received.'  
    return  
  end  
  
  response.messages.each do |message|  
    puts '-' * 20  
    puts "Message body: #{message.body}"  
    puts "Message ID:  #{message.message_id}"  
  end  
rescue StandardError => e  
  puts "Error receiving messages: #{e.message}"  
end  
  
# Full example call:  
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.  
def run_me  
  region = 'us-west-2'  
  queue_name = 'my-queue'  
  max_number_of_messages = 10  
  
  sts_client = Aws::STS::Client.new(region: region)  
  
  # For example:  
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
```

```

queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

sqs_client = Aws::SQS::Client.new(region: region)

puts "Receiving messages from queue '#{queue_name}'..."

receive_messages(sqs_client, queue_url, max_number_of_messages)
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__

```

- Per i dettagli sull'API, [ReceiveMessage](#) consulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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(())
}

```

- Per i dettagli sulle API, consulta la [ReceiveMessage](#) guida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Riceve un messaggio da una coda Amazon SQS.

```
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.
```

Riceve un messaggio da una coda Amazon SQS utilizzando il supporto del long polling.

```
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.
```


- Per i dettagli sulle API, [ReceiveMessage](#) consulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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("---")
}
```

- Per i dettagli sull'API, consulta la [ReceiveMessage](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **RemovePermission** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `RemovePermission`.

CLI

AWS CLI

Come rimuovere un'autorizzazione

Questo esempio rimuove l'autorizzazione con l'etichetta specificata dalla coda specificata.

Comando:

```
aws sqs remove-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessageFromMyQueue
```

Output:

```
None.
```

- Per i dettagli sull'API, consulta [RemovePermission AWS CLI](#) Command Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio rimuove le impostazioni delle autorizzazioni con l'etichetta specificata dalla coda specificata.

```
Remove-SQSPermission -Label SendMessageFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [RemovePermission](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio rimuove le impostazioni delle autorizzazioni con l'etichetta specificata dalla coda specificata.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [RemovePermission](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **SendMessage** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare SendMessage.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Gestire messaggi di grandi dimensioni con S3](#)
- [Inviare e ricevere batch di messaggi](#)

.NET

SDK per .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea una coda Amazon SQS e invia un messaggio a tale coda.

```
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;
}
}

```

- Per i dettagli sull'API, [SendMessage](#) consulta AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    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();
}

```

- Per i dettagli sull'API, [SendMessage](#) consulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come inviare un messaggio

Questo esempio invia alla coda specificata un messaggio con il corpo del messaggio, il periodo di ritardo e gli attributi del messaggio specificato.

Comando:

```

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

```

File di input (send-message.json):

```

{
  "City": {
    "DataType": "String",

```

```
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

Output:

```
{
  "MD50fMessageBody": "51b0a325...39163aa0",
  "MD50fMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- Per i dettagli sull'API, consulta [SendMessage AWS CLI](#) Command Reference.

Java

SDK per Java 2.x

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Seguono due esempi dell'operazione SendMessage:

- Inviare un messaggio con un corpo e un ritardo
- Inviare un messaggio con un corpo e gli attributi del messaggio

Invia un messaggio con un corpo e un ritardo.

```
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);
    }
}
}

```

Invia un messaggio con un corpo e gli attributi del messaggio.

```

/**
 * <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);
        }
    }
}

```

- Per i dettagli sull'API, [SendMessage](#) consulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio a una coda Amazon SQS.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
    MessageBody:
      "Information about current NY Times fiction bestseller for week of 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

```
};
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, [SendMessage](#) consulta AWS SDK per JavaScript API Reference.

SDK per JavaScript (v2)

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Invia un messaggio a una coda 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 = {
  // 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);
  }
});
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK per JavaScript](#).
- Per i dettagli sull'API, [SendMessage](#) consulta AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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.")
    }
}
```

- Per i dettagli sull'API, [SendMessage](#) consulta AWS SDK for Kotlin API reference.

PowerShell

Strumenti per V4 PowerShell

Esempio 1: questo esempio invia un messaggio con gli attributi e il corpo del messaggio specificati alla coda specificata con un ritardo di consegna del messaggio di 10 secondi.

```
$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
```

Output:

MD5ofMessageAttributes	MD5ofMessageBody	MessageId
-----	-----	-----
1d3e51347bc042efbdf6dda31EXAMPLE c739-4d0c-818b-1820eEXAMPLE	51b0a3256d59467f973009b73EXAMPLE	c35fed8f-

- Per i dettagli sull'API, vedere [SendMessage](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio invia un messaggio con gli attributi e il corpo del messaggio specificati alla coda specificata con un ritardo di consegna del messaggio di 10 secondi.

```
$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
```

Output:

MD5ofMessageAttributes	MD5ofMessageBody	MessageId
-----	-----	-----
1d3e51347bc042efbdf6dda31EXAMPLE c739-4d0c-818b-1820eEXAMPLE	51b0a3256d59467f973009b73EXAMPLE	c35fed8f-

- Per i dettagli sull'API, vedere [SendMessage](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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

```

- Per i dettagli sull'API, consulta [SendMessage AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK per Ruby

Note

C'è di più su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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__
```

- Per i dettagli sull'API, [SendMessage](#) consulta AWS SDK per Ruby API Reference.

Rust

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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(())
}
```

- Per i dettagli sulle API, consulta la [SendMessage](#) guida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
TRY.  
    oo_result = lo_sqs->sendmessage(                " oo_result is returned for  
testing purposes. "  
        iv_queueurl = iv_queue_url  
        iv_messagebody = iv_message ).  
    MESSAGE 'Message sent to SQS queue.' TYPE 'I'.  
CATCH /aws1/cx_sqsinvalidmsgconts.  
    MESSAGE 'Message contains non-valid characters.' TYPE 'E'.  
CATCH /aws1/cx_sqsunsupportedop.  
    MESSAGE 'Operation not supported.' TYPE 'E'.  
ENDTRY.
```

- Per i dettagli sulle API, [SendMessage](#) consulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **SendMessageBatch** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare SendMessageBatch.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- [Inviare e ricevere batch di messaggi](#)

CLI

AWS CLI

Come inviare più messaggi in batch

Questo esempio invia alla coda specificata 2 messaggi con il corpo del messaggio, il periodo di ritardo e gli attributi del messaggio specificati.

Comando:

```
aws sqs send-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-batch.json
```

File di input (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"
    }
  }
}
]

```

Output:

```

{
  "Successful": [
    {
      "MD5OfMessageBody": "203c4a38...7943237e",
      "MD5OfMessageAttributes": "10809b55...baf283ef",
      "Id": "FuelReport-0001-2015-09-16T140731Z",
      "MessageId": "d175070c-d6b8-4101-861d-adeb3EXAMPLE"
    },
  ],
}

```

```
{
  "MD5OfMessageBody": "2cf0159a...c1980595",
  "MD5OfMessageAttributes": "55623928...ae354a25",
  "Id": "FuelReport-0002-2015-09-16T140930Z",
  "MessageId": "f9b7d55d-0570-413e-b9c5-a9264EXAMPLE"
}
```

- Per i dettagli sull'API, consulta AWS CLI Command [SendMessageBatch](#) Reference.

Java

SDK per Java 2.x

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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);
```

- Per i dettagli sull'API, [SendMessageBatch](#) consulta AWS SDK for Java 2.x API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio invia 2 messaggi con gli attributi e i corpi dei messaggi specificati alla coda specificata. La distribuzione viene ritardata di 15 secondi per il primo messaggio e di 10 secondi per il secondo messaggio.

```
$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
```

Output:

Failed	Successful
-----	-----
{}	{FirstMessage, SecondMessage}

- Per i dettagli sull'API, vedere [SendMessageBatch](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio invia 2 messaggi con gli attributi e i corpi dei messaggi specificati alla coda specificata. La distribuzione viene ritardata di 15 secondi per il primo messaggio e di 10 secondi per il secondo messaggio.

```
$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
```

Output:

```
Failed    Successful
-----
{}        {FirstMessage, SecondMessage}
```

- Per i dettagli sull'API, vedere [SendMessageBatch](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, consulta [SendMessageBatch AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK per Ruby

Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, [SendMessageBatch](#) consulta AWS SDK per Ruby API Reference.

SAP ABAP

SDK per SAP ABAP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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_sqsemttybatchrequest.  
    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.
```

- Per i dettagli sulle API, [SendMessageBatch](#) consulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **SetQueueAttributes** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare SetQueueAttributes.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- [Pubblicazione di messaggi nelle code](#)

.NET

SDK per .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Imposta l'attributo della policy di una coda per un argomento.

```

/// <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;
}

```

- Per i dettagli sull'API, [SetQueueAttributes](#) consulta AWS SDK per .NET API Reference.

C++

SDK per C++

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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();
}

```

Configura una coda DLQ.

```

    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();
}

```

Configura una coda Amazon SQS per utilizzare il long polling.

```

    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();
}

```

- Per i dettagli sull'API, [SetQueueAttributes](#) consulta AWS SDK per C++ API Reference.

CLI

AWS CLI

Come impostare gli attributi della coda

Questo esempio imposta la coda specificata su un ritardo di distribuzione di 10 secondi, una dimensione massima del messaggio di 128 KB (128 KB * 1.024 byte), un periodo di conservazione dei messaggi di 3 giorni (3 giorni * 24 ore * 60 minuti * 60 secondi), un tempo di attesa per la ricezione dei messaggi di 20 secondi e un timeout di visibilità predefinito di 60 secondi. Questo esempio associa inoltre la coda DLQ specificata a un numero massimo di ricezione di 1.000 messaggi.

Comando:

```

aws sqs set-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-attributes.json

```

File di input (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"
}
```

Output:

None.

- Per i dettagli sull'API, consulta AWS CLI Command [SetQueueAttributes](#) Reference.

Go

SDK per Go V2

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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
```

- Per i dettagli sull'API, [SetQueueAttributes](#) consulta AWS SDK per Go API Reference.

Java

SDK per Java 2.x

Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Configura Amazon SQS per utilizzare la crittografia lato server (SSE) utilizzando una chiave KMS personalizzata.

```
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();
        }
    }
}

```

- Per i dettagli sull'API, [SetQueueAttributes](#) consulta AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
};
```

Configura una coda Amazon SQS per utilizzare il long polling.

```
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;
};
```

Configura una coda DLQ.

```
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;
};
```

- Per i dettagli sull'API, [SetQueueAttributes](#) consulta AWS SDK per JavaScript API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio mostra come impostare una policy per sottoscrivere una coda a un argomento SNS. Quando si pubblica un messaggio in un argomento, viene inviato un messaggio a ogni coda sottoscritta.

```
# 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 }
```

Esempio 2: questo esempio imposta gli attributi specificati della coda specificata.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
  "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Per i dettagli sull'API, vedere [SetQueueAttributes](#) in AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio mostra come impostare una policy per sottoscrivere una coda a un argomento SNS. Quando si pubblica un messaggio in un argomento, viene inviato un messaggio a ogni coda sottoscritta.

```
# 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 }

```

Esempio 2: questo esempio imposta gli attributi specificati della coda specificata.

```

Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
"131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

```

- Per i dettagli sull'API, vedere [SetQueueAttributes](#) in AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Imposta l'attributo della policy di una coda per un argomento.

```

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

```

- Per i dettagli sull'API, consulta [SetQueueAttributes AWSSDK for Python \(Boto3\) API Reference](#).

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

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.")  
    }
```

- Per i dettagli sull'API, consulta la [SetQueueAttributes](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Utilizzo di Amazon SQS con un SDK AWS](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scenari per l'utilizzo di Amazon SQS AWS SDKs

I seguenti esempi di codice mostrano come implementare scenari comuni in Amazon SQS con. AWS SDKs Questi scenari illustrano come eseguire attività specifiche chiamando più funzioni all'interno di Amazon SQS o in combinazione con altri Servizi AWS. Ogni scenario include un collegamento al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice.

Gli scenari sono relativi a un livello intermedio di esperienza per aiutarti a comprendere le azioni di servizio nel contesto.

Esempi

- [Creare un'applicazione web che invia e recupera i messaggi utilizzando Amazon SQS](#)
- [Creazione di un'applicazione di messaggistica con Step Functions](#)
- [Creazione di un'applicazione Amazon Textract explorer](#)
- [Crea e pubblica su un argomento FIFO Amazon SNS utilizzando un SDK AWS](#)
- [Rileva persone e oggetti in un video con Amazon Rekognition utilizzando un SDK AWS](#)
- [Gestisci messaggi Amazon SQS di grandi dimensioni utilizzando Amazon S3 con un SDK AWS](#)
- [Ricevi ed elabora le notifiche degli eventi di Amazon S3 utilizzando un SDK AWS](#)
- [Pubblica messaggi Amazon SNS nelle code Amazon SQS utilizzando un SDK AWS](#)
- [Invia e ricevi batch di messaggi con Amazon SQS utilizzando un SDK AWS](#)
- [Utilizza il AWS Message Processing Framework for .NET per pubblicare e ricevere messaggi Amazon SQS](#)

- [Utilizzare la raccolta di messaggistica Amazon SQS Java per utilizzare l'interfaccia Servizio messaggi Java \(JMS\) per Amazon SQS](#)
- [Lavora con i tag di coda e Amazon SQS utilizzando un SDK AWS](#)

Creare un'applicazione web che invia e recupera i messaggi utilizzando Amazon SQS

Gli esempi di codice seguenti mostrano come creare un'applicazione di messaggistica utilizzando Amazon SQS.

Java

SDK per Java 2.x

Mostra come utilizzare l'API Amazon SQS per sviluppare una REST API Spring che invia e recupera i messaggi.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK per Kotlin

Mostra come utilizzare l'API Amazon SQS per sviluppare una REST API Spring che invia e recupera i messaggi.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Amazon SQS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione di messaggistica con Step Functions

Il seguente esempio di codice mostra come creare un'applicazione di AWS Step Functions messaggistica che recupera i record dei messaggi da una tabella di database.

Python

SDK per Python (Boto3)

Mostra come usare AWS SDK per Python (Boto3) with per creare un'applicazione di messaggistica che AWS Step Functions recupera i record dei messaggi da una tabella Amazon DynamoDB e li invia con Amazon Simple Queue Service (Amazon SQS). La macchina a stati si integra con una AWS Lambda funzione per scansione il database alla ricerca di messaggi non inviati.

- Crea una macchina a stati che recuperi e aggiorni i record di messaggi da una tabella Amazon DynamoDB.
- Aggiorna la definizione della macchina a stati per inviare messaggi anche ad Amazon Simple Queue Service (Amazon SQS).
- Avvia e arresta l'esecuzione della macchina a stati.
- Connettiti a Lambda, DynamoDB e Amazon SQS da una macchina a stati utilizzando le integrazioni di servizi.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su [GitHub](#)

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione Amazon Textract explorer

Gli esempi di codice seguenti mostrano come esplorare l'output di Amazon Textract tramite un'applicazione interattiva.

JavaScript

SDK per JavaScript (v3)

Mostra come utilizzare per AWS SDK per JavaScript creare un'applicazione React che utilizza Amazon Textract per estrarre dati dall'immagine di un documento e visualizzarli in una pagina Web interattiva. Questo esempio viene eseguito in un browser Web e richiede, come credenziali, un'identità autenticata Amazon Cognito. Utilizza Amazon Simple Storage Service (Amazon S3) per l'archiviazione e per le notifiche esegue il polling di una coda Amazon Simple Queue Service (Amazon SQS) sottoscritta a un argomento Amazon Simple Notification Service (Amazon SNS).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Cognito Identity
- Simple Storage Service (Amazon S3)
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK per Python (Boto3)

Mostra come utilizzarlo AWS SDK per Python (Boto3) con Amazon Textract per rilevare elementi di testo, moduli e tabelle nell'immagine di un documento. L'immagine di input e

l'output di Amazon Textract sono mostrati in un'applicazione Tkinter che consente di esplorare gli elementi rilevati.

- Invia un'immagine del documento ad Amazon Textract ed esplora l'output degli elementi rilevati.
- Invia immagini direttamente ad Amazon Textract o tramite un bucket Amazon Simple Storage Service (Amazon S3).
- Utilizza asynchronous APIs per avviare un processo che pubblica una notifica su un argomento di Amazon Simple Notification Service (Amazon SNS) al termine del processo.
- Esegue il polling di una coda Amazon Simple Queue Service (Amazon SQS) per un messaggio di completamento del processo e visualizza i risultati.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- Amazon Cognito Identity
- Simple Storage Service (Amazon S3)
- Amazon SNS
- Amazon SQS
- Amazon Textract

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Crea e pubblica su un argomento FIFO Amazon SNS utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come creare un argomento FIFO Amazon SNS e pubblicare un messaggio su di esso.

Java

SDK per Java 2.x

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

In questo esempio

- viene creato un argomento Amazon SNS FIFO, due code FIFO Amazon SQS e una coda Standard.
- viene effettuata la sottoscrizione all'argomento e pubblicato un messaggio nell'argomento.

Il [test](#) verifica la ricezione del messaggio in ogni coda. L'[esempio completo](#) mostra anche l'aggiunta di policy di accesso e l'eliminazione delle risorse alla fine.

```
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);
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
 - [CreateTopic](#)
 - [Pubblicare](#)
 - [Subscribe](#)

Python

SDK per Python (Boto3)

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea un argomento FIFO, sottoscrivi una coda FIFO di Amazon SQS all'argomento e pubblica un messaggio su un argomento Amazon SNS.

```
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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Python (Boto3).
 - [CreateTopic](#)
 - [Pubblicare](#)
 - [Subscribe](#)

SAP ABAP

SDK per SAP ABAP

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea un argomento FIFO, sottoscrivi una coda Amazon SQS FIFO all'argomento e pubblica un messaggio su un argomento Amazon SNS.

```

" 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.

```

- Per informazioni dettagliate sull'API, consulta gli argomenti seguenti nella documentazione di riferimento dell'API AWS SDK per SAP ABAP.
 - [CreateTopic](#)
 - [Pubblicare](#)
 - [Subscribe](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Rileva persone e oggetti in un video con Amazon Rekognition utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come rilevare persone e oggetti in un video con Amazon Rekognition.

Java

SDK per Java 2.x

Mostra come utilizzare l'API Java di Amazon Rekognition per creare un'applicazione che rileva volti e oggetti nei video situati in un bucket Amazon Simple Storage Service (Amazon S3). L'applicazione invia all'amministratore una notifica e-mail sui risultati tramite Amazon Simple Email Service (Amazon SES).

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK per Python (Boto3)

Usa Amazon Rekognition per rilevare volti, oggetti e persone nei video avviando processi di rilevamento asincrono. Questo esempio, inoltre, configura Amazon Rekognition per notificare un argomento Amazon Simple Notification Service (Amazon SNS) al completamento dei processi e sottoscrive una coda Amazon Simple Queue Service (Amazon SQS) all'argomento. Quando la coda riceve un messaggio su un processo, questo viene recuperato e vengono restituiti i risultati.

Questo esempio è visualizzato al meglio su GitHub. Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SES
- Amazon SNS
- Amazon SQS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Gestisci messaggi Amazon SQS di grandi dimensioni utilizzando Amazon S3 con un SDK AWS

L'esempio di codice seguente mostra come utilizzare la libreria client ampia di Amazon SQS utilizzare messaggi Amazon SQS di grandi dimensioni.

Java

SDK per Java 2.x

Note

C'è altro su GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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;
    }
}
```

- Per ulteriori informazioni, consulta la [Guida per sviluppatori di AWS SDK for Java 2.x](#).
- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
 - [CreateBucket](#)
 - [PutBucketLifecycleConfiguration](#)
 - [ReceiveMessage](#)
 - [SendMessage](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Ricevi ed elabora le notifiche degli eventi di Amazon S3 utilizzando un SDK AWS

L'esempio di codice seguente mostra come utilizzare le notifiche di eventi S3 in modo orientato agli oggetti.

Java

SDK per Java 2.x

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Questo esempio mostra come elaborare un evento di notifica S3 utilizzando Amazon SQS.

```
/**
 * 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 S3EventNotification
API.
 * <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()) {

sqscClient.deleteMessageBatch(DeleteMessageBatchRequest.builder()
            .queueUrl(queueUrl)
            .entries(messagesToDelete)
            .build()
        ).join();
    }
} // End of while block.
} catch (InterruptedException | ExecutionException e) {
    throw new RuntimeException(e);
}
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
 - [DeleteMessageBatch](#)
 - [GetQueueAttributes](#)
 - [PutBucketNotificationConfiguration](#)
 - [ReceiveMessage](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Pubblica messaggi Amazon SNS nelle code Amazon SQS utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

.NET

SDK per .NET

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
/// <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\n"}and queue URL {queueUrl}" +  
                $"{"\r\n"}has 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;
    }
}

```

Crea una classe che avvolge le operazioni di Amazon SQS.

```

    /// <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;
}
}

```

Crea una classe che avvolge le operazioni di Amazon SNS.

```

/// <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;
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per .NET .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Pubblicare](#)
 - [ReceiveMessage](#)

- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

C++

SDK per C++

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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();
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per C++ .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Pubblicare](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)

- [Unsubscribe](#)

Go

SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Definisce una struttura per il wrapping delle azioni di Amazon SNS utilizzate in questo esempio.

```
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
}

```

Definisce una struttura per il wrapping delle azioni di Amazon SQS utilizzate in questo esempio.

```

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
}

```

Eliminare le risorse.

```

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)
        }
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)

- [GetQueueAttributes](#)
- [Pubblicare](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Java

SDK per Java 2.x

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 "";
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Pubblicare](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

JavaScript

SDK per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Questo è il punto di ingresso per questo scenario.

```
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();
};
```

Il codice precedente fornisce le dipendenze necessarie e avvia lo scenario. La sezione successiva contiene il blocco principale dell'esempio.

```
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();
    }
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per JavaScript .
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Pubblicare](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Kotlin

SDK per Kotlin

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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 json0bject = gson.fromJson(jsonString, JsonObject::class.java)
            val toneArray = json0bject.getAsJsonArray("tone")
            for (value: String? in filterList) {
                toneArray.add(JsonPrimitive(value))
            }

            val updatedJsonString: String = gson.toJson(json0bject)

```

```

        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
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Kotlin.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)

- [GetQueueAttributes](#)
- [Pubblicare](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Python

SDK per Python (Boto3)

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
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_yn)
        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.")

```

Crea classi che racchiudono le operazioni di Amazon SNS e Amazon SQS da utilizzare nello scenario.

```

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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Python (Boto3).
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Pubblicare](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Swift

SDK per Swift

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
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)
        }
    }
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API SDK AWS per Swift.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Pubblicare](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Invia e ricevi batch di messaggi con Amazon SQS utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come:

- Creare una coda Amazon SQS.
- Inviare batch di messaggi alla coda.
- Ricevere batch di messaggi dalla coda.
- Eliminare batch di messaggi dalla coda.

Java

SDK per Java 2.x

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Come illustrato negli esempi seguenti, puoi gestire le operazioni di messaggistica batch con Amazon SQS utilizzando due approcci diversi con AWS SDK for Java 2.x:

`SendRecvBatch.java` utilizza operazioni batch esplicite. È necessario creare manualmente batch di messaggi e chiamare `sendMessageBatch()` e `deleteMessageBatch()` direttamente, nonché gestire le risposte in batch, inclusi eventuali messaggi non riusciti. Questo approccio garantisce il pieno controllo sul dimensionamento dei batch e sulla gestione degli errori. Tuttavia, richiede più codice per gestire la logica di batching.

`SimpleProducerConsumer.java` utilizza la `SqsAsyncBatchManager` libreria di alto livello per il raggruppamento automatico delle richieste. Vengono effettuate chiamate individuali `sendMessage()` e `deleteMessage()` con le stesse firme dei metodi del client standard. L'SDK memorizza automaticamente queste chiamate nel buffer e le invia come operazioni batch. Questo approccio richiede modifiche minime al codice e offre vantaggi in termini di prestazioni del batching.

Utilizza il batching esplicito quando è necessario un controllo preciso sulla composizione dei batch e sulla gestione degli errori. Utilizza il batching automatico quando è necessario ottimizzare le prestazioni con modifiche minime al codice.

`SendRecvBatch.java`: utilizza operazioni batch esplicite con i messaggi.

```
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: utilizza il raggruppamento automatico dei messaggi.

```

package com.example.sqs;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsAsyncClient;
import software.amazon.awssdk.services.sqs.batchmanager.SqsAsyncBatchManager;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageResponse;
import software.amazon.awssdk.core.exception.SdkException;

import java.math.BigInteger;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Demonstrates the AWS SDK for Java 2.x Automatic Request Batching API for
 * Amazon SQS.
 *
 * This example showcases the high-level SqsAsyncBatchManager library that
 * provides
 * efficient batching and buffering for SQS operations. The batch manager offers

```

- * methods that directly mirror SqsAsyncClient methods—sendMessage, changeMessageVisibility,
- * deleteMessage, and receiveMessage—making it a drop-in replacement with minimal code changes.
- *
- * Key features of the SqsAsyncBatchManager:
 - * - Automatic batching: The SDK automatically buffers individual requests and sends them
 - * as batches when maxBatchSize (default: 10) or sendRequestFrequency (default: 200ms) thresholds are reached
 - * - Familiar API: Method signatures match SqsAsyncClient exactly, requiring no learning curve
 - * - Background optimization: The batch manager maintains internal buffers and handles batching logic transparently
 - * - Asynchronous operations: All methods return CompletableFuture for non-blocking execution
- *
- * Performance benefits demonstrated:
 - * - Reduced API calls: Multiple individual requests are consolidated into single batch operations
 - * - Lower costs: Fewer API calls result in reduced SQS charges
 - * - Higher throughput: Batch operations process more messages per second
 - * - Efficient resource utilization: Fewer network round trips and better connection reuse
- *
- * This example compares:
 - * 1. Single-message operations using SqsAsyncClient directly
 - * 2. Batch operations using SqsAsyncBatchManager with identical method calls
- *
- * Usage patterns:
 - * - Set batch size to 1 to use SqsAsyncClient for baseline performance measurement
 - * - Set batch size > 1 to use SqsAsyncBatchManager for optimized batch processing
 - * - Monitor real-time throughput metrics to observe performance improvements
- *
- * Prerequisites:
 - * - AWS SDK for Java 2.x version 2.28.0 or later
 - * - An existing SQS queue
 - * - Valid AWS credentials configured
- *

```
* The program displays real-time metrics showing the dramatic performance
difference
* between individual operations and automatic batching.
*/
public class SimpleProducerConsumer {

    // The maximum runtime of the program.
    private final static int MAX_RUNTIME_MINUTES = 60;
    private final static Logger log =
    LoggerFactory.getLogger(SimpleProducerConsumer.class);

    /**
     * Runs the SQS batching demonstration with user-configured parameters.
     *
     * Prompts for queue name, thread counts, batch size, message size, and
runtime.
     * Creates producer and consumer threads to demonstrate batching performance.
     *
     * @param args command line arguments (not used)
     * @throws InterruptedException if thread operations are interrupted
     */
    public static void main(String[] args) throws InterruptedException {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the number of producers: ");
        final int producerCount = input.nextInt();

        System.out.print("Enter the number of consumers: ");
        final int consumerCount = input.nextInt();

        System.out.print("Enter the number of messages per batch: ");
        final int batchSize = input.nextInt();

        System.out.print("Enter the message size in bytes: ");
        final int messageSizeByte = input.nextInt();

        System.out.print("Enter the run time in minutes: ");
        final int runTimeMinutes = input.nextInt();

        // Create SQS async client and batch manager for all operations.
```



```
// The SqsAsyncBatchManager is created from the SqsAsyncClient using the
// batchManager() factory method, which provides default batching
configuration.
// This high-level library automatically handles request buffering and
batching
// while maintaining the same method signatures as SqsAsyncClient.
final SqsAsyncClient sqsAsyncClient = SqsAsyncClient.create();
final SqsAsyncBatchManager batchManager = sqsAsyncClient.batchManager();

final String queueUrl =
sqsAsyncClient.getQueueUrl(GetQueueUrlRequest.builder()
    .queueName(queueName)
    .build()).join().queueUrl();

// The flag used to stop producer, consumer, and monitor threads.
final AtomicBoolean stop = new AtomicBoolean(false);

// Start the producers.
final AtomicInteger producedCount = new AtomicInteger();
final Thread[] producers = new Thread[producerCount];
for (int i = 0; i < producerCount; i++) {
    if (batchSize == 1) {
        producers[i] = new Producer(sqsAsyncClient, queueUrl,
messageSizeByte,
            producedCount, stop);
    } else {
        producers[i] = new BatchProducer(batchManager, queueUrl,
batchSize,
            messageSizeByte, producedCount, stop);
    }
    producers[i].start();
}

// Start the consumers.
final AtomicInteger consumedCount = new AtomicInteger();
final Thread[] consumers = new Thread[consumerCount];
for (int i = 0; i < consumerCount; i++) {
    if (batchSize == 1) {
        consumers[i] = new Consumer(sqsAsyncClient, queueUrl,
consumedCount, stop);
    } else {
        consumers[i] = new BatchConsumer(batchManager, queueUrl,
batchSize,
            consumedCount, stop);
    }
}
```

```

        }
        consumers[i].start();
    }

    // Start the monitor thread.
    final Thread monitor = new Monitor(producedCount, consumedCount, stop);
    monitor.start();

    // Wait for the specified amount of time then stop.
    Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runTimeMinutes,
        MAX_RUNTIME_MINUTES)));
    stop.set(true);

    // Join all threads.
    for (int i = 0; i < producerCount; i++) {
        producers[i].join();
    }

    for (int i = 0; i < consumerCount; i++) {
        consumers[i].join();
    }

    monitor.interrupt();
    monitor.join();

    // Close resources
    batchManager.close();
    sqsAsyncClient.close();
}

/**
 * Creates a random string of approximately the specified size in bytes.
 *
 * @param sizeByte the target size in bytes for the generated string
 * @return a random string encoded in base-32
 */
private static String makeRandomString(int sizeByte) {
    final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
    new Random().nextBytes(bs);
    bs[0] = (byte) ((bs[0] | 64) & 127);
    return new BigInteger(bs).toString(32);
}

/**

```

```

    * Sends messages individually using SqsAsyncClient for baseline performance
    measurement.
    *
    * This producer demonstrates traditional single-message operations without
    batching.
    * Each sendMessage() call results in a separate API request to SQS,
    providing
    * a performance baseline for comparison with the batch operations.
    *
    * The sendMessage() method signature is identical to
    SqsAsyncBatchManager.sendMessage(),
    * showing how the high-level batching library maintains API compatibility
    while
    * adding automatic optimization behind the scenes.
    */
    private static class Producer extends Thread {
        final SqsAsyncClient sqsAsyncClient;
        final String queueUrl;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        /**
         * Creates a producer thread for single-message operations.
         *
         * @param sqsAsyncClient the SQS client for sending messages
         * @param queueUrl the URL of the target queue
         * @param messageSizeByte the size of messages to generate
         * @param producedCount shared counter for tracking sent messages
         * @param stop shared flag to signal thread termination
         */
        Producer(SqsAsyncClient sqsAsyncClient, String queueUrl, int
        messageSizeByte,
            AtomicInteger producedCount, AtomicBoolean stop) {
            this.sqsAsyncClient = sqsAsyncClient;
            this.queueUrl = queueUrl;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /**
         * Continuously sends messages until the stop flag is set.
         *

```

```

    * Uses SqsAsyncClient.sendMessage() directly, resulting in one API call
    per message.
    * This approach provides baseline performance metrics for comparison
    with batching.
    * Each call blocks until the individual message is sent, demonstrating
    traditional
    * one-request-per-operation behavior.
    */
    public void run() {
        try {
            while (!stop.get()) {
                sqsAsyncClient.sendMessage(SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody(theMessage)
                    .build()).join();
                producedCount.incrementAndGet();
            }
        } catch (SdkException | java.util.concurrent.CompletionException e) {
            // Handle both SdkException and CompletionException from async
operations.
            // If this unlikely condition occurs, stop.
            log.error("Producer: " + e.getMessage());
            System.exit(1);
        }
    }
}

/**
 * Sends messages using SqsAsyncBatchManager for automatic request batching
and optimization.
 *
 * This producer demonstrates the AWS SDK for Java 2.x high-level batching
library.
 * The SqsAsyncBatchManager automatically buffers individual sendMessage()
calls and
 * sends them as batches when thresholds are reached:
 * - maxBatchSize: Maximum 10 messages per batch (default)
 * - sendRequestFrequency: 200ms timeout before sending partial batches
(default)
 *
 * Key advantages of the batching approach:
 * - Identical API: batchManager.sendMessage() has the same signature as
sqsAsyncClient.sendMessage()
 * - Automatic optimization: No code changes needed to benefit from batching

```

```

    * - Transparent buffering: The SDK handles batching logic internally
    * - Reduced API calls: Multiple messages sent in single batch requests
    * - Lower costs: Fewer API calls result in reduced SQS charges
    * - Higher throughput: Batch operations process significantly more messages
per second
    */
    private static class BatchProducer extends Thread {
        final SqsAsyncBatchManager batchManager;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        /**
         * Creates a producer thread for batch operations.
         *
         * @param batchManager the batch manager for efficient message sending
         * @param queueUrl the URL of the target queue
         * @param batchSize the number of messages to send per batch
         * @param messageSizeByte the size of messages to generate
         * @param producedCount shared counter for tracking sent messages
         * @param stop shared flag to signal thread termination
         */
        BatchProducer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                        int messageSizeByte, AtomicInteger producedCount,
                        AtomicBoolean stop) {
            this.batchManager = batchManager;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /**
         * Continuously sends batches of messages using the high-level batching
library.
         *
         * Notice how batchManager.sendMessage() uses the exact same method
signature
         * and request builder pattern as SqsAsyncClient.sendMessage(). This
demonstrates

```

```

    * the drop-in replacement capability of the SqsAsyncBatchManager.
    *
    * The SDK automatically:
    * - Buffers individual sendMessage() calls internally
    * - Groups them into batch requests when thresholds are met
    * - Sends SendMessageBatchRequest operations to SQS
    * - Returns individual CompletableFuture responses for each message
    *
    * This transparent batching provides significant performance
improvements
    * without requiring changes to application logic or error handling
patterns.
    */
    public void run() {
        try {
            while (!stop.get()) {
                // Send multiple messages using the high-level batch manager.
                // Each batchManager.sendMessage() call uses identical syntax
to
                // sqsAsyncClient.sendMessage(), demonstrating API
compatibility.
                // The SDK automatically buffers these calls and sends them
as
                // batch operations when maxBatchSize (10) or
sendRequestFrequency (200ms)
                // thresholds are reached, significantly improving
throughput.
                for (int i = 0; i < batchSize; i++) {
                    CompletableFuture<SendMessageResponse> future =
batchManager.sendMessage(
                        SendMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .messageBody(theMessage)
                            .build());

                    // Handle the response asynchronously
                    future.whenComplete((response, throwable) -> {
                        if (throwable == null) {
                            producedCount.incrementAndGet();
                        } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
!(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {

```

```

        log.error("BatchProducer: Failed to send
message", throwable);
    }
    // Ignore CancellationException and executor shutdown
errors - expected during shutdown
    });
}

    // Small delay to allow batching to occur
    Thread.sleep(10);
}
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    log.error("BatchProducer interrupted: " + e.getMessage());
} catch (SdkException | java.util.concurrent.CompletionException e) {
    log.error("BatchProducer: " + e.getMessage());
    System.exit(1);
}
}
}

/**
 * Receives and deletes messages individually using SqsAsyncClient for
baseline measurement.
 *
 * This consumer demonstrates traditional single-message operations without
batching.
 * Each receiveMessage() and deleteMessage() call results in separate API
requests,
 * providing a performance baseline for comparison with batch operations.
 *
 * The method signatures are identical to SqsAsyncBatchManager methods:
 * - receiveMessage() matches batchManager.receiveMessage()
 * - deleteMessage() matches batchManager.deleteMessage()
 *
 * This API consistency allows easy migration to the high-level batching
library.
 */
private static class Consumer extends Thread {
    final SqsAsyncClient sqsAsyncClient;
    final String queueUrl;
    final AtomicInteger consumedCount;
    final AtomicBoolean stop;

```

```

/**
 * Creates a consumer thread for single-message operations.
 *
 * @param sqsAsyncClient the SQS client for receiving messages
 * @param queueUrl the URL of the source queue
 * @param consumedCount shared counter for tracking processed messages
 * @param stop shared flag to signal thread termination
 */
Consumer(SqsAsyncClient sqsAsyncClient, String queueUrl, AtomicInteger
consumedCount,
        AtomicBoolean stop) {
    this.sqsAsyncClient = sqsAsyncClient;
    this.queueUrl = queueUrl;
    this.consumedCount = consumedCount;
    this.stop = stop;
}

/**
 * Continuously receives and deletes messages using traditional single-
request operations.
 *
 * Uses SqsAsyncClient methods directly:
 * - receiveMessage(): One API call per receive operation
 * - deleteMessage(): One API call per delete operation
 *
 * This approach demonstrates the baseline performance without batching
optimization.
 * Compare these method calls with the identical signatures used in
BatchConsumer
 * to see how the high-level batching library maintains API
compatibility.
 */
public void run() {
    try {
        while (!stop.get()) {
            try {
                final ReceiveMessageResponse result =
sqsAsyncClient.receiveMessage(
                    ReceiveMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .build()).join();

                if (!result.messages().isEmpty()) {
                    final Message m = result.messages().get(0);

```



```

        // Note: deleteMessage() signature identical to
batchManager.deleteMessage()

sqsAsyncClient.deleteMessage(DeleteMessageRequest.builder()
    .queueUrl(queueUrl)
    .receiptHandle(m.receiptHandle())
    .build()).join();
    consumedCount.incrementAndGet();
    }
    } catch (SdkException |
java.util.concurrent.CompletionException e) {
    log.error(e.getMessage());
    }
    }
    } catch (SdkException | java.util.concurrent.CompletionException e) {
        // Handle both SdkException and CompletionException from async
operations.
        // If this unlikely condition occurs, stop.
        log.error("Consumer: " + e.getMessage());
        System.exit(1);
    }
    }
    }

/**
 * Receives and deletes messages using SqsAsyncBatchManager for automatic
optimization.
 *
 * This consumer demonstrates the AWS SDK for Java 2.x high-level batching
library
 * for message consumption. The SqsAsyncBatchManager provides two key
optimizations:
 *
 * 1. Receive optimization: Maintains an internal buffer of messages fetched
in the
 * background, so receiveMessage() calls return immediately from the
buffer
 * 2. Delete batching: Automatically buffers deleteMessage() calls and sends
them
 * as DeleteMessageBatchRequest operations when thresholds are reached
 *
 * Key features:
 * - Identical API: receiveMessage() and deleteMessage() have the same
signatures

```

```

    * as SqsAsyncClient methods, making this a true drop-in replacement
    * - Background fetching: The batch manager continuously fetches messages to
keep
    * the internal buffer populated, reducing receive latency
    * - Automatic delete batching: Individual deleteMessage() calls are buffered
and
    * sent as batch operations (up to 10 per batch, 200ms frequency)
    * - Transparent optimization: No application logic changes needed to benefit
    *
    * Performance benefits:
    * - Reduced API calls through automatic batching of delete operations
    * - Lower latency for receives due to background message buffering
    * - Higher overall throughput with fewer network round trips
    */
private static class BatchConsumer extends Thread {
    final SqsAsyncBatchManager batchManager;
    final String queueUrl;
    final int batchSize;
    final AtomicInteger consumedCount;
    final AtomicBoolean stop;

    /**
     * Creates a consumer thread for batch operations.
     *
     * @param batchManager the batch manager for efficient message processing
     * @param queueUrl the URL of the source queue
     * @param batchSize the maximum number of messages to receive per batch
     * @param consumedCount shared counter for tracking processed messages
     * @param stop shared flag to signal thread termination
     */
    BatchConsumer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                  AtomicInteger consumedCount, AtomicBoolean stop) {
        this.batchManager = batchManager;
        this.queueUrl = queueUrl;
        this.batchSize = batchSize;
        this.consumedCount = consumedCount;
        this.stop = stop;
    }

    /**
     * Continuously receives and deletes messages using the high-level
batching library.
     *

```

```

    * Demonstrates the key advantage of SqsAsyncBatchManager: identical
method signatures
    * with automatic optimization. Notice how:
    *
    * - batchManager.receiveMessage() uses the same syntax as
sqsAsyncClient.receiveMessage()
    * - batchManager.deleteMessage() uses the same syntax as
sqsAsyncClient.deleteMessage()
    *
    * Behind the scenes, the batch manager:
    * 1. Maintains an internal message buffer populated by background
fetching
    * 2. Returns messages immediately from the buffer (reduced latency)
    * 3. Automatically batches deleteMessage() calls into
DeleteMessageBatchRequest operations
    * 4. Sends batch deletes when maxBatchSize (10) or sendRequestFrequency
(200ms) is reached
    *
    * This provides significant performance improvements with zero code
changes
    * compared to traditional SqsAsyncClient usage patterns.
    */
    public void run() {
        try {
            while (!stop.get()) {
                // Receive messages using the high-level batch manager.
                // This call uses identical syntax to
sqsAsyncClient.receiveMessage()
                // but benefits from internal message buffering for improved
performance.

                final ReceiveMessageResponse result =
batchManager.receiveMessage(
                    ReceiveMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .maxNumberOfMessages(Math.min(batchSize, 10))
                        .build()).join();

                if (!result.messages().isEmpty()) {
                    final List<Message> messages = result.messages();

                    // Delete messages using the batch manager.
                    // Each deleteMessage() call uses identical syntax to
SqsAsyncClient

```

```

        // but the SDK automatically buffers these calls and
sends them

        // as DeleteMessageBatchRequest operations for optimal
performance.

        for (Message message : messages) {
            CompletableFuture<DeleteMessageResponse> future =
batchManager.deleteMessage(
                DeleteMessageRequest.builder()
                    .queueUrl(queueUrl)

                .receiptHandle(message.receiptHandle())

                    .build());

            future.whenComplete((response, throwable) -> {
                if (throwable == null) {
                    consumedCount.incrementAndGet();
                } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
                    !(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {
                    log.error("BatchConsumer: Failed to delete
message", throwable);
                }
                // Ignore CancellationException and executor
shutdown errors - expected during shutdown
            });
        }
    }

    // Small delay to prevent tight polling
    Thread.sleep(10);
}
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    log.error("BatchConsumer interrupted: " + e.getMessage());
} catch (SdkException | java.util.concurrent.CompletionException e) {
    // Handle both SdkException and CompletionException from async
operations.

    // If this unlikely condition occurs, stop.
    log.error("BatchConsumer: " + e.getMessage());
    System.exit(1);
}
}
}
}

```

```

/**
 * Displays real-time throughput statistics every second.
 *
 * This thread logs the current count of produced and consumed messages
 * to help you monitor the performance comparison.
 */
private static class Monitor extends Thread {
    private final AtomicInteger producedCount;
    private final AtomicInteger consumedCount;
    private final AtomicBoolean stop;

    /**
     * Creates a monitoring thread that displays throughput statistics.
     *
     * @param producedCount shared counter for messages sent
     * @param consumedCount shared counter for messages processed
     * @param stop shared flag to signal thread termination
     */
    Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
            AtomicBoolean stop) {
        this.producedCount = producedCount;
        this.consumedCount = consumedCount;
        this.stop = stop;
    }

    /**
     * Logs throughput statistics every second until stopped.
     *
     * Displays the current count of produced and consumed messages
     * to help monitor the performance comparison between batching
strategies.
     */
    public void run() {
        try {
            while (!stop.get()) {
                Thread.sleep(1000);
                log.info("produced messages = " + producedCount.get()
                        + ", consumed messages = " + consumedCount.get());
            }
        } catch (InterruptedException e) {
            // Allow the thread to exit.
        }
    }
}

```

```
}  
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
 - [CreateQueue](#)
 - [DeleteMessage](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [ReceiveMessage](#)
 - [SendMessage](#)
 - [SendMessageBatch](#)

Python

SDK per Python (Boto3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea funzioni per il wrapping delle funzioni dei messaggi Amazon SQS.

```
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

```

Utilizza le funzioni wrapper per inviare e ricevere messaggi in batch.

```

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)
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Python (Boto3).
 - [CreateQueue](#)
 - [DeleteMessage](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [ReceiveMessage](#)
 - [SendMessage](#)
 - [SendMessageBatch](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizza il AWS Message Processing Framework for .NET per pubblicare e ricevere messaggi Amazon SQS

Il seguente esempio di codice mostra come creare applicazioni che pubblicano e ricevono messaggi Amazon SQS utilizzando il AWS Message Processing Framework for .NET.

.NET

SDK per .NET

Fornisce un tutorial per il AWS Message Processing Framework per .NET. Il tutorial crea un'applicazione web che consente all'utente di pubblicare un messaggio Amazon SQS e un'applicazione della riga di comando che riceve il messaggio.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta il [tutorial completo](#) nella Guida per gli AWS SDK per .NET sviluppatori e l'esempio su [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon SQS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare la raccolta di messaggistica Amazon SQS Java per utilizzare l'interfaccia Servizio messaggi Java (JMS) per Amazon SQS

L'esempio di codice seguente mostra come utilizzare la raccolta di messaggistica Amazon SQS Java da utilizzare con l'interfaccia JMS.

Java

SDK per Java 2.x

Note

C'è altro su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

I seguenti esempi funzionano con le code Amazon SQS standard e includono:

- Invio di un messaggio di testo.
- Ricezione di messaggi in modo sincrono.
- Ricezione di messaggi in modo asincrono.

- Ricezione di messaggi utilizzando la modalità CLIENT_ACKNOWLEDGE.
- Ricezione di messaggi utilizzando la modalità UNORDERED_ACKNOWLEDGE.
- Utilizzo di Spring per iniettare dipendenze.
- Una classe di utilità che fornisce metodi comuni utilizzati dagli altri esempi.

Per ulteriori informazioni sull'utilizzo di JMS con Amazon SQS, consulta [Guida per gli sviluppatori di Amazon SQS](#).

Invio di un messaggio di testo.

```
/**
 * This method establishes a connection to a standard Amazon SQS queue using
 the Amazon SQS
 * Java Messaging Library and sends text messages to it. It uses JMS (Java
 Message Service) API
 * with automatic acknowledgment mode to ensure reliable message delivery,
 and automatically
 * manages all messaging resources.
 *
 * @throws JMSException If there is a problem connecting to or sending
 messages to the queue
 */
public static void doSendTextMessage() throws JMSException {
    // Create a connection factory.
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        SqsClient.create()
    );

    // Create the connection in a try-with-resources statement so that it's
 closed automatically.
    try (SQSConnection connection = connectionFactory.createConnection()) {

        // Create the queue if needed.
        SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
 SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

        // Create a session that uses the JMS auto-acknowledge mode.
        Session session = connection.createSession(false,
 Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer =
 session.createProducer(session.createQueue(QUEUE_NAME));
```

```

        createAndSendMessages(session, producer);
    } // The connection closes automatically. This also closes the session.
    LOGGER.info("Connection closed");
}

/**
 * This method reads text input from the keyboard and sends each line as a
 * separate message
 * to a standard Amazon SQS queue using the Amazon SQS Java Messaging
 * Library. It continues
 * to accept input until the user enters an empty line, using JMS (Java
 * Message Service) API to
 * handle the message delivery.
 *
 * @param session The JMS session used to create messages
 * @param producer The JMS message producer used to send messages to the
 * queue
 */
private static void createAndSendMessages(Session session, MessageProducer
producer) {
    BufferedReader inputReader = new BufferedReader(
        new InputStreamReader(System.in, Charset.defaultCharset()));

    try {
        String input;
        while (true) {
            LOGGER.info("Enter message to send (leave empty to exit): ");
            input = inputReader.readLine();
            if (input == null || input.isEmpty()) break;

            TextMessage message = session.createTextMessage(input);
            producer.send(message);
            LOGGER.info("Send message {}", message.getJMSMessageID());
        }
    } catch (EOFException e) {
        // Just return on EOF
    } catch (IOException e) {
        LOGGER.error("Failed reading input: {}", e.getMessage(), e);
    } catch (JMSException e) {
        LOGGER.error("Failed sending message: {}", e.getMessage(), e);
    }
}

```

Ricezione di messaggi in modo sincrono.

```
/**
 * 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);
    }
}

```

Ricezione di messaggi in modo asincrono.

```

/**
 * 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(QUEUE_NAME));
                // Provide an implementation of the MessageListener interface,
            which has a single 'onMessage' method.
                // We use a lambda expression for the implementation.
                consumer.setMessageListener(message -> {
                    try {
                        SqsJmsExampleUtils.handleMessage(message);
                        message.acknowledge();
                    } catch (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" );
}

```

Ricezione di messaggi utilizzando la modalità `CLIENT_ACKNOWLEDGE`.

```

/**
 * 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(QueueName));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(QueueName));

        // 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,
        LOG.info("Waiting for visibility timeout...");
        try {
            Thread.sleep(TIME_OUT_MILLIS);
        } catch (InterruptedException e) {
            LOG.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();
    }
}

```

Ricezione di messaggi utilizzando la modalità UNORDERED_ACKNOWLEDGE.

```

/**
 * 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(QueueName));
MessageConsumer consumer =
session.createConsumer(session.createQueue(QueueName));

// 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();
    }
}
}

```

Utilizzo di Spring per iniettare dipendenze.

```
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(STRING_XML_CONFIG_FILE);
        File springFile = new File(resource.getFile());
        if (!springFile.exists() || !springFile.canRead()) {
            LOGGER.error("File " + STRING_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.sendMessage(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);
}
}
}

```

Definizioni dei bean Spring.

```

<?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>
```

Una classe di utilità che fornisce metodi comuni utilizzati dagli altri esempi.

```
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);
}
};
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
 - [CreateQueue](#)
 - [DeleteQueue](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Lavora con i tag di coda e Amazon SQS utilizzando un SDK AWS

L'esempio di codice seguente mostra come eseguire operazioni di tagging con Amazon SQS.

Java

SDK per Java 2.x

Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

L'esempio seguente crea tag per una coda, elenca i tag e rimuove un tag.

```
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 tag a queue.
 * @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);
    }
}
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
 - [ListQueueTags](#)
 - [TagQueue](#)
 - [UntagQueue](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi serverless per Amazon SQS

I seguenti esempi di codice mostrano come usare Amazon SQS con AWS SDKs

Esempi

- [Invocazione di una funzione Lambda da un trigger Amazon SQS](#)
- [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS](#)

Invocazione di una funzione Lambda da un trigger Amazon SQS

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

.NET

SDK per .NET

Note

C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite .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
            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 per Go V2

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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 per Java 2.x

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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 per JavaScript (v3)

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento SQS con JavaScript Lambda utilizzando.

```
// 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;  
  }  
}
```

Consumo di un evento SQS con TypeScript Lambda utilizzando.

```
// 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 per PHP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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 per Python (Boto3)

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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 per Ruby

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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
    end
```

```

    rescue StandardError => err
      puts "An error occurred"
      raise err
    end
  end
end

```

Rust

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

.NET

SDK per .NET

Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite .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 per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Java 2.x

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per JavaScript (v3)

Note

C'è altro da fare. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione degli errori degli elementi batch SQS utilizzando Lambda. 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}`);
}

```

Segnalazione degli errori degli elementi batch SQS utilizzando Lambda. 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.');
```

PHP

SDK per PHP

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Python (Boto3)

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Ruby

Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 << {"itemIdIdentifier" => record['messageId']}
        end
      end

      sqs_batch_response["batchItemFailures"] = batch_item_failures
      return sqs_batch_response
    end
  end
end
```

Rust

SDK per Rust

Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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  
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Utilizzo di Amazon SQS con un SDK AWS](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Risoluzione dei problemi in Amazon SQS

Questo argomento fornisce consigli per la risoluzione di errori e problemi comuni che potresti riscontrare quando usi la console Amazon SQS, l'API Amazon SQS o altri strumenti con Amazon SQS. Se scopri un problema che non è elencato qui di seguito, puoi utilizzare il pulsante Feedback in questa pagina per segnalarlo.

Per ulteriori suggerimenti sulla risoluzione dei problemi e per risposte a domande comuni relative al supporto, visitare il [Knowledge Center di AWS](#).

Argomenti

- [Risolvi un errore di accesso negato in Amazon SQS](#)
- [Risolvi gli errori dell'API Amazon SQS](#)
- [Risolvi i problemi relativi alla coda di lettera morta di Amazon SQS e al redrive DLQ](#)
- [Risolvi i problemi di limitazione FIFO in Amazon SQS](#)
- [Risolvi i problemi relativi ai messaggi non restituiti per una chiamata API Amazon ReceiveMessage SQS](#)
- [Risolvi gli errori di rete di Amazon SQS](#)
- [Risoluzione dei problemi delle code di Amazon Simple Queue Service utilizzando AWS X-Ray](#)

Risolvi un errore di accesso negato in Amazon SQS

I seguenti argomenti trattano le cause `AccessDenied` o `AccessDeniedException` gli errori più comuni nelle chiamate API di Amazon SQS. Per ulteriori informazioni su come risolvere questi errori, consulta [Come si risolvono gli errori "" o "AccessDenied" nelle chiamate API AccessDeniedException di Amazon SQS?](#) nella Knowledge Center Guide.AWS

Esempi di messaggi di errore:

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

- oppure -

```
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.
```

Politica di coda di Amazon SQS e politica IAM

Per verificare se il richiedente dispone delle autorizzazioni appropriate per eseguire un'operazione Amazon SQS, procedi come segue:

- Identifica il principale IAM che effettua la chiamata all'API Amazon SQS. Se il principale IAM proviene dallo stesso account, la policy di coda di Amazon SQS o la policy AWS Identity and Access Management (IAM) devono includere le autorizzazioni per consentire esplicitamente l'accesso all'azione.
- Se il principale è un'entità IAM:
 - Puoi identificare il tuo utente o ruolo IAM controllando l'angolo in alto a destra di o utilizzando il Console di gestione AWS comando. [aws sts get-caller-identity](#)
 - Controlla tutte le policy IAM collegate al ruolo o all'utente IAM. È possibile utilizzare uno dei seguenti metodi:
 - Testa le policy IAM con IAM Policy [Simulator](#).
 - Esamina i vari [tipi di policy IAM](#).
 - Se necessario, [modifica la policy utente IAM](#).
 - Controlla la politica di coda e [modificala](#) se necessario.
- Se il principale è un AWS servizio, la policy di coda di Amazon SQS deve consentire esplicitamente l'accesso.
- Se il principale è un account principale, sia la policy di coda di Amazon SQS che la policy IAM devono consentire esplicitamente l'accesso.
- Se la policy utilizza un elemento condizionale, verifica che la condizione limiti l'accesso.

Important

Un rifiuto esplicito in entrambi i criteri ha la precedenza su un permesso esplicito. Ecco alcuni esempi di base di policy di [Amazon SQS](#).

AWS Key Management Service autorizzazioni

Se la tua coda Amazon SQS ha la [crittografia lato server \(SSE\)](#) attivata con un servizio gestito dal cliente AWS KMS key, le autorizzazioni devono essere concesse sia ai produttori che ai consumatori. Per confermare se una coda è crittografata, puoi utilizzare l'**KmsMasterKeyId** attributo [GetQueueAttributes](#) API o dalla console di coda in Encryption.

- [Autorizzazioni richieste per i produttori:](#)

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "<Key ARN>"
}
```

- [Autorizzazioni richieste per i consumatori:](#)

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "<Key ARN>"
}
```

- Autorizzazioni richieste per l'accesso [tra](#) account diversi:

```
{
  "Effect": "Allow",
  "Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "<Key ARN>"
}
```

Scegli una delle seguenti opzioni per abilitare la crittografia per una coda Amazon SQS:

- [SSE-Amazon SQS](#) (chiave di crittografia creata e gestita dal servizio Amazon SQS).
- [AWS chiave predefinita gestita](#) (alias/aws/sqs)
- [Chiave gestita dal cliente](#)

Tuttavia, se utilizzi una [chiave KMS AWS](#) gestita, non puoi modificare la politica delle chiavi predefinita. Pertanto, per fornire l'accesso ad altri servizi e account multipli, utilizza la chiave gestita dal cliente. In questo modo è possibile modificare la politica chiave.

Policy degli endpoint VPC

Se accedi ad [Amazon SQS tramite un endpoint Amazon Virtual Private Cloud \(Amazon VPC\)](#), la [policy degli endpoint VPC di Amazon SQS deve consentire](#) l'accesso. Puoi creare una policy per gli endpoint Amazon VPC per Amazon SQS, in cui puoi specificare quanto segue:

1. Il principale che può eseguire azioni.
2. Le azioni che possono essere eseguite.
3. Le risorse sui cui si possono eseguire azioni.

Nell'esempio seguente, la policy degli endpoint VPC specifica che l'utente IAM *MyUser* è autorizzato a inviare messaggi alla coda Amazon SQS. *MyQueue* Ad altre azioni, agli utenti IAM e alle risorse Amazon SQS viene negato l'accesso tramite l'endpoint VPC.

```
{
  "Statement": [{
    "Action": ["sqs:SendMessage"],
    "Effect": "Allow",
    "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

Politica di controllo dei servizi dell'organizzazione

Se fai Account AWS parte di un'organizzazione, AWS Organizations le policy possono impedirti di accedere alle code di Amazon SQS. Per impostazione predefinita, AWS Organizations le policy non bloccano alcuna richiesta ad Amazon SQS. Tuttavia, assicurati che AWS Organizations le tue policy non siano state configurate per bloccare l'accesso alle code di Amazon SQS. Per istruzioni su come controllare le tue AWS Organizations politiche, consulta [Elenco di tutte le politiche nella Guida](#) per l'AWS Organizations utente.

Risolvi gli errori dell'API Amazon SQS

I seguenti argomenti trattano gli errori più comuni restituiti durante le chiamate API di Amazon SQS e come risolverli.

QueueDoesNotExist errore

Questo errore viene restituito quando il servizio Amazon SQS non riesce a trovare la coda menzionata per l'azione Amazon SQS.

Possibili cause e mitigazioni:

- Regione errata: esamina la configurazione del client Amazon SQS per confermare di aver configurato la regione corretta sul client. Se non configuri una regione sul client, utilizza l'SDK o AWS CLI scegli la regione dal [file di configurazione o dalla variabile](#) di ambiente. Se l'SDK non trova una regione nel file di configurazione, l'SDK imposta la regione su us-east-1 per impostazione predefinita.
- La coda potrebbe essere stata eliminata di recente: se la coda è stata eliminata prima di effettuare la chiamata API, la chiamata API restituirà questo errore. Verifica la presenza CloudTrail di eventuali [DeleteQueue](#) operazioni prima del momento in cui si è verificato l'errore.
- Problemi di autorizzazione: se l'utente o il ruolo richiedente AWS Identity and Access Management (IAM) non dispone delle autorizzazioni richieste, potresti ricevere il seguente errore:

```
The specified queue does not exist or you do not have access to it.
```

Controlla le autorizzazioni ed effettua la chiamata API con le autorizzazioni corrette.

Per ulteriori dettagli sulla risoluzione dell'`QueueDoesNotExist` errore, consulta [Come posso risolvere l' `QueueDoesNotExist` errore quando effettuo chiamate API alla mia coda Amazon SQS?](#) nella Knowledge Center Guide.AWS

InvalidAttributeValue errore

Questo errore verrà restituito dopo l'aggiornamento della policy delle risorse di coda di Amazon SQS o delle proprietà con una policy o un principal errati.

Possibili cause e mitigazioni:

- Politica sulle risorse non valida: verifica che la politica delle risorse contenga tutti i campi obbligatori. Per ulteriori informazioni, consulta [IAM JSON Policy Elements Reference](#) e [Validating IAM Policy](#). Puoi anche utilizzare il [generatore di policy IAM](#) per creare e testare una policy di risorse Amazon SQS. Assicurati che la policy sia in formato JSON.
- Principal non valido: assicurati che l'Principal elemento esista nella politica delle risorse e che il valore sia valido. Se l'Principal elemento della policy delle risorse di Amazon SQS include un'entità IAM, assicurati che l'entità esista prima di utilizzare la policy. Amazon SQS convalida la politica delle risorse e verifica l'entità IAM. Se l'entità IAM non esiste, riceverai un errore. Per confermare le entità IAM, usa [GetRole](#) e [GetUser](#) APIs.

Per ulteriori informazioni su come risolvere un `InvalidAttributeValue` errore, consulta [Come si risolve l' `QueueDoesNotExist` errore quando effettuo chiamate API alla mia coda Amazon SQS?](#) nella Knowledge Center Guide.AWS

ReceiptHandle errore

Al momento di effettuare una chiamata [DeleteMessage](#) API, l'errore `ReceiptHandleIsInvalid` o `InvalidParameterValue` potrebbe essere restituito se l'handle di ricezione non è corretto o è scaduto.

- `ReceiptHandleIsInvalid` errore: Se la gestione delle ricevute non è corretta, riceverai un errore simile a questo esempio:

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- `InvalidParameterValue` errore: Se il codice di ricezione è scaduto, riceverai un errore simile a questo esempio:

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:  
Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The  
receipt handle has expired.
```

Possibili cause e mitigazioni:

L'handle di ricezione viene creato per ogni messaggio ricevuto ed è valido solo per il periodo di timeout di visibilità. Quando il periodo di timeout di visibilità scade, il messaggio diventa visibile in coda per i consumatori. Quando ricevi nuovamente il messaggio dal consumatore, ricevi una nuova maniglia di ricezione. Per evitare errori di gestione delle ricevute errati o scaduti, utilizza il codice di ricezione corretto per eliminare il messaggio entro il periodo di timeout di visibilità della coda di Amazon SQS.

Per ulteriori informazioni su come risolvere un `ReceiptHandle` errore, consulta [Come si risolvono gli errori "" e "ReceiptHandleInvalid" quando utilizzo InvalidParameterValue la chiamata all'API Amazon SQS?](#) DeleteMessage nella Knowledge Center Guide.AWS

Risolvi i problemi relativi alla coda di lettera morta di Amazon SQS e al redrive DLQ

I seguenti argomenti trattano le cause più comuni dei problemi di Amazon SQS DLQ e DLQ redrive e come risolverli. Per ulteriori informazioni, consulta [Come posso risolvere i problemi di redrive DLQ di Amazon SQS?](#) nella Knowledge Center Guide.AWS

Problemi relativi al DLQ

Scopri i problemi DLQ più comuni e come risolverli.

La visualizzazione dei messaggi utilizzando la console può comportarne il trasferimento a una coda DLQ

Amazon SQS conteggia la visualizzazione di un messaggio nella console rispetto alle policy di redrive della coda corrispondente. Pertanto, se si visualizza un messaggio nella console il numero di volte specificato nella politica di redrive della coda corrispondente, il messaggio viene spostato nella coda di lettere morte della coda corrispondente.

Per correggere questo comportamento, puoi procedere in uno dei seguenti modi:

- Aumentare l'impostazione Maximum Receives (Ricezioni massime) per la policy di reindirizzamento della coda corrispondente.
- Evitare di visualizzare i messaggi della coda corrispondente nella console.

NumberOfMessagesSent e NumberOfMessagesReceived per una coda DLQ non corrispondono

Se invii un messaggio a una coda DLQ manualmente, viene acquisito dal parametro [NumberOfMessagesSent](#). Tuttavia, se viene inviato un messaggio a una coda DLQ a seguito di un tentativo di elaborazione non riuscito, non viene acquisito da questo parametro. Pertanto, è possibile che i valori [NumberOfMessagesReceived](#) di NumberOfMessagesSent e siano diversi.

Creazione e configurazione di un redrive per una coda di lettere non scritte

Il redrive della coda di lettere morte richiede l'impostazione delle autorizzazioni appropriate [per](#) Amazon SQS per ricevere messaggi dalla coda di lettere non scritte e inviare messaggi alla coda di destinazione. Se non disponi delle autorizzazioni corrette, l'operazione di reindirizzamento della coda con lettere morte può fallire. Puoi visualizzare lo stato dell'attività di reindirizzamento dei messaggi per risolvere i problemi e riprovare.

Gestione degli errori dei messaggi di coda standard e FIFO

[Le code standard](#) continuano a elaborare i messaggi fino alla scadenza del [periodo di conservazione](#). Questa elaborazione continua riduce al minimo le possibilità che la coda venga bloccata da messaggi non utilizzati. La presenza di un gran numero di messaggi che l'utente non riesce a eliminare ripetutamente può aumentare i costi e comportare un carico aggiuntivo sull'hardware. Per contenere i costi, sposta i messaggi non riusciti nella coda delle lettere non valide.

Le code standard consentono inoltre un numero elevato di messaggi in volo. Se la maggior parte dei tuoi messaggi non può essere consumata e non viene inviata a una coda di lettere morte, la velocità di elaborazione dei messaggi può rallentare. Per mantenere l'efficienza della coda, assicurati che l'applicazione gestisca correttamente l'elaborazione dei messaggi.

Le [code FIFO](#) garantiscono l'elaborazione "exactly-once" utilizzando messaggi in sequenza da un gruppo di messaggi. Pertanto, sebbene l'utente possa continuare a recuperare i messaggi ordinati da un altro gruppo di messaggi, il primo gruppo di messaggi rimane non disponibile fino a quando il messaggio che blocca la coda non viene elaborato correttamente o spostato in una coda di lettere morte.

Inoltre, le code FIFO consentono un numero inferiore di messaggi in transito. Per evitare che la coda FIFO venga bloccata da un messaggio, assicurati che l'applicazione gestisca correttamente l'elaborazione dei messaggi.

Per ulteriori informazioni, consultare [Quote di messaggi Amazon SQS](#) e [Best practice di Amazon SQS](#).

Problemi con DLQ-Redrive

Scopri i problemi più comuni di DLQ-Redrive e come risolverli.

AccessDenied problema di autorizzazione

L'AccessDenied errore si verifica quando il redrive DLQ fallisce perché l'entità AWS Identity and Access Management (IAM) non dispone delle autorizzazioni richieste.

Esempio di messaggio di errore:

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

Le seguenti autorizzazioni API sono necessarie per effettuare richieste di redrive DLQ:

Per avviare il reindirizzamento di un messaggio:

- Autorizzazioni per la coda di lettere non scritte:
 - `sqs:StartMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs>DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `kms:Decrypt`— Quando la coda delle lettere non scritte o la coda dei sorgenti originali sono crittografate.
- Autorizzazioni per la coda di destinazione:
 - `sqs:SendMessage`
 - `kms:GenerateDataKey`— Quando la coda di destinazione è crittografata.
 - `kms:Decrypt`— Quando la coda di destinazione è crittografata.

Per annullare la reindirizzazione di un messaggio in corso:

- Autorizzazioni per la coda di lettere non scritte:
 - `sqs:CancelMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `kms:Decrypt`— Quando la coda delle lettere non scritte o la coda dei sorgenti originali sono crittografate.

Per mostrare lo stato dello spostamento di un messaggio:

- Autorizzazioni per la coda di lettere non scritte:
 - `sqs:ListMessageMoveTasks`
 - `sqs:GetQueueAttributes`

Errore **NonExistentQueue**

L'`NonExistentQueue` errore si verifica quando la coda di origine di Amazon SQS non esiste o è stata eliminata. Controlla e reindirizza a una coda Amazon SQS presente.

Esempio di messaggio di errore:

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource errore

L'`CouldNotDetermineMessageSource` errore si verifica quando si tenta di avviare un redrive DLQ con i seguenti scenari:

- Un messaggio Amazon SQS inviato direttamente al DLQ con API. [SendMessage](#)
- Un messaggio dall'argomento AWS Lambda o dalla funzione Amazon Simple Notification Service (Amazon SNS) con il DLQ configurato.

Per risolvere questo errore, scegli Redrive verso una destinazione personalizzata quando avvii il redrive. Quindi, inserisci l'ARN della coda Amazon SQS per spostare tutti i messaggi dal DLQ alla coda di destinazione.

Esempio di messaggio di errore:

```
Failed: CouldNotDetermineMessageSource
```

Risolvi i problemi di limitazione FIFO in Amazon SQS

Per impostazione predefinita, le code FIFO supportano 300 transazioni al secondo, per azione API per [SendMessage](#) e [ReceiveMessageDeleteMessage](#). Le richieste superiori a 300 TPS generano l'`ThrottlingException` errore anche se i messaggi in coda sono disponibili. Per mitigare questo problema, puoi utilizzare i seguenti metodi:

- [Abilitazione di un throughput elevato per le code FIFO in Amazon SQS](#).
- Utilizza le azioni `SendMessageBatch` batch dell'API Amazon SQS `ChangeMessageVisibilityBatch` per aumentare il limite TPS fino a 3.000 messaggi al secondo per azione API e per ridurre i costi. `DeleteMessageBatch` Per l'`ReceiveMessageAPI`, imposta il `MaxNumberOfMessages` parametro per ricevere fino a dieci messaggi per transazione. Per ulteriori informazioni, consulta [Operazioni in batch per Amazon SQS](#).
- Per le code FIFO con velocità effettiva elevata, segui i consigli per [ottimizzare](#) l'utilizzo delle partizioni. Invia messaggi con lo stesso gruppo di messaggi in batch. IDs Elimina i messaggi o modifica i valori di timeout di visibilità dei messaggi in batch con handle di ricezione delle stesse `ReceiveMessage` richieste API.
- Aumenta il numero di valori univoci [MessageGroupId](#). Ciò consente una distribuzione uniforme tra le partizioni di coda FIFO. Per ulteriori informazioni, consulta [Using the Message Group ID di Amazon SQS](#).

Per ulteriori informazioni, consulta [Perché la mia coda FIFO di Amazon SQS non restituisce tutti i messaggi o i messaggi in altri gruppi di messaggi?](#) nella AWS Knowledge Center Guide.

Risolvi i problemi relativi ai messaggi non restituiti per una chiamata API Amazon ReceiveMessage SQS

I seguenti argomenti illustrano le cause più comuni per cui un messaggio Amazon SQS non può essere restituito ai consumatori e come risolverle. Per ulteriori informazioni, consulta [Perché non posso ricevere messaggi dalla mia coda Amazon SQS?](#) nella AWS Knowledge Center Guide.

Coda vuota

Per determinare se una coda è vuota, utilizza long polling per chiamare l'API. [ReceiveMessage](#). Puoi anche utilizzare le metriche

`ApproximateNumberOfMessagesVisible`, `ApproximateNumberOfMessagesNotVisible`, e `ApproximateNumberOfMessagesDelayed` CloudWatch . Se tutti i valori delle metriche sono impostati su 0 per diversi minuti, la coda viene considerata vuota.

È stato raggiunto il limite di volo

[Se utilizzi un polling lungo e se il limite di attesa della coda \(120000 per impostazione predefinita\) viene superato, Amazon SQS non restituirà messaggi di errore che superano i limiti di quota.](#)

Ritardo dei messaggi

Se la coda di Amazon SQS è configurata come coda di [ritardo](#) o i messaggi sono stati inviati con [timer per i messaggi](#), i messaggi non sono visibili fino al termine del periodo di ritardo. Per verificare se una coda è configurata come coda di ritardo, utilizza l'`DelaySeconds` attributo [GetQueueAttributes](#) API o dalla console di coda in Delivery delay. Controlla la [ApproximateNumberOfMessagesDelayed](#) CloudWatch metrica per capire se ci sono messaggi in ritardo.

Il messaggio è in corso

Se il messaggio è stato interrogato da un altro consumatore, il messaggio rimarrà inattivo o invisibile per il periodo di timeout di [visibilità](#). I sondaggi aggiuntivi potrebbero restituire una ricezione vuota. Controlla la [ApproximateNumberOfMessagesVisible](#) CloudWatch metrica per comprendere il numero di messaggi disponibili per la ricezione. Nel caso delle code FIFO, se un messaggio con l'ID del gruppo di messaggi è in transito, non verranno restituiti altri messaggi a meno che non elimini il messaggio o non diventi visibile. Questo perché l'[ordine dei messaggi](#) viene mantenuto a livello di gruppo di messaggi in una coda FIFO.

Metodo di sondaggio

Se utilizzi il [polling breve](#), (`WaitTimeSeconds` è 0) Amazon SQS campiona un sottoinsieme dei suoi server e restituisce messaggi solo da tali server. Pertanto, potresti non ricevere i messaggi anche se sono disponibili per la ricezione. Le successive richieste di sondaggio restituiranno i messaggi.

Se utilizzi un [polling lungo](#), Amazon SQS esegue il polling di tutti i server e invia una risposta dopo aver raccolto almeno un messaggio disponibile e fino al numero massimo specificato. Se il valore di `ReceiveMessageWaitTimeSeconds` è troppo basso, potresti non ricevere tutti i messaggi disponibili.

Risolvi gli errori di rete di Amazon SQS

I seguenti argomenti trattano le cause più comuni dei problemi di rete in Amazon SQS e come risolverli.

ETIMOUT error

L'ETIMOUT errore si verifica quando il client non riesce a stabilire una connessione TCP a un endpoint Amazon SQS.

Risoluzione dei problemi

- Controlla la connessione di rete

Verifica la tua connessione di rete ad Amazon SQS eseguendo comandi come `telnet`

Example: `telnet sqs.us-east-1.amazonaws.com 443`

- Controlla le impostazioni di rete
 - Assicurati che le regole del firewall locale, i percorsi e gli elenchi di controllo degli accessi (ACLs) consentano il traffico sulla porta che utilizzi.
 - Le regole in uscita (uscita) del gruppo di sicurezza devono consentire il traffico verso la porta 80 o 443.
 - Le regole ACL di rete in uscita (uscita) devono consentire il traffico verso la porta TCP 80 o 443.
 - Le regole ACL di rete in entrata (ingresso) devono consentire il traffico sulle porte TCP 1024-65535.
 - Le istanze Amazon Elastic Compute Cloud (Amazon EC2) che si connettono alla rete Internet pubblica devono disporre di connettività [Internet](#).
- Endpoint Amazon Virtual Private Cloud (Amazon VPC)

Se accedi ad Amazon SQS tramite un endpoint Amazon VPC, il gruppo di sicurezza degli endpoint deve consentire il traffico in entrata al gruppo di sicurezza del client sulla porta 443. L'ACL di rete associato alla sottorete dell'endpoint VPC deve avere questa configurazione:

- Le regole ACL di rete in uscita (uscita) devono consentire il traffico sulle porte TCP 1024-65535 (porte temporanee).
- Le regole ACL di rete in entrata (ingresso) devono consentire il traffico sulla porta 443.

Inoltre, la policy degli endpoint VPC AWS Identity and Access Management (IAM) di Amazon SQS deve consentire l'accesso. Il seguente esempio di politica degli endpoint VPC specifica che l'utente IAM *MyUser* è autorizzato a inviare messaggi alla coda Amazon SQS. *MyQueue*. Ad altre azioni, agli utenti IAM e alle risorse Amazon SQS viene negato l'accesso tramite l'endpoint VPC.

```
{
  "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

L'UnknownHostException errore si verifica quando non è possibile determinare l'indirizzo IP dell'host.

Risoluzione dei problemi

Utilizzate l'nslookup utilità per restituire l'indirizzo IP associato al nome host:

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI o SDK per gli endpoint legacy di Python:

```
nslookup <region>.queue.amazonaws.com
```

Se hai ricevuto un output non riuscito, segui le istruzioni in [Come funziona il DNS e come posso risolvere gli errori DNS](#) parziali o intermittenti? nella Knowledge Center Guide.AWS

Se hai ricevuto un output valido, è probabile che si tratti di un problema a livello di applicazione. Per risolvere i problemi a livello di applicazione, prova i seguenti metodi:

- Riavvia l'applicazione.
- Verifica che l'applicazione Java non abbia una cache DNS difettosa. Se possibile, configura l'applicazione in modo che aderisca al TTL DNS. Per ulteriori informazioni, vedere [Impostazione del TTL JVM per le ricerche dei nomi DNS](#).

Per ulteriori informazioni su come risolvere gli errori di rete, consulta [Come si risolvono gli errori di connessione «ETIMETIMEOUT» e «» di Amazon SQS](#)? UnknownHostException AWS nella Knowledge Center Guide.

Risoluzione dei problemi delle code di Amazon Simple Queue Service utilizzando AWS X-Ray

AWS X-Ray raccoglie dati sulle richieste servite dall'applicazione e consente di visualizzare e filtrare i dati per identificare potenziali problemi e opportunità di ottimizzazione. Per ogni richiesta tracciata all'applicazione, è possibile visualizzare informazioni dettagliate sulla richiesta, sulla risposta e sulle chiamate effettuate dall'applicazione alle AWS risorse a valle, ai microservizi, ai database e al Web HTTP. APIs

Per inviare intestazioni di AWS X-Ray traccia tramite Amazon SQS, puoi effettuare una delle seguenti operazioni:

- Usare l'[intestazione di traccia](#) X-Amzn-Trace-Id.
- Usare gli [attributi del sistema di messaggi](#) AWSTraceHeader.

Per raccogliere dati relativi a errori e latenza, è necessario analizzare il client [AmazonSQS](#) utilizzando l'[AWS SDK X-Ray](#).

Puoi utilizzare la AWS X-Ray console per visualizzare la mappa delle connessioni tra Amazon SQS e altri servizi utilizzati dall'applicazione. È inoltre possibile utilizzare la console per visualizzare i parametri come la latenza media e le percentuali di errore. Per ulteriori informazioni, consultare [Amazon SQS e AWS X-Ray](#) nella Guida per sviluppatori di AWS X-Ray .

Sicurezza in Amazon SQS

Questa sezione fornisce informazioni sulla sicurezza di Amazon SNS, l'autenticazione e il controllo degli accessi, oltre alla sintassi delle policy di accesso Amazon SQS.

Argomenti

- [Protezione dei dati in Amazon SQS](#)
- [Identity and Access Management in Amazon SQS](#)
- [Registrazione e monitoraggio in Amazon SQS](#)
- [Convalida della conformità per Amazon SQS](#)
- [Resilienza in Amazon SQS](#)
- [Sicurezza dell'infrastruttura in Amazon SQS](#)
- [Best practice di sicurezza per Amazon SQS](#)

Protezione dei dati in Amazon SQS

Il modello di [responsabilità AWS condivisa modello](#) si applica alla protezione dei dati in Amazon Simple Queue Service. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per maggiori informazioni sulla privacy dei dati, consulta le [Domande frequenti sulla privacy dei dati](#). Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al [AWS Modello di responsabilità condivisa e GDPR](#) nel AWS Blog sulla sicurezza.

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- SSL/TLS Da utilizzare per comunicare con AWS le risorse. È richiesto TLS 1.2 ed è consigliato TLS 1.3.

- Configura l'API e la registrazione delle attività degli utenti con AWS CloudTrail. Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta [Lavorare con i CloudTrail percorsi](#) nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il [Federal Information Processing Standard \(FIPS\) 140-3](#).

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori con Amazon SQS o altro Servizi AWS utilizzando la console, l'API o AWS CLI AWS SDKs I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per la fatturazione o i log di diagnostica. Quando si fornisce un URL a un server esterno, suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la richiesta al server.

Le sezioni seguenti forniscono informazioni sulla protezione dei dati in Amazon SQS.

Crittografia dei dati in Amazon SQS

La protezione dei dati ha lo scopo di proteggere i dati sia in transito (durante la trasmissione verso e da Amazon SQS), sia quando sono inattivi (ovvero quando sono archiviati su disco nei data center Amazon SQS). È possibile proteggere i dati in transito tramite il protocollo Secure Sockets Layer (SSL) o tramite la crittografia lato client. Per impostazione predefinita, Amazon SQS archivia messaggi e file utilizzando la crittografia del disco. Puoi proteggere i dati inattivi richiedendo ad Amazon SQS di crittografare i tuoi messaggi prima di salvarli nel file system crittografato dei suoi data center. Amazon SQS consiglia di utilizzare SSE per la crittografia ottimizzata dei dati.

Argomenti

- [Crittografia inattiva in Amazon SQS](#)
- [Gestione delle chiavi Amazon SQS](#)

Crittografia inattiva in Amazon SQS

La crittografia lato server (SSE) consente di trasmettere dati sensibili in code crittografate. SSE protegge il contenuto dei messaggi nelle code utilizzando chiavi di crittografia gestite da SQL (SSE-SQS) o chiavi gestite in (SSE-KMS). AWS Key Management Service Per informazioni sulla gestione di SSE tramite, consulta quanto segue: Console di gestione AWS

- [Configurare SSE-SQS per una coda \(console\)](#)
- [Configurare SSE-KMS per una coda \(console\)](#)

Per informazioni sulla gestione di SSE utilizzando le [GetQueueAttributes](#) azioni AWS SDK per Java (e [CreateQueueSetQueueAttributes](#), e), consulta i seguenti esempi:

- [Utilizzo della crittografia lato server con le code Amazon SQS](#)
- [Configurazione delle autorizzazioni KMS per Servizi AWS](#)

SSE esegue la crittografia dei messaggi non appena vengono ricevuti da Amazon SQS. I messaggi sono archiviati in forma crittografata e decrittografati da Amazon SQS solo quando vengono inviati a un consumatore autorizzato.

Important

Tutte le richieste alle code con la funzione SSE abilitata devono utilizzare HTTPS e [Signature Version 4](#).

Una [coda crittografata](#) che utilizza la chiave predefinita (chiave KMS AWS gestita per Amazon SQS) non può richiamare una funzione Lambda in un'altra. Account AWS

Alcune funzionalità dei AWS servizi che possono inviare notifiche ad Amazon SQS utilizzando l' AWS Security Token Service [AssumeRole](#) azione sono compatibili con SSE ma funzionano solo con le code standard:

- [Hook del ciclo di vita di dimensionamento automatico](#)
- [AWS Lambda Code DLQ](#)

Per informazioni sulla compatibilità di altri servizi con code crittografate, consulta [AWS Configura le autorizzazioni KMS per i servizi](#) e la documentazione del servizio.

AWS KMS combina hardware e software sicuri e ad alta disponibilità per fornire un sistema di gestione delle chiavi scalabile per il cloud. Quando usi Amazon SQS con AWS KMS, anche [le chiavi dati](#) che crittografano i dati dei messaggi vengono crittografate e archiviate con i dati che proteggono.

Di seguito sono elencati i vantaggi derivanti dall'uso di AWS KMS:

- È possibile creare e gestire [AWS KMS keys](#) in modo autonomo.
- Puoi anche utilizzare la chiave KMS AWS gestita per Amazon SQS, che è unica per ogni account e regione.
- Gli standard AWS KMS di sicurezza possono aiutarti a soddisfare i requisiti di conformità relativi alla crittografia.

Per ulteriori informazioni, consulta [Cos'è AWS Key Management Service?](#) nella Guida per gli sviluppatori AWS Key Management Service .

Ambito della crittografia

SSE crittografa il corpo di un messaggio in una coda Amazon SQS.

SSE non esegue la crittografia di quanto segue:

- Metadati della coda (nome e attributi della coda)
- Metadati del messaggio (ID messaggio, timestamp e attributi)
- Metriche per coda

La crittografia di un messaggio ne rende i contenuti non disponibili a utenti non autorizzati o anonimi. Con SSE abilitato, le richieste anonime `SendMessage` e `ReceiveMessage` alla coda crittografata verranno rifiutate. Le best practice di sicurezza di Amazon SQS consigliano di non utilizzare richieste anonime. Se desideri inviare richieste anonime a una coda Amazon SQS, assicurati di disabilitare SSE. Ciò non ha implicazioni sul normale funzionamento di Amazon SQS:

- Un messaggio viene crittografato solo se inviato dopo che la crittografia di una coda è abilitata. Amazon SQS non crittografa i messaggi in backlog.
- Tutti i messaggi crittografati restano crittografati anche se la crittografia della relativa coda è disabilitata.

Lo spostamento di un messaggio a una [coda dead-letter](#) non ne pregiudica la crittografia:

- Quando Amazon SQS sposta un messaggio da una coda di origine crittografata a una coda DLQ non crittografata, il messaggio rimane crittografato.
- Quando Amazon SQS sposta un messaggio da una coda di origine non crittografata a una coda DLQ crittografata, il messaggio rimane non crittografato.

Termini chiave

I seguenti termini chiave possono aiutarti a comprendere meglio le funzionalità di SSE . Per una descrizione dettagliata, consulta la [Documentazione di riferimento per l'API Amazon Simple Notification Service](#).

Chiave di dati

La chiave (DEK) responsabile della crittografia dei contenuti dei messaggi Amazon SQS.

Per ulteriori informazioni, consulta [Chiavi di dati](#) nella Guida per sviluppatori di AWS Key Management Service nella Guida per sviluppatori di AWS Encryption SDK .

Periodo di riutilizzo della chiave di dati

Il periodo di tempo, in secondi, durante il quale Amazon SQS può riutilizzare una chiave dati per crittografare o decrittografare i messaggi prima di effettuare una nuova chiamata. AWS KMS Numero intero che rappresenta secondi, tra 60 (1 minuto) e 86.400 (24 ore). Il valore predefinito è 300 (5 minuti). Per ulteriori informazioni, consulta [Informazioni sul periodo di riutilizzo della chiave di dati](#).

Note

Nell'improbabile eventualità di impossibilità di raggiungerla AWS KMS, Amazon SQS continua a utilizzare la chiave dati memorizzata nella cache fino a quando non viene ristabilita una connessione.

ID della chiave KMS

L'alias, l'alias ARN, l'ID della chiave o l'ARN della chiave KMS AWS gestita o di una chiave KMS personalizzata, nel tuo account o in un altro account. Sebbene l'alias della chiave KMS AWS gestita per Amazon SQS sia `alias/aws/sqs` sempre, l'alias di una chiave KMS personalizzata può, ad esempio, essere. `alias/MyAlias` Puoi utilizzare queste chiavi KMS per proteggere i messaggi nelle code Amazon SQS.

 Note

Ricorda quanto segue:

- Se non specifichi una chiave KMS personalizzata, Amazon SQS utilizza AWS la chiave KMS gestita per Amazon SQS.
- La prima volta che utilizzi Console di gestione AWS per specificare la chiave KMS AWS gestita per Amazon SQS per una coda AWS KMS , crea AWS la chiave KMS gestita per Amazon SQS.
- In alternativa, la prima volta che utilizzi l'SendMessageBatchazione SendMessage o su una coda con SSE abilitato, AWS KMS crea la chiave KMS AWS gestita per Amazon SQS.

Puoi creare chiavi KMS, definire le politiche che controllano l'utilizzo delle chiavi KMS e controllare l'utilizzo delle chiavi KMS utilizzando la sezione Customer managed keys della console o dell'azione. AWS KMS [CreateKey](#) AWS KMS Per ulteriori informazioni, consulta [Chiavi KMS](#) e [Creazione di chiavi](#) nella Guida per gli sviluppatori di AWS Key Management Service . Per altri esempi di identificatori di chiave KMS, consulta [KeyId](#) l'API Reference.AWS Key Management Service Per ulteriori informazioni su come individuare gli identificatori KMS, consulta [Trovare l'ARN e l'ID chiave](#) nella Guida per sviluppatori di AWS Key Management Service .

 Important

Sono previsti costi aggiuntivi per l'utilizzo. AWS KMS Per ulteriori informazioni, consulta [Stima dei costi AWS KMS](#) e [Prezzi di AWS Key Management Service](#).

Crittografia a busta

La sicurezza dei dati crittografati dipende in parte dalla protezione della chiave di dati che può decrittarli. Amazon SQS utilizza la chiave KMS per crittografare la chiave di dati, quindi la chiave di dati crittografata viene archiviata con il messaggio crittografato. Questa pratica di utilizzare una chiave KMS per crittografare le chiavi di dati è nota come crittografia a busta.

Per ulteriori informazioni, consulta [Crittografia a busta](#) nella Guida per gli sviluppatori di AWS Encryption SDK .

Gestione delle chiavi Amazon SQS

Amazon SQS si integra con AWS Key Management Service (KMS) per gestire le [chiavi KMS per la crittografia lato server \(SSE\)](#). Per informazioni su SSE e definizioni di gestione delle chiavi, consulta la sezione [Crittografia inattiva in Amazon SQS](#). Amazon SQS utilizza le chiavi KMS per convalidare e proteggere le chiavi di dati che crittografano e decrittografano i messaggi. Nelle sezioni seguenti vengono fornite informazioni sull'utilizzo delle chiavi KMS e delle chiavi di dati nel servizio Amazon SQS.

Configurazione delle autorizzazioni AWS KMS

Ogni chiave KMS deve avere una policy delle chiavi. Tieni presente che non puoi modificare la politica delle chiavi di una chiave KMS AWS gestita per Amazon SQS. La policy per questa chiave KMS include le autorizzazioni per tutte le entità principali nell'account (che sono autorizzate a utilizzare Amazon SQS) per usare le code crittografate.

Amazon SQS distingue i chiamanti in base alle loro AWS credenziali, indipendentemente dal fatto che utilizzino AWS account, utenti IAM o ruoli IAM diversi. Inoltre, lo stesso ruolo IAM con politiche di scoping diverse verrà trattato come richiedenti distinti. Tuttavia, quando si utilizzano le sessioni di ruolo IAM, variando solo `RoleSessionName` mantenendo le stesse politiche di ruolo e ambito IAM non si creano richiedenti distinti. Pertanto, quando specifichi i principi AWS KMS chiave delle policy, evita di fare affidamento solo sulle `RoleSessionName` differenze, poiché queste sessioni verranno trattate come lo stesso richiedente.

In alternativa, puoi specificare le autorizzazioni richieste in una policy IAM assegnata alle entità principal che producono e utilizzano messaggi crittografati. Per ulteriori informazioni, consulta [Utilizzo delle policy IAM con AWS KMS](#) nella Guida per gli sviluppatori di AWS Key Management Service .

Note

Sebbene sia possibile configurare le autorizzazioni globali per l'invio e la ricezione da Amazon SQS AWS KMS , è necessario denominare in modo esplicito l'ARN completo delle chiavi KMS in regioni Resource specifiche nella sezione di una policy IAM.

AWS Configura le autorizzazioni KMS per i servizi

Diversi AWS servizi fungono da sorgenti di eventi in grado di inviare eventi alle code di Amazon SQS. Per consentire a queste fonti di eventi di funzionare con code crittografate, devi creare una chiave

KMS gestita dal cliente e aggiungere le autorizzazioni nella policy chiave affinché il servizio utilizzi i metodi API richiesti. AWS KMS Esegui la procedura seguente per configurare le autorizzazioni.

Warning

Quando modifichi la chiave KMS per crittografare i tuoi messaggi Amazon SQS, tieni presente che i messaggi esistenti crittografati con la vecchia chiave KMS rimarranno crittografati con quella chiave. Per decrittografare questi messaggi, devi conservare la vecchia chiave KMS e assicurarti che la relativa politica delle chiavi conceda ad Amazon SQS le autorizzazioni per e. `kms:Decrypt` `kms:GenerateDataKey` Dopo l'aggiornamento a una nuova chiave KMS per crittografare nuovi messaggi, assicurati che tutti i messaggi esistenti crittografati con la vecchia chiave KMS vengano elaborati e rimossi dalla coda prima di eliminare o disabilitare la vecchia chiave KMS.

1. Crea una chiave KMS gestita dal cliente Per ulteriori informazioni, consulta [Creazione di chiavi](#) nella Guida per gli sviluppatori di AWS Key Management Service .
2. Per consentire all'origine degli eventi del AWS servizio di utilizzare i metodi `kms:Decrypt` e `kms:GenerateDataKey` API, aggiungi la seguente dichiarazione alla politica delle chiavi KMS.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "service.amazonaws.com"
    },
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": "*"
  }]
}
```


Sostituire "service" nell'esempio precedente con il nome del servizio dell'origine evento. Le origini eventi includono i seguenti servizi.

Origine eventi	Nome del servizio
CloudWatch Eventi Amazon	events.amazonaws.com
Notifiche di eventi Amazon S3	s3.amazonaws.com
Sottoscrizioni ad argomenti Amazon SNS	sns.amazonaws.com

3. [Configura una coda SSE esistente](#) utilizzando l'ARN della tua chiave KMS.
4. Fornire l'ARN della coda crittografata per l'origine eventi.

Configura le AWS KMS autorizzazioni per i produttori

Quando scade il [periodo di riutilizzo della chiave dati](#), la successiva chiamata del produttore a SendMessage o SendMessageBatch attiva anche le chiamate a kms:Decrypt e kms:GenerateDataKey. La chiamata a kms:Decrypt è per verificare l'integrità della nuova chiave dati prima di utilizzarla. Il produttore deve quindi avere le autorizzazioni kms:Decrypt e kms:GenerateDataKey per la chiave KMS.

Aggiungere la seguente istruzione alla policy IAM del produttore. Ricordarsi di utilizzare i valori ARN corretti per la risorsa chiave e la risorsa coda.

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"
}
]
```

Configura le AWS KMS autorizzazioni per i consumatori

Quando scade il periodo di riutilizzo della chiave dati, la successiva chiamata del consumatore a `ReceiveMessage` attiva anche una chiamata a `kms:Decrypt` per verificare l'integrità della nuova chiave dati prima di utilizzarla. Il consumatore deve avere l'autorizzazione `kms:Decrypt` per qualsiasi chiave KMS che viene utilizzata per crittografare i messaggi nella coda specificata. Se una coda funge da [coda DLQ](#), il consumatore deve avere anche l'autorizzazione `kms:Decrypt` per qualsiasi chiave KMS che viene utilizzata per crittografare i messaggi nella coda di origine. Aggiungere la seguente istruzione alla policy IAM del consumatore. Ricordarsi di utilizzare i valori ARN corretti per la risorsa chiave e la risorsa coda.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-2:123456789012:key/11112222233333"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
    }
  ]
}
```

```
    ]
  }
}
```

Configura le AWS KMS autorizzazioni con una protezione sostitutiva confusa

Quando il principale di una istruzione della policy della chiave è un [Principale del servizio AWS](#), è possibile utilizzare le chiavi di condizione globali [aws:SourceArn](#) o [aws:SourceAccount](#) per proteggersi dal [problema del "confused deputy"](#). Per utilizzare queste chiavi di condizione, impostare il valore sul nome della risorsa Amazon (ARN) della risorsa crittografata. Se non si conosce l'ARN della risorsa, utilizzare `aws:SourceAccount`.

In questa policy della chiave KMS, una risorsa specifica del servizio di proprietà dell'account 111122223333 può richiamare KMS per le operazioni Decrypt e GenerateDataKey che si verificano durante l'utilizzo SSE di Amazon SQS.

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"
          ]
        }
      }
    }
  ]
}
```

Quando si utilizzano code Amazon SQS abilitate per SSE, sono supportati i seguenti servizi `aws:SourceArn`:

- Amazon SNS
- Simple Storage Service (Amazon S3)
- CloudWatch Eventi
- AWS Lambda
- CodeBuild
- Amazon Connect Customer Profiles
- AWS Auto Scaling
- Amazon Chime

Informazioni sul periodo di riutilizzo della chiave di dati

Il [periodo di riutilizzo della chiave di dati](#) definisce la durata massima per Amazon SQS per riutilizzare la stessa chiave dati. Quando termina il periodo di riutilizzo della chiave di dati, Amazon SQS genera una nuova chiave di dati. Prendere nota delle seguenti linee guida sul periodo di riutilizzo.

- Un periodo di riutilizzo più breve offre una maggiore sicurezza, ma comporta un maggior numero di chiamate verso AWS KMS, il che potrebbe comportare addebiti oltre il piano gratuito.
- Anche se la chiave di dati viene memorizzata nella cache separatamente per la crittografia e la decrittografia, il periodo di riutilizzo si applica a entrambe le copie della chiave di dati.
- Al termine del periodo di riutilizzo della chiave dati, la chiamata successiva `SendMessage` o `SendMessageBatch` in genere attiva una chiamata al AWS KMS `GenerateDataKey` metodo per ottenere una nuova chiave dati. Inoltre, ogni chiamata successiva a `SendMessage` e `ReceiveMessage` attiverà ciascuna una chiamata AWS KMS `Decrypt` a per verificare l'integrità della chiave dati prima di utilizzarla.
- [I responsabili](#) (Account AWS o gli utenti) non condividono le chiavi dati (i messaggi inviati da destinatari univoci ottengono sempre chiavi dati uniche). Pertanto, il volume delle chiamate verso AWS KMS è un multiplo del numero di principali univoci in uso durante il periodo di riutilizzo della chiave dati.

Stima dei costi AWS KMS

Per prevedere i costi e comprendere meglio la AWS fattura, potresti voler sapere con quale frequenza Amazon SQS utilizza la tua chiave KMS.

Note

Anche se la seguente formula può darti un'idea molto precisa dei costi previsti, i costi effettivi potrebbero essere più elevati a causa della natura diffusa di Amazon SQS.

Per calcolare il numero di richieste API (R) per coda, usa la formula seguente:

$$R = (B / D) * (2 * P + C)$$

B è il periodo di fatturazione (in secondi).

D è il [periodo di riutilizzo della chiave di dati](#) (in secondi).

P è il numero di [principali](#) produttori che effettuano invii alla coda Amazon SQS.

C è il numero di principali consumatori che ricevono dalla coda Amazon SQS.

Important

In generale, ai principali produttori viene addebitato un importo doppio rispetto ai principali consumatori. Per ulteriori informazioni, consulta [Informazioni sul periodo di riutilizzo della chiave di dati](#).

Se il produttore e l'utilizzatore hanno utenti diversi, il costo aumenta.

Di seguito vengono riportati esempi di calcolo. Per informazioni dettagliate sui prezzi, consulta [Prezzi di AWS Key Management Service](#).

Esempio 1: calcolo del numero di chiamate AWS KMS API per 2 principali e 1 coda

Questo esempio assume quanto segue:

- Il periodo di fatturazione è compreso tra il 1° e il 31 gennaio (2.678.400 secondi).

- Il periodo di riutilizzo della chiave di dati è impostato su 5 minuti (300 secondi).
- C'è una coda.
- C'è 1 principale produttore e 1 principale consumatore.

$$(2,678,400 / 300) * (2 * 1 + 1) = 26,784$$

Esempio 2: calcolo del numero di chiamate AWS KMS API per più produttori e consumatori e 2 code

Questo esempio assume quanto segue:

- Il periodo di fatturazione è compreso tra il 1° e il 28 febbraio (2.419.200 secondi).
- Il periodo di riutilizzo della chiave di dati è impostato su 24 ore (86.400 secondi).
- Ci sono 2 code.
- La prima coda ha 3 principali produttori e 1 principale consumatore.
- La seconda coda ha 5 principali produttori e 2 principali consumatori.

$$(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532$$

AWS KMS errori

Quando lavori con Amazon SQS e AWS KMS, potresti riscontrare errori. I seguenti riferimenti descrivono gli errori e le possibili soluzioni di risoluzione dei problemi.

- [Errori comuni AWS KMS](#)
- [Errori di decrittografia AWS KMS](#)
- [AWS KMS GenerateDataKey errori](#)

Privacy del traffico di rete in Amazon SQS

Un endpoint Amazon Virtual Private Cloud (Amazon VPC) per Amazon SQS è un'entità logica all'interno di un VPC che consente la connettività solo ad Amazon SQS. Il VPC instrada le richieste ad Amazon SQS e le risposte al VPC. Nelle sezioni seguenti vengono fornite informazioni sull'utilizzo degli endpoint VPC e sulla creazione delle policy di endpoint VPC.

Endpoint di Amazon Virtual Private Cloud per Amazon SQS

Se utilizzi Amazon VPC per ospitare AWS le tue risorse, puoi stabilire una connessione tra il tuo VPC e Amazon SQS. Puoi utilizzare questa connessione per inviare messaggi alle code Amazon SQS senza utilizzare la rete Internet pubblica.

Amazon VPC ti consente di avviare AWS risorse in una rete virtuale personalizzata. Puoi utilizzare un VPC per controllare le impostazioni di rete, come l'intervallo di indirizzi IP, le sottoreti, le tabelle di routing e i gateway di rete. Per ulteriori informazioni VPCs, consulta la [Amazon VPC User Guide](#).

Per connettere il tuo VPC a Amazon SQS, devi innanzitutto definire un'interfaccia dell'endpoint VPC, la quale ti consente di connettere il VPC ad altri servizi AWS . L'endpoint offre una connettività dimensionabile e affidabile a Amazon SQS senza richiedere un gateway internet, un'istanza NAT (Network Address Translation) o una connessione VPN. Per ulteriori informazioni, consulta [Tutorial: Invio di un messaggio a una coda Amazon SQS da Amazon Virtual Private Cloud](#) e [Esempio 5: Negare l'accesso se non è un endpoint VPC](#) in questa guida ed [Endpoint VPC dell'interfaccia \(AWS PrivateLink\)](#) nella Guida per l'utente di Amazon VPC.

Important

- Puoi utilizzare Amazon Virtual Private Cloud solo con endpoint Amazon SQS HTTPS.
- Quando configuri Amazon SQS per inviare messaggi da Amazon VPC, devi abilitare il DNS privato e specificare gli endpoint nel formato `sqs.us-east-2.amazonaws.com` o per l'endpoint dual-stack `sqs.us-east-2.api.aws`
- Amazon SQS supporta anche gli endpoint FIPS tramite l' PrivateLink utilizzo del servizio endpoint. `com.amazonaws.region.sqs-fips` Puoi connetterti agli endpoint FIPS nel formato `sqs-fips.region.amazonaws.com`
- Quando si utilizza l'endpoint dual-stack in Amazon Virtual Private Cloud, le richieste verranno inviate utilizzando e. IPv4 IPv6
- Il DNS privato non supporta endpoint precedenti come `queue.amazonaws.com` o `us-east-2.queue.amazonaws.com`.

Creazione di una policy per endpoint VPC di Amazon per Amazon SQS

È possibile creare una policy per gli endpoint VPC di Amazon per Amazon SQS per specificare quanto segue:

- Il principale che può eseguire azioni.
- Le azioni che possono essere eseguite.
- Le risorse sui cui si possono eseguire azioni.

Per ulteriori informazioni, consulta [Controllo degli accessi ai servizi con endpoint VPC](#) nella Guida per l'utente di Amazon VPC.

Il seguente esempio di policy di endpoint VPC specifica che l'utente `MyUser` è autorizzato a inviare messaggi alla coda Amazon SQS `MyQueue`.

```
{
  "Statement": [{
    "Action": ["sqs:SendMessage"],
    "Effect": "Allow",
    "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

Non si può accedere a quanto segue:

- Altre azioni API Amazon SQS, come `sqs:CreateQueue` e `sqs>DeleteQueue`.
- Altri utenti e ruoli che provano a utilizzare questo endpoint VPC.
- Invio di messaggi da `MyUser` a una coda Amazon SQS diversa.

Note

L'utente può ancora utilizzare altre azioni API Amazon SQS dall'esterno del VPC. Per ulteriori informazioni, consulta [Esempio 5: Negare l'accesso se non è un endpoint VPC](#).

Connect ad Amazon SQS utilizzando endpoint Dual-stack (e) IPv4 IPv6

Gli endpoint dual-stack supportano sia il traffico che il traffico. IPv4 IPv6 Quando effettui una richiesta a un endpoint dual-stack, l'URL dell'endpoint si risolve in un indirizzo o in un indirizzo. IPv4 IPv6 [Per ulteriori informazioni sugli endpoint dual-stack e FIPS, consulta la guida di riferimento SDK.](#)

Amazon SQS supporta endpoint dual-stack regionali, il che significa che devi specificare la AWS regione come parte del nome dell'endpoint. I nomi degli endpoint dual-stack utilizzano la seguente convenzione di denominazione: `sqs.Region.amazonaws.com`. Ad esempio, il nome dell'endpoint dual-stack per la Regione eu-west-1 è `sqs.eu-west-1.amazonaws.com`.

[Per l'elenco completo degli endpoint Amazon SQS, consulta la AWS Guida generale.](#)

Identity and Access Management in Amazon SQS

AWS Identity and Access Management (IAM) è un sistema Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM controllano chi può essere autenticato (accesso effettuato) e autorizzato (dispone di autorizzazioni) per utilizzare le risorse Amazon SQS. IAM è un software Servizio AWS che puoi utilizzare senza costi aggiuntivi.

Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia in base al tuo ruolo:

- Utente del servizio: richiedi le autorizzazioni all'amministratore se non riesci ad accedere alle funzionalità (consulta [Risoluzione dei problemi di identità e accesso ad Amazon Simple Queue Service](#))
- Amministratore del servizio: determina l'accesso degli utenti e invia le richieste di autorizzazione (consulta [Come funziona Amazon Simple Queue Service con IAM](#))
- Amministratore IAM: scrivi policy per gestire l'accesso (consulta [Best practice per le policy](#))

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi autenticarti come utente IAM o assumendo un ruolo IAM. Utente root dell'account AWS

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come AWS IAM Identity Center (IAM Identity Center), autenticazione Single Sign-On o credenziali. Google/Facebook Per ulteriori informazioni sull'accesso, consulta [Come accedere all' Account AWS](#) nella Guida per l'utente di Accedi ad AWS .

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta [AWS Signature Version 4 per le richieste API](#) nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un Account AWS, si inizia con un'identità di accesso denominata utente Account AWS root che ha accesso completo a tutte Servizi AWS le risorse. Consigliamo vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali dell'utente root, consulta [Attività che richiedono le credenziali dell'utente root](#) nella Guida per l'utente IAM.

Identità federata

Come procedura ottimale, richiedi agli utenti umani di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente della directory aziendale, del provider di identità Web o Directory Service che accede Servizi AWS utilizzando le credenziali di una fonte di identità. Le identità federate assumono ruoli che forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, si consiglia di utilizzare AWS IAM Identity Center. Per ulteriori informazioni, consulta [Che cos'è il Centro identità IAM?](#) nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un [utente IAM](#) è una identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ti consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali a lungo termine. Per ulteriori informazioni, consulta [Richiedere agli utenti umani di utilizzare la federazione con un provider di identità per accedere AWS utilizzando credenziali temporanee](#) nella Guida per l'utente IAM.

Un [gruppo IAM](#) specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta [Casi d'uso per utenti IAM](#) nella Guida per l'utente di IAM.

Ruoli IAM

Un [ruolo IAM](#) è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi assumere un ruolo [passando da un ruolo utente a un ruolo IAM \(console\)](#) o chiamando un'operazione AWS CLI o AWS API. Per ulteriori informazioni, consulta [Metodi per assumere un ruolo](#) nella Guida per l'utente di IAM.

I ruoli IAM sono utili per l'accesso degli utenti federati, le autorizzazioni utente IAM temporanee, l'accesso multi-account, l'accesso multi-servizio e le applicazioni in esecuzione su Amazon EC2. Per

maggiori informazioni, consultare [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.

Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e associandole a AWS identità o risorse. Una policy definisce le autorizzazioni quando è associata a un'identità o a una risorsa. AWS valuta queste politiche quando un preside effettua una richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta [Panoramica delle policy JSON](#) nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy controllano le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come creare una policy basata su identità, consultare [Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente](#) nella Guida per l'utente IAM.

Le policy basate su identità possono essere policy in linea (con embedding direttamente in una singola identità) o policy gestite (policy autonome collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta [Scegliere tra policy gestite e policy in linea](#) nella Guida per l'utente di IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi includono le policy di trust dei ruoli IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. In una policy basata sulle risorse è obbligatorio [specificare un'entità principale](#).

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi che possono impostare le autorizzazioni massime concesse dai tipi di policy più comuni:

- Limiti delle autorizzazioni: imposta il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta [Limiti delle autorizzazioni per le entità IAM](#) nella Guida per l'utente di IAM.
- Politiche di controllo del servizio (SCPs): specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa in AWS Organizations. Per ulteriori informazioni, consultare [Policy di controllo dei servizi](#) nella Guida per l'utente di AWS Organizations.
- Politiche di controllo delle risorse (RCPs): imposta le autorizzazioni massime disponibili per le risorse nei tuoi account. Per ulteriori informazioni, consulta [Politiche di controllo delle risorse \(RCPs\)](#) nella Guida per l'AWS Organizations utente.
- Policy di sessione: policy avanzate passate come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare [Policy di sessione](#) nella Guida per l'utente IAM.

Più tipi di policy

Quando a una richiesta si applicano più tipi di policy, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta [Logica di valutazione delle policy](#) nella IAM User Guide.

Panoramica sulla gestione degli accessi in Amazon SQS

Ogni AWS risorsa è di proprietà di un Account AWS utente e le autorizzazioni per creare o accedere a una risorsa sono regolate da politiche di autorizzazione. Un amministratore account può collegare le autorizzazioni di policy a identità IAM (utenti, gruppi e ruoli) e alcuni servizi (ad esempio Amazon SQS) supportano anche il collegamento delle policy di autorizzazione alle risorse.

Note

Un amministratore account (o un utente amministratore) è un utente con privilegi di amministratore. Per ulteriori informazioni, consulta [Best practice IAM](#) nella Guida per l'utente di IAM.

Nel concedere le autorizzazioni, devi specificare quali utenti riceveranno le autorizzazioni, la risorsa per cui acquisiscono le autorizzazioni e le azioni specifiche che desideri consentire sulla risorsa.

Risorse e operazioni di Amazon Simple Queue Service

In Amazon SQS, l'unica risorsa è la coda. In una policy, utilizza un Amazon Resource Name (ARN) per identificare la risorsa a cui si applica la policy stessa. La seguente risorsa dispone di un ARN univoco associato:

Tipo di risorsa	Formato ARN
Queue	<code>arn:aws:sqs: <i>region</i>:<i>account_id</i> :<i>queue_name</i></code>

Di seguito sono elencati alcuni esempi di formato ARN per le code:

- Un ARN per una coda denominata `my_queue` nella regione Stati Uniti orientali (Ohio), appartenente all'account 123456789012: AWS

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- Un ARN per una coda denominata `my_queue` in ciascuna delle diverse regioni che Amazon SQS supporta:

```
arn:aws:sqs:*:123456789012:my_queue
```

- Un ARN che utilizza `*` o `?` come carattere jolly per il nome della coda. Nei seguenti esempi, l'ARN corrisponde a tutte le code con prefisso `my_prefix_`:

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

Puoi ottenere il valore ARN per una coda esistente chiamando l'azione [GetQueueAttributes](#). Il valore dell'attributo `QueueArn` è l'ARN della coda. [Per ulteriori informazioni su ARNs, consulta IAM nella IAM User Guide. ARNs](#)

Amazon SQS offre un set di azioni che funzionano con la risorsa della coda. Per ulteriori informazioni, consulta [Autorizzazioni API Amazon SQS: riferimento a operazioni e risorse](#).

Informazioni sulla proprietà delle risorse

Account AWS Possiede le risorse create nell'account, indipendentemente da chi le ha create. Nello specifico, il proprietario della risorsa è l' Account AWS dell'entità principale (ovvero l'account root, un utente IAM o un ruolo IAM) che autentica la richiesta di creazione della risorsa. Negli esempi seguenti viene illustrato il funzionamento:

- Se utilizzi le credenziali dell'account root del tuo account Account AWS per creare una coda Amazon SQS, Account AWS sei il proprietario della risorsa (in Amazon SQS, la risorsa è la coda Amazon SQS).
- Se crei un utente nel tuo Account AWS e concedi le autorizzazioni per creare una coda all'utente, l'utente può creare la coda. Tuttavia, tieni presente che Account AWS (a cui appartiene l'utente) è il proprietario della risorsa della coda.
- Se crei un ruolo IAM Account AWS con le autorizzazioni per creare una coda Amazon SQS, chiunque possa assumere il ruolo può creare una coda. Il tuo Account AWS (a cui appartiene il ruolo) possiede la risorsa della coda.

Gestione dell'accesso alle risorse

Una policy di autorizzazioni descrive le autorizzazioni concesse agli account. Nella sezione seguente vengono descritte le opzioni disponibili per la creazione di policy relative alle autorizzazioni.

Note

In questa sezione viene descritto IAM nel contesto di Amazon SQS. Non vengono fornite informazioni dettagliate sul servizio IAM. Per la documentazione di IAM completa, consulta la pagina [Che cos'è IAM?](#) nella Guida per l'utente di IAM. Per informazioni sulla sintassi delle policy IAM e le rispettive descrizioni, consulta [Riferimento alle policy IAM di AWS](#) nella Guida per l'utente di IAM.

Le policy collegate a un'identità IAM vengono definite policy basate su identità (policy IAM), mentre quelle collegate a una risorsa vengono definite policy basate su risorse.

Policy basate sull'identità

Sono disponibili due modi per offrire agli utenti autorizzazioni per le code Amazon SQS: tramite il sistema di policy Amazon SQS e il sistema di policy IAM. Puoi usare uno dei due sistemi, oppure entrambi, per collegare policy a utenti o ruoli. Nella maggior parte dei casi, puoi ottenere lo stesso risultato utilizzando uno dei due sistemi. Ad esempio, puoi eseguire le operazioni seguenti:

- Collegare una policy di autorizzazione a un utente o a un gruppo nel tuo account: per concedere a un utente le autorizzazioni per creare una coda Amazon SQS, puoi associare una policy di autorizzazioni a un utente o a un gruppo a cui appartiene l'utente.
- Allega una politica di autorizzazione a un utente in un altro Account AWS: puoi allegare una politica di autorizzazioni a un utente di un altro utente Account AWS per consentirgli di interagire con una coda Amazon SQS. Tuttavia, le autorizzazioni per più account non si applicano alle seguenti azioni:

Le autorizzazioni per più account non sono applicabili alle azioni seguenti:

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

Per concedere l'accesso a queste azioni, l'utente deve appartenere allo stesso Account AWS proprietario della coda Amazon SQS.

- Associa una politica di autorizzazione a un ruolo (concedi autorizzazioni per più account): per concedere autorizzazioni su più account a una coda SQS, devi combinare sia le politiche IAM che quelle basate sulle risorse:

1. Nell'account A (che possiede la coda):

- Allega una policy basata sulle risorse alla coda SQS. Questa policy deve concedere esplicitamente le autorizzazioni necessarie (ad esempio [SendMessage](#), [ReceiveMessage](#)) al principale dell'Account B (come un ruolo IAM).

2. Nell'Account A, crea un ruolo IAM:

- Una politica di fiducia che consente all'account B o al Servizio AWS di assumere il ruolo.

Note

Se desideri che un Servizio AWS (come Lambda o EventBridge) assuma il ruolo, specifica il principale del servizio (ad esempio, `lambda.amazonaws.com`) nella politica di fiducia.

- Una politica basata sull'identità che concede al ruolo assunto le autorizzazioni per interagire con la coda.

3. Nell'Account B, concedi il permesso di assumere il ruolo nell'Account A.

È necessario configurare la politica di accesso della coda per consentire il collegamento principale tra account. Le policy basate sull'identità IAM da sole non sono sufficienti per l'accesso tra account diversi alle code SQS.

Per ulteriori informazioni sull'uso di IAM per delegare le autorizzazioni, consulta [Access Management](#) nella IAM User Guide (Guida per l'utente di IAM).

Anche se utilizza le policy IAM, Amazon SQS ha una propria infrastruttura di policy. Puoi utilizzare una policy Amazon SQS con una coda per specificare quali AWS account hanno accesso alla coda. Puoi specificare il tipo e le condizioni di accesso (per esempio, una condizione che consente di concedere autorizzazioni per l'utilizzo di `SendMessage`, `ReceiveMessage` se la richiesta viene effettuata prima del 31 dicembre 2010). Le azioni specifiche per cui puoi concedere autorizzazioni sono un sottoinsieme dell'intero elenco di azioni Amazon SQS. Quando scrivi una policy Amazon SQS e specifichi * per "consentire tutte le azioni Amazon SQS", significa che un utente può eseguire tutte le azioni in questo sottogruppo.

Il seguente diagramma mostra il concetto di una di queste policy Amazon SQS di base che copre il sottoinsieme di azioni. La policy è destinata a `queue_xyz` e concede all'AWS Account 1 e all'AWS Account 2 le autorizzazioni per utilizzare qualsiasi azione consentita con la coda specificata.

Note

La risorsa nella politica è specificata come `123456789012/queue_xyz`, dove `123456789012` è l'ID AWS account dell'account proprietario della coda.

SQS Policy on queue_xyz

Allow who:

AWS account 1

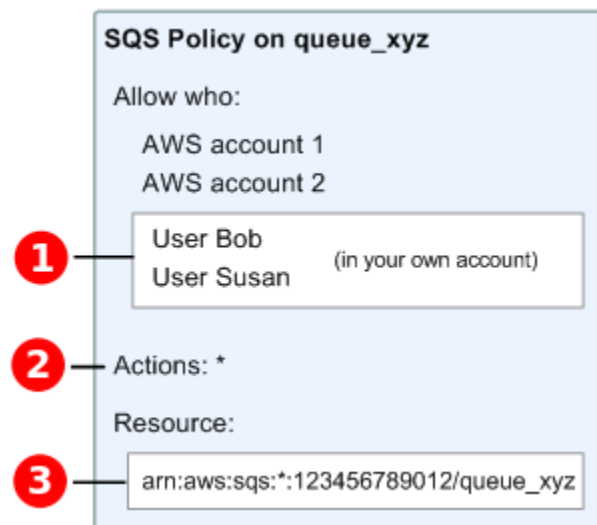
AWS account 2

Actions: *

Resource:

123456789012/queue_xyz

Con l'introduzione di IAM e i concetti di Users e Amazon Resource Names (ARNs), alcune cose sono cambiate nelle policy SQS. La tabella e il diagramma seguenti descrivono le modifiche.



1 Per informazioni su come concedere le autorizzazioni agli utenti in diversi account, consulta il [tutorial: Delegare l'accesso tra AWS account utilizzando ruoli IAM nella IAM User Guide](#).

2 Il sottoinsieme di operazioni incluse in * si è ampliato. Per un elenco di operazioni consentite, consulta [Autorizzazioni API Amazon SQS: riferimento a operazioni e risorse](#).

3

Puoi specificare la risorsa utilizzando il nome della risorsa Amazon (ARN), la modalità standard di specificare le risorse in policy IAM. Per maggiori informazioni sul formato ARN per le code Amazon SQS, consulta [Risorse e operazioni di Amazon Simple Queue Service](#).

Ad esempio, in base alla politica di Amazon SQS nel diagramma precedente, chiunque possieda le credenziali di sicurezza per l' AWS Account 1 o l'Account 2 può accedere. AWS queue_xyz Inoltre, gli utenti Bob e Susan nell'account AWS (con ID 123456789012) possono accedere alla coda.

Prima dell'introduzione di IAM, Amazon SQS forniva automaticamente al creatore di una coda il controllo completo sulla stessa (ovvero, l'accesso a tutte le azioni Amazon SQS possibili su tale coda). Questo non è più vero, a meno che il creatore non usi le credenziali di sicurezza AWS . Qualsiasi utente che dispone di autorizzazioni per creare una coda deve disporre anche delle autorizzazioni per utilizzare altre azioni Amazon SQS al fine di eseguire qualsiasi operazione con le code create.

Di seguito è riportato un esempio di policy che consente a un utente di utilizzare tutte le azioni Amazon SQS, ma solo con le code i cui nomi hanno come prefisso la stringa letterale bob_queue_.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
  }]
}
```

Per ulteriori informazioni, consulta [Utilizzo delle politiche con Amazon SQS](#) e [Identità \(utenti, gruppi e ruoli\)](#) nella Guida per l'utente IAM.

Specifica degli elementi delle policy: operazioni, effetti, risorse ed entità

Per ogni [risorsa Amazon Simple Queue Service](#), il servizio definisce un set di [azioni](#). Per concedere le autorizzazioni per queste azioni, Amazon SQS definisce un set di azioni che puoi specificare in una policy.

 Note

L'esecuzione di un'azione può richiedere le autorizzazioni per più di un'azione. Quando si concedono autorizzazione per operazioni specifiche, si identifica anche la risorsa per cui le operazioni sono concesse o negate.

Di seguito sono elencati gli elementi di base di una policy:

- **Risorsa:** in una policy si utilizza il nome della risorsa Amazon (ARN) per identificare la risorsa a cui si applica la policy stessa.
- **Operazione:** si utilizzano parole chiave per identificare le azioni sulla risorsa da consentire o rifiutare. Ad esempio, l'autorizzazione `sqs:CreateQueue` consente all'utente di eseguire l'azione Amazon Simple Queue Service `CreateQueue`.
- **Effetto:** l'effetto prodotto quando l'utente richiede l'operazione specifica, ovvero un'autorizzazione o un rifiuto. Se non concedi esplicitamente l'accesso a una risorsa, l'accesso viene implicitamente rifiutato. È anche possibile rifiutare esplicitamente l'accesso a una risorsa, per garantire che un utente non sia in grado di accedervi, anche se l'accesso viene concesso da un'altra policy.
- **Principale:** nelle policy basate su identità (policy IAM), l'utente a cui la policy è collegata è il principale implicito. Per policy basate su risorse, specifichi l'utente, l'account, il servizio o un'altra entità che desideri riceva le autorizzazioni (si applica solo alle policy basate su risorse).

Per ulteriori informazioni sulla sintassi e le descrizioni delle policy Amazon SQS, consulta [Riferimento alle policy IAM AWS](#) nella Guida per l'utente di IAM.

Per una tabella che mostra tutte le azioni Amazon Simple Queue Service e le risorse a cui si applicano, consulta [Autorizzazioni API Amazon SQS: riferimento a operazioni e risorse](#).

Come funziona Amazon Simple Queue Service con IAM

Prima di utilizzare IAM per gestire l'accesso ad Amazon SQS, scopri quali funzionalità IAM sono disponibili per l'uso con Amazon SQS.

Funzionalità IAM utilizzabili con Amazon Simple Queue Service

Funzionalità IAM	Supporto per Amazon SQS
Policy basate sull'identità	Sì
Policy basate su risorse	Sì
Operazioni di policy	Sì
Risorse relative alle policy	Sì
Chiavi di condizione della policy (specifica del servizio)	Sì
ACLs	No
ABAC (tag nelle policy)	Parziale
Credenziali temporanee	Sì
Inoltro delle sessioni di accesso (FAS)	Sì
Ruoli di servizio	Sì
Ruoli collegati al servizio	No

Per avere una visione di alto livello di come Amazon SQS e AWS altri servizi funzionano con la maggior parte delle funzionalità IAM, [AWS consulta i servizi che funzionano con](#) IAM nella IAM User Guide.

Controllo accessi

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano. AWS WAF ACLs Per ulteriori informazioni ACLs, consulta la [panoramica della lista di controllo degli accessi \(ACL\)](#) nella Amazon Simple Storage Service Developer Guide.

 Note

È importante capire che tutti Account AWS possono delegare le proprie autorizzazioni agli utenti tramite i propri account. L'accesso su più account ti consente di condividere l'accesso alle tue AWS risorse senza dover gestire utenti aggiuntivi. Per ulteriori informazioni sull'utilizzo dell'accesso multi-account, consultare la sezione relativa all'[abilitazione dell'accesso multiaccount](#) nella Guida dell'utente IAM.

Per ulteriori dettagli sulle autorizzazioni per più contenuti e sui codici di condizione all'interno delle policy personalizzate di Amazon SQS, consulta [Limitazioni delle politiche personalizzate di Amazon SQS](#).

Policy basate sull'identità per Amazon SQS

Supporta le policy basate sull'identità: sì

Le policy basate sull'identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta [Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente](#) nella Guida per l'utente di IAM.

Con le policy basate sull'identità di IAM, è possibile specificare quali operazioni e risorse sono consentite o respinte, nonché le condizioni in base alle quali le operazioni sono consentite o respinte. Per informazioni su tutti gli elementi utilizzabili in una policy JSON, consulta [Guida di riferimento agli elementi delle policy JSON IAM](#) nella Guida per l'utente IAM.

Esempi di policy basate su identità per Amazon SQS

Per visualizzare degli esempi di policy basate sull'identità di Amazon SQS, consulta la pagina [Best practice per le policy](#).

Policy basate sulle risorse all'interno di Amazon SQS

Supporta le policy basate sulle risorse: sì

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli IAM e le policy di bucket

Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. In una policy basata sulle risorse è obbligatorio [specificare un'entità principale](#). I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Per consentire l'accesso multi-account, è possibile specificare un intero account o entità IAM in un altro account come entità principale in una policy basata sulle risorse. Per ulteriori informazioni, consulta [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.

Operazioni delle policy per Amazon SQS

Supporta le operazioni di policy: si

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso in una policy. Includere le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per un elenco di operazioni di Amazon SQS, consulta [Risorse definite da Amazon Simple Queue Service](#) nella Documentazione di riferimento all'autorizzazione del servizio.

Le azioni di policy in Amazon SQS utilizzano il seguente prefisso prima dell'azione:

```
sqs
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [  
    "sqs:action1",  
    "sqs:action2"  
]
```

Per alcuni esempi di policy basate sull'identità di Amazon SQS, consulta la pagina [Best practice per le policy](#).

Risorse di policy per Amazon SQS

Supporta le risorse relative alle policy: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Come best practice, specifica una risorsa utilizzando il suo [nome della risorsa Amazon \(ARN\)](#). Per le azioni che non supportano le autorizzazioni a livello di risorsa, si utilizza un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

Per visualizzare un elenco dei tipi di risorse Amazon SQS e relativi ARNs, consulta [Actions Defined by Amazon Simple Queue Service nel Service Authorization](#) Reference. Per informazioni sulle azioni con cui è possibile specificare l'ARN di ciascuna risorsa, consulta [Risorse definite da Amazon Simple Queue Service](#).

Per alcuni esempi di policy basate sull'identità di Amazon SQS, consulta la pagina [Best practice per le policy](#).

Chiavi di condizione delle policy per Amazon SQS

Supporta le chiavi di condizione delle policy specifiche del servizio: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Condition` specifica quando le istruzioni vengono eseguite in base a criteri definiti. È possibile compilare espressioni condizionali che utilizzano [operatori di condizione](#), ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di [contesto delle condizioni AWS globali nella Guida](#) per l'utente IAM.

Per un elenco completo delle chiavi di condizione di Amazon SQS, consulta [Chiavi di condizione per Amazon Simple Queue Service](#) nella documentazione di riferimento all'autorizzazione di servizio. Per informazioni su operazioni e risorse con cui è possibile utilizzare una chiave di condizione, consulta [Risorse definite da Amazon Simple Queue Service](#).

Per alcuni esempi di policy basate sull'identità di Amazon SQS, consulta la pagina [Best practice per le policy](#).

ACLs in Amazon SQS

Supporti ACLs: No

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

ABAC con Amazon SQS

Supporta ABAC (tag nelle policy): parzialmente

Il controllo degli accessi basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi, chiamati tag. È possibile allegare tag a entità e AWS risorse IAM, quindi progettare politiche ABAC per consentire operazioni quando il tag del principale corrisponde al tag sulla risorsa.

Per controllare l'accesso basato su tag, fornire informazioni sui tag nell'[elemento condizione](#) di una policy utilizzando le chiavi di condizione `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Se un servizio supporta tutte e tre le chiavi di condizione per ogni tipo di risorsa, il valore per il servizio è Sì. Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per maggiori informazioni su ABAC, consulta [Definizione delle autorizzazioni con autorizzazione ABAC](#) nella Guida per l'utente di IAM. Per visualizzare un tutorial con i passaggi per l'impostazione di ABAC, consulta [Utilizzo del controllo degli accessi basato su attributi \(ABAC\)](#) nella Guida per l'utente di IAM.

Utilizzo di credenziali temporanee con Amazon SQS

Supporta le credenziali temporanee: sì

Le credenziali temporanee forniscono l'accesso a breve termine alle AWS risorse e vengono create automaticamente quando si utilizza la federazione o si cambia ruolo. AWS consiglia di generare dinamicamente credenziali temporanee anziché utilizzare chiavi di accesso a lungo termine. Per

ulteriori informazioni, consulta [Credenziali di sicurezza temporanee in IAM](#) e [Servizi AWS compatibili con IAM](#) nella Guida per l'utente IAM.

Sessioni di accesso diretto per Amazon SQS

Supporta l'inoltro delle sessioni di accesso (FAS): sì

Le sessioni di accesso inoltrato (FAS) utilizzano le autorizzazioni del principale chiamante al Servizio AWS, combinate con la richiesta di effettuare richieste Servizio AWS ai servizi downstream. Per i dettagli delle policy relative alle richieste FAS, consulta [Forward access sessions](#).

Ruoli di servizio per Amazon SQS

Supporta i ruoli di servizio: sì

Un ruolo di servizio è un [ruolo IAM](#) che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta [Create a role to delegate permissions to an Servizio AWS](#) nella Guida per l'utente IAM.

Warning

La modifica delle autorizzazioni per un ruolo di servizio potrebbe compromettere la funzionalità di Amazon SQS. Modifica i ruoli del servizio solo quando Amazon SQS fornisce le indicazioni per farlo.

Ruoli collegati ai servizi per Amazon SQS

Supporta i ruoli collegati ai servizi: no

Un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un Servizio AWS. Il servizio può assumere il ruolo per eseguire un'operazione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati al servizio, ma non modificarle.

Per ulteriori informazioni su come creare e gestire i ruoli collegati ai servizi, consulta [Servizi AWS supportati da IAM](#). Trova un servizio nella tabella che include un Yes nella colonna Service-linked role (Ruolo collegato ai servizi). Scegli il collegamento Sì per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Amazon SQS si aggiorna alle AWS policy gestite

Per aggiungere le autorizzazioni a utenti, gruppi e ruoli, è più semplice utilizzare policy gestite da AWS piuttosto che scrivere autonomamente le policy. La [creazione di policy gestite dai clienti IAM](#) che forniscono al tuo team solo le autorizzazioni di cui ha bisogno richiede tempo e competenza. Per iniziare rapidamente, utilizzare le nostre policy gestite da AWS. Queste policy coprono i casi d'uso più comuni e sono disponibili nel tuo account AWS. Per ulteriori informazioni sulle policy AWS gestite, consulta le [policy AWS gestite](#) nella IAM User Guide.

AWS i servizi mantengono e aggiornano le politiche AWS gestite. Non è possibile modificare le autorizzazioni nelle politiche AWS gestite. I servizi aggiungono occasionalmente autorizzazioni aggiuntive a una policy AWS gestita per supportare nuove funzionalità. Questo tipo di aggiornamento interessa tutte le identità (utenti, gruppi e ruoli) a cui è collegata la policy. È più probabile che i servizi aggiornino una politica AWS gestita quando viene lanciata una nuova funzionalità o quando diventano disponibili nuove operazioni. I servizi non rimuovono le autorizzazioni da una policy AWS gestita, quindi gli aggiornamenti delle policy non comprometteranno le autorizzazioni esistenti.

Inoltre, AWS supporta politiche gestite per le funzioni lavorative che si estendono su più servizi. Ad esempio, la policy ReadOnlyAccess AWS gestita fornisce l'accesso in sola lettura a tutti i AWS servizi e le risorse. Quando un servizio lancia una nuova funzionalità, AWS aggiunge autorizzazioni di sola lettura per nuove operazioni e risorse. Per l'elenco e la descrizione delle policy di funzione dei processi, consultare la sezione [Policy gestite da AWS per funzioni di processi](#) nella Guida per l'utente di IAM.

AWS politica gestita: Amazon SQSFull Access

È possibile allegare la policy AmazonSQSFullAccess alle identità Amazon SQS. Questa policy concede le autorizzazioni che consentono l'accesso completo ad Amazon SQS.

Per visualizzare le autorizzazioni per questa politica, consulta [Amazon SQSFull Access](#) nel AWS Managed Policy Reference.

AWS politica gestita: Amazon SQSRead OnlyAccess

È possibile allegare la policy AmazonSQSReadOnlyAccess alle identità Amazon SQS. Questa policy concede le autorizzazioni che consentono l'accesso in sola lettura ad Amazon SQS.

Per visualizzare le autorizzazioni per questa politica, consulta [Amazon SQSRead OnlyAccess](#) nel AWS Managed Policy Reference.

AWS politica gestita: SQSUnlock QueuePolicy

Se hai configurato in modo errato la politica di coda per un account membro in modo da negare a tutti gli utenti l'accesso alla tua coda Amazon SQS, puoi utilizzare la policy SQSUnlockQueuePolicy AWS gestita per sbloccare la coda.

Per ulteriori informazioni su come rimuovere una politica di coda mal configurata che impedisce a tutti i principali di accedere a una coda Amazon SQS, consulta [Esegui un' AWS Organizations attività privilegiata su un](#) account membro nella IAM User Guide.

Amazon SQS si aggiorna alle AWS policy gestite

Visualizza i dettagli sugli aggiornamenti delle politiche AWS gestite per Amazon SQS da quando questo servizio ha iniziato a tracciare queste modifiche. Per gli avvisi automatici sulle modifiche apportate a questa pagina, effettua la sottoscrizione al feed RSS nella pagina [Cronologia dei documenti](#) di Amazon SQS.

Modifica	Descrizione	Data
SQSUnlockQueuePolicy	Amazon SQS ha aggiunto una nuova policy AWS gestita chiamata SQSUnlockQueuePolicy per sbloccare una coda e rimuovere una policy di coda non configurata correttamente che impedisce a tutti i principali di accedere a una coda Amazon SQS.	15 novembre 2024
AmazonSQSReadOnlyAccess	Amazon SQS ha aggiunto l' ListQueueTags azione, che recupera tutti i tag associati a una coda Amazon SQS specificata. Consente	20 giugno 2024

Modifica	Descrizione	Data
	di visualizzare le coppie chiave-valore che sono state assegnate alla coda per scopi organizzativi o relativi ai metadati. Questa operazione è associata all'operazione API <code>ListQueueTags</code> .	
AmazonSQSReadOnlyAccess	Amazon SQS ha aggiunto una nuova azione che consente di elencare le attività di spostamento dei messaggi più recenti (fino a 10) in una coda di origine specifica. Questa operazione è associata all'operazione API ListMessageMoveTasks .	9 giugno 2023

Risoluzione dei problemi di identità e accesso ad Amazon Simple Queue Service

Utilizza le informazioni seguenti per eseguire la diagnosi e risolvere i problemi comuni che possono verificarsi durante l'utilizzo di Amazon SQS e IAM.

Non dispongo dell'autorizzazione per eseguire un'azione in Amazon SQS

Se ricevi un errore che indica che non disponi dell'autorizzazione per eseguire un'operazione, le tue policy devono essere aggiornate in modo che ti sei consentito eseguire tale operazione.

Il seguente esempio di errore si verifica quando l'utente `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa `my-example-widget` fittizia, ma non dispone di autorizzazioni `sqs:GetWidget` fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
sqs:GetWidget on resource: my-example-widget
```

In questo caso, la policy deve essere aggiornata in modo che Mateo possa accedere alla risorsa *my-example-widget* mediante l'operazione `sqs:GetWidget`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un errore che indica che non disponi dell'autorizzazione per eseguire l'azione `iam:PassRole`, per poter passare un ruolo ad Amazon SQS dovrai aggiornare le policy.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

Il seguente esempio di errore si verifica quando un utente IAM denominato `marymajor` cerca di utilizzare la console per eseguire un'azione in Amazon SQS. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse Amazon SQS

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per consentire alle persone di accedere alle tue risorse.

Per maggiori informazioni, consulta gli argomenti seguenti:

- Per sapere se Amazon SQS supporta queste funzionalità, consulta [Come funziona Amazon Simple Queue Service con IAM](#).

- Per scoprire come fornire l'accesso alle risorse di tua proprietà, consulta [Fornire l'accesso a un utente IAM in Account AWS un altro Account AWS di tua proprietà nella IAM User Guide](#).
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta [Fornire l'accesso a soggetti Account AWS di proprietà di terze parti](#) nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta [Fornire l'accesso a utenti autenticati esternamente \(federazione delle identità\)](#) nella Guida per l'utente IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente di IAM.

Voglio sbloccare la mia coda

Se fai Account AWS parte di un'organizzazione, AWS Organizations le policy possono impedirti di accedere alle risorse di Amazon SQS. Per impostazione predefinita, AWS Organizations le policy non bloccano alcuna richiesta ad Amazon SQS. Tuttavia, assicurati che AWS Organizations le tue policy non siano state configurate per bloccare l'accesso alle code di Amazon SQS. Per istruzioni su come controllare le tue AWS Organizations politiche, consulta [Elenco di tutte le politiche nella Guida](#) per l'AWS Organizations utente.

Inoltre, se hai configurato in modo errato la tua politica di coda per un account membro in modo da negare a tutti gli utenti l'accesso alla tua coda Amazon SQS, puoi sbloccare la coda avviando una sessione privilegiata per l'account membro in IAM. Una volta avviata una sessione privilegiata, puoi eliminare la politica di coda non configurata correttamente per riottenere l'accesso alla coda. Per ulteriori informazioni, consulta [Eseguire un'attività privilegiata su un account AWS Organizations membro](#) nella Guida per l'utente IAM.

Utilizzo delle politiche con Amazon SQS

In questo argomento vengono forniti esempi di policy basate su identità in cui un amministratore account può collegare policy di autorizzazione a identità IAM ovvero utenti, gruppi e ruoli.

Important

In primo luogo, è consigliabile esaminare gli argomenti introduttivi in cui vengono spiegati i concetti di base e le opzioni disponibili per gestire l'accesso alle risorse di Amazon Simple Queue Service. Per ulteriori informazioni, consulta [Panoramica sulla gestione degli accessi in Amazon SQS](#).

Ad eccezione di `ListQueues`, tutte le azioni Amazon SQS supportano autorizzazioni a livello di risorsa. Per ulteriori informazioni, consulta [Autorizzazioni API Amazon SQS: riferimento a operazioni e risorse](#).

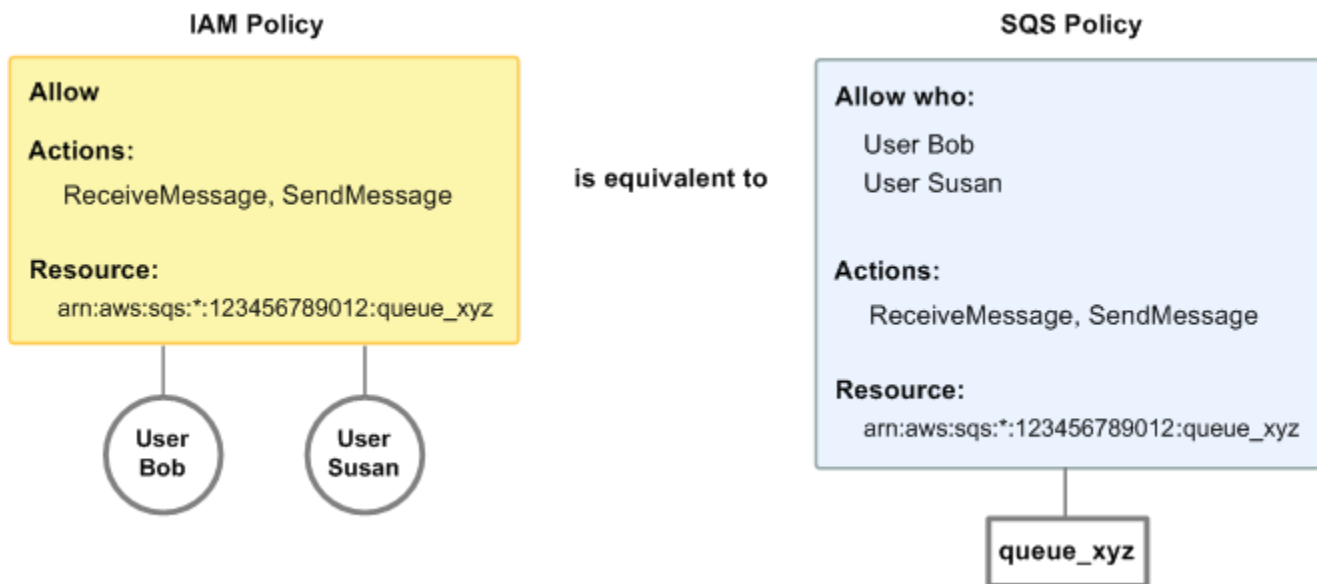
Utilizzo di policy Amazon SQS e IAM

Esistono due modi per concedere ai tuoi utenti le autorizzazioni per le tue risorse Amazon SQS: utilizzando il sistema di policy Amazon SQS (politiche basate sulle risorse) e utilizzando il sistema di policy IAM (politiche basate sull'identità). Puoi utilizzare uno o entrambi i metodi, ad eccezione dell'`ListQueues` azione, che è un'autorizzazione regionale che può essere impostata solo in una policy IAM.

Ad esempio, il diagramma seguente mostra l'equivalenza tra una policy IAM e una policy Amazon SQS. La policy IAM concede i diritti su `Amazon ReceiveMessage` SQS `SendMessage` e sulle azioni per la coda `queue_xyz` richiamata nel AWS tuo account e la policy è allegata agli utenti di nome Bob e Susan (Bob e Susan dispongono delle autorizzazioni indicate nella policy). Questa policy Amazon SQS fornisce inoltre a Bob e Susan i diritti per le azioni `ReceiveMessage` e `SendMessage` per la stessa coda.

Note

L'esempio seguente mostra politiche semplici senza condizioni. Puoi specificare una particolare condizione in una qualsiasi delle policy e ottenere lo stesso risultato.



C'è una differenza importante tra le policy di IAM e Amazon SQS: il sistema di policy di Amazon SQS consente di concedere l'autorizzazione ad AWS altri account, mentre IAM no.

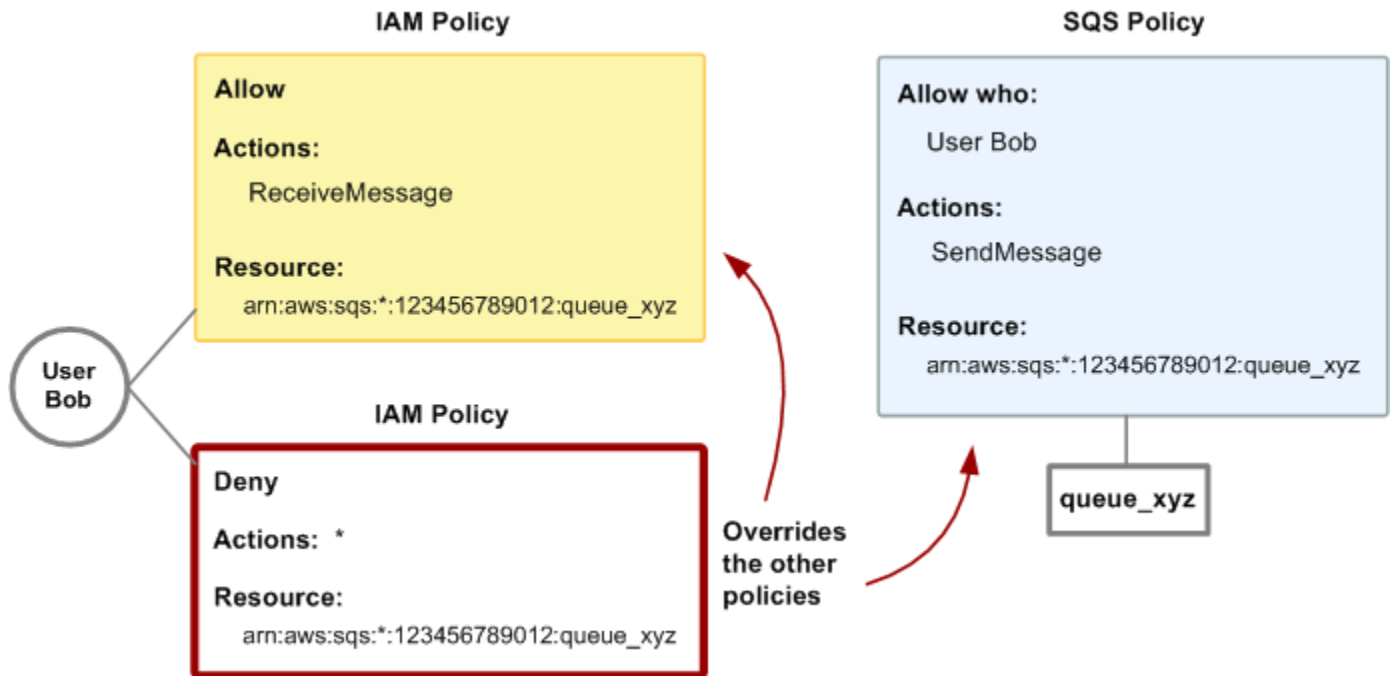
Sta a te decidere se utilizzare insieme i sistemi per gestire le autorizzazioni. Gli esempi seguenti mostrano il modo in cui i due sistemi di policy interagiscono.

- Nel primo esempio, Bob dispone sia di una policy IAM sia di una policy Amazon SQS applicabili all'account. La policy IAM concede all'account di Bob l'autorizzazione per l'azione ReceiveMessage su queue_xyz, mentre la policy Amazon SQS concede all'account l'autorizzazione per l'operazione SendMessage sulla stessa coda. Il diagramma seguente illustra questo concetto.



Se Bob invia una richiesta `ReceiveMessage` a `queue_xyz`, la policy IAM consente l'azione. Se Bob invia una richiesta `SendMessage` a `queue_xyz`, la policy Amazon SQS consente l'azione.

- Nel secondo esempio, Bob abusa del suo accesso a `queue_xyz`, pertanto è necessario rimuovere il suo intero accesso alla coda. La cosa più semplice da fare è aggiungere una policy che gli nega l'accesso a tutte le azioni per la coda. Questa policy sostituisce le altre due perché un `deny` esplicito sostituisce sempre un `allow`. Per ulteriori informazioni sulla logica di valutazione della policy, consulta [Utilizzo di policy personalizzate con la sintassi delle policy di accesso Amazon SQS](#). Il diagramma seguente illustra questo concetto.



Puoi anche aggiungere un'ulteriore istruzione alla policy Amazon SQS che nega a Bob qualsiasi tipo di accesso alla coda. Ha lo stesso effetto dell'aggiunta di una policy IAM che nega a Bob l'accesso alla coda. Per esempi di policy che coprono le azioni e le risorse di Amazon SQS, consulta [Esempi di base di policy Amazon SQS](#). Per ulteriori informazioni sulla sintassi di policy Amazon SQS, consulta [Utilizzo di policy personalizzate con la sintassi delle policy di accesso Amazon SQS](#).

Autorizzazioni necessarie per utilizzare la console Amazon SQS

Un utente che desidera utilizzare la console Amazon SQS deve disporre di un set di autorizzazioni minimo per utilizzare le code Amazon SQS nell'account Account AWS dell'utente. Ad esempio, l'utente deve avere l'autorizzazione di chiamare l'azione `ListQueues` per essere in grado di elencare

le code, o l'autorizzazione di chiamare l'azione `CreateQueue` per essere in grado di creare code. Oltre alle autorizzazioni Amazon SQS, per iscrivere una coda Amazon SQS a un argomento Amazon SNS, la console richiede anche le autorizzazioni per operazioni Amazon SNS.

Se decidi di creare una policy IAM più restrittiva relativa alle autorizzazioni minime richieste, la console potrebbe non funzionare come previsto per gli utenti con tale policy IAM.

Non è necessario consentire autorizzazioni minime per la console per gli utenti che effettuano chiamate solo verso le azioni AWS CLI o Amazon SQS.

Esempi di policy basate su identità per Amazon SQS

Per impostazione predefinita, gli utenti e i ruoli IAM non dispongono dell'autorizzazione per creare o modificare risorse Amazon SQS. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta [Creazione di policy IAM \(console\)](#) nella Guida per l'utente di IAM.

Per informazioni dettagliate sulle azioni e sui tipi di risorse definiti da Amazon SQS, incluso il formato di ARNs per ogni tipo di risorsa, consulta [Actions, Resources and Condition Keys for Amazon Simple Queue Service nel Service](#) Authorization Reference.

Note

Quando configuri gli hook del ciclo di vita per il Dimensionamento automatico Amazon EC2, non è necessario scrivere una policy per inviare messaggi a una coda Amazon SQS. Per ulteriori informazioni, consulta [Amazon EC2 Auto Scaling](#) Lifecycle Hooks nella Amazon EC2 User Guide.

Best practice per le policy

Le policy basate sull'identità determinano se qualcuno può creare, accedere o eliminare risorse Amazon SQS nell'account. Queste azioni possono comportare costi aggiuntivi per l'Account AWS. Quando si creano o modificano policy basate sull'identità, seguire queste linee guida e raccomandazioni:

- Inizia con le policy AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le policy gestite che concedono

le autorizzazioni per molti casi d'uso comuni. AWS Sono disponibili nel tuo Account AWS. Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai AWS clienti specifiche per i tuoi casi d'uso. Per maggiori informazioni, consulta [Policy gestite da AWS](#) o [Policy gestite da AWS per le funzioni dei processi](#) nella Guida per l'utente di IAM.

- Applicazione delle autorizzazioni con privilegio minimo - Quando si impostano le autorizzazioni con le policy IAM, concedere solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegio minimo. Per maggiori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta [Policy e autorizzazioni in IAM](#) nella Guida per l'utente di IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso - Per limitare l'accesso ad azioni e risorse è possibile aggiungere una condizione alle policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio CloudFormation. Per maggiori informazioni, consultare la sezione [Elementi delle policy JSON di IAM: condizione](#) nella Guida per l'utente di IAM.
- Utilizzo dello strumento di analisi degli accessi IAM per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali - Lo strumento di analisi degli accessi IAM convalida le policy nuove ed esistenti in modo che aderiscano al linguaggio (JSON) della policy IAM e alle best practice di IAM. Lo strumento di analisi degli accessi IAM offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per maggiori informazioni, consultare [Convalida delle policy per il Sistema di analisi degli accessi IAM](#) nella Guida per l'utente di IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungere le condizioni MFA alle policy. Per maggiori informazioni, consultare [Protezione dell'accesso API con MFA](#) nella Guida per l'utente di IAM.

Per maggiori informazioni sulle best practice in IAM, consulta [Best practice di sicurezza in IAM](#) nella Guida per l'utente di IAM.

Utilizzo della console Amazon SQS

Per accedere alla console Amazon Simple Queue Service, è necessario disporre di un set di autorizzazioni minimo. Queste autorizzazioni devono consentirti di elencare e visualizzare i dettagli sulle risorse Amazon SQS presenti nel tuo Account AWS. Se crei una policy basata sull'identità più

restrittiva rispetto alle autorizzazioni minime richieste, la console non funzionerà nel modo previsto per le entità (utenti o ruoli) associate a tale policy.

Non è necessario consentire autorizzazioni minime di console per gli utenti che effettuano chiamate solo verso AWS CLI o l'API. AWS Al contrario, è opportuno concedere l'accesso solo alle azioni che corrispondono all'operazione API che stanno cercando di eseguire.

Per garantire che utenti e ruoli possano continuare a utilizzare la console Amazon SQS, collega anche la policy gestita di Amazon AmazonSQSReadOnlyAccess AWS SQS alle entità. Per maggiori informazioni, consulta [Aggiunta di autorizzazioni a un utente](#) nella Guida per l'utente di IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono collegate alla relativa identità utente. Questa politica include le autorizzazioni per completare questa azione sulla console o utilizzando l'API o a livello di codice. AWS CLI 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": "*"
}
]
```

Consenti a un utente di creare code

Nell'esempio seguente creiamo una policy che consente a Bob di accedere a tutte le operazioni Amazon SQS, ma solo con code i cui nomi hanno come prefisso la stringa letterale `alice_queue_`.

Amazon SQS non concede automaticamente al creatore di una coda le autorizzazioni per utilizzare la coda. Pertanto, dobbiamo concedere esplicitamente a Bob le autorizzazioni per l'utilizzo di tutte le azioni Amazon SQS, oltre all'azione `CreateQueue` nella policy IAM.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
  }]
}
```

Consenti agli sviluppatori di scrivere messaggi in una coda condivisa

Nell'esempio seguente, creiamo un gruppo per sviluppatori e alleghiamo una policy che consente al gruppo di utilizzare l'azione `SendMessage` Amazon SQS, ma solo con la coda che appartiene a quella specificata Account AWS e ha un nome. `MyCompanyQueue`

JSON

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

```
"Action": "sqs:SendMessage",  
"Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"  
  }  
}
```

È possibile utilizzare `*` invece di `SendMessage` per concedere le seguenti operazioni a un principale su una coda condivisa: `ChangeMessageVisibility`, `DeleteMessage`, `GetQueueAttributes`, `GetQueueUrl`, `ReceiveMessage` e `SendMessage`.

Note

Sebbene `*` includa l'accesso fornito da altri tipi di autorizzazione, Amazon SQS considera le autorizzazioni separatamente. Ad esempio, è possibile concedere le autorizzazioni `*` e `SendMessage` a un utente, anche se un `*` include l'accesso fornito da `SendMessage`. Questo concetto si applica anche quando rimuovi un'autorizzazione. Se un principale dispone solo di un'autorizzazione `*`, la richiesta di rimuovere un'autorizzazione `SendMessage`, non lascia al principale un'autorizzazione di tipo tutto tranne. Al contrario, la richiesta non ha effetto, perché il principale non dispone di un'autorizzazione `SendMessage` esplicita. Per lasciare al principale solo l'autorizzazione `ReceiveMessage`, aggiungi prima l'autorizzazione `ReceiveMessage`, quindi rimuovi l'autorizzazione `*`.

Consenti ai manager di ottenere la dimensione generale delle code

Nell'esempio seguente, creiamo un gruppo per manager e alleghiamo una policy che consente al gruppo di utilizzare l'operazione `GetQueueAttributes` Amazon SQS con tutte le code che appartengono all'account specificato. AWS

JSON

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

Consenti a un partner di inviare messaggi a una coda specifica

Puoi eseguire questa operazione utilizzando una policy Amazon SQS o IAM. Se il tuo partner dispone di una Account AWS policy Amazon SQS, potrebbe essere più semplice utilizzare una policy Amazon SQS. Tuttavia, qualsiasi utente dell'azienda del partner che possiede le credenziali AWS di sicurezza può inviare messaggi alla coda. Se desideri limitare l'accesso a un determinato utente o applicazione, devi trattare il partner come un utente della tua azienda e utilizzare una policy IAM anziché una policy Amazon SQS.

In questo esempio vengono effettuate le seguenti operazioni:

1. Crea un gruppo chiamato WidgetCo a rappresentare l'azienda partner.
2. Creazione di un utente per l'utente o l'applicazione specifica presso l'azienda del partner che ha bisogno di accedere.
3. Aggiungere l'utente al gruppo .
4. Collegamento di una policy che concedere al gruppo l'accesso solo all'azione `SendMessage` solo per la coda denominata `WidgetPartnerQueue`.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
  }]
}
```

Esempi di base di policy Amazon SQS

Questa sezione mostra le policy di esempio per i casi d'uso Amazon SQS più comuni.

Puoi utilizzare la console per verificare gli effetti di ogni policy quando la allegghi all'utente.

Inizialmente l'utente non dispone di alcuna autorizzazione e non sarà in grado di eseguire alcuna operazione nella console. Quando le policy vengono collegate all'utente, è possibile verificare che l'utente possa eseguire diverse operazioni nella console.

 Note

Ti consigliamo di utilizzare due finestre del browser: una per concedere le autorizzazioni e l'altra per accedere Console di gestione AWS utilizzando le credenziali dell'utente per verificare le autorizzazioni man mano che le concedi all'utente.

Esempio 1: concedere un'autorizzazione a uno Account AWS

La seguente politica di esempio concede 111122223333 al Account AWS numero l'SendMessageautorizzazione per la coda denominata 444455556666/queue1 nella regione Stati Uniti orientali (Ohio).

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"
  }]
}
```

Esempio 2: concedere due autorizzazioni a una Account AWS

La politica di esempio seguente concede 111122223333 sia il Account AWS numero che SendMessage l'ReceiveMessageautorizzazione per la coda denominata. 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"
  }]
}
```

Esempio 3: concedere tutte le autorizzazioni a due Account AWS

La seguente politica di esempio concede due Account AWS numeri diversi (111122223333e444455556666) l'autorizzazione a utilizzare tutte le azioni a cui Amazon SQS consente l'accesso condiviso per la coda 123456789012/queue1 denominata nella regione Stati Uniti orientali (Ohio).

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"
  }]
}
```

```
}]
}
```

Esempio 4: Concedere autorizzazioni multi-account a un ruolo e un nome utente

La seguente politica di esempio concede `role1` e `username1` in base al Account AWS numero, 111122223333 l'autorizzazione per più account a utilizzare tutte le azioni a cui Amazon SQS consente l'accesso condiviso per la coda 123456789012/queue1 denominata nella regione Stati Uniti orientali (Ohio).

Le autorizzazioni per più account non sono applicabili alle azioni seguenti:

- [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"
}]
}

```

Esempio 5: Concedere un'autorizzazione a tutti gli utenti

La seguente policy di esempio concede a tutti gli utenti (utenti anonimi) l'autorizzazione `ReceiveMessage` per la coda denominata `111122223333/queue1`.

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"
  }]
}

```

Esempio 6: Concedere un'autorizzazione limitata nel tempo a tutti gli utenti

La seguente policy di esempio concede a tutti gli utenti (utenti anonimi) l'autorizzazione `ReceiveMessage` per la coda denominata `111122223333/queue1`, ma solo tra le ore 12 (mezzogiorno) e le ore 15 del 31 gennaio 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"
    }
  }
}]
}

```

Esempio 7: Concedere tutte le autorizzazioni a tutti gli utenti in un intervallo CIDR

La seguente policy di esempio concede a tutti gli utenti (anonimi) l'autorizzazione a utilizzare tutte le azioni Amazon SQS possibili che possono essere condivise per la coda denominata 111122223333/queue1, ma solo se la richiesta proviene dall'intervallo CIDR 192.0.2.0/24.

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"
      }
    }
  }]
}

```

Esempio 8: Autorizzazioni per l'elenco di elementi consentiti e l'elenco di elementi bloccati per utenti in diversi intervalli CIDR

La policy di esempio seguente dispone di due istruzioni:

- La prima dichiarazione concede a tutti gli utenti (utenti anonimi) nell'intervallo CIDR 192.0.2.0/24 (fatta eccezione per 192.0.2.188) l'autorizzazione per utilizzare l'azione SendMessage per la coda denominata 111122223333/queue1.
- La seconda dichiarazione blocca tutti gli utenti (utenti anonimi) nell'intervallo CIDR 12.148.72.0/23 in modo che non possano utilizzare la coda.

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"
      }
    }
  }
}
```

```
}]  
}
```

Utilizzo di policy personalizzate con la sintassi delle policy di accesso Amazon SQS

Per concedere autorizzazioni di base (come [SendMessage](#) o [ReceiveMessage](#)) basate solo su un Account AWS ID, non è necessario scrivere una politica personalizzata. Utilizza invece l'azione Amazon SQS. [AddPermission](#)

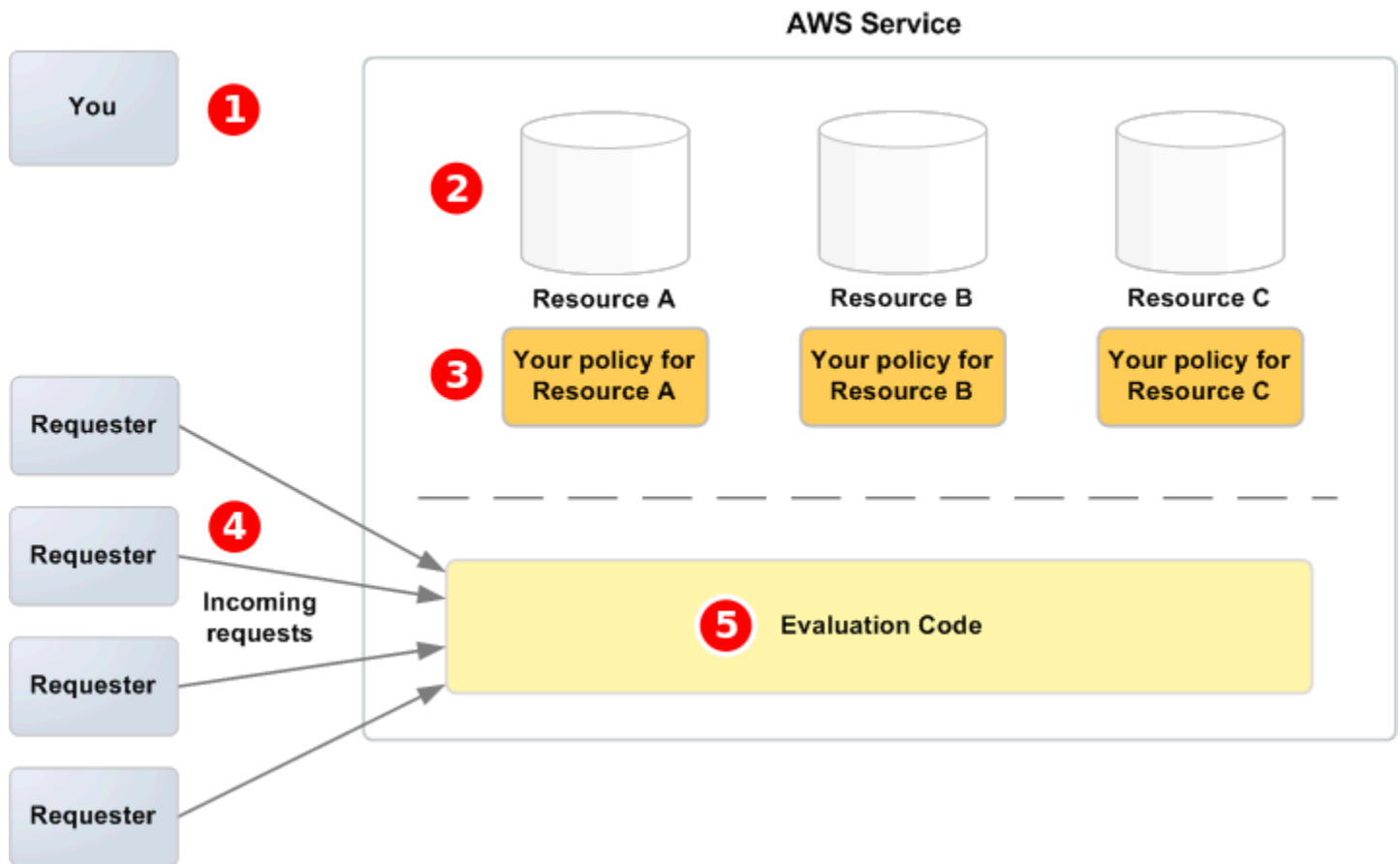
Per consentire o negare l'accesso in base a condizioni specifiche, come l'ora della richiesta o l'indirizzo IP del richiedente, devi creare una policy Amazon SQS personalizzata e caricarla utilizzando l'azione. [SetQueueAttributes](#)

Argomenti

- [Architettura di controllo degli accessi Amazon SQS](#)
- [Flusso di lavoro del processo di controllo degli accessi Amazon SQS](#)
- [Concetti chiave della sintassi delle policy di accesso di Amazon SQS](#)
- [Logica di valutazione della sintassi delle policy di accesso Amazon SQS](#)
- [Relazioni tra negazioni esplicite e predefinite nella sintassi delle policy di accesso di Amazon SQS](#)
- [Limitazioni delle politiche personalizzate di Amazon SQS](#)
- [Esempi di sintassi delle policy di accesso di Amazon SQS personalizzata](#)

Architettura di controllo degli accessi Amazon SQS

Il diagramma riportato di seguito descrive il controllo degli accessi per le risorse Amazon SQS.



1

Tu, il proprietario delle risorse.

2

Le tue risorse contenute nel AWS servizio (ad esempio, le code Amazon SQS).

3

Le tue policy. È buona norma avere una policy per ogni risorsa. Il AWS servizio fornisce un'API che utilizzi per caricare e gestire le tue politiche.

4

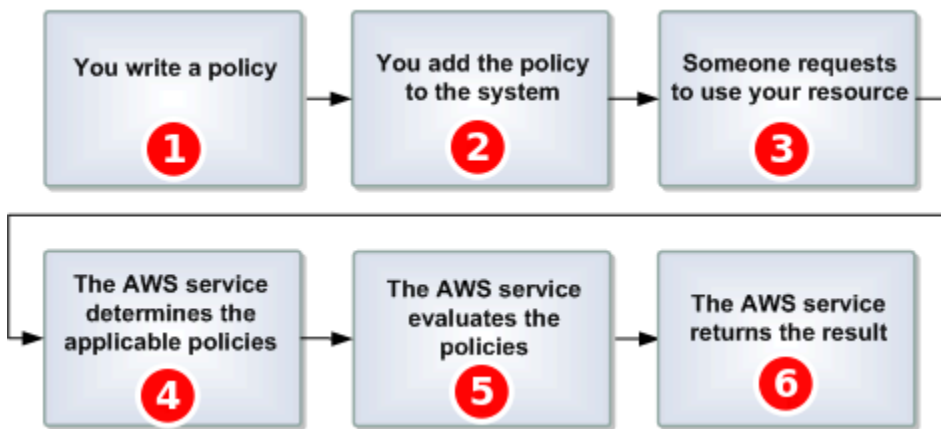
I richiedenti e le relative richieste in arrivo al servizio AWS .

5

Codice di valutazione della sintassi delle policy di accesso. Questo è il set di codice all'interno del AWS servizio che valuta le richieste in arrivo rispetto alle politiche applicabili e determina se al richiedente è consentito l'accesso alla risorsa.

Flusso di lavoro del processo di controllo degli accessi Amazon SQS

Nel seguente diagramma viene descritto il flusso di lavoro di controllo degli accessi generale con la sintassi delle policy di accesso Amazon SQS.



1

Scrivi una policy Amazon SQS per la tua coda.

2

Carichi la tua politica su AWS. Il servizio AWS fornisce un'API che utilizzi per caricare le tue politiche. Ad esempio, usa l'azione `SetQueueAttributes` di Amazon SQS per caricare una policy per una particolare coda Amazon SQS.

3

Qualcuno invia una richiesta per usare la tua coda Amazon SQS.

4

Amazon SQS esamina tutte le policy Amazon SQS disponibili e determina quali sono applicabili.

5

Amazon SQS valuta le policy e determina se il richiedente è autorizzato a utilizzare la tua coda o meno.

6

In base al risultato della valutazione della policy, Amazon SQS restituisce un errore `Access denied` al richiedente o continua a elaborare la richiesta.

Concetti chiave della sintassi delle policy di accesso di Amazon SQS

Per scrivere le tue proprie policy, devi avere familiarità con [JSON](#) e una serie di concetti fondamentali.

Abilita

Il risultato di una [Statement](#) che [Effetto](#) impostato su allow.

Azione

L'attività che [Principale](#) ha l'autorizzazione di eseguire, in genere una richiesta a AWS.

Default-deny

Il risultato di una [Statement](#) priva di impostazioni [Abilita](#) o [Explicit-deny](#).

Condition

Qualsiasi restrizione o dettaglio relativi a una [Autorizzazione](#). Condizioni tipiche relative a data e ora e indirizzi IP.

Effetto

Il risultato che desideri che la [Statement](#) di una [Policy](#) restituisca al momento della valutazione. Puoi specificare il valore deny o allow quando scrivi la dichiarazione della policy. Possono esserci tre risultati possibili al momento della valutazione della policy: [Default-deny](#), [Abilita](#) e [Explicit-deny](#).

Explicit-deny

Il risultato di una [Statement](#) che [Effetto](#) impostato su deny.

Valutazione

Il processo che Amazon SQS utilizza per determinare se una richiesta in entrata deve essere negata o consentita in base a una [Policy](#).

Approvatore

L'utente che scrive una [Policy](#) per concedere autorizzazioni a una risorsa. L'emittente, per definizione, è sempre il proprietario della risorsa. AWS non consente agli utenti di Amazon SQS di creare policy per risorse che non possiedono.

Chiave

La caratteristica specifica che costituisce la base per la limitazione dell'accesso.

Autorizzazione

La nozione di consentire o rifiutare l'accesso a una risorsa utilizzando una [Condition](#) e una [Chiave](#).

Policy

Il documento che agisce da container per una o più [istruzioni](#).



Amzon SQS utilizza la policy per determinare se concedere o meno a un utente l'accesso a una risorsa.

Principale

L'utente che riceve la [Autorizzazione](#) nella [Policy](#).

Risorsa

L'oggetto a cui le richiesta del [Principale](#) accedono.

Statement

La descrizione formale di una singola autorizzazione, scritta nella sintassi delle policy di accesso come parte di un documento di [Policy](#) più ampio.

Richiedente

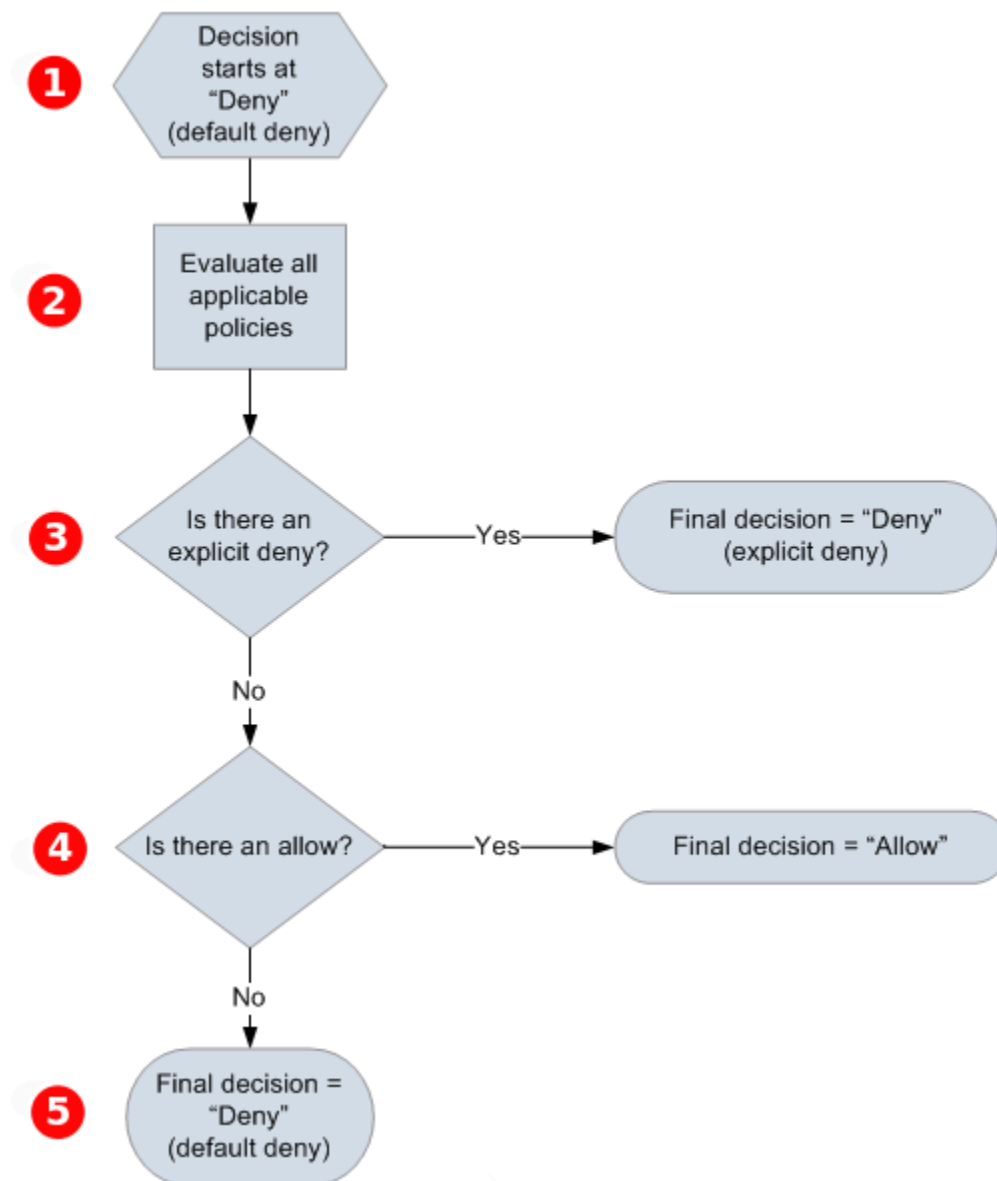
L'utente che invia una richiesta per l'accesso a una [Risorsa](#).

Logica di valutazione della sintassi delle policy di accesso Amazon SQS

Al momento della valutazione, Amazon SQS determina se le richieste da parte di utenti diversi dal proprietario della risorsa devono essere concesse o negate. La logica della valutazione segue diverse regole di base:

- Per impostazione predefinita, tutte le richieste per utilizzare la tua risorsa che provengono da chiunque eccetto te, vengono rifiutate.
- Un [Abilita](#) sostituisce qualsiasi [Default-deny](#).
- Un [Explicit-deny](#) sovrascrive qualsiasi allow.
- L'ordine in cui vengono valutate le policy non è importante.

Il diagramma seguente descrive in dettaglio come Amazon SQS valuta le decisioni circa le autorizzazioni di accesso.



1

La decisione inizia con un default-deny.

2

Il codice di applicazione valuta tutte le policy applicabili alla richiesta (in base alla risorsa, all'entità principale, all'operazione e alle condizioni). L'ordine in cui il codice di attuazione valuta le policy non è importante.

3

Il codice di attuazione cerca una dichiarazione explicit-deny che può essere applicata alla richiesta. Se ne trova uno, il codice di attuazione restituisce una decisione di rifiuto e il processo termina.

4

Se non viene trovato un explicit-deny, il codice di applicazione cerca un'istruzione allow da applicare alla richiesta. Se ne trova anche uno, il codice di attuazione restituisce una decisione di consenso e il processo termina (il servizio continua ad elaborare la richiesta).

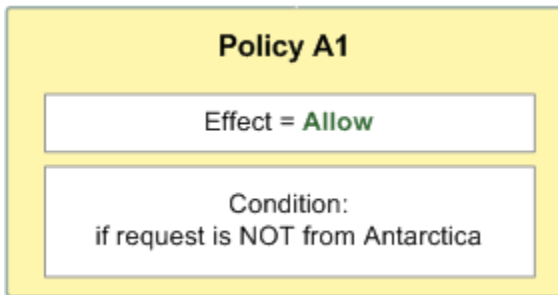
5

Se non viene trovato alcun allow, la decisione finale è il rifiuto (dal momento che non è stato trovato un explicit-deny o un allow, questo viene considerato un default-deny).

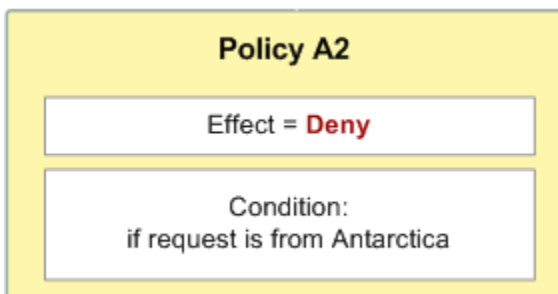
Relazioni tra negazioni esplicite e predefinite nella sintassi delle policy di accesso di Amazon SQS

Se una policy Amazon SQS non si applica direttamente a una richiesta, questa comporta un [Default-deny](#). Ad esempio, se un utente richiede l'autorizzazione per utilizzare Amazon SQS, ma l'unica policy che si applica all'utente può utilizzare DynamoDB, la richiesta comporta un default-deny.

Se una condizione in una dichiarazione non viene soddisfatta, la richiesta comporta un default-deny. Se tutte le condizioni nella dichiarazione sono soddisfatte, la richiesta restituisce un [Abilita](#) o un [Explicit-deny](#) a seconda del valore dell'elemento [Effetto](#) della policy. Le policy non specificano cosa fare se una condizione non viene soddisfatta e quindi il risultato predefinito in quel caso è un default-deny. Ad esempio, supponiamo che tu voglia impedire le richieste provenienti dall'Antartide. Scrivi una Policy A1 che consente una richiesta solo se non proviene dall'Antartide. Il diagramma seguente illustra la policy Amazon SQS.

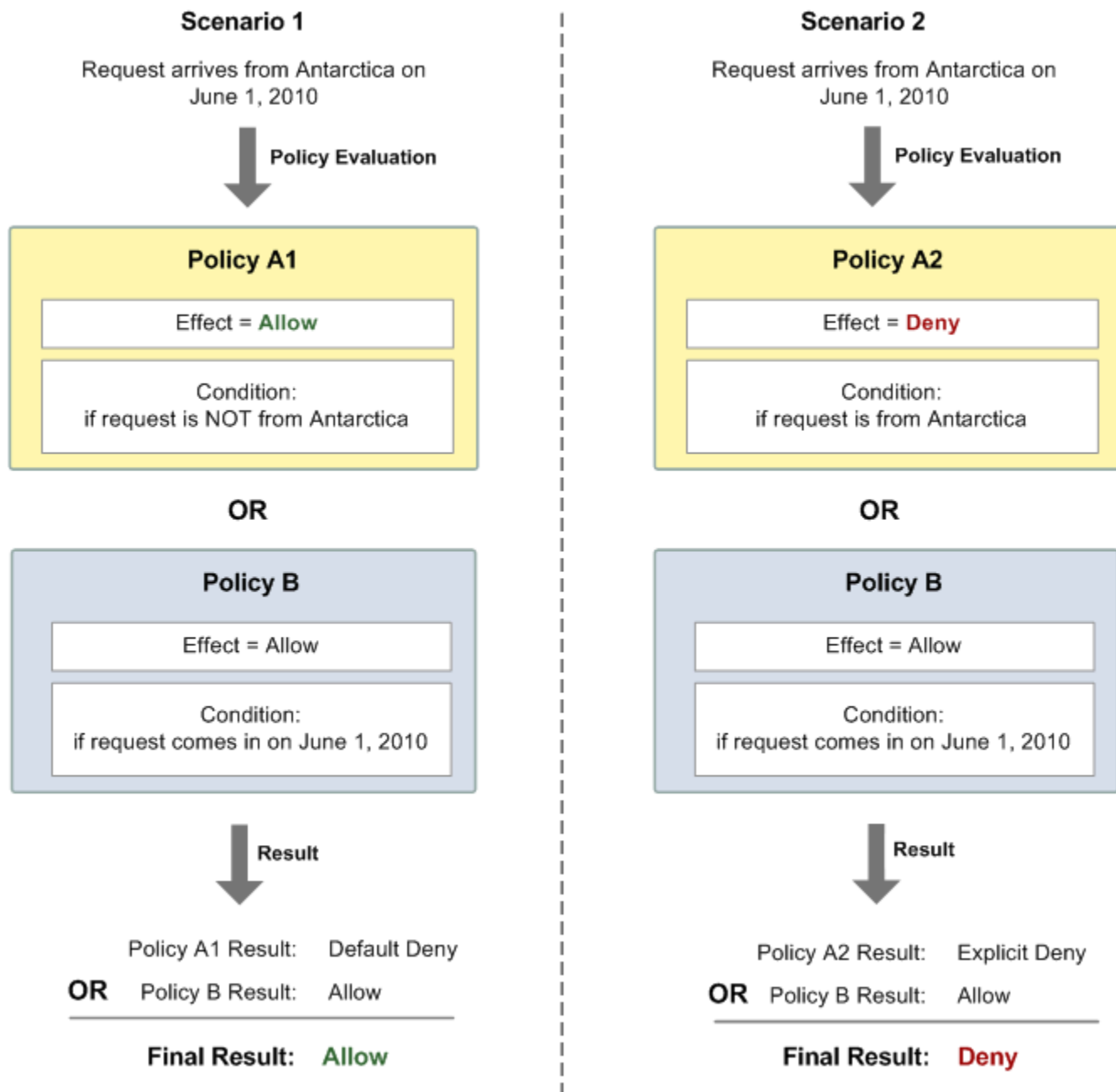


Se un utente invia una richiesta dagli Stati Uniti, la condizione è soddisfatta (la richiesta non proviene dall'Antartide) e la richiesta comporta un allow. Tuttavia, se un utente invia una richiesta dall'Antartide, la condizione non viene soddisfatta e la richiesta comporta, per impostazione predefinita, un default-deny. Puoi modificare il risultato a un explicit-deny scrivendo una Policy A2 che nega esplicitamente una richiesta se proviene da Antartide. Il diagramma seguente illustra la policy.



Se un utente invia una richiesta dall'Antartide, la condizione viene soddisfatta e la richiesta comporta un explicit-deny.

La differenza tra un default-deny e un explicit-deny è importante perché un allow può sovrascrivere il primo ma non l'ultimo. Ad esempio, Policy B consente le richieste se arrivano il 1° giugno 2010. Il seguente diagramma confronta la combinazione di questa policy con Policy A1 e Policy A2.



Nello Scenario 1, Policy A1 comporta un default-deny e Policy B comporta un allow perché la policy consente le richieste pervenute il 1° giugno 2010. Il allow dalla policy B sovrascrive il default-deny dalla policy A1 e pertanto la richiesta è consentita.

Nello Scenario 2, Policy B2 comporta un explicit-deny e Policy B comporta un allow. L'explicit-deny dalla policy A2 sovrascrive il allow dalla policy B e pertanto la richiesta viene rifiutata.

Limitazioni delle politiche personalizzate di Amazon SQS

Accesso multi-account

Le autorizzazioni per più account non sono applicabili alle azioni seguenti:

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

Chiavi di condizione

Al momento Amazon SQS supporta solo un sottoinsieme limitato delle [chiavi di condizione disponibili in IAM](#). Per ulteriori informazioni, consulta [Autorizzazioni API Amazon SQS: riferimento a operazioni e risorse](#).

Esempi di sintassi delle policy di accesso di Amazon SQS personalizzata

I seguenti sono esempi tipici di policy di accesso Amazon SQS.

Esempio 1: Concedere l'autorizzazione a un account

La policy Amazon SQS di esempio seguente fornisce all'account Account AWS l'autorizzazione 111122223333 per inviare e ricevere dalla queue2 di proprietà dell' Account AWS 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"
}]
}
```

Esempio 2: Concedere l'autorizzazione a uno o più account

Il seguente esempio di policy Amazon SQS fornisce uno o più Account AWS accessi alle code di proprietà del tuo account per un periodo di tempo specifico. È necessario scrivere questa policy e caricarla in Amazon SQS utilizzando l'operazione [SetQueueAttributes](#) perché l'operazione [AddPermission](#) non consente di specificare una limitazione di tempo quando si concede l'accesso a una coda.

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"
        }
    }
}
}]
}

```

Esempio 3: Concedere autorizzazioni a richieste provenienti da istanze Amazon EC2

La policy di esempio Amazon SQS seguente fornisce l'accesso a richieste provenienti da istanze Amazon EC2. Questo esempio si basa sull'esempio "[Esempio 2: Concedere l'autorizzazione a uno o più account](#)": limita l'accesso a prima del 30 giugno 2009 alle 12.00 (UTC), limita l'accesso all'intervallo IP 203.0.113.0/24. È necessario scrivere questa policy e caricarla su Amazon SQS utilizzando l'azione [SetQueueAttributes](#) perché l'azione [AddPermission](#) non consente di specificare una limitazione di indirizzo IP al momento di concedere l'accesso a una coda.

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"
      }
    }
  }
}

```

Esempio 4: Negare l'accesso a un account specifico

Il seguente esempio di policy di Amazon SQS nega un Account AWS accesso specifico alla tua coda. Questo esempio si basa sull'esempio "[Esempio 1: Concedere l'autorizzazione a un account](#)": nega l'accesso a ciò che è specificato. Account AWS È necessario scrivere questa policy e caricarla in Amazon SQS utilizzando l'operazione [SetQueueAttributes](#) perché l'operazione [AddPermission](#) non consente di negare l'accesso a una coda (concede solo l'accesso a una coda).

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"
    }
  ]
}

```

Esempio 5: Negare l'accesso se non è un endpoint VPC

La seguente policy Amazon SQS di esempio limita l'accesso alla queue1: 111122223333 può eseguire le azioni [SendMessage](#) e [ReceiveMessage](#) solo dall'ID dell'endpoint VPC vpce-1a2b3c4d (specificato usando la condizione `aws:sourceVpce`). Per ulteriori informazioni, consulta [Endpoint di Amazon Virtual Private Cloud per Amazon SQS](#).

Note

- La condizione `aws:sourceVpce` non richiede un ARN per la risorsa dell'endpoint VPC, ma solo l'ID dell'endpoint VPC.
- Puoi modificare l'esempio che segue per limitare tutte le operazioni per un determinato endpoint VPC negando tutte le operazioni Amazon SQS (`sqs:*`) nella seconda istruzione. Tuttavia, questa istruzione della policy stabilisce che tutte le operazioni (comprese le operazioni amministrative necessarie per modificare le autorizzazioni della coda) devono essere eseguite tramite l'endpoint VPC specifico definito nella policy, impedendo potenzialmente all'utente di modificare successivamente le autorizzazioni della coda.

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"
        }
    }
}
```

Utilizzo delle credenziali di sicurezza temporanee con Amazon SQS

Oltre a creare utenti con le proprie credenziali di sicurezza, IAM ti consente anche di concedere credenziali di sicurezza temporanee a qualsiasi utente, consentendogli di accedere ai tuoi servizi e risorse. AWS Puoi gestire utenti che hanno Account AWS. Puoi anche gestire gli utenti del tuo sistema che non ne hanno Account AWS (utenti federati). Inoltre, anche le applicazioni create per accedere alle AWS risorse possono essere considerate «utenti».

Queste credenziali di sicurezza temporanee sono utilizzate per effettuare delle richieste ad Amazon SQS. Le librerie API calcolano il valore di firma necessario utilizzando tali credenziali per autenticare la richiesta dell'utente. In caso di invio di richieste tramite l'utilizzo di credenziali scadute, Amazon SQS rifiuta la richiesta.

Note

Non puoi impostare una policy in base alle credenziali temporanee.

Prerequisiti

1. Usare IAM per creare credenziali di sicurezza provvisorie:

- Token di sicurezza

- ID chiave di accesso
 - Secret Access Key
2. Prepara la tua stringa a effettuare l'accesso con l'ID chiave di accesso temporanea e il token di sicurezza.
 3. Utilizza la Secret Access Key temporanea anziché la tua Secret Access Key per firmare la tua richiesta di API query.

Note

Quando invii la richiesta di API query firmata, utilizza l'ID chiave di accesso temporaneo anziché l'ID chiave di accesso e includi i token di sicurezza. Per ulteriori informazioni sul supporto IAM per le credenziali di sicurezza temporanee, consulta [Granting Temporary Access to Your AWS Resources](#) nella IAM User Guide.

Chiamare un'azione API query Amazon SQS utilizzando le credenziali di sicurezza temporanee

1. Richiedi un token di sicurezza temporaneo utilizzando AWS Identity and Access Management. Per ulteriori informazioni, consulta [Creazione di credenziali di sicurezza temporanee per abilitare l'accesso per utenti IAM](#) nella Guida per l'utente IAM.

IAM restituisce un token di sicurezza, un ID chiave di accesso e una chiave di accesso segreta.

2. Prepara la tua query utilizzando l'ID chiave di accesso temporaneo anziché l'ID chiave di accesso e includi i token di sicurezza. Firma la tua richiesta utilizzando la Secret Access Key temporaneo anziché quella effettiva.
3. Invia la stringa della query firmata con l'ID chiave di accesso temporanea e il token di sicurezza.

Nell'esempio seguente viene mostrato come utilizzare credenziali di sicurezza temporanee per autenticare una richiesta Amazon SQS. La struttura di **AUTHPARAMS** dipende dalla modalità di firma della richiesta API. Per ulteriori informazioni, consulta [Signing AWS API Requests](#) nel Amazon Web Services General Reference.

```
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
```

L'esempio seguente utilizza credenziali di sicurezza temporanee per inviare due messaggi con l'operazione SendMessageBatch.

```
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
```

Gestione degli accessi per code crittografate di Amazon SQS con politiche di privilegi minimi

Puoi utilizzare Amazon SQS per lo scambio di dati sensibili tra applicazioni utilizzando la crittografia lato server (SSE) integrata con [AWS Key Management Service \(KMS\)](#). Con l'integrazione di Amazon SQS e AWS KMS, puoi gestire centralmente le chiavi che proteggono Amazon SQS, nonché le chiavi che proteggono le altre risorse. AWS

Più AWS servizi possono fungere da sorgenti di eventi che inviano eventi ad Amazon SQS. [Per consentire a una fonte di eventi di accedere alla coda crittografata di Amazon SQS, devi configurare la coda con una chiave gestita dal cliente.](#) AWS KMS Quindi, utilizza la policy chiave per consentire al servizio di utilizzare i metodi API richiesti. AWS KMS Il servizio richiede inoltre le autorizzazioni per autenticare l'accesso per consentire alla coda di inviare eventi. Puoi raggiungere questo obiettivo utilizzando una policy di Amazon SQS, che è una policy basata sulle risorse che puoi utilizzare per controllare l'accesso alla coda Amazon SQS e ai relativi dati.

Le seguenti sezioni forniscono informazioni su come controllare l'accesso alla coda crittografata di Amazon SQS tramite la policy di Amazon SQS e la policy chiave. AWS KMS Le policy di questa guida ti aiuteranno a ottenere il [privilegio minimo](#).

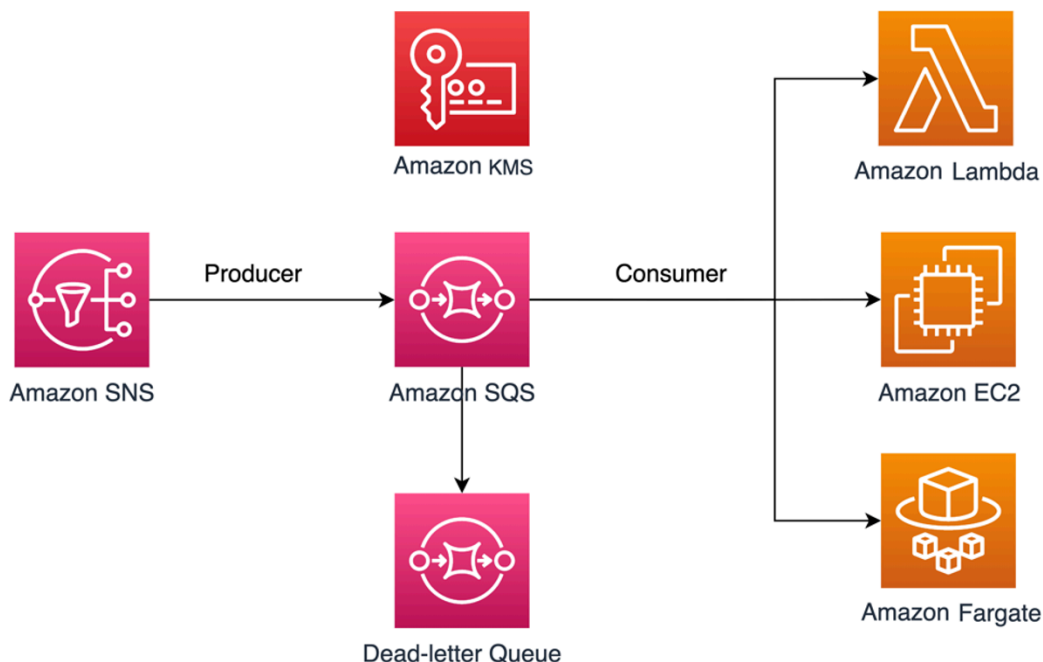
Questa guida descrive anche come le policy basate sulle risorse risolvono il [problema confused-deputy](#) utilizzando le chiavi di contesto delle condizioni IAM globali [aws:SourceArn](#), [aws:SourceAccount](#) e [aws:PrincipalOrgID](#).

Argomenti

- [Panoramica di](#)
- [Policy della chiave con privilegio minimo per Amazon SQS](#)
- [Istruzioni di policy di Amazon SQS per la coda DLQ](#)
- [Prevenzione del problema "confused deputy" tra servizi](#)
- [Utilizza IAM Access Analyzer per esaminare l'accesso multi-account](#)

Panoramica di

In questo argomento, descriveremo un caso d'uso comune per illustrare come creare la policy della chiave e la policy di coda di Amazon SQS. Questo caso d'uso viene mostrato nell'immagine seguente.



In questo esempio, il produttore del messaggio è un argomento di [Amazon Simple Notification Service \(SNS\)](#), configurato per separare i messaggi dalla coda crittografata di Amazon SQS. Il consumatore di messaggi è un servizio di elaborazione, come una funzione [AWS Lambda](#), un'istanza [Amazon Elastic Compute Cloud \(EC2\)](#) o un container [AWS Fargate](#). La coda Amazon SQS viene quindi configurata per inviare messaggi non riusciti a una coda [DLQ](#). Ciò è utile per eseguire il debug dell'applicazione o del sistema di messaggistica perché DLQs consente di isolare i messaggi non utilizzati per determinare il motivo per cui la loro elaborazione non è riuscita. Nella soluzione definita in questo argomento, un servizio di elaborazione come una funzione Lambda viene utilizzato per elaborare i messaggi archiviati nella coda Amazon SQS. Se l'utente dei messaggi si trova in un cloud privato virtuale (VPC), l'istruzione della policy [DenyReceivingIfNotThroughVPCE](#) inclusa in questa guida consente di limitare la ricezione dei messaggi a quel VPC specifico.

Note

Questa guida contiene solo le autorizzazioni IAM richieste sotto forma di istruzioni di policy. Per creare la policy, devi aggiungere le dichiarazioni alla tua policy Amazon SQS o alla AWS KMS tua policy chiave. Questa guida non fornisce istruzioni su come creare la coda o la chiave Amazon SQS. AWS KMS Per istruzioni su come creare queste risorse, consulta [Creazione di una coda Amazon SQS](#) e [Creazione di chiavi](#). La policy di Amazon SQS definita in questa guida non supporta il reindirizzamento dei messaggi direttamente nella stessa coda Amazon SQS o in un'altra.

Policy della chiave con privilegio minimo per Amazon SQS

In questa sezione, descriviamo le autorizzazioni con privilegi minimi richieste AWS KMS per la chiave gestita dal cliente che utilizzi per crittografare la tua coda Amazon SQS. Con queste autorizzazioni, puoi limitare l'accesso solo alle entità previste implementando il privilegio minimo. La policy della chiave deve contenere le seguenti istruzioni di policy, che descriveremo in dettaglio di seguito:

- [AWS KMS Concedi le autorizzazioni di amministratore alla chiave](#)
- [Concedi l'accesso in sola lettura ai metadati della chiave](#)
- [Concedi ad Amazon SNS le autorizzazioni KMS per pubblicare i messaggi nella coda](#)
- [Consenti ai consumatori di decrittografare i messaggi dalla coda](#)

AWS KMS Concedi le autorizzazioni di amministratore alla chiave

Per creare una AWS KMS chiave, devi fornire le autorizzazioni di AWS KMS amministratore per il ruolo IAM che utilizzi per distribuire la chiave. AWS KMS Queste autorizzazioni di amministratore sono definite nella seguente istruzione di policy AllowKeyAdminPermissions. Quando aggiungi questa dichiarazione alla tua policy AWS KMS chiave, assicurati di sostituirla *<admin-role ARN>* con l'Amazon Resource Name (ARN) del ruolo IAM utilizzato per distribuire la AWS KMS chiave, gestire la AWS KMS chiave o entrambi. Questo può essere il ruolo IAM della tua pipeline di implementazione o il [ruolo di amministratore per la tua organizzazione](#) nelle [AWS Organizations](#).

```
{
  "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 una policy AWS KMS chiave, il valore dell'`Resource` elemento deve essere*, il che significa «questa AWS KMS chiave». L'asterisco (*) identifica la AWS KMS chiave a cui è allegata la politica chiave.

Concedi l'accesso in sola lettura ai metadati della chiave

Per concedere ad altri ruoli IAM l'accesso in sola lettura ai metadati della chiave, aggiungi l'istruzione `AllowReadAccessToKeyMetaData` alla policy della tua chiave. Ad esempio, la seguente dichiarazione consente di elencare tutte le AWS KMS chiavi del proprio account a fini di controllo. Questa istruzione concede all'utente AWS root l'accesso in sola lettura ai metadati chiave. Pertanto, qualsiasi principale IAM dell'account può avere accesso ai metadati della chiave se le relative policy basate sull'identità includono autorizzazioni elencate nella seguente istruzione: `kms:Describe*`, `kms:Get*` e `kms:List*`. Assicurati di sostituirla `<account-ID>` con le tue informazioni.

```
{
  "Sid": "AllowReadAccesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}
```

Concedi ad Amazon SNS le autorizzazioni KMS per pubblicare i messaggi nella coda

Per consentire al tuo argomento Amazon SNS di pubblicare messaggi nella coda crittografata di Amazon SQS, aggiungi l'informativa sulla policy `AllowSNSToSendToSQS` alla policy della chiave. Questa dichiarazione concede ad Amazon SNS le autorizzazioni per utilizzare AWS KMS la chiave per la pubblicazione nella tua coda Amazon SQS. Assicurati di sostituirla con le tue informazioni `<account-ID>`.

 Note

ConditionNella dichiarazione limita l'accesso solo al servizio Amazon SNS nello stesso AWS account.

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
}
```

Consenti ai consumatori di decrittografare i messaggi dalla coda

La seguente istruzione AllowConsumersToReceiveFromTheQueue concede all'utente di messaggi Amazon SQS le autorizzazioni necessarie per decrittografare i messaggi ricevuti dalla coda crittografata di Amazon SQS. Quando alleggi la policy statement, sostituiscila *<consumer's runtime role ARN>* con l'ARN del ruolo di runtime IAM del consumer del messaggio.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
}
```

```
"Action": [  
    "kms:Decrypt"  
],  
"Resource": "*" ]}
```

Policy relativa al privilegio minimo per Amazon SQS

Questa sezione illustra le politiche di coda di Amazon SQS con privilegio minimo per il caso d'uso trattato in questa guida (ad esempio, da Amazon SNS ad Amazon SQS). La policy definita è progettata per prevenire accessi involontari utilizzando una combinazione delle istruzioni Deny e Allow. Le istruzioni Allow concedono l'accesso all'entità o alle entità previste. Le istruzioni Deny impediscono ad altre entità indesiderate di accedere alla coda di Amazon SQS, escludendo al contempo l'entità prevista dalla condizione della politica.

La policy di Amazon SQS include le seguenti istruzioni, che descriviamo in dettaglio di seguito:

- [Limita le autorizzazioni di gestione di Amazon SQS](#)
- [Limita le azioni di coda di Amazon SQS dall'organizzazione specificata](#)
- [Concedi le autorizzazioni Amazon SQS ai consumatori](#)
- [Applica la crittografia in transito](#)
- [Limita la trasmissione di messaggi a un argomento specifico di Amazon SNS](#)
- [\(Opzionale\) Limitazione della ricezione dei messaggi a un endpoint VPC specifico](#)

Limita le autorizzazioni di gestione di Amazon SQS

La seguente istruzione della policy `RestrictAdminQueueActions` limita le autorizzazioni di gestione di Amazon SQS solo al ruolo o ai ruoli IAM utilizzati per implementare la coda, gestire la coda o entrambe le cose. Assicurarsi di sostituire *<placeholder values>* con le proprie informazioni. Specificare l'ARN del ruolo IAM utilizzato per distribuire la coda Amazon SQS, nonché il ruolo di amministratore che dovrebbe avere le autorizzazioni ARNs di gestione di Amazon SQS.

```
{  
    "Sid": "RestrictAdminQueueActions",  
    "Effect": "Deny",  
    "Principal": {  
        "AWS": "*" ]  
    },  
    "Action": [  
        "kms:Decrypt"  
    ],  
    "Resource": "*" ]}
```

```

    "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>"
      ]
    }
  }
}

```

Limita le azioni di coda di Amazon SQS dall'organizzazione specificata

Per proteggere le tue risorse Amazon SQS dall'accesso esterno (accesso da parte di un'entità esterna all'[organizzazione AWS](#)), utilizza la seguente istruzione. Questa istruzione limita l'accesso alla coda di Amazon SQS all'organizzazione specificata in `Condition`. Assicurati di sostituirlo `<SQS queue ARN>` con l'ARN del ruolo IAM utilizzato per distribuire la coda Amazon SQS e poi con l'ID della `<org-id>` tua organizzazione.

```

{
  "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>"
      ]
    }
  }
}

```

```
    ]
  }
}
}
```

Concedi le autorizzazioni Amazon SQS ai consumatori

Per ricevere messaggi dalla coda Amazon SQS, devi fornire al consumatore dei messaggi le autorizzazioni necessarie. La seguente istruzione di policy concede al consumatore, da te specificato, le autorizzazioni necessarie per utilizzare i messaggi dalla coda di Amazon SQS. Quando aggiungi l'istruzione alla tua policy Amazon SQS, assicurati di sostituirla *<consumer's IAM runtime role ARN>* con l'ARN del ruolo di runtime IAM utilizzato dal consumatore e *<SQS queue ARN>* con l'ARN del ruolo IAM utilizzato per distribuire la coda Amazon SQS.

```
{
  "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>"
}
```

Per impedire ad altre entità di ricevere messaggi dalla coda di Amazon SQS, aggiungi l'istruzione `DenyOtherConsumersFromReceiving` alla policy della coda di Amazon SQS. Questa istruzione limita il consumo di messaggi al consumatore specificato, senza consentire ad altri consumatori di accedervi, anche se le autorizzazioni di identità consentirebbero loro l'accesso. Assicurati di sostituire e con le tue informazioni *<SQS queue ARN>*. *<consumer's runtime role ARN>*

```
{
  "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>"
    }
  }
}

```

Applica la crittografia in transito

La seguente istruzione di policy `DenyUnsecureTransport` impone a consumatori e produttori di utilizzare canali sicuri (connessioni TLS) per inviare e ricevere messaggi dalla coda di Amazon SQS. Assicurati di sostituirlo `<SQS queue ARN>` con l'ARN del ruolo IAM utilizzato per distribuire la coda Amazon SQS.

```

{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}

```

Limita la trasmissione di messaggi a un argomento specifico di Amazon SNS

Il seguente è un esempio di istruzione di policy AllowSNSToSendToTheQueue che consente all'argomento Amazon SNS di inviare messaggi alla coda Amazon SQS. Assicurati di sostituirlo *<SQS queue ARN>* con l'ARN del ruolo IAM utilizzato per distribuire la coda Amazon SQS e con *<SNS topic ARN>* l'argomento ARN di Amazon SNS.

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

La seguente istruzione di policy DenyAllProducersExceptSNSFromSending impedisce ad altri produttori di inviare messaggi alla coda. Sostituisci *<SQS queue ARN>* e con le tue informazioni. *<SNS topic ARN>*

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```


(Opzionale) Limitazione della ricezione dei messaggi a un endpoint VPC specifico

Per limitare la ricezione di messaggi solo a un [endpoint VPC](#) specifico, aggiungi la seguente istruzione di policy alla tua policy di coda di Amazon SQS. Questa istruzione impedisce a un utente che consuma messaggi di ricevere messaggi dalla coda a meno che i messaggi non provengano dall'endpoint VPC desiderato. Sostituisci *<SQS queue ARN>* con l'ARN del ruolo IAM utilizzato per distribuire la coda Amazon SQS e con *<vpce_id>* l'ID dell'endpoint VPC.

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

Istruzioni di policy di Amazon SQS per la coda DLQ

Aggiungi le seguenti istruzioni di policy, identificate dal relativo ID, alla tua policy di accesso DLQ:

- RestrictAdminQueueActions
- DenyQueueActionsOutsideOrg
- AllowConsumersToReceiveFromTheQueue
- DenyOtherConsumersFromReceiving
- DenyUnsecureTransport

Oltre ad aggiungere le precedenti istruzioni di policy alla tua policy di accesso DLQ, devi aggiungere anche un'istruzione per limitare la trasmissione di messaggi alle code di Amazon SQS, come descritto nella sezione seguente.

Limitazione della trasmissione di messaggi a code Amazon SQS

Per limitare l'accesso solo alle code Amazon SQS dallo stesso account, aggiungi la seguente istruzione di policy `DenyAnyProducersExceptSQS` alla policy di coda DLQ. Questa istruzione non limita la trasmissione dei messaggi a una coda specifica perché prima di creare la coda principale devi implementare il DLQ, quindi non conoscerai l'ARN di Amazon SQS quando crei la DLQ. Se devi limitare l'accesso a una sola coda Amazon SQS, modifica la `aws:SourceArn` in `Condition` con l'ARN della tua coda di origine Amazon SQS, quando lo conosci.

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

Important

Le policy di coda di Amazon SQS definite in questa guida non limitano l'operazione `sqs:PurgeQueue` a uno o più ruoli IAM specifici. L'azione `sqs:PurgeQueue` consente di eliminare tutti i messaggi nella coda Amazon SQS. Puoi anche utilizzare questa azione per apportare modifiche al formato del messaggio senza sostituire la coda Amazon SQS. Durante il debug di un'applicazione, puoi cancellare la coda di Amazon SQS per rimuovere messaggi potenzialmente errati. Durante il test dell'applicazione, puoi indirizzare un volume di messaggi elevato attraverso la coda di Amazon SQS e quindi eliminare la coda per ricominciare da capo prima di entrare in produzione. Il motivo per cui non si limita questa azione a un determinato ruolo è che questo ruolo potrebbe non essere noto durante l'implementazione della coda Amazon SQS. Dovrai aggiungere questa autorizzazione alla policy basata sull'identità del ruolo per poter eliminare la coda.

Prevenzione del problema "confused deputy" tra servizi

Il [confused deputy](#) è un problema di sicurezza in cui un'entità che non dispone dell'autorizzazione per eseguire un'azione può costringere un'entità dotata di privilegi maggiori a eseguire l'azione. Per evitare che ciò accada, AWS fornisce strumenti che aiutano a proteggere il proprio account se si fornisce a terzi (cosiddetti cross-account) o ad altri AWS servizi (noti come cross-service) l'accesso alle risorse del proprio account. Le istruzioni di policy in questa sezione possono aiutarti a evitare il problema del "confused deputy" tra servizi.

La rappresentazione tra servizi può verificarsi quando un servizio (il servizio chiamante) effettua una chiamata a un altro servizio (il servizio chiamato). Il servizio chiamante può essere manipolato per utilizzare le proprie autorizzazioni e agire sulle risorse di un altro cliente, a cui normalmente non avrebbe accesso. Per contribuire alla protezione da questo problema, le policy basate sulle risorse definite in questo post utilizzano le chiavi di contesto delle condizioni IAM globali [aws:SourceArn](#), [aws:SourceAccount](#) e [aws:PrincipalOrgID](#). Ciò limita le autorizzazioni di cui dispone un servizio per una risorsa specifica, un account specifico o un'organizzazione specifica in AWS Organizations.

Utilizza IAM Access Analyzer per esaminare l'accesso multi-account

Puoi utilizzare [AWS IAM Access Analyzer](#) per esaminare le policy di coda AWS KMS e le policy chiave di Amazon SQS e avvisarti quando una coda o AWS KMS una chiave Amazon SQS concede l'accesso a un'entità esterna. IAM Access Analyzer consente di identificare le [risorse](#) nell'organizzazione e negli account che sono condivisi con un'entità esterna. Questa zona di fiducia può essere un AWS account o l'organizzazione all'interno di AWS Organizations che specifichi quando abiliti IAM Access Analyzer.

IAM Access Analyzer identifica le risorse condivise con responsabili esterni utilizzando un ragionamento basato sulla logica per analizzare le politiche basate sulle risorse nel tuo ambiente. AWS Per ogni istanza di una risorsa condivisa al di fuori della zona di fiducia, Access Analyzer genera una ricerca. I [risultati](#) comprendono informazioni sull'accesso e sull'entità esterna a cui è concesso. È possibile rivedere i risultati per determinare se l'accesso è intenzionale e sicuro o se invece è involontario e rappresenta un rischio per la sicurezza. Per eventuali accessi non intenzionali, rivedi la policy interessata e correggila. Fai riferimento a questo [post del blog](#) per ulteriori informazioni su come AWS IAM Access Analyzer identifica l'accesso non intenzionale alle tue risorse. AWS

Per ulteriori informazioni su AWS IAM Access Analyzer, consulta la documentazione di [AWS IAM Access Analyzer](#).

Autorizzazioni API Amazon SQS: riferimento a operazioni e risorse

Quando configuri [Controllo accessi](#) e scrivi policy di autorizzazione che è possibile collegare a un'identità IAM, utilizza la tabella seguente come riferimento. L' include ogni azione di Amazon Simple Queue Service, le azioni corrispondenti per le quali è possibile concedere le autorizzazioni per eseguire l'azione e la AWS risorsa per la quale è possibile concedere le autorizzazioni.

Puoi specificare le operazioni nel campo `Action` della policy e il valore della risorsa nel campo `Resource`. Per specificare un'operazione, utilizzare il prefisso `sqs :` seguito dal nome dell'azione (ad esempio `sqs:CreateQueue`).

Attualmente, Amazon SQS supporta le [chiavi di contesto delle condizioni globali disponibili in IAM](#).

API di Amazon Simple Queue Service e autorizzazioni richieste per le azioni

[AddPermission](#)

Operazioni: `sqs:AddPermission`

Risorsa: `arn:aws:sqs:region:account_id:queue_name`

[ChangeMessageVisibility](#)

Operazioni: `sqs:ChangeMessageVisibility`

Risorsa: `arn:aws:sqs:region:account_id:queue_name`

[ChangeMessageVisibilityBatch](#)

Operazioni: `sqs:ChangeMessageVisibilityBatch`

Risorsa: `arn:aws:sqs:region:account_id:queue_name`

[CreateQueue](#)

Operazioni: `sqs:CreateQueue`

Risorsa: `arn:aws:sqs:region:account_id:queue_name`

[DeleteMessage](#)

Operazioni: `sqs>DeleteMessage`

Risorsa: `arn:aws:sqs:region:account_id:queue_name`

[DeleteMessageBatch](#)

Operazioni: `sqs>DeleteMessageBatch`

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[DeleteQueue](#)

Operazioni: sqs:DeleteQueue

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[GetQueueAttributes](#)

Operazioni: sqs:GetQueueAttributes

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[GetQueueUrl](#)

Operazioni: sqs:GetQueueUrl

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[ListDeadLetterSourceQueues](#)

Operazioni: sqs:ListDeadLetterSourceQueues

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[ListQueues](#)

Operazioni: sqs:ListQueues

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[ListQueueTags](#)

Operazioni: sqs:ListQueueTags

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[PurgeQueue](#)

Operazioni: sqs:PurgeQueue

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[ReceiveMessage](#)

Operazioni: sqs:ReceiveMessage

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[RemovePermission](#)

Operazioni: sqs:RemovePermission

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[SendMessage](#) e [SendMessageBatch](#)

Operazioni: sqs:SendMessage

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[SetQueueAttributes](#)

Operazioni: sqs:SetQueueAttributes

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[TagQueue](#)

Operazioni: sqs:TagQueue

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

[UntagQueue](#)

Operazioni: sqs:UntagQueue

Risorsa: arn:aws:sqs:*region*:*account_id*:*queue_name*

Registrazione e monitoraggio in Amazon SQS

Amazon Simple Queue Service è integrato con [AWS CloudTrail](#), un servizio che fornisce un registro delle azioni intraprese da un utente, ruolo o un Servizio AWS. CloudTrail acquisisce tutte le chiamate API per Amazon SQS come eventi. Le chiamate acquisite includono chiamate dalla console Amazon SQS e chiamate di codice alle operazioni dell'API Amazon SQS. Utilizzando le informazioni raccolte da CloudTrail, puoi determinare la richiesta effettuata ad Amazon SQS, l'indirizzo IP da cui è stata effettuata la richiesta, quando è stata effettuata e dettagli aggiuntivi.

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con le credenziali utente root o utente.
- Se la richiesta è stata effettuata per conto di un utente del Centro identità IAM.

- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro Servizio AWS.

CloudTrail è attivo nel tuo account Account AWS quando crei l'account e hai automaticamente accesso alla cronologia degli CloudTrail eventi. La cronologia CloudTrail degli eventi fornisce un record visualizzabile, ricercabile, scaricabile e immutabile degli ultimi 90 giorni di eventi di gestione registrati in un. Regione AWS Per ulteriori informazioni, consulta [Lavorare con la cronologia degli CloudTrail eventi](#) nella Guida per l'utente.AWS CloudTrail Non sono CloudTrail previsti costi per la visualizzazione della cronologia degli eventi.

Per una registrazione continua degli eventi degli Account AWS ultimi 90 giorni, crea un trail o un data store di eventi [CloudTrailLake](#).

CloudWatch Allarmi Amazon

Monitora una singola metrica in un periodo di tempo specificato e intraprendi una o più azioni in base al valore della metrica rispetto a una soglia definita su più periodi. Ad esempio, puoi configurare un CloudWatch allarme per inviare una notifica a un argomento Amazon SNS o attivare un'azione per inviare un messaggio a una coda Amazon SQS. CloudWatch gli allarmi non eseguono azioni semplicemente perché si trovano in uno stato specifico; lo stato deve cambiare e rimanere in quello stato per un numero definito di periodi.

Per ulteriori informazioni, consultare [Creazione di CloudWatch allarmi per i parametri di Amazon SQS](#) e [Creazione di allarmi per code di lettere scadenti con Amazon CloudWatch](#).

CloudWatch Registri Amazon

Monitora, archivia e accedi ai file di registro relativi ad Amazon SQS configurando le applicazioni o le funzioni Lambda che elaborano i messaggi per inviare i log ai log. CloudWatch Puoi utilizzare questi log per analizzare l'elaborazione dei messaggi, eseguire il debug dei problemi e monitorare le prestazioni dei flussi di lavoro Amazon SQS.

Per ulteriori informazioni, consulta [Registrazione delle chiamate API di Amazon Simple Queue Service utilizzando AWS CloudTrail](#).

CloudWatch Eventi Amazon

Usa Amazon CloudWatch Events per rilevare modifiche o eventi specifici nel tuo AWS ambiente e indirizzarli a una coda Amazon SQS. Ciò ti consente di acquisire dati sugli eventi, attivare flussi di lavoro o archiviare eventi per elaborarli in un secondo momento.

Per ulteriori informazioni, [Automazione delle notifiche dai AWS servizi ad Amazon SQS tramite Amazon EventBridge](#) consulta questa guida ed [EventBridge è l'evoluzione di Amazon CloudWatch Events](#) nella Amazon EventBridge User Guide.

AWS CloudTrail Registri

CloudTrail acquisisce un record dettagliato delle azioni eseguite su Amazon SQS da utenti, ruoli o. Servizi AWS Questi log consentono di tenere traccia delle chiamate API, ad esempio, or [SendMessageReceiveMessageDeleteQueue](#), e di fornire dettagli chiave come chi ha effettuato la richiesta, quando si è verificata e l'indirizzo IP di origine.

Per ulteriori informazioni, consulta [Registrazione delle chiamate API di Amazon Simple Queue Service utilizzando AWS CloudTrail](#).

AWS Trusted Advisor

Trusted Advisor utilizza le best practice sviluppate servendo AWS i clienti per aiutarti a ottimizzare l'utilizzo di Amazon SQS. Esamina le code Amazon SQS e fornisce consigli pratici per migliorare la sicurezza, migliorare l'affidabilità dell'elaborazione dei messaggi e ridurre i costi. Ad esempio, può suggerire di abilitare le code di posta indesiderata o di migliorare le politiche di accesso alle code per garantire operazioni sicure.

Per ulteriori informazioni, consulta [AWS Trusted Advisor](#) nella Guida per l'utente di Supporto .

CloudTrail sentieri

Un trail consente di CloudTrail inviare file di log a un bucket Amazon S3. Tutti i percorsi creati utilizzando il Console di gestione AWS sono multiregionali. È possibile creare un trail per una singola Regione o per più Regioni tramite AWS CLI. La creazione di un percorso multiregionale è consigliata in quanto consente di registrare l'intera attività del proprio Regioni AWS account. Se si crea un trail per una singola Regione, è possibile visualizzare solo gli eventi registrati nella Regione AWS del trail. Per ulteriori informazioni sui trail, consulta [Creating a trail for your Account AWS](#) e [Creating a trail for an organization](#) nella Guida per l'utente di AWS CloudTrail .

Puoi inviare gratuitamente una copia dei tuoi eventi di gestione in corso al tuo bucket Amazon S3 CloudTrail creando un percorso, tuttavia ci sono costi di storage di Amazon S3. [Per ulteriori informazioni sui CloudTrail prezzi, consulta la pagina Prezzi.AWS CloudTrail](#) Per informazioni sui prezzi di Amazon S3, consulta [Prezzi di Amazon S3](#).

CloudTrail Archivi di dati sugli eventi di Lake

CloudTrail Lake ti consente di eseguire query basate su SQL sui tuoi eventi. CloudTrail [Lake converte gli eventi esistenti in formato JSON basato su righe in formato Apache ORC](#). ORC è un

formato di archiviazione a colonne ottimizzato per il recupero rapido dei dati. Gli eventi vengono aggregati in archivi di dati degli eventi, che sono raccolte di eventi immutabili basate sui criteri selezionati applicando i [selettori di eventi avanzati](#). I selettori applicati a un archivio di dati degli eventi controllano quali eventi persistono e sono disponibili per l'esecuzione della query. Per ulteriori informazioni su CloudTrail Lake, consulta [Working with AWS CloudTrail Lake](#) nella Guida per l'utente AWS CloudTrail.

CloudTrail Gli archivi e le richieste di dati sugli eventi di Lake comportano dei costi. Quando crei un datastore di eventi, scegli l'[opzione di prezzo](#) da utilizzare per tale datastore. L'opzione di prezzo determina il costo per l'importazione e l'archiviazione degli eventi, nonché il periodo di conservazione predefinito e quello massimo per il datastore di eventi. [Per ulteriori informazioni sui CloudTrail prezzi, consulta la sezione Prezzi AWS CloudTrail](#)

Registrazione delle chiamate API di Amazon Simple Queue Service utilizzando AWS CloudTrail

CloudTrail consente di registrare e monitorare le operazioni di Amazon SQS utilizzando due tipi di eventi: eventi relativi ai dati ed eventi di gestione. Ciò semplifica il monitoraggio e la verifica dell'attività di Amazon SQS nel tuo account.

Eventi relativi ai dati di Amazon SQS in CloudTrail

[Gli eventi sui dati](#) forniscono informazioni sulle operazioni sulle risorse eseguite su o in una risorsa (ad esempio, l'invio di messaggi a un oggetto Amazon SQS). Queste operazioni sono definite anche operazioni del piano dei dati. Gli eventi di dati sono spesso attività che interessano volumi elevati di dati. Per impostazione predefinita, CloudTrail non registra gli eventi relativi ai dati. La cronologia CloudTrail degli eventi non registra gli eventi relativi ai dati.

Per gli eventi di dati sono previsti costi aggiuntivi. Per ulteriori informazioni sui CloudTrail prezzi, consulta la sezione [AWS CloudTrail Prezzi](#).

Puoi registrare gli eventi relativi ai dati per i tipi di risorse Amazon SQS utilizzando la CloudTrail console o AWS CLI le operazioni CloudTrail API. Per ulteriori informazioni su come registrare gli eventi relativi ai dati, consulta [Registrazione di eventi relativi ai dati con Console di gestione AWS](#) e [Registrazione di eventi di dati con AWS Command Line Interface](#) nella Guida per l'utente AWS CloudTrail .

Per registrare gli eventi relativi ai dati di Amazon SQS CloudTrail, devi utilizzare selettori di eventi avanzati per configurare le risorse o le azioni specifiche di Amazon SQS che desideri registrare.

Includi il tipo di risorsa [AWS::SQS::Queue](#) per acquisire le azioni relative alla coda. Puoi perfezionare ulteriormente le tue preferenze di registrazione utilizzando filtri come `eventName` (come gli eventi). [SendMessage](#) Per ulteriori informazioni, consulta [AdvancedEventSelector](#) nella documentazione di riferimento dell'API CloudTrail .

Tipo di evento di dati (console)	valore <code>resources.type</code>	Dati registrati APIs su CloudTrail
Coda Amazon SQS	AWS::SQS::Queue	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch • GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

Utilizza selettori di eventi avanzati per filtrare i campi e registrare solo gli eventi importanti. Per maggiori informazioni su questi campi, consulta [AdvancedFieldSelector](#) nella Guida di riferimento delle API di AWS CloudTrail .

Eventi di gestione di Amazon SQS in CloudTrail

[Gli eventi](#) di gestione forniscono informazioni sulle operazioni di gestione eseguite sulle risorse del tuo Account AWS. Queste operazioni sono definite anche operazioni del piano di controllo (control-plane). Per impostazione predefinita, CloudTrail registra gli eventi di gestione.

Amazon SQS registra le seguenti operazioni del piano di controllo CloudTrail come eventi di gestione.

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTasks](#)
- [PurgeQueue](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

Esempio di evento Amazon SQS

Un evento rappresenta una singola richiesta proveniente da qualsiasi fonte e include informazioni sull'operazione API richiesta, la data e l'ora dell'operazione, i parametri della richiesta e così via. CloudTrail i file di registro non sono una traccia ordinata dello stack delle chiamate API pubbliche, quindi gli eventi non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra un CloudTrail evento che dimostra l'SendMessageoperazione.

```
{
  "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

L'ListQueuesoperazione è un caso unico perché non agisce su una risorsa specifica. Di conseguenza, il campo ARN non include un nome di coda e utilizza invece un carattere jolly (*).

Per informazioni sui contenuti dei CloudTrail record, consulta i contenuti dei [CloudTrailrecord nella Guida](#) per l'AWS CloudTrail utente.

Monitoraggio delle code Amazon SQS tramite CloudWatch

Amazon SQS e Amazon CloudWatch sono integrati, quindi puoi utilizzarli CloudWatch per visualizzare e analizzare i parametri per le tue code Amazon SQS. [Puoi visualizzare e analizzare i parametri delle code dalla console Amazon SQS, dalla console, utilizzando o utilizzando l'API AWS CLI. CloudWatch CloudWatch](#) Puoi anche [impostare CloudWatch allarmi per i parametri](#) di Amazon SQS.

CloudWatch le metriche per le code Amazon SQS vengono raccolte automaticamente e inserite a intervalli di un minuto. CloudWatch Queste metriche vengono raccolte su tutte le code che soddisfano le linee guida per essere attive. CloudWatch CloudWatch considera attiva una coda per un massimo di sei ore se contiene messaggi o se un'azione vi accede.

Quando una coda Amazon SQS rimane inattiva per più di sei ore, il servizio Amazon SQS viene considerato inattivo e smette di fornire i parametri al servizio. CloudWatch I dati mancanti, o i dati che rappresentano zero, non possono essere visualizzati nelle CloudWatch metriche di Amazon SQS per il periodo di tempo in cui la coda Amazon SQS era inattiva.

Note

- Una coda Amazon SQS può essere attivata quando l'utente che chiama un'API contro la coda non è autorizzato e la richiesta ha esito negativo.
- La console Amazon SQS esegue una chiamata [GetQueueAttributes](#)API all'apertura della pagina della coda. La richiesta GetQueueAttributes API attiva la coda.
- Quando una coda viene attivata da uno stato inattivo, si verifica un ritardo fino a 15 minuti nelle CloudWatch metriche.

- Non sono previsti costi per i parametri di Amazon SQS riportati in CloudWatch. Sono fornite come parte del servizio Amazon SQS.
- CloudWatch le metriche sono supportate sia per le code standard che per quelle FIFO.

Accesso ai CloudWatch parametri per Amazon SQS

Amazon SQS e Amazon CloudWatch sono integrati, quindi puoi utilizzarli CloudWatch per visualizzare e analizzare i parametri per le tue code Amazon SQS. [Puoi visualizzare e analizzare i parametri delle code dalla console Amazon SQS, dalla console, utilizzando o utilizzando l'API AWS CLI. CloudWatch CloudWatch](#) Puoi anche [impostare CloudWatch allarmi per i parametri](#) di Amazon SQS.

Utilizzo della console Amazon SQS

Utilizza la console Amazon SQS per accedere e analizzare i parametri per un massimo di 10 code Amazon SQS.

1. Accedere alla [console Amazon SQS](#).
2. Nell'elenco delle code, scegli (seleziona) le caselle per le code ai cui parametri desideri accedere. Puoi visualizzare i parametri per un massimo di 10 code.
3. Scegli la scheda Monitoraggio.

Nella sezione relativa ai parametri SQS vengono visualizzati diversi grafici.

4. Per comprendere ciò che un determinato grafico rappresenta, passa il mouse su  accanto al grafico desiderato, oppure consulta [CloudWatch Metriche disponibili per Amazon SQS](#).
5. Per modificare l'intervallo di tempo per tutti i grafici contemporaneamente, per Time Range (Intervallo di tempo) scegli l'intervallo di tempo desiderato (per esempio, Last Hour (Ultima ora)).
6. Per visualizzare statistiche aggiuntive per un singolo grafico, scegli il grafico.
7. Nella finestra di dialogo Monitoring Details (Dettagli monitoraggio) CloudWatch, seleziona una Statistic (Statistica), (ad esempio, Sum (Somma)). Per un elenco di statistiche supportate, consulta [CloudWatch Metriche disponibili per Amazon SQS](#).
8. Per modificare l'intervallo di tempo o il periodo di tempo per il quale un singolo grafico viene visualizzato (ad esempio, per mostrare un intervallo di tempo delle ultime 24 ore anziché gli ultimi

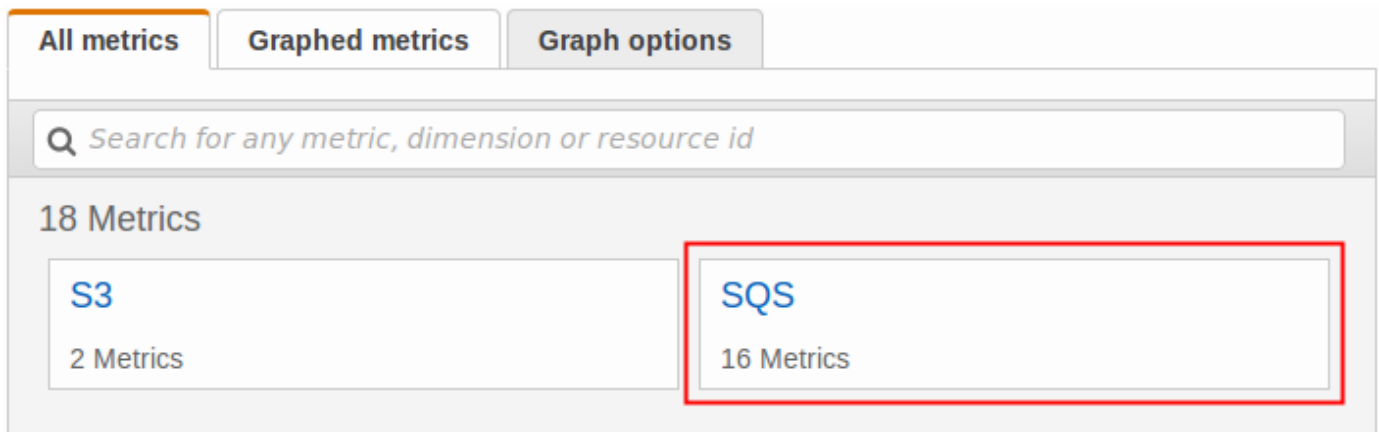
5 minuti, o per visualizzare un periodo di tempo di ogni ora anziché ogni 5 minuti), con la finestra di dialogo del grafico ancora visualizzata, per Time Range (Intervallo di tempo) scegli l'intervallo di tempo desiderato (per esempio, Last 24 Hours (Ultime 24 ore)). Per Period (Periodo) scegli il periodo di tempo desiderato entro l'intervallo di tempo specificato (ad esempio, 1 Hour (1 ora)). Dopo aver finito di esaminare il grafico, scegli Close (Chiudi).

9. (Facoltativo) Per utilizzare CloudWatch funzionalità aggiuntive, nella scheda Monitoraggio, scegli Visualizza tutti i CloudWatch parametri, quindi segui le istruzioni della procedura. [Utilizzo della CloudWatch console Amazon](#)

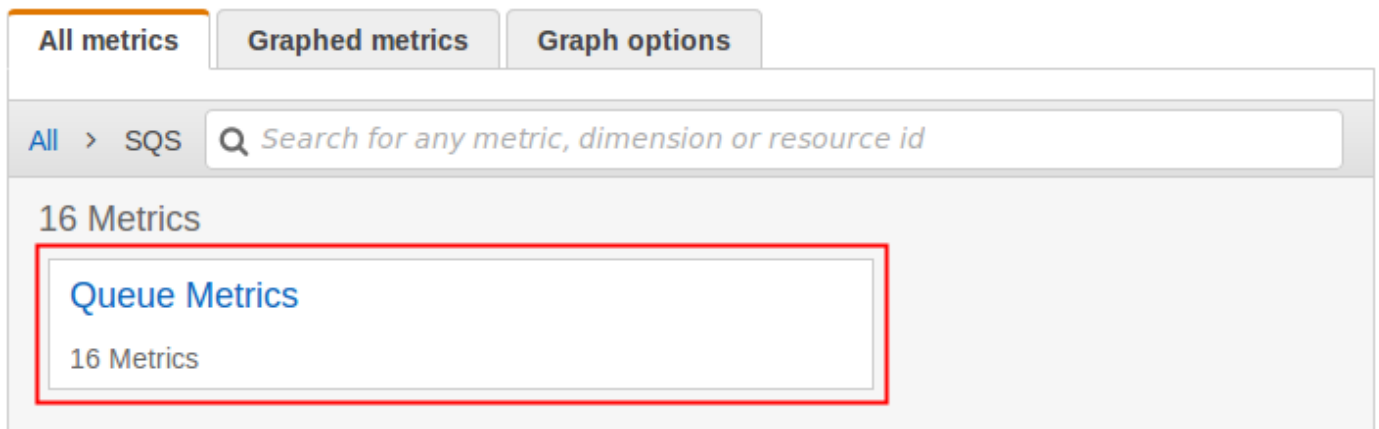
Utilizzo della CloudWatch console Amazon

Usa la CloudWatch console per accedere e analizzare i parametri di Amazon SQS.

1. Accedi alla [console CloudWatch](#).
2. Nel pannello di navigazione, scegli Metrics (Parametri).
3. Seleziona namespace parametro SQS.



4. Seleziona la dimensione parametro Queue Metrics (parametri coda).



5. Puoi ora quindi esaminare le metriche Amazon SQS:

- Per ordinare i parametri, utilizza l'intestazione della colonna.
- Per creare il grafico di un parametro, seleziona la casella di controllo accanto al parametro.
- Per filtrare in base a un parametro, scegli il nome del parametro e quindi Add to search (Aggiungi alla ricerca).

All metrics			Graphed metrics	Graph options
All > SQS > Queue Metrics			Search for any metric, dimension or resource id	
☐	QueueName (16)	Metric Name		
☐	MyQueue	ApproximateAgeOfOldestMessage		
☐	MyQueue	MessagesDelayed		
☐	MyQueue	MessagesNotVisible		
☐	MyQueue	MessagesVisible		
☐	MyQueue			
☐	MyQueue			
☐	MyQueue			
☐	MyQueue			
☐	MyQueue			
☐	MyQueue	NumberOfMessagesSent		

Per ulteriori informazioni e opzioni aggiuntive, consulta [Graph Metrics](#) e [Using Amazon CloudWatch Dashboards](#) nella Amazon CloudWatch User Guide.

Usando il AWS Command Line Interface

Per accedere ai parametri di Amazon SQS utilizzando AWS CLI, esegui il comando. [get-metric-statistics](#)

Per ulteriori informazioni, consulta [Get Statistics for a Metric](#) nella Amazon CloudWatch User Guide.

Utilizzo dell'API CloudWatch

Per accedere ai parametri di Amazon SQS utilizzando l' CloudWatch API, utilizza l'azione. [GetMetricStatistics](#)

Per ulteriori informazioni, consulta [Get Statistics for a Metric](#) nella Amazon CloudWatch User Guide.

Creazione di CloudWatch allarmi per i parametri di Amazon SQS

CloudWatch consente di attivare allarmi quando una metrica raggiunge una soglia specificata. Ad esempio, puoi creare un allarme per il parametro `NumberOfMessagesSent`. Ad esempio, se vengono inviati più di 100 messaggi alla coda `MyQueue` in 1 ora, viene inviata una notifica tramite e-mail. Per ulteriori informazioni, consulta [Creating Amazon CloudWatch Alarms](#) nella Amazon CloudWatch User Guide.

1. Accedi Console di gestione AWS e apri la CloudWatch console all'indirizzo <https://console.aws.amazon.com/cloudwatch/>.
2. Seleziona Alarms (Allarmi), quindi Create Alarm (Crea allarme).
3. Nella sezione Select Metric (Seleziona parametro) della finestra di dialogo Create Alarm (Crea allarme), scegli Browse Metrics (Sfoglia parametri), SQS.
4. Per SQS > Queue Metrics, scegli il nome `QueueName` e la metrica per cui impostare un allarme, quindi scegli Avanti. Per un elenco di parametri disponibili, consulta [CloudWatch Metriche disponibili per Amazon SQS](#).

Nel seguente esempio, la selezione è per un allarme per il parametro `NumberOfMessagesSent` per la coda `MyQueue`. I trigger di allarme quando il numero di messaggi inviati supera 100.

5. Nella sezione Define Alarm (Definisci allarme) della finestra di dialogo Create Alarm (Crea allarme), procedi come segue:
 - a. In Alarm Threshold (Soglia allarme), digita Name (Nome) e Description (Descrizione) per l'allarme.
 - b. Imposta is su > 100.
 - c. Imposta for (per) su 1 out of 1 datapoints (1 di 1 punto dati).
 - d. In Alarm preview (Anteprima allarme), imposta Period (Periodo) su 1 Hour (1 ora).
 - e. Imposta Statistic (Statistica) su Standard, Sum (Somma).
 - f. In Actions (Operazioni), imposta Whenever this alarm (Ogni volta che l'allarme) su State is ALARM (Stato è ALLARME).

Se desideri CloudWatch inviare una notifica quando viene attivato l'allarme, seleziona un argomento Amazon SNS esistente o scegli Nuovo elenco e inserisci gli indirizzi e-mail separati da virgole.

 Note

Se crei un nuovo argomento Amazon SNS, gli indirizzi e-mail devono essere verificati prima che possano ricevere le notifiche. Se lo stato dell'allarme cambia prima della verifica degli indirizzi e-mail, le notifiche non vengono inviate.

6. Scegli Crea allarme.

L'allarme viene creato.

CloudWatch Metriche disponibili per Amazon SQS

Amazon SQS invia le seguenti metriche a CloudWatch

 Note

Per alcuni parametri, il risultato è approssimativo a causa dell'architettura distribuita di Amazon SQS. Nella maggior parte dei casi, il conteggio deve essere vicino al numero effettivo di messaggi in coda.

Metriche Amazon SQS

Amazon SQS pubblica automaticamente i parametri operativi [su CloudWatch Amazon](#) nello spazio dei nomi. AWS/SQS Queste metriche ti aiutano a monitorare lo stato e le prestazioni delle code. Data la natura distribuita di SQS, molti valori sono approssimativi, ma sufficientemente accurati per la maggior parte delle decisioni operative.

 Note

- Tutte le metriche emettono valori non negativi solo quando la coda è attiva.
- Alcune metriche (ad esempio `SentMessageSize`) non vengono emesse finché non viene inviato almeno un messaggio.

Metrica	Description	unità	Comportamento dei report	Note chiave
ApproximateAgeOfOldestMessage	L'età del messaggio non elaborato più vecchio presente nella coda.	Secondi	Segnalato se la coda contiene almeno un messaggio attivo.	• Per le code standard, se un messaggio viene ricevuto tre o più volte e non viene eliminato, SQS lo sposta in fondo alla coda. La metrica riflette quindi l'età del messaggio successivo o che non ha superato la soglia di ricezione. Questo riordino si verifica anche quando è in vigore una politica di redrive. • I messaggi Poison-pill (quelli ricevuti ripetutamente ma mai eliminati) vengono esclusi da questa metrica fino a quando non vengono elaborati correttamente. • Quando un messaggio viene spostato in un DLQ dopo aver superato il, la data di scadenza viene ripristinata. <code>maxReceiveCount</code>. In tal caso, la metrica del DLQ riflette l'ora in cui il messaggio è stato spostato, non quando è stato originariamente inviato. • Le code FIFO non riordinano i messaggi per mantenere l'ordine. Un messaggio non

Metrica	Description	unità	Comportamento dei report	Note chiave
				riuscito blocca il relativo gruppo di messaggi fino alla sua eliminazione o alla scadenza. Se è configurato un DLQ, il messaggio viene inviato lì dopo il raggiungimento della soglia di ricezione.

Metrica	Description	unità	Comportamento dei report	Note chiave
ApproximateNumberOfGroupsWithInflightMessages	Solo per FIFO. Il numero di gruppi di messaggi con uno o più messaggi in corso.	Conteggio	Segnalato se la coda FIFO è attiva.	- Un messaggio viene considerato in transito dopo che è stato ricevuto dalla coda da un consumatore ma non è ancora stato eliminato o è scaduto. - Questa metrica consente di risolvere i problemi e ottimizzare la velocità effettiva delle code FIFO. I valori elevati di solito indicano una forte concorrenza. - Se la coda presenta un notevole arretrato e questo valore rimane basso, valuta la possibilità di ridimensionare i consumatori o aumentare il numero di gruppi di messaggi attivi. - Per informazioni sulla velocità effettiva e sui limiti in volo, consulta. Quote di Amazon SQS

Metrica	Description	unità	Comportamento dei report	Note chiave
ApproximateNumberOfMessagesDelayed	Il numero di messaggi in coda che sono in ritardo e non sono immediatamente disponibili per il recupero.	Conteggio	Segnalato se nella coda sono presenti messaggi in ritardo.	• Si applica alle code configurate con un ritardo predefinito e ai singoli messaggi inviati con un parametro. <code>DelaySeconds</code> • I messaggi ritardati rimangono nascosti ai consumatori fino alla scadenza del periodo di ritardo, il che può influire sulla percezione dell'arretrato o del throughput delle code.
ApproximateNumberOfMessagesNotVisible	Il numero di messaggi in volo che sono stati ricevuti ma non ancora eliminati o scaduti.	Conteggio	Segnalato se esistono messaggi in volo.	• I messaggi entrano nello stato in corso dopo essere stati inviati a un consumatore tramite l'ReceiveMessage API. • Questi messaggi sono temporaneamente nascosti agli altri consumatori durante la finestra di timeout di visibilità. • Utilizza questa metrica per tenere traccia dei ritardi nell'elaborazione dei messaggi o dei consumatori bloccati.

Metrica	Description	unità	Comportamento dei report	Note chiave
ApproximateNumberOfMessagesVisible	Il numero di messaggi attualmente disponibili per il recupero e l'elaborazione.	Conteggio	Segnalato se la coda è attiva.	- Riflette l'attuale arretrato di elaborazione nella coda. Non esiste un limite rigido al numero di messaggi che possono accumularsi, ma sono soggetti al periodo di conservazione configurato dalla coda. - Un valore costantemente elevato può indicare che i consumatori non dispongono di risorse sufficienti o che la logica di elaborazione è bloccata.
NumberOfEmptyReceives ¹	Il numero di chiamate ReceiveMessage API che non hanno restituito messaggi.	Conteggio	Segnalato durante le operazioni di ricezione.	- Questa metrica può aiutare a identificare le inefficienze nel comportamento dei sondaggi o i casi di consumo sottoutilizzati. - Possono verificarsi valori elevati quando la coda è vuota, il consumatore utilizza sondaggi brevi o i messaggi vengono elaborati più velocemente di quanto vengano prodotti. - Questo non è un indicatore preciso dello stato della coda. Riflette il comportamento sul lato del servizio e può includere nuovi tentativi.

Metrica	Description	unità	Comportamento dei report	Note chiave
NumberOfDeduplicatedSentMessages	Solo per FIFO. Il numero di messaggi inviati che sono stati deduplicati e non aggiunti alla coda.	Conteggio	Segnalato se vengono rilevati MessageDeduplicationId valori o contenuti duplicati.	• SQS deduplica i messaggi in base all'hashing basato sul MessageDeduplicationId contenuto (se abilitato). • Un valore elevato può indicare che un produttore e invia ripetutamente lo stesso messaggio entro la finestra di deduplicazione di 5 minuti. • Utilizzate questa metrica per risolvere i problemi della logica ridondante del produttore o per confermare che la deduplicazione funzioni come previsto.

Metrica	Description	unità	Comportamento dei report	Note chiave
NumberOfMessagesDeleted ¹	Il numero di messaggi eliminati con successo dalla coda.	Conteggio	Segnalato per ogni richiesta di eliminazione con un handle di ricezione valido.	- Questa metrica conta tutte le operazioni di eliminazione riuscite, anche se lo stesso messaggio viene eliminato più di una volta. - I motivi più comuni alla base dei valori includono: higher-than-expected - Eliminazioni multiple dello stesso messaggio utilizzando diversi handle di ricezione, dopo la scadenza del timeout di visibilità e la ricezione del messaggio. - Eliminazioni duplicate utilizzando lo stesso handle di ricezione, che restituiscono comunque lo stato di successo e incrementano la metrica. - Utilizzate questa metrica per tenere traccia del successo dell'elaborazione dei messaggi, ma non trattatela come un conteggio esatto dei messaggi unici eliminati.

Metrica	Description	unità	Comportamento dei report	Note chiave
NumberOfMessagesReceived ¹	Il numero di messaggi restituiti dall' ReceiveMessageAPI .	Conteggio	Segnalato durante le operazioni di ricezione.	• Ciò include tutti i messaggi restituiti ai consumatori, compresi quelli che vengono successivamente rimessi in coda a causa della scadenza del timeout di visibilità. • Un singolo messaggio può essere ricevuto più volte se non viene eliminato, il che può far sì che questa metrica superi il numero di messaggi inviati. • Utilizzala per tracciare l'attività dei consumatori, ma non considerarla come un numero di messaggi unici elaborati.

Metrica	Description	unità	Comportamento dei report	Note chiave
NumberOfMessagesSent ¹	Il numero di messaggi aggiunti con successo a una coda.	Conteggio	Segnalato per ogni invio manuale riuscito.	• <code>SendMessageBatch</code> Vengono conteggiate le chiamate manuali verso <code>SendMessage</code> o verso un DLQ, incluse quelle dirette a un DLQ. • I messaggi che vengono spostati automaticamente in un DLQ dopo aver superato il limite <code>non maxReceiveCount</code> sono inclusi in questa metrica. • Di conseguenza, <code>NumberOfMessagesSent</code> potrebbe essere inferiore a, <code>NumberOfMessagesReceived</code> specialmente se le politiche di <code>redrive</code> spostano molti messaggi in background. DLQs

Metrica	Description	unità	Comportamento dei report	Note chiave
SentMessageSize ¹	La dimensione dei messaggi inviati con successo alla coda.	Byte	Non viene emesso finché non viene inviato almeno un messaggio.	- Questa metrica non verrà visualizzata nella CloudWatch console finché la coda non riceverà il primo messaggio. - Utilizza questa metrica per tenere traccia della dimensione di ogni messaggio in byte. Ciò è utile per analizzare le tendenze del payload o stimare i costi di throughput. - La dimensione massima dei messaggi per SQS è di 1 MiB.
ApproximateNumberOfNoisyGroups	Il numero di gruppi di messaggi considerati rumorosi in una coda equa. Un gruppo di messaggi rumoroso rappresenta un inquilino vicino rumoroso di una coda multi-tenant.	Conteggio	Se la coda è attiva, viene riportato un valore non negativo.	- Aiuta a identificare potenziali problemi legati alla rumorosità dei vicini in ambienti multi-tenant tracciando i gruppi di messaggi che consumano risorse sproporzionate. - Utilizzate questa metrica per impostare allarmi che si attivano quando il numero di gruppi rumorosi supera la soglia accettabile, indicando potenziali problemi di equità delle code.

Metrica	Description	unità	Comportamento dei report	Note chiave
ApproximateNumberOfMessagesVisibleInQuietGroups	Il numero di messaggi visibili esclusi i messaggi provenienti da gruppi di messaggi rumorosi.	Conteggio	Se la coda è attiva, viene riportato un valore non negativo.	- Fornisce visibilità sul backlog della coda per i gruppi di messaggi con tariffa standard, esclusi i messaggi provenienti da vicini rumorosi. - Aiuta a identificare il vero backlog di elaborazione per i gruppi di messaggi tipici filtrando l'impatto dei vicini rumorosi.
ApproximateNumberOfMessagesNotVisibleInQuietGroups	Il numero di messaggi in transito, esclusi i messaggi provenienti da gruppi di messaggi rumorosi.	Conteggio	Se la coda è attiva, viene riportato un valore non negativo.	- Tiene traccia dei messaggi in corso (in fase di elaborazione ma non ancora eliminati) provenienti da gruppi di messaggi che si comportano bene. - Utilizza questa metrica per monitorare la velocità di elaborazione dei normali gruppi di messaggi e rilevare rallentamenti nell'elaborazione che non sono causati da vicini rumorosi.

Metrica	Description	unità	Comportamento dei report	Note chiave
ApproximateNumberOfMessagesDelayedInQuietGroups	Il numero di messaggi, esclusi i messaggi provenienti da gruppi di messaggi rumorosi, che vengono ritardati e non sono disponibili per la lettura immediata. I messaggi ritardati si verificano o quando la coda è configurata come coda di ritardo o quando un messaggio è stato inviato con un parametro di ritardo.	Conteggio	Se la coda è attiva, viene riportato un valore non negativo.	• Aiuta a monitorare l'arretrato dei messaggi in ritardo provenienti da gruppi di messaggi con schemi di throughput normali o previsti (a differenza dei gruppi ad alto volume o rumorosi) • Utile per comprendere i requisiti di elaborazione futuri e la pianificazione della capacità per i carichi di lavoro tipici.

Metrica	Description	unità	Comportamento dei report	Note chiave
ApproximateAgeOfOldestMessageInQuietGroups	L'età del messaggio non eliminato più vecchio nella coda, esclusi i messaggi provenienti da gruppi di messaggi rumorosi.	Secondi	Se la coda è attiva, viene riportato un valore non negativo.	- Utilizzato per monitorare e la conformità agli SLA e rilevare i rallentamenti nell'elaborazione in gruppi di messaggi con modelli di throughput normali o previsti (al contrario dei gruppi di messaggi ad alto volume o rumorosi che altrimenti potrebbero alterare la metrica). - Utilizzate questa metrica per impostare allarmi per i timeout di elaborazione dei messaggi che ignorano i messaggi invecchiati artificialmente provenienti da vicini rumorosi.

¹ Queste metriche riflettono l'attività a livello di sistema e possono includere nuovi tentativi, duplicati o messaggi ritardati. Non utilizzare conteggi non elaborati per stimare lo stato della coda in tempo reale senza tenere conto del comportamento del ciclo di vita dei messaggi.

Code e metriche contenenti lettere non scritte () DLQs CloudWatch

Quando si lavora con DLQs, è importante capire come si comportano i parametri di Amazon SQS:

- **NumberOfMessagesSent**— Questa metrica si comporta in modo diverso per: DLQs
 - Invio manuale: i messaggi inviati manualmente a un DLQ vengono acquisiti da questa metrica.
 - Reindirizzamento automatico: i messaggi spostati automaticamente su un DLQ a causa di errori di elaborazione non vengono acquisiti da questa metrica. Di conseguenza, le

`NumberOfMessagesReceived` metriche `NumberOfMessagesSent` e potrebbero mostrare discrepanze per. DLQs

- **Metrica consigliata per DLQs:** per monitorare lo stato di un DLQ, utilizza la metrica. `ApproximateNumberOfMessagesVisible` Questa metrica indica il numero di messaggi attualmente disponibili per l'elaborazione nel DLQ.

Code e metriche eque CloudWatch

Quando utilizzi [code corrette](#), Amazon SQS emette i seguenti parametri aggiuntivi:

- `ApproximateNumberOfNoisyGroups`
- `ApproximateNumberOfMessagesVisibleInQuietGroups`
- `ApproximateNumberOfMessagesNotVisibleInQuietGroups`
- `ApproximateNumberOfMessagesDelayedInQuietGroups`
- `ApproximateAgeOfOldestMessageInQuietGroups`

Note

Ogni `QuietGroup` metrica è un sottoinsieme della `Approximate` metrica standard equivalente a livello di coda, ma esclude i messaggi provenienti da gruppi di vicini rumorosi.

Gruppi rumorosi

Un gruppo di messaggi rumoroso rappresenta un inquilino vicino rumoroso di una coda multi-tenant.

Gruppi silenziosi

Gruppi di messaggi esclusi i gruppi rumorosi.

Osservazione del comportamento delle code corrette di SQS

Per monitorare l'effetto delle code eque di Amazon SQS, puoi confrontare i `Approximate..InQuietGroups` parametri con i parametri standard a livello di coda. Durante i picchi di traffico per uno specifico tenant, le metriche generali a livello di coda possono rivelare un aumento degli arretrati o una vecchiaia dei messaggi più vecchia. Tuttavia, esaminando isolatamente

i gruppi silenziosi, è possibile identificare che la maggior parte dei gruppi di messaggi o degli inquilini non rumorosi non ne risente e fornire una stima del numero totale di gruppi di messaggi interessati.

Sebbene queste nuove metriche forniscano una buona panoramica del comportamento delle code corrette di Amazon SQS, può essere utile capire quale tenant specifico sta causando il carico.

[Amazon CloudWatch Contributor Insights](#) ti consente di visualizzare le metriche relative ai contributori principali, al numero totale di contributori unici e al loro utilizzo. Ciò è particolarmente utile negli scenari in cui si ha a che fare con migliaia di inquilini che altrimenti porterebbero a dati (e costi) ad alta cardinalità quando si emettono metriche tradizionali.

Per un esempio di configurazione del monitoraggio per le code eque, consultate l'esempio su [GitHub](#).

Dimensioni per le metriche Amazon SQS

Le metriche di Amazon SQS CloudWatch utilizzano un'unica dimensione: **QueueName**. Tutti i dati metrici vengono raggruppati e filtrati in base al nome della coda.

Suggerimenti per il monitoraggio

Monitora SQS in modo efficace utilizzando metriche e CloudWatch allarmi chiave per rilevare gli arretrati nelle code, ottimizzare le prestazioni e rispettare i limiti del servizio.

- Imposta [CloudWatch allarmi](#) in base `ApproximateNumberOfMessagesVisible` a catch backlog growth.
- Monitora `NumberOfEmptyReceives` per ottimizzare la frequenza dei sondaggi e ridurre i costi delle API.
- Utilizzalo `ApproximateNumberOfGroupsWithInflightMessages` nelle code FIFO per diagnosticare i limiti di throughput.
- Rivedi le [quote SQS per comprendere le soglie metriche e](#) i limiti del servizio.

Convalida della conformità per Amazon SQS

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione [Scope by Compliance Program Servizi AWS](#) e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di [AWS conformità Programmi](#) di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta [Scaricamento dei report in AWS Artifact](#) .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. Per ulteriori informazioni sulla responsabilità di conformità durante l'utilizzo Servizi AWS, consulta [AWS la documentazione sulla sicurezza](#).

Resilienza in Amazon SQS

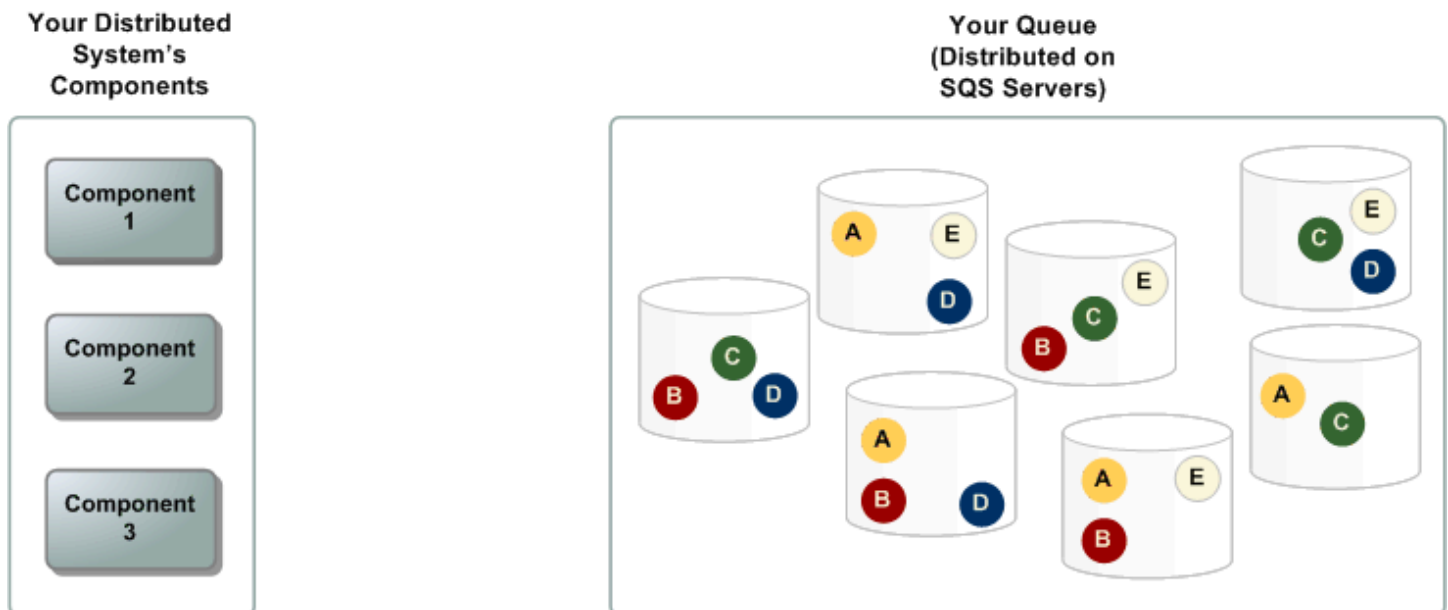
L'infrastruttura AWS globale è costruita attorno a AWS regioni e zone di disponibilità. AWS Le regioni offrono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, throughput elevato e altamente ridondante. Con le zone di disponibilità è possibile progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali. [Per ulteriori informazioni su AWS regioni e zone di disponibilità, consulta Global Infrastructure.AWS](#)

Oltre all'infrastruttura AWS globale, Amazon SQS offre code distribuite.

Code distribuite

Un sistema di messaggistica distribuito si compone di tre parti principali: i componenti del sistema distribuito, la coda (distribuita sui server Amazon SQS) e i messaggi in coda.

Nel seguente scenario, il sistema dispone di diversi produttori, ossia componenti che inviano messaggi alla coda, e consumatori, componenti che ricevono messaggi dalla coda. La coda (che contiene i messaggi da A a E) archivia in modo ridondante i messaggi su più server Amazon SQS.



Sicurezza dell'infrastruttura in Amazon SQS

In quanto servizio gestito, Amazon SQS è protetto dalle procedure di sicurezza di rete AWS globali descritte nel white paper [Amazon Web Services: Overview of Security Processes](#).

Utilizza azioni API AWS pubblicate per accedere ad Amazon SQS attraverso la rete. I client devono supportare Transport Layer Security (TLS) 1.2 o versioni successive. I client devono, inoltre, supportare le suite di cifratura con PFS (Perfect Forward Secrecy), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE).

Le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associate a un principal IAM. In alternativa, è possibile utilizzare [AWS Security Token Service](#) (AWS STS) per generare credenziali di sicurezza temporanee per le richieste di firma.

Puoi richiamare queste operazioni API da qualsiasi posizione di rete, ma Amazon SQS supporta le policy di accesso basate sulle risorse, che possono includere limitazioni in base all'indirizzo IP di origine. Puoi anche utilizzare le policy di Amazon SQS per controllare l'accesso da endpoint Amazon VPC specifici o specifici VPCs. Ciò isola efficacemente l'accesso alla rete a una determinata coda Amazon SQS solo dal VPC specifico all'interno della rete. AWS Per ulteriori informazioni, consulta [Esempio 5: Negare l'accesso se non è un endpoint VPC](#).

Best practice di sicurezza per Amazon SQS

AWS offre molte funzionalità di sicurezza per Amazon SQS, che dovresti esaminare nel contesto della tua politica di sicurezza. Di seguito sono riportate le best practice di prevenzione per la sicurezza per Amazon SQS.

Note

Le linee guida specifiche per l'implementazione fornite sono per i casi d'uso e le implementazioni comuni. Si consiglia di visualizzare best practice nel contesto del caso d'uso specifico, dell'architettura e del modello di minaccia.

Assicurarsi che le code non siano accessibili pubblicamente

A meno che tu non richieda esplicitamente a chiunque su Internet di essere in grado di leggere o scrivere nella tua coda Amazon SQS, devi assicurarti che la coda non sia accessibile al pubblico (accessibile da tutti nel mondo o da qualsiasi utente autenticato). AWS

- Evitare di creare policy con Principal impostato su "".
- Evitare di utilizzare un carattere jolly (*). Assegnare invece un nome a uno o più utenti specifici.

Implementazione dell'accesso con privilegi minimi

Quando si concedono autorizzazioni, si decide chi le riceve, per quali code sono le autorizzazioni e operazioni API specifiche che si desidera consentire per queste code. Implementare privilegi minimi è importante per ridurre i rischi per la sicurezza e ridurre l'effetto di errori o intenti dannosi.

Seguire i consigli di sicurezza standard relativi alla concessione dei privilegi minimi. Cioè, concedere solo le autorizzazioni necessarie per eseguire un'attività specifica. È possibile effettuare questa implementazione utilizzando una combinazione di policy di sicurezza.

Amazon SQS utilizza il modello produttore-consumatore, che richiede tre tipi di accesso all'account utente:

- Amministratori: accesso alla creazione, alla modifica e all'eliminazione di code. Gli amministratori controllano anche le policy della coda.
- Produttori: accesso all'invio di messaggi alle code.
- Consumatori: accesso alla ricezione e all'eliminazione di messaggi dalle code.

Per ulteriori informazioni, consulta le sezioni seguenti:

- [Identity and Access Management in Amazon SQS](#)
- [Autorizzazioni API Amazon SQS: riferimento a operazioni e risorse](#)
- [Utilizzo di policy personalizzate con la sintassi delle policy di accesso Amazon SQS](#)

Usa i ruoli IAM per applicazioni e AWS servizi che richiedono l'accesso ad Amazon SQS

Affinché applicazioni o AWS servizi come Amazon EC2 possano accedere alle code di Amazon SQS, devono utilizzare credenziali AWS valide nelle richieste API. AWS Poiché queste credenziali non vengono ruotate automaticamente, non è necessario archiviare AWS le credenziali direttamente nell'applicazione o nell'istanza EC2.

È preferibile usare un ruolo IAM per gestire credenziali provvisorie per le applicazioni o i servizi che devono accedere ad Amazon SQS. Quando utilizzi un ruolo, non devi distribuire credenziali a lungo termine (come nome utente, password e chiavi di accesso) a un'istanza o servizio EC2 come AWS Lambda. Al contrario, il ruolo fornisce autorizzazioni temporanee che le applicazioni possono utilizzare quando effettuano chiamate ad altre risorse. AWS

Per ulteriori informazioni, consulta [Ruoli IAM](#) e [Scenari comuni per ruoli: utenti, applicazioni e servizi](#) nella Guida per l'utente IAM.

Implementare la crittografia lato server

Per attenuare i problemi di perdita di dati, utilizzare la crittografia dei dati inattivi per crittografare i messaggi utilizzando una chiave memorizzata in un percorso diverso da quello in cui vengono archiviati i messaggi. La crittografia lato server (SSE) fornisce la crittografia dei dati inattivi. Amazon SQS esegue la crittografia dei dati a livello di messaggio quando li memorizza e decrittografa i messaggi per l'utente quando vi accede. SSE utilizza chiavi gestite in AWS Key Management Service. Finché si autentica la richiesta e si dispone delle autorizzazioni di accesso, non vi è alcuna differenza tra l'accesso alle code crittografate e non crittografate.

Per ulteriori informazioni, consultare [Crittografia inattiva in Amazon SQS](#) e [Gestione delle chiavi Amazon SQS](#).

Applica la crittografia dei dati in transito

Senza HTTPS (TLS), un utente malintenzionato basato sulla rete può intercettare il traffico di rete o manipolarlo, utilizzando un attacco come man-in-the-middle. Consenti solo connessioni crittografate tramite HTTPS (TLS) utilizzando la condizione [aws:SecureTransport](#) nella policy della coda per forzare le richieste a utilizzare SSL.

Prendere in considerazione l'utilizzo di endpoint VPC per accedere a Amazon SQS

Se si dispone di code con cui è necessario essere in grado di interagire ma che non devono assolutamente essere esposte a Internet, utilizzare gli endpoint VPC per accordare l'accesso solo agli host all'interno di un determinato VPC. Puoi utilizzare le policy di coda per controllare l'accesso alle code da endpoint Amazon VPC specifici o da specifici VPCs.

Gli endpoint VPC di Amazon SQS offrono due modi per controllare l'accesso ai messaggi:

- È possibile controllare le richieste, gli utenti o i gruppi autorizzati tramite un endpoint VPC specifico.
- Puoi controllare quali endpoint VPCs o VPC hanno accesso alla tua coda utilizzando una politica di coda.

Per ulteriori informazioni, consultare [Endpoint di Amazon Virtual Private Cloud per Amazon SQS](#) e [Creazione di una policy per endpoint VPC di Amazon per Amazon SQS](#).

Risorse Amazon SQS correlate

La tabella seguente elenca le risorse correlate che possono essere utili durante l'utilizzo di questo servizio.

Risorsa	Descrizione
Documentazione di riferimento delle API di Amazon Simple Queue Service	Descrizione di operazioni, parametri e tipi di dati e un elenco di errori generati dal servizio.
Amazon SQS nel Riferimento ai comandi di AWS CLI	Descrizioni dei AWS CLI comandi che è possibile utilizzare e per lavorare con le code.
Regioni ed endpoint	Informazioni su regioni ed endpoint di Amazon SQS
Pagina del prodotto	La pagina Web principale che include informazioni su Amazon SQS.
Forum di discussione	Forum basato su community per sviluppatori per la discussione di questioni tecniche correlate ad Amazon SQS.
AWS Informazioni sull'assistenza Premium	La pagina Web principale per informazioni su AWS Premium Supportone-on-one, un canale di supporto a risposta rapida che consente di creare ed eseguire applicazioni sui servizi di AWS infrastruttura.

Cronologia della documentazione

Nella tabella seguente sono descritte le modifiche importanti apportate alla Guida per gli sviluppatori di Amazon Simple Queue Service da gennaio 2019. Per ricevere notifiche sugli aggiornamenti della documentazione, è possibile iscriversi al [feed RSS](#).

Le funzionalità del servizio vengono talvolta implementate in modo incrementale AWS nelle regioni in cui il servizio è disponibile. Aggiorniamo questa documentazione solo per la prima versione. Non forniamo informazioni sulla disponibilità delle regioni e non annunciamo implementazioni successive delle regioni. Per informazioni sulla disponibilità regionale delle funzionalità del servizio e per iscriverti alle notifiche sugli aggiornamenti, vedi [Cosa c'è di AWS nuovo?](#) .

Modifica	Descrizione	Data
Supporto per Fair Queues	È stato aggiunto il supporto Fair Queues per le code standard.	21 luglio 2025
È stato aggiunto il supporto per gli endpoint dual-stack (e) IPv4 IPv6	Amazon SQS ora supporta gli endpoint dual-stack (IPv4 and IPv6), consentendo l'accesso alle code tramite entrambi i protocolli IP.	17 aprile 2025
CloudTrail integrazione per tutti gli Amazon SQS APIs	CloudTrail integrazione aggiunta per tutti gli Amazon APIs SQS.	10 gennaio 2025
SQSUnlockQueuePolicy	Amazon SQS ha aggiunto una nuova policy AWS gestita chiamata SQSUnlockQueuePolicy per sbloccare una coda e rimuovere una policy di coda non configurata correttamente che impedisce a tutti i	15 novembre 2024

principali di accedere a una coda Amazon SQS.

[AWS kms:Decrypt](#)

Amazon SQS non richiede più l'`kms:Decrypt` autorizzazione per l'`SendMessage` API. I clienti ora necessitano solo dell'`kms:GenerateDataKey` autorizzazione sulla chiave KMS utilizzata per crittografare la coda, ma hanno comunque bisogno `kms:Decrypt` dell'autorizzazione per chiamare `ReceiveMessage`.

24 luglio 2024

[Aggiornamento delle metriche FIFO](#)

Supporta `NumberOfDuplicatedSentMessages` e `ApproximateNumberOfGroupsWithInflightMessages` aggiunge parametri FIFO di Amazon SQS.

3 luglio 2024

[ListQueueTags azione supportata nella policy SQSRead OnlyAccess gestita da Amazon](#)

La policy `SQSRead OnlyAccess` gestita di Amazon supporta `ListQueueTags` il recupero di tutti i tag associati a una coda Amazon SQS specificata.

2 maggio 2024

[AWS Protocollo JSON](#)

Effettua richieste API utilizzando il protocollo AWS JSON.

27 luglio 2023

<u>Nuova sezione che descrive le politiche AWS gestite per Amazon SQS e gli aggiornamenti a queste politiche</u>	Amazon SQS ha aggiunto una nuova azione che consente di elencare le attività di spostamento dei messaggi più recenti (fino a 10) in una coda di origine specifica. Questa operazione è associata all'operazione API <code>ListMessageMoveTasks</code> .	7 giugno 2023
<u>Utilizzo del redrive della coda Dead-Letter APIs</u>	Configura i redrive delle code dead-letter utilizzando Amazon SQS. APIs	7 giugno 2023
<u>ABAC per Amazon SQS</u>	Controllo degli accessi basato su attributi (ABAC) con tag di coda per autorizzazioni di accesso flessibili e scalabili.	10 novembre 2022
<u>Aumenta il limite di velocità di trasmissione effettiva elevata FIFO</u>	Aumento delle quote predefinite per la modalità FIFO ad alta velocità di trasmissione effettiva nelle regioni commerciali, oltre all'ottimizzazione dei documenti FIFO ad alta velocità di trasmissione effettiva.	20 ottobre 2022
<u>La crittografia lato server (SSE) predefinita è disponibile</u>	Crittografia lato server (SSE) con crittografia di proprietà SQS (SSE-SQS) per impostazioni predefinite.	26 settembre 2022

[È disponibile il supporto per la protezione alternativa confusa di Amazon SQS](#)

La protezione aggiuntiva confusa consente di specificare nuove intestazioni nelle richieste, che vengono verificate in base alle condizioni della policy KMS quando si utilizza l'SSE gestito da Amazon SQS.

29 dicembre 2021

[L'SSE gestito è disponibile](#)

SSE gestito da Amazon SQS (SSE-SQS) è una crittografia gestita lato server che utilizza chiavi di crittografia di proprietà di Amazon SQS per proteggere i dati sensibili inviati tramite code di messaggi.

23 novembre 2021

[È disponibile il redrive per le code DLQ](#)

Amazon SQS supporta il [redrive delle code DLQ](#) per le code standard.

10 novembre 2021

[È disponibile una velocità di trasmissione effettiva elevata per i messaggi nelle code FIFO](#)

La velocità di trasmissione effettiva elevata per le code FIFO di Amazon SQS offre un numero maggiore di transazioni al secondo (TPS) per i messaggi nelle code FIFO. Per informazioni sulle quote di velocità di trasmissione effettiva, consulta [Quote relative ai messaggi](#).

27 maggio 2021

[È disponibile una velocità di trasmissione effettiva elevata per i messaggi nelle code FIFO nella versione in anteprima](#)

La velocità di trasmissione effettiva elevata per le code FIFO Amazon SQS è in versione di anteprima ed è soggetta a modifiche. Questa funzionalità offre un numero maggiore di transazioni al secondo (TPS) per i messaggi nelle code FIFO. Per informazioni sulle quote di velocità di trasmissione effettiva, consulta [Quote relative ai messaggi](#).

17 dicembre 2020

[Nuovo design della console Amazon SQS](#)

Per semplificare i flussi di lavoro di sviluppo e produzione, la console Amazon SQS offre [una nuova esperienza utente](#).

8 luglio 2020

[Amazon SQS supporta l'impaginazione per ListQueues e listDeadLetter SourceQueues](#)

È possibile specificare il numero massimo di risultati da restituire da un [listDeadLetterSourceQueuesListQueues](#) o da una richiesta.

22 giugno 2020

[Amazon SQS supporta i parametri CloudWatch Amazon di 1 minuto in AWS tutte le regioni, ad eccezione AWS GovCloud delle regioni \(Stati Uniti\)](#)

Il CloudWatch parametro di un minuto per Amazon SQS è disponibile in tutte le regioni, ad eccezione delle regioni AWS GovCloud (US)

09 gennaio 2020

[Amazon SQS supporta parametri da 1 minuto CloudWatch](#)

La CloudWatch metrica di un minuto per Amazon SQS è attualmente disponibile solo nelle seguenti regioni: Stati Uniti orientali (Ohio), Europa (Irlanda), Europa (Stoccolma) e Asia Pacifico (Tokyo).

25 novembre 2019

[AWS Lambda sono disponibili i trigger per le code FIFO di Amazon SQS](#)

È possibile configurare i messaggi in ingresso in una coda FIFO per attivare una funzione Lambda.

25 novembre 2019

[La crittografia lato server \(SSE\) per Amazon SQS è disponibile nelle regioni della Cina](#)

SSE per Amazon SQS è disponibile nelle regioni della Cina.

13 novembre 2019

[Le code FIFO sono disponibili nella Regione Medio Oriente \(Bahrein\)](#)

Le code FIFO sono disponibili nella Regione Medio Oriente (Bahrein).

10 ottobre 2019

[Gli endpoint Amazon Virtual Private Cloud \(Amazon VPC\) per Amazon SQS sono disponibili nelle regioni AWS GovCloud \(Stati Uniti orientali\) e \(Stati Uniti occidentali\) AWS GovCloud](#)

Puoi inviare messaggi alle code Amazon SQS da Amazon VPC nelle regioni AWS GovCloud (Stati Uniti orientali) e (Stati Uniti occidentali). AWS GovCloud

5 settembre 2019

[Amazon SQS consente la risoluzione dei problemi delle code AWS X-Ray utilizzando gli attributi del sistema di messaggistica](#)

È possibile risolvere i problemi dei messaggi che passano attraverso le code Amazon SQS utilizzando X-Ray. Questa versione aggiunge il parametro di richiesta `MessageSystemAttribute` alle operazioni API `SendMessageBatch` e `SendMessage` (consentendo di inviare intestazioni della traccia X-Ray tramite Amazon SQS), l'attributo `AWSTraceHeader` all'operazione API [`ReceiveMessage`](#) e il tipo di dati `MessageSystemAttributeValue`.

28 agosto 2019

[È possibile contrassegnare con tag le code Amazon SQS al momento della creazione](#)

Puoi utilizzare una singola chiamata API Amazon SQS, una funzione AWS SDK o un comando AWS Command Line Interface (AWS CLI) per creare contemporaneamente una coda e specificarne i tag. Inoltre, Amazon SQS supporta le chiavi `aws:TagKeys` and `aws:RequestTag` AWS Identity and Access Management (IAM).

22 agosto 2019

[Temporary Queue Client per Amazon SQS è ora disponibile](#)

Le code temporanee consentono di risparmiare tempo di sviluppo e costi di implementazione quando si utilizzano schemi di messaggi comuni come request-response. È possibile utilizzare il [Temporary Queue Client](#) per creare code temporanee ad alta velocità, convenienti e gestite dalle applicazioni.

25 luglio 2019

[SSE per Amazon SQS è disponibile nella regione \(Stati Uniti AWS GovCloud orientali\)](#)

La crittografia lato server (SSE) per Amazon SQS è disponibile nella regione (Stati Uniti orientali). AWS GovCloud

20 giugno 2019

[Le code FIFO sono disponibili nelle regioni Asia Pacifico \(Hong Kong\), Cina \(Pechino\), \(Stati Uniti orientali\) e AWS GovCloud AWS GovCloud \(Stati Uniti occidentali\)](#)

Le code FIFO sono disponibili nelle regioni di Asia Pacifico (Hong Kong), Cina (Pechino), AWS GovCloud (Stati Uniti orientali) e AWS GovCloud (Stati Uniti occidentali).

15 maggio 2019

[Le policy endpoint Amazon VPC sono disponibili per Amazon SQS](#)

Puoi creare policy di endpoint Amazon VPC per Amazon SQS.

4 aprile 2019

[Le code FIFO sono disponibili nelle regioni Europa \(Stoccolma\) e Cina \(Ningxia\)](#)

Le code FIFO sono disponibili nelle regioni Europa (Stoccolma) e Cina (Ningxia).

14 marzo 2019

[Le code FIFO sono disponibili in tutte le regioni in cui è disponibile Amazon SQS](#)

Le code FIFO sono disponibili nelle regioni Stati Uniti orientali (Virginia settentrionale), Stati Uniti orientali (Ohio), Stati Uniti occidentali (California settentrionale), Stati Uniti occidentali (Oregon), Asia Pacifico (Mumbai), Asia Pacifico (Seoul), Asia Pacifico (Singapore), Asia Pacifico (Sydney), Asia Pacifico (Tokyo), Canada (Centrale), Europa (Francoforte), Europa (Irlanda), Europa (Londra), Europa (Parigi) e Sud America (San Paolo).

7 febbraio 2019

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.