aws

Panduan Developer

Amazon Simple Queue Service



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

Amazon Simple Queue Service: Panduan Developer

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

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan antara para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Apa itu Amazon SQS? 1
Manfaat menggunakan Amazon SQS 1
Arsitektur dasar 1
Antrian terdistribusi 2
Siklus hidup pesan 2
Perbedaan antara Amazon SQS, Amazon MQ, dan Amazon SNS 4
Memulai 6
Penyiapan 6
Mendaftar untuk Akun AWS 6
Bersiaplah untuk menggunakan kode contoh 6
Langkah selanjutnya 7
Memahami konsol Amazon SQS 7
Jenis antrian 8
Menerapkan sistem permintaan-respons di Amazon SQS 12
Membuat antrian standar 12
Membuat antrian 12
Mengirim pesan menggunakan antrian standar 14
Membuat antrian FIFO 15
Membuat antrean 15
Mengirim pesan menggunakan antrian FIFO 18
Tugas umum 18
Mengelola antrian 20
Mengedit antrian 20
Menerima dan menghapus pesan 20
Mengonfirmasi antrian kosong 22
Menghapus antrian 23
Membersihkan antrian 24
Antrian standar 26
Pengiriman Amazon at-least-once SQS 27
Antrian dan pengidentifikasi pesan 27
Pengidentifikasi untuk antrian standar 27
Antrian FIFO 29
Istilah kunci antrian FIFO 30
Logika pengiriman FIFO 31

Mengirim pesan ... 31
Menerima pesan ... 32
Mencoba lagi beberapa kali ... 33
Catatan tambahan tentang perilaku FIFO ... 34
Contoh untuk pemahaman yang lebih baik ... 34
Persis-sekali diproses ... 35
Pindah dari antrian standar ke antrian FIFO ... 35
Antrian FIFO dan perilaku konkurensi Lambda ... 37
Pengelompokan pesan antrian FIFO ... 37
Konkurensi Lambda dengan antrian FIFO ... 38
Contoh kasus penggunaan ... 38
Throughput tinggi untuk antrian FIFO ... 38
Kasus penggunaan ... 39
Partisi dan distribusi data ... 40
Mengaktifkan throughput tinggi untuk antrian FIFO ... 42
Antrian dan pengidentifikasi pesan ... 43
Pengidentifikasi untuk antrian FIFO ... 27
Pengidentifikasi tambahan untuk antrian FIFO ... 45
Kuota ... 46
Kuota antrian FIFO ... 46
Kuota Amazon SQS ... 46
Kuota antrian standar ... 48
Kuota pesan ... 49
Kuota kebijakan ... 56
Fitur dan kemampuan ... 57
Dead-letter antrian ... 57
Menggunakan kebijakan untuk antrian surat mati ... 58
Memahami periode retensi pesan untuk antrian surat mati ... 58
Mengonfigurasi antrean surat mati ... 59
Mengkonfigurasi redrive antrian huruf mati ... 59
CloudTrail Persyaratan pembaruan dan izin ... 70
Membuat alarm untuk antrian huruf mati menggunakan Amazon CloudWatch ... 74
Metadata pesan untuk Amazon SQS ... 74
Atribut pesan ... 74
Atribut sistem pesan ... 78
Sumber daya yang diperlukan untuk memproses pesan ... 78

Daftar antrian pagination ... 79
Tag alokasi biaya ... 80
Polling pendek dan panjang ... 81
Mengkonsumsi pesan menggunakan polling pendek ... 81
Mengkonsumsi pesan menggunakan polling panjang ... 82
Perbedaan antara polling panjang dan pendek ... 83
Batas waktu visibilitas ... 83
Kasus penggunaan batas waktu visibilitas ... 84
Mengatur dan menyesuaikan batas waktu visibilitas ... 84
Dalam pesan penerbangan dan kuota ... 85
Memahami batas waktu visibilitas dalam antrian standar dan FIFO ... 85
Penanganan kegagalan ... 86
Mengubah dan mengakhiri batas waktu visibilitas ... 86
Praktik terbaik ... 87
Antrian yang adil ... 87
Cara menggunakan antrian yang adil ... 89
Kapan harus menggunakan antrian yang adil ... 90
Metrik antrian CloudWatch yang adil ... 90
Bagaimana antrian yang adil bekerja ... 91
Antrian tunda ... 93
Antrian sementara ... 95
Antrian virtual ... 95
Request-response pola pesan (antrian virtual) ... 96
Contoh skenario: Memproses permintaan login ... 97
Membersihkan antrian ... 99
Pengatur waktu pesan ... 100
Mengakses pipa EventBridge ... 101
Mengelola pesan besar ... 102
Menggunakan Extended Client Library untuk Java ... 102
Menggunakan Extended Client Library untuk Python ... 109
Mengkonfigurasi Amazon SQS ... 112
ABAC untuk Amazon SQS ... 112
Apa itu ABAC? ... 112
Mengapa saya harus menggunakan ABAC di Amazon SQS? ... 113
Penandaan untuk kontrol akses ... 114
Membuat pengguna IAM dan antrian Amazon SQS ... 114

Menguji kontrol akses berbasis atribut 117
Mengkonfigurasi parameter antrian 118
Mengonfigurasi kebijakan akses 120
Mengkonfigurasi SSE-SQS untuk antrian 121
Mengkonfigurasi SSE-KMS untuk antrian 122
Mengkonfigurasi tag untuk antrian 124
Berlangganan antrean ke topik 125
Langganan lintas akun 126
Langganan lintas wilayah 126
Mengkonfigurasi pemicu Lambda 127
Prasyarat 128
Mengotomatiskan notifikasi menggunakan EventBridge 129
Atribut pesan 129
Praktik terbaik 131
Penanganan kesalahan dan pesan bermasalah 131
Menangani kesalahan permintaan di Amazon SQS 131
Menangkap pesan bermasalah di Amazon SQS 132
Mengatur retensi antrian huruf mati di Amazon SQS 132
Deduplikasi dan pengelompokan pesan 132
Menghindari pemrosesan pesan yang tidak konsisten di Amazon SQS 133
Menggunakan ID deduplikasi pesan 133
Menggunakan ID grup pesan 136
Menggunakan ID percobaan permintaan terima 137
Pemrosesan dan pengaturan waktu pesan 138
Memproses pesan tepat waktu di Amazon SQS 138
Mengatur polling panjang di Amazon SQS 139
Menggunakan mode polling yang sesuai di Amazon SQS 140
Contoh SDK Java 141
Menggunakan enkripsi sisi server 141
Menambahkan SSE ke antrian yang ada 141
Menonaktifkan SSE untuk antrian 142
Membuat antrian dengan SSE 143
Mengambil atribut SSE 144
Mengonfigurasi tag 144
Mencantumkan tanda 144
Menambahkan atau memperbarui tag 145

Menghapus tanda ... 145
Mengirim atribut pesan ... 146
Mendefinisikan atribut ... 146
Mengirim pesan dengan atribut ... 148
Menggunakan API ... 149
Membuat permintaan API kueri menggunakan protokol AWS JSON ... 150
Membangun titik akhir ... 150
Membuat permintaan POST ... 151
Menafsirkan tanggapan Amazon SQS JSON API ... 152
Protokol JSON Amazon SQS AWS FAQs ... 153
Membuat permintaan API kueri menggunakan protokol AWS kueri ... 156
Membangun titik akhir ... 157
Membuat permintaan GET ... 157
Membuat permintaan POST ... 151
Menafsirkan tanggapan Amazon SQS XMLAPI ... 159
Mengotentikasi permintaan ... 160
Proses otentikasi dasar dengan HMAC-SHA ... 161
Bagian 1: Permintaan dari pengguna ... 162
Bagian 2: Tanggapan dari AWS ... 163
Tindakan Batch ... 164
Tindakan pesan batching ... 165
Mengaktifkan buffering sisi klien dan batching permintaan dengan Amazon SQS ... 166
Meningkatkan throughput menggunakan penskalaan horizontal dan batching aksi dengan Amazon SQS ... 178
Bekerja dengan AWS SDK ... 191
Menggunakan JMS ... 193
Prasyarat ... 193
Menggunakan Java Messaging Library ... 194
Membuat koneksi JMS ... 195
Membuat antrian Amazon SQS ... 196
Mengirim pesan secara sinkron ... 197
Menerima pesan secara sinkron ... 198
Menerima pesan secara asinkron ... 199
Menggunakan mode pengakuan klien ... 201
Menggunakan mode pengakuan tidak berurutan ... 202
Menggunakan Klien JMS dengan klien Amazon SQS lainnya ... 202

Contoh Java yang berfungsi untuk menggunakan JMS dengan antrian standar 204
ExampleConfiguration.jawa 204
TextMessageSender.jawa 207
SyncMessageReceiver.jawa 208
AsyncMessageReceiver.jawa 210
SyncMessageReceiverClientAcknowledge.jawa 212
SyncMessageReceiverUnorderedAcknowledge.jawa 216
SpringExampleConfiguration.xml.xl 219
SpringExample.jawa 221
ExampleCommon.jawa 223
Implementasi JMS 1.1 yang didukung 225
Antarmuka umum yang didukung 225
Jenis pesan yang didukung 225
Mode pengakuan pesan yang didukung 226
Header yang ditentukan JMS dan properti yang dicadangkan 226
Tutorial 228
Membuat antrian Amazon SQS menggunakan CloudFormation 228
Mengirim pesan dari VPC 230
Langkah 1: Buat pasangan kunci Amazon EC2 231
Langkah 2: Buat AWS sumber daya 231
Langkah 3: Konfirmasikan bahwa instans EC2 Anda tidak dapat diakses publik 232
Langkah 4: Buat titik akhir Amazon VPC untuk Amazon SQS 233
Langkah 5: Kirim pesan ke antrian Amazon SQS Anda 234
Contoh kode 236
Hal-hal mendasar 237
Halo Amazon SQS 238
Tindakan 250
Skenario 423
Buat aplikasi perpesanan 424
Membuat aplikasi messenger 425
Membuat aplikasi penjelajah Amazon Textract 426
Buat dan publikasikan ke topik FIFO 428
Mendeteksi orang dan objek dalam video 440
Kelola pesan besar menggunakan S3 441
Memproses pemberitahuan acara S3 445
Publikasikan pesan ke antrian 449

Mengirim dan menerima batch pesan 585
Menggunakan AWS Message Processing Framework untuk.NET dengan Amazon SQS 616
Gunakan Amazon SQS Java Messaging Library untuk bekerja dengan antarmuka JMS 617
Bekerja dengan tag antrian 639
Contoh nirserver 643
Memanggil fungsi Lambda dari pemicu Amazon SQS 643
Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS 652
Pemecahan Masalah 662
Kesalahan akses ditolak 662
Kebijakan antrian Amazon SQS dan kebijakan IAM 663
AWS Key Management Service (AWS KMS) izin 664
Kebijakan titik akhir VPC 665
Kebijakan kontrol layanan organisasi 666
Kesalahan API 666
QueueDoesNotExist kesalahan 666
InvalidAttributeValue kesalahan 667
ReceiptHandle kesalahan 667
Masalah redrive DLQ dan DLQ 668
Masalah DLQ 668
Masalah DLQ-redrive 670
Masalah pelambatan FIFO 672
Pesan tidak dikembalikan untuk panggilan ReceiveMessage API 672
Antrian kosong 672
Dalam batas penerbangan tercapai 673
Penundaan pesan 673
Pesan dalam penerbangan 673
Metode polling 673
Kesalahan jaringan 674
ETIMEOUT error 674
UnknownHostException error 675
Mengatasi masalah antrian menggunakan X-Ray 676
Keamanan 677
Perlindungan data 677
Enkripsi data 678
Privasi lalu lintas antarjaringan 690
Menggunakan titik akhir dual-stack untuk konektivitas 693

Manajemen identitas dan akses 693
Audiens 693
Mengautentikasi dengan identitas 693
Mengelola akses menggunakan kebijakan 695
Gambaran umum 697
Cara kerja Amazon Simple Queue Service dengan IAM 704
AWS kebijakan yang dikelola 710
Pemecahan masalah 713
Menggunakan kebijakan 715
Pencatatan log dan pemantauan 763
Mencatat panggilan API 766
Memantau antrian 770
Validasi kepatuhan 791
Ketahanan 792
Antrian terdistribusi 792
Keamanan infrastruktur 793
Praktik terbaik 794
Pastikan antrian tidak dapat diakses publik 794
Menerapkan akses hak istimewa yang paling rendah 794
Gunakan peran IAM untuk aplikasi dan AWS layanan yang memerlukan akses Amazon SQS 795
Menerapkan enkripsi sisi server 795
Menegakkan enkripsi data saat transit 796
Pertimbangkan untuk menggunakan titik akhir VPC untuk mengakses Amazon SQS 796
Sumber daya terkait 797
Riwayat dokumentasi 798
..... dcccvi

Apa itu Amazon Simple Queue Service?

Amazon Simple Queue Service (Amazon SQS) menawarkan antrian yang aman, tahan lama, dan tersedia yang memungkinkan Anda mengintegrasikan dan memisahkan sistem dan komponen perangkat lunak terdistribusi. Amazon SQS menawarkan konstruksi umum seperti antrian surat mati dan tag alokasi biaya. ??? Ini menyediakan API layanan web generik yang dapat Anda akses menggunakan bahasa pemrograman apa pun yang AWS didukung SDK.

Manfaat menggunakan Amazon SQS

- Keamanan — Anda mengontrol siapa yang dapat mengirim pesan ke dan menerima pesan dari antrian Amazon SQS. Anda dapat memilih untuk mengirimkan data sensitif dengan melindungi isi pesan dalam antrian dengan menggunakan enkripsi sisi server terkelola (SSE) default Amazon SQS, atau dengan menggunakan kunci SSE khusus yang dikelola dalam (). AWS Key Management Service AWS KMS
- Daya Tahan — Untuk keamanan pesan Anda, Amazon SQS menyimpannya di beberapa server. Antrian standar mendukung pengiriman pesan setidaknya sekali, dan antrian FIFO mendukung pemrosesan pesan tepat sekali dan mode throughput tinggi. ???
- Ketersediaan — Amazon SQS menggunakan infrastruktur redundan untuk menyediakan akses yang sangat bersamaan ke pesan dan ketersediaan tinggi untuk memproduksi dan mengkonsumsi pesan.
- Skalabilitas — Amazon SQS dapat memproses setiap permintaan buffer secara independen, menskalakan secara transparan untuk menangani peningkatan atau lonjakan beban apa pun tanpa instruksi penyediaan apa pun.
- Keandalan — Amazon SQS mengunci pesan Anda selama pemrosesan, sehingga beberapa produsen dapat mengirim dan beberapa konsumen dapat menerima pesan secara bersamaan.
- Kustomisasi — Antrian Anda tidak harus sama persis — misalnya, Anda dapat mengatur penundaan default pada antrian. Anda dapat menyimpan konten pesan yang lebih besar dari 1 MiB menggunakan Amazon Simple Storage Service (Amazon S3) atau Amazon DynamoDB, dengan Amazon SQS memegang penunjuk ke objek Amazon S3, atau Anda dapat membagi pesan besar menjadi pesan yang lebih kecil.

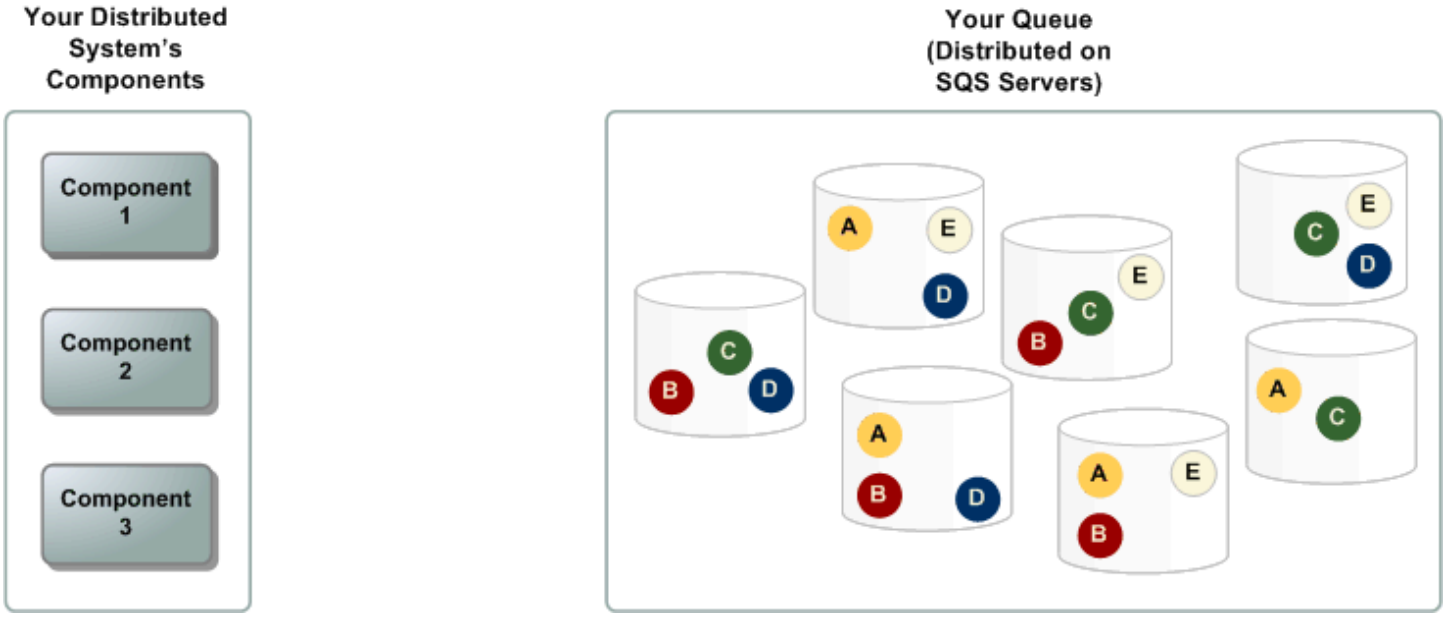
Arsitektur dasar Amazon SQS

Bagian ini menjelaskan komponen sistem pesan terdistribusi dan menjelaskan siklus hidup pesan Amazon SQS.

Antrian terdistribusi

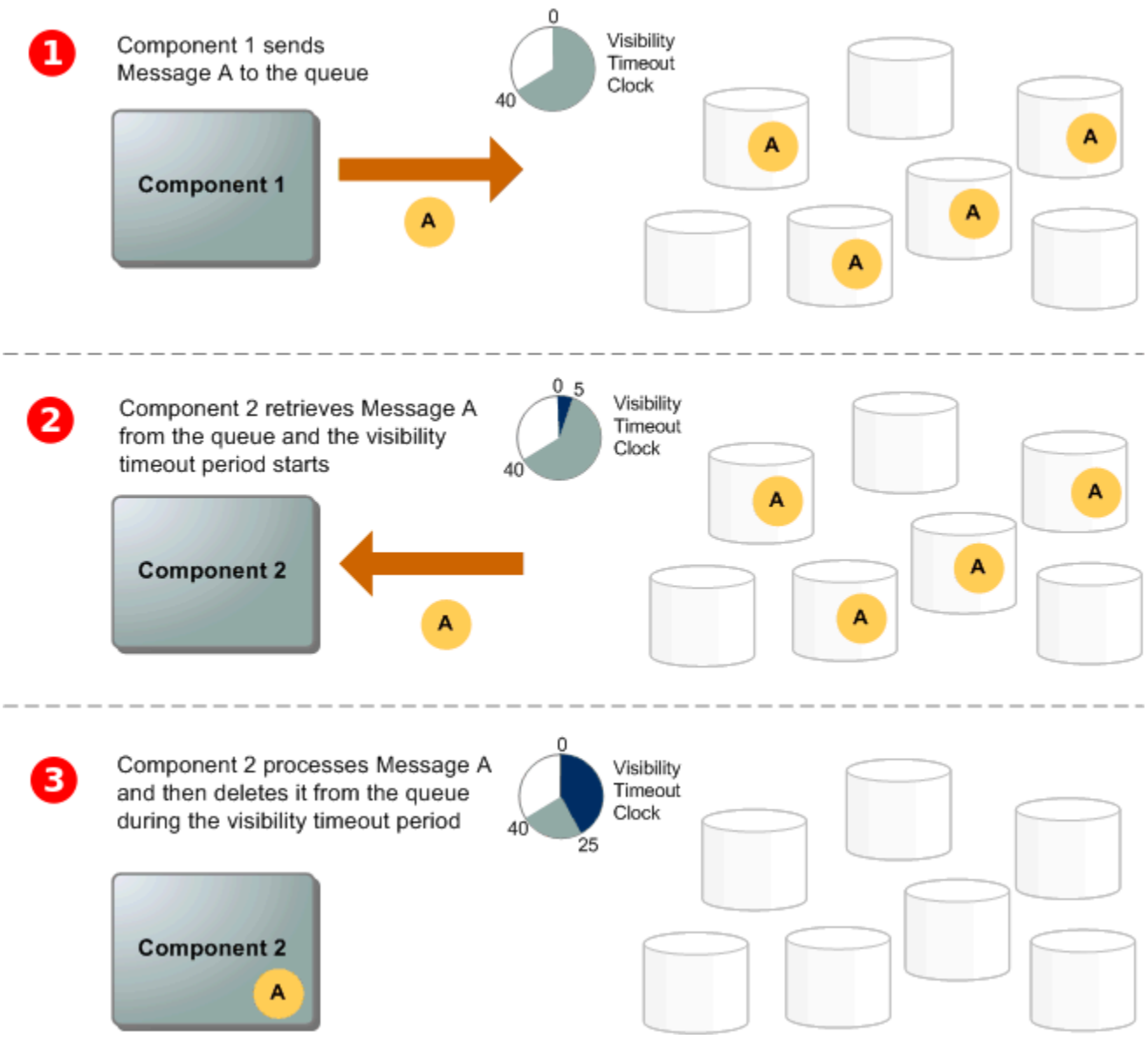
Ada tiga bagian utama dalam sistem pesan terdistribusi: komponen sistem terdistribusi Anda, antrian Anda (didistribusikan di server Amazon SQS), dan pesan dalam antri an.

Dalam skenario berikut, sistem Anda memiliki beberapa produsen (komponen yang mengirim pesan ke antrian) dan konsumen (komponen yang menerima pesan dari antrian). Antrian (yang menyimpan pesan A hingga E) secara redundansi menyimpan pesan di beberapa server Amazon SQS.



Siklus hidup pesan

Skenario berikut menjelaskan siklus hidup pesan Amazon SQS dalam antrian, dari pembuatan hingga penghapusan.



1 Produs (Komponen 1) mengirimkan pesan A ke antrian, dan pesan tersebut didistribusikan ke seluruh server Amazon SQS secara redundansi.

2 Ketika konsumen (Komponen 2) siap untuk memproses pesan, ia mengkonsumsi pesan dari antrian, dan pesan A dikembalikan. Saat pesan A sedang diproses, pesan tersebut tetap berada dalam antrian dan tidak dikembalikan ke permintaan penerimaan berikutnya selama batas waktu visibilitas.

3 Konsu

(Komponen 2) menghapus pesan A dari antrian untuk mencegah pesan diterima dan diproses lagi ketika batas waktu visibilitas berakhir.

> **Note**
>
> Amazon SQS secara otomatis menghapus pesan yang telah berada dalam antrian lebih dari periode retensi pesan maksimum. Periode penyimpanan pesan default adalah 4 hari. Namun, Anda dapat mengatur periode penyimpanan pesan ke nilai dari 60 detik hingga 1.209.600 detik (14 hari) menggunakan tindakan tersebut. `SetQueueAttributes`

Perbedaan antara Amazon SQS, Amazon MQ, dan Amazon SNS

Amazon SQS, Amazon SNS, dan Amazon MQ menawarkan layanan pesan terkelola yang sangat skalabel dan mudah digunakan, masing-masing dirancang untuk peran tertentu dalam sistem terdistribusi. Berikut adalah ikhtisar yang disempurnakan tentang perbedaan antara layanan ini:

Amazon SQS memis ahkan dan menskalakan sistem dan komponen perangkat lunak terdistribusi sebagai layanan antrian. Ini memproses pesan melalui satu pelanggan biasanya, ideal untuk alur kerja di mana pencegahan pesanan dan kerugian sangat penting. Untuk distribusi yang lebih luas, mengintegrasikan Amazon SQS dengan Amazon SNS memungkinkan pola pesan fanout, secara efektif mendorong pesan ke beberapa pelanggan sekaligus.

Amazon SNS memungkinkan penerbit mengirim pesan ke beberapa pelanggan melalui topik, yang berfungsi sebagai saluran komunikasi. Pelanggan menerima pesan yang dipublikasikan menggunakan jenis titik akhir yang didukung, seperti Amazon Data Firehose, Amazon SQS, Lambda, HTTP, email, notifikasi push seluler, dan pesan teks seluler (SMS). Layanan ini sangat ideal untuk skenario yang membutuhkan pemberitahuan segera, seperti keterlibatan pengguna real-time atau sistem alarm. Untuk mencegah hilangnya pesan saat pelanggan sedang offline, mengintegrasikan Amazon SNS dengan pesan antrian Amazon SQS memastikan pengiriman yang konsisten.

Amazon MQ paling cocok dengan perusahaan yang ingin bermigrasi dari broker pesan tradisional, mendukung protokol pesan standar seperti AMQP dan MQTT, bersama dengan Apache ActiveMQ dan RabbitMQ. https://www.rabbitmq.com/ Ini menawarkan kompatibilitas dengan sistem lama yang membutuhkan pesan yang stabil dan andal tanpa konfigurasi ulang yang signifikan.

Bagan berikut memberikan gambaran umum dari setiap jenis sumber daya layanan:

Tipe sumber daya	Amazon SNS	Amazon SQS	Amazon MQ
Sinkron	Tidak	Tidak	Ya
Asinkron	Ya	Ya	Ya
Antrean	Tidak	Ya	Ya
Publisher-subscriber pesan	Ya	Tidak	Ya
Broker pesan	Tidak	Tidak	Ya

Baik Amazon SQS dan Amazon SNS direkomendasikan untuk aplikasi baru yang dapat memperoleh manfaat dari skalabilitas yang hampir tidak terbatas dan API sederhana. Mereka umumnya menawarkan solusi yang lebih hemat biaya untuk aplikasi volume tinggi dengan harga pay-as-you-go mereka. Kami merekomendasikan Amazon MQ untuk memigrasikan aplikasi dari broker pesan yang ada yang mengandalkan kompatibilitas dengan API seperti JMS atau protokol seperti Advanced Message Queuing Protocol (AMQP), MQTT OpenWire, dan Simple Text Oriented Message Protocol (STOMP).

Mulai menggunakan Amazon SQS

Topik ini memandu Anda menggunakan konsol Amazon SQS untuk membuat dan mengelola antrian standar dan antrian FIFO. Anda akan mempelajari cara menavigasi konsol, melihat atribut antrian, dan membedakan antara jenis antrian. Tugas utama termasuk mengirim, menerima, dan mengonfigurasi pesan, menyesuaikan parameter seperti batas waktu visibilitas dan penyimpanan pesan, dan mengelola akses antrian melalui kebijakan.

Topik

- Menyiapkan Amazon SQS
- Memahami konsol Amazon SQS
- Jenis antrian Amazon SQS
- Membuat antrian standar Amazon SQS dan mengirim pesan
- Membuat antrian Amazon SQS FIFO dan mengirim pesan
- Tugas umum untuk memulai dengan Amazon SQS

Menyiapkan Amazon SQS

Sebelum Anda dapat menggunakan Amazon SQS untuk pertama kalinya, Anda harus menyelesaikan langkah-langkah berikut:

Mendaftar untuk Akun AWS

Untuk memulai AWS, Anda membutuhkan Akun AWS. Untuk informasi tentang membuat Akun AWS, lihat Memulai dengan Akun AWS di Panduan AWS Account Management Referensi.

Bersiaplah untuk menggunakan kode contoh

Panduan ini mencakup contoh yang menggunakan AWS SDK for Java. Untuk menjalankan kode contoh, ikuti petunjuk penyiapan di Memulai dengan AWS SDK for Java 2.0.

Anda dapat mengembangkan AWS aplikasi dalam bahasa pemrograman lain, seperti GoJavaScript, Python dan Ruby. Untuk informasi selengkapnya, lihat Alat untuk Dibangun AWS.

> **Note**
>
> Anda dapat menjelajahi Amazon SQS tanpa menulis kode dengan alat seperti AWS Command Line Interface (AWS CLI) atau Windows. PowerShell Anda dapat menemukan AWS CLI contoh di bagian Amazon SQS dari AWS CLI Command Reference. Anda dapat menemukan PowerShell contoh Windows di bagian Layanan Antrian Sederhana Amazon dari Referensi Alat AWS untuk PowerShell Cmdlet.

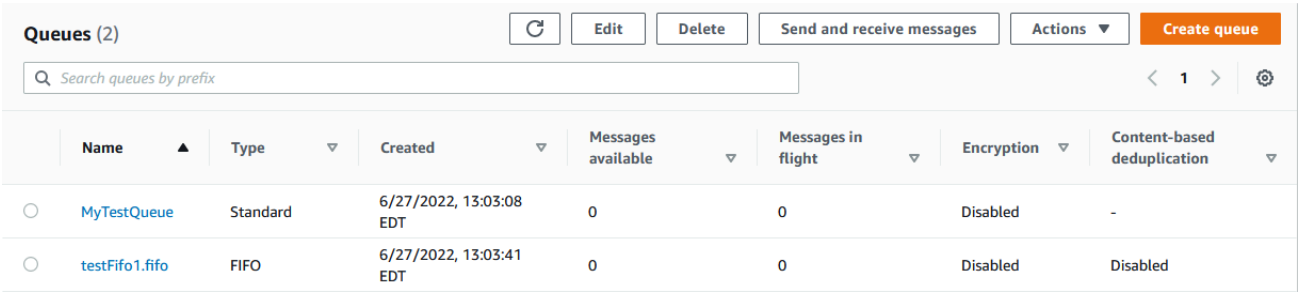
Langkah selanjutnya

Anda sekarang siap untuk Memulai mengelola antrian dan pesan Amazon SQS menggunakan file. Konsol Manajemen AWS

Memahami konsol Amazon SQS

Saat Anda membuka konsol Amazon SQS, pilih Antrian dari panel navigasi. Halaman Antrian memberikan informasi tentang semua antrian Anda di wilayah aktif.

Setiap entri antrian memberikan informasi penting tentang antrian, termasuk tipe dan atribut kuncinya. Antrian standar, dioptimalkan untuk throughput maksimum dan pemesanan pesan upaya terbaik, dibedakan dari antrian First-In-First-Out (FIFO), yang memprioritaskan pemesanan pesan dan keunikan untuk aplikasi yang membutuhkan pengurutan pesan yang ketat.

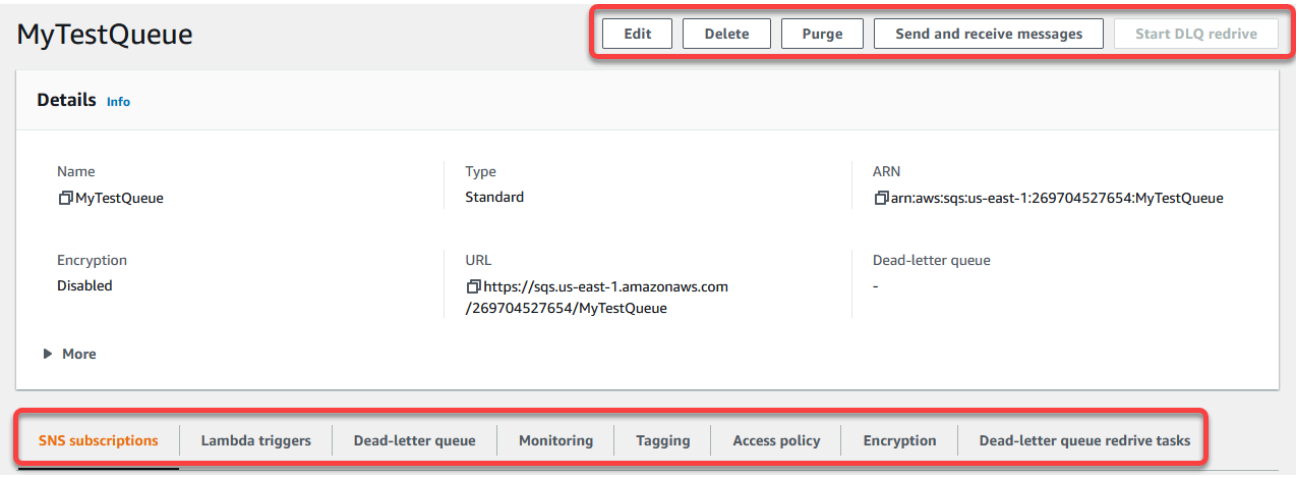


Elemen dan tindakan interaktif

Dari halaman Antrian, Anda memiliki beberapa opsi untuk mengelola antrian Anda:

1. Tindakan Cepat — Berdekatan dengan setiap nama antrian, menu tarik-turun menawarkan akses cepat ke tindakan umum seperti mengirim pesan, melihat atau menghapus pesan, mengonfigurasi pemicu, dan menghapus antrian itu sendiri.

2. Tampilan dan Konfigurasi Terperinci — Mengklik nama antrian membuka halaman Detailnya, di mana Anda dapat mempelajari lebih dalam pengaturan dan konfigurasi antrian. Di sini, Anda dapat menyesuaikan parameter seperti periode penyimpanan pesan, batas waktu visibilitas, dan ukuran pesan maksimum untuk menyesuaikan antrian dengan persyaratan aplikasi Anda.



Pemilihan wilayah dan tag sumber daya

Pastikan Anda berada di tempat yang benar Wilayah AWS untuk mengakses dan mengelola antrian Anda secara efektif. Selain itu, pertimbangkan untuk menggunakan tag sumber daya untuk mengatur dan mengkategorikan antrian Anda, memungkinkan manajemen sumber daya yang lebih baik, alokasi biaya, dan kontrol akses dalam lingkungan bersama Anda. AWS

Dengan memanfaatkan fitur dan fungsi yang ditawarkan dalam konsol Amazon SQS, Anda dapat mengelola infrastruktur pesan secara efisien, mengoptimalkan kinerja antrian, dan memastikan pengiriman pesan yang andal untuk aplikasi Anda.

Jenis antrian Amazon SQS

Amazon SQS mendukung dua jenis antrian: antrian standar dan antrian FIFO. Gunakan tabel berikut untuk menentukan antrian mana yang paling sesuai dengan kebutuhan Anda.

Antrian standar	Antrian FIFO
Through put tak terbatas — Antrian standar mendukung jumlah panggilan API yang sangat tinggi dan hampir tidak terbatas per detik,	Throughput tinggi — Saat Anda menggunakan batching, antrian FIFO memproses hingga 3.000 pesan per detik per metode API

Antrian standar	Antrian FIFO
per tindakan (`SendMessage`, `ReceiveMessage`, atau `DeleteMessage`). Throughput tinggi ini membuatnya ideal untuk kasus penggunaan yang memerlukan pemrosesan pesan dalam jumlah besar dengan cepat, seperti streaming data real-time atau aplikasi skala besar. Sementara antrian standar berskala secara otomatis sesuai permintaan, penting untuk memantau pola penggunaan untuk memastikan kinerja yang optimal, terutama di wilayah dengan beban kerja yang lebih tinggi. At-least-once pengiriman - Dijamin pengiriman setidaknya sekali, yang berarti bahwa setiap pesan dikirimkan setidaknya sekali, tetapi dalam beberapa kasus, pesan dapat dikirimkan lebih dari sekali karena percobaan ulang atau penundaan jaringan. Anda harus merancang aplikasi Anda untuk menangani pesan duplikat potensial dengan menggunakan operasi idempotent, yang memastikan bahwa memproses pesan yang sama beberapa kali tidak akan mempengaruhi status sistem. Best-effort pemesanan — Menyediakan pemesanan dengan upaya terbaik, yang berarti bahwa meskipun Amazon SQS mencoba mengirimkan pesan sesuai urutan pengiriman, itu tidak menjamin hal ini. Dalam beberapa kasus, pesan mungkin datang rusak, terutama dalam kondisi throughput tinggi atau pemulihan kegagalan. Untuk aplikasi di mana urutan pemrosesan pesan sangat penting, Anda harus menangani logika penataan ulang dalam	(`SendMessageBatch`, `ReceiveMessage`, atau). `DeleteMessageBatch` Throughput ini bergantung pada 300 panggilan API per detik, dengan setiap panggilan API menangani batch 10 pesan.Dengan mengaktifkan mode throughput tinggi, Anda dapat menskalakan hingga 30.000 transaksi per detik (TPS) dengan pemesanan santai dalam grup pesan. Tanpa batching, antrian FIFO mendukung hingga 300 panggilan API per detik per metode API (`SendMessage` ,`ReceiveMessage` , or). `DeleteMessage` Jika Anda membutuhkan lebih banyak throughput, Anda dapat meminta peningkatan kuota melalui Pusat AWS Dukungan. Untuk mengaktifkan mode throughput tinggi, lihatMengaktifkan throughput tinggi untuk antrian FIFO di Amazon SQS. Exactly-once pemrosesan - Antrian FIFO mengirimkan setiap pesan sekali dan menyimpannya tetap tersedia sampai Anda memproses dan menghapusnya. Dengan menggunakan fitur seperti `MessageDeduplicationId` atau deduplikasi berbasis konten, Anda mencegah pesan duplikat, bahkan saat mencoba lagi karena masalah jaringan atau batas waktu. First-in-first-out pengiriman — Antrian FIFO memastikan bahwa Anda menerima pesan dalam urutan pengiriman dalam setiap grup pesan. Dengan mendistribusikan pesan ke beberapa grup, Anda dapat memprosesnya secara paralel sambil tetap mempertahankan urutan dalam setiap grup.

Antrian standar	Antrian FIFO
aplikasi atau menggunakan antrian FIFO untuk jaminan pemesanan yang ketat. Daya tahan dan redundansi — Antrian standar memastikan daya tahan tinggi dengan menyimpan beberapa salinan dari setiap pesan di beberapa Zona Keter AWS sediaan. Ini memastikan bahwa pesan tidak hilang, bahkan jika terjadi kegagalan infrastruktur. Batas waktu visibilitas — Amazon SQS memungkinkan Anda mengonfigurasi batas waktu visibilitas untuk mengontrol berapa lama pesan tetap tersembunyi setelah diterima, memastikan bahwa konsumen lain tidak memproses pesan sampai sepenuhnya ditangani atau batas waktu berakhir.	
	

Antrian standar	Antrian FIFO
Gunakan antrian standar untuk mengirim data antar aplikasi ketika throughput sangat penting, misalnya: • Pisahkan permintaan pengguna langsung dari pekerjaan latar belakang intensif. Izinkan pengguna mengunggah media dengan cepat saat Anda memproses tugas seperti mengubah ukuran atau pengkodean di latar belakang, memastikan waktu respons yang cepat tanpa membebani sistem. • Mengalokasikan tugas ke beberapa node pekerja. Mendistribusikan sejumlah besar permintaan validasi kartu kredit di beberapa node pekerja, dan tangani pesan duplikat dengan operasi idempotent untuk menghindari kesalahan pemrosesan. • Pesan batch untuk pemrosesan di masa mendatang. Antrekan beberapa entri untuk penambahan batch ke database. Karena urutan pesan tidak dijamin, rancang sistem Anda untuk menangani pemrosesan yang tidak sesuai pesanan jika perlu.	Gunakan antrian FIFO untuk mengirim data antar aplikasi ketika urutan peristiwa penting, misalnya: • Pastikan perintah yang dimasukkan pengguna dijalankan dalam urutan yang benar. Ini adalah kasus penggunaan utama untuk antrian FIFO, di mana perintah perintah sangat penting. Misalnya, jika pengguna melakukan urutan tindakan dalam aplikasi, antrian FIFO memastikan tindakan diproses dalam urutan yang sama dengan yang dimasukkan. • Tampilkan harga produk yang benar dengan mengirimkan modifikasi harga dalam urutan yang benar. Antrian FIFO memastikan bahwa beberapa pembaruan pada harga produk tiba dan diproses secara berurutan . Tanpa FIFO, pengurangan harga dapat diproses setelah kenaikan harga, menyebabkan data yang salah ditampilkan. • Mencegah siswa mendaftar di kursus sebelum mendaftar untuk akun. Dengan menggunakan antrian FIFO, Anda memastikan bahwa proses pendaftaran terjadi dalam urutan yang benar. Sistem memproses pendaftaran akun terlebih dahulu dan kemudian pendaftaran kursus, mencegah permintaan pendaftaran dieksekusi sebelum waktunya.

Menerapkan sistem permintaan-respons di Amazon SQS

Saat menerapkan sistem request-response atau remote procedure call (RPC), ingatlah praktik terbaik berikut:

- Buat antrian balasan saat start-up — Alih-alih membuat antrian balasan per pesan, buatlah antrian balasan saat start-up, per produsen. Gunakan atribut pesan ID korelasi untuk memetakan balasan permintaan secara efisien.
- Hindari berbagi antrian balasan di antara produsen — Pastikan setiap produsen memiliki antrian balasannya sendiri. Berbagi antrian balasan dapat mengakibatkan produsen menerima pesan respons yang ditujukan untuk produsen lain.

Untuk informasi selengkapnya tentang menerapkan pola permintaan-respons menggunakan Klien Antrian Sementara, lihat. Request-response pola pesan (antrian virtual)

Membuat antrian standar Amazon SQS dan mengirim pesan

Anda dapat membuat antrian standar dan mengirim pesan menggunakan konsol Amazon SQS. Topik ini juga menekankan praktik terbaik, termasuk menghindari informasi sensitif dalam nama antrian dan memanfaatkan enkripsi sisi server yang dikelola.

Membuat antrian standar menggunakan konsol Amazon SQS

> **Important**
>
> Pada 17 Agustus 2022, enkripsi sisi server default (SSE) diterapkan ke semua antrian Amazon SQS.
> Jangan menambahkan informasi identitas pribadi (PII) atau informasi rahasia atau sensitif lainnya dalam nama antrian. Nama antrian dapat diakses oleh banyak Amazon Web Services, termasuk penagihan dan CloudWatch log. Nama antrian tidak dimaksudkan untuk digunakan untuk data pribadi atau sensitif.

Untuk membuat antrean standar Amazon SQS

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Pilih Buat antrean.

3. Untuk Type, tipe antrian Standar diatur secara default.

> **Note**
>
> Anda tidak dapat mengubah jenis antrian setelah Anda membuat antrian.

4. Masukkan Nama untuk antrian Anda.
5. (Opsional) Konsol menetapkan nilai default untuk parameter konfigurasi antrian. Di bawah Konfigurasi, Anda dapat mengatur nilai baru untuk parameter berikut:

   a. Untuk batas waktu Visibilitas, masukkan durasi dan unit. Kisarannya dari 0 detik hingga 12 jam. Nilai default-nya adalah 30 detik.

   b. Untuk periode penyimpanan Pesan, masukkan durasi dan unit. Kisarannya dari 1 menit hingga 14 hari. Nilai defaultnya adalah 4 hari.

   c. Untuk keterlambatan Pengiriman, masukkan durasi dan unit. Kisarannya dari 0 detik hingga 15 menit. Nilai defaultnya adalah 0 detik.

   d. Untuk Ukuran pesan maksimum, masukkan nilai. Kisarannya dari 1 KiB hingga 1024 KiB. Nilai defaultnya adalah 1024 KiB.

   e. Untuk Menerima waktu tunggu pesan, masukkan nilai. Kisarannya dari 0 hingga 20 detik. Nilai defaultnya adalah 0 detik, yang menetapkan polling pendek. Setiap nilai bukan nol menetapkan polling panjang.

6. (Opsional) Tentukan kebijakan Akses. Kebijakan akses mendefinisikan akun, pengguna, dan peran yang dapat mengakses antrian. Kebijakan akses juga mendefinisikan tindakan (seperti `SendMessage`, `ReceiveMessage`, atau `DeleteMessage`) yang dapat diakses pengguna. Kebijakan default hanya mengizinkan pemilik antrian untuk mengirim dan menerima pesan.

   Untuk menentukan kebijakan akses, lakukan salah satu hal berikut:

   - Pilih Dasar untuk mengonfigurasi siapa yang dapat mengirim pesan ke antrian dan siapa yang dapat menerima pesan dari antrian. Konsol membuat kebijakan berdasarkan pilihan Anda dan menampilkan kebijakan akses yang dihasilkan di panel JSON hanya-baca.
   - Pilih Advanced untuk mengubah kebijakan akses JSON secara langsung. Ini memungkinkan Anda menentukan serangkaian tindakan khusus yang dapat dilakukan oleh setiap prinsipal (akun, pengguna, atau peran).

7. Untuk kebijakan Izinkan Remrive, pilih Diaktifkan. Pilih salah satu dari berikut ini: Izinkan semua, Dengan antrian, atau Tolak semua. Saat memilih Dengan antrian, tentukan daftar hingga 10 antrian sumber berdasarkan Nama Sumber Daya Amazon (ARN).
8. Amazon SQS menyediakan enkripsi sisi server terkelola secara default. Untuk memilih jenis kunci enkripsi, atau menonaktifkan enkripsi sisi server terkelola Amazon SQS, perluas Enkripsi. Untuk informasi lebih lanjut tentang jenis kunci enkripsi, lihat Mengonfigurasi enkripsi sisi server untuk antrian menggunakan kunci enkripsi yang dikelola SQS danMengonfigurasi enkripsi sisi server untuk antrian menggunakan konsol Amazon SQS.

   > **Note**
   >
   > Dengan SSE diaktifkan, anonim `SendMessage` dan `ReceiveMessage` permintaan ke antrian terenkripsi akan ditolak. Praktik terbaik keamanan Amazon SQS merekomendasikan agar tidak menggunakan permintaan anonim. Jika Anda ingin mengirim permintaan anonim ke antrian Amazon SQS, pastikan untuk menonaktifkan SSE.

9. (Opsional) Untuk mengonfigurasi antrian surat mati untuk menerima pesan yang tidak terkirim, perluas antrian Dead-letter.
10. (Opsional) Untuk menambahkan tag ke antrian, perluas Tag.
11. Pilih Buat antrean. Amazon SQS membuat antrian dan menampilkan halaman Detail antrian.

Amazon SQS menyebarkan informasi tentang antrian baru di seluruh sistem. Karena Amazon SQS adalah sistem terdistribusi, Anda mungkin mengalami sedikit penundaan sebelum konsol menampilkan antrian di halaman Antrian.

Mengirim pesan menggunakan antrian standar

Setelah antrian Anda dibuat, Anda dapat mengirim pesan ke sana.

1. Dari panel navigasi kiri, pilih Antrian. Dari daftar antrian, pilih antrian yang Anda buat.
2. Dari Tindakan, pilih Kirim dan terima pesan.

   Konsol menampilkan halaman Kirim dan terima pesan.
3. Di badan Pesan, masukkan teks pesan.

4. Untuk antrian standar, Anda dapat memasukkan nilai untuk penundaan Pengiriman dan memilih unit. Misalnya, masukkan 60 dan pilih detik. Untuk informasi selengkapnya, lihat Pengatur waktu pesan Amazon SQS.
5. Pilih Kirim pesan.

   Saat pesan Anda dikirim, konsol menampilkan pesan sukses. Pilih Lihat detail untuk menampilkan informasi tentang pesan terkirim.

Membuat antrian Amazon SQS FIFO dan mengirim pesan

Anda dapat membuat antrean Amazon SQS FIFO dan mengirim pesan menggunakan konsol. Topik ini menjelaskan cara mengatur parameter antrian, termasuk batas waktu visibilitas, penyimpanan pesan, dan deduplikasi, sambil mengikuti praktik terbaik keamanan seperti menghindari informasi sensitif dalam nama antrian dan mengaktifkan enkripsi sisi server. Ini juga mencakup mendefinisikan kebijakan akses, mengonfigurasi antrian huruf mati, dan mengirim pesan dengan atribut khusus FIFO seperti ID grup pesan dan ID deduplikasi.

Membuat antrian FIFO menggunakan konsol Amazon SQS

Anda dapat menggunakan konsol Amazon SQS untuk membuat antrian FIFO. Konsol menyediakan nilai default untuk semua pengaturan kecuali untuk nama antrian.

> **Important**
>
> Pada 17 Agustus 2022, enkripsi sisi server default (SSE) diterapkan ke semua antrian Amazon SQS.
> Jangan menambahkan informasi identitas pribadi (PII) atau informasi rahasia atau sensitif lainnya dalam nama antrian. Nama antrian dapat diakses oleh banyak Amazon Web Services, termasuk penagihan dan CloudWatch log. Nama antrian tidak dimaksudkan untuk digunakan untuk data pribadi atau sensitif.

Untuk membuat antrean Amazon SQS FIFO

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Pilih Buat antrean.
3. Untuk Type, tipe antrian Standar diatur secara default. Untuk membuat antrian FIFO, pilih FIFO.

> **Note**
>
> Anda tidak dapat mengubah jenis antrian setelah Anda membuat antrian.

4. Masukkan Nama untuk antrian Anda.

   Nama antrian FIFO harus diakhiri dengan akhiran. `.fifo` Akhiran dihitung terhadap kuota nama antrian 80 karakter. Untuk menentukan apakah antrian adalah FIFO, Anda dapat memeriksa apakah nama antrian diakhiri dengan akhiran.

5. (Opsional) Konsol menetapkan nilai default untuk parameter konfigurasi antrian. Di bawah Konfigurasi, Anda dapat mengatur nilai baru untuk parameter berikut:

   a. Untuk batas waktu Visibilitas, masukkan durasi dan unit. Kisarannya dari 0 detik hingga 12 jam. Nilai default-nya adalah 30 detik.

   b. Untuk periode penyimpanan Pesan, masukkan durasi dan unit. Kisarannya dari 1 menit hingga 14 hari. Nilai defaultnya adalah 4 hari.

   c. Untuk keterlambatan Pengiriman, masukkan durasi dan unit. Kisarannya dari 0 detik hingga 15 menit. Nilai defaultnya adalah 0 detik.

   d. Untuk Ukuran pesan maksimum, masukkan nilai. Kisarannya dari 1 KiB hingga 1024 KiB. Nilai defaultnya adalah 1024 KiB.

   e. Untuk Menerima waktu tunggu pesan, masukkan nilai. Kisarannya dari 0 hingga 20 detik. Nilai defaultnya adalah 0 detik, yang menetapkan polling pendek. Setiap nilai bukan nol menetapkan polling panjang.

   f. Untuk antrian FIFO, pilih deduplikasi berbasis konten untuk mengaktifkan deduplikasi berbasis konten. Pengaturan default dinonaktifkan.

   g. (Opsional) Untuk antrian FIFO untuk mengaktifkan throughput yang lebih tinggi untuk mengirim dan menerima pesan dalam antrian, pilih Aktifkan FIFO throughput tinggi.

      Memilih opsi ini mengubah opsi terkait (cakupan Deduplikasi dan batas throughput FIFO) ke pengaturan yang diperlukan untuk mengaktifkan throughput tinggi untuk antrian FIFO. Jika Anda mengubah salah satu pengaturan yang diperlukan untuk menggunakan FIFO throughput tinggi, throughput normal berlaku untuk antrian, dan deduplikasi terjadi seperti yang ditentukan. Untuk informasi selengkapnya, lihat Throughput tinggi untuk antrian FIFO di Amazon SQS dan Kuota pesan Amazon SQS.

6. (Opsional) Tentukan kebijakan Akses. Kebijakan akses mendefinisikan akun, pengguna, dan peran yang dapat mengakses antrian. Kebijakan akses juga mendefinisikan tindakan (seperti `SendMessage`, `ReceiveMessage`, atau `DeleteMessage`) yang dapat diakses pengguna. Kebijakan default hanya mengizinkan pemilik antrian untuk mengirim dan menerima pesan.

   Untuk menentukan kebijakan akses, lakukan salah satu hal berikut:

   - Pilih Dasar untuk mengonfigurasi siapa yang dapat mengirim pesan ke antrian dan siapa yang dapat menerima pesan dari antrian. Konsol membuat kebijakan berdasarkan pilihan Anda dan menampilkan kebijakan akses yang dihasilkan di panel JSON hanya-baca.
   - Pilih Advanced untuk mengubah kebijakan akses JSON secara langsung. Ini memungkinkan Anda menentukan serangkaian tindakan khusus yang dapat dilakukan oleh setiap prinsipal (akun, pengguna, atau peran).

7. Untuk kebijakan Izinkan Remrive, pilih Diaktifkan. Pilih salah satu dari berikut ini: Izinkan semua, Dengan antrian, atau Tolak semua. Saat memilih Dengan antrian, tentukan daftar hingga 10 antrian sumber berdasarkan Nama Sumber Daya Amazon (ARN).

8. Amazon SQS menyediakan enkripsi sisi server terkelola secara default. Untuk memilih jenis kunci enkripsi, atau menonaktifkan enkripsi sisi server terkelola Amazon SQS, perluas Enkripsi. Untuk informasi lebih lanjut tentang jenis kunci enkripsi, lihat Mengonfigurasi enkripsi sisi server untuk antrian menggunakan kunci enkripsi yang dikelola SQS danMengonfigurasi enkripsi sisi server untuk antrian menggunakan konsol Amazon SQS.

   > **Note**
   >
   > Dengan SSE diaktifkan, anonim `SendMessage` dan `ReceiveMessage` permintaan ke antrian terenkripsi akan ditolak. Praktik terbaik keamanan Amazon SQS merekomendasikan agar tidak menggunakan permintaan anonim. Jika Anda ingin mengirim permintaan anonim ke antrian Amazon SQS, pastikan untuk menonaktifkan SSE.

9. (Opsional) Untuk mengonfigurasi antrian surat mati untuk menerima pesan yang tidak terkirim, perluas antrian Dead-letter.

10. (Opsional) Untuk menambahkan tag ke antrian, perluas Tag.

11. Pilih Buat antrean. Amazon SQS membuat antrian dan menampilkan halaman Detail antrian.

Amazon SQS menyebarkan informasi tentang antrian baru di seluruh sistem. Karena Amazon SQS adalah sistem terdistribusi, Anda mungkin mengalami sedikit penundaan sebelum konsol menampilkan antrian di halaman Antrian.

Setelah membuat antrian, Anda dapat mengirim pesan ke sana, dan menerima serta menghapus pesan. Anda juga dapat mengedit pengaturan konfigurasi antrian apa pun kecuali jenis antrian.

Mengirim pesan menggunakan antrian FIFO

Setelah Anda membuat antrian, Anda dapat mengirim pesan ke sana.

1. Dari panel navigasi kiri, pilih Antrian. Dari daftar antrian, pilih antrian yang Anda buat.
2. Dari Tindakan, pilih Kirim dan terima pesan.

   Konsol menampilkan halaman Kirim dan terima pesan.
3. Di badan Pesan, masukkan teks pesan.
4. Untuk antrian First-In-First-Out (FIFO), masukkan ID grup Pesan. Untuk informasi selengkapnya, lihat Logika pengiriman antrian FIFO di Amazon SQS.
5. (Opsional) Untuk antrian FIFO, Anda dapat memasukkan ID deduplikasi Pesan. Jika Anda mengaktifkan deduplikasi berbasis konten untuk antrian, ID deduplikasi pesan tidak diperlukan. Untuk informasi selengkapnya, lihat Logika pengiriman antrian FIFO di Amazon SQS.
6. Antrian FIFO tidak mendukung pengatur waktu pada pesan individual. Untuk informasi selengkapnya, lihat Pengatur waktu pesan Amazon SQS.
7. Pilih Kirim pesan.

   Saat pesan Anda dikirim, konsol menampilkan pesan sukses. Pilih Lihat detail untuk menampilkan informasi tentang pesan terkirim.

Tugas umum untuk memulai dengan Amazon SQS

Setelah Anda membuat antrian dan mempelajari cara mengirim, menerima, dan menghapus pesan, Anda mungkin ingin mencoba yang berikut ini:

- Memicu fungsi Lambda untuk memproses pesan masuk secara otomatis, memungkinkan alur kerja berbasis peristiwa tanpa perlu polling berkelanjutan.
- Konfigurasikan antrian, termasuk SSE dan fitur lainnya.
- Kirim pesan dengan atribut.

- Kirim pesan dari VPC.
- Temukan fungsionalitas dan arsitektur Amazon SQS.
- Temukan panduan dan peringatan yang akan membantu Anda memaksimalkan Amazon SQS.
- Jelajahi contoh Amazon SQS untuk AWS SDK, seperti Panduan Pengembang.AWS SDK for Java 2.x
- Pelajari tentang perintah Amazon SQS. AWS CLI
- Pelajari tentang tindakan Amazon SQS API.
- Pelajari cara berinteraksi dengan Amazon SQS secara terprogram. Lihat Bekerja dengan APIs dan menjelajahi Pusat AWS Pengembangan:
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows & .NET
- Pelajari cara memantau biaya dan sumber daya.
- Pelajari cara melindungi data Anda.
- Pelajari lebih lanjut tentang alur kerja Amazon SQS.

Mengelola antrian Amazon SQS

Pelajari cara mengelola antrian Amazon SQS menggunakan konsol, termasuk mengedit setelan antrian, menerima dan menghapus pesan, mengonfirmasi kekosongan antrian, dan menghapus atau membersihkan antrian. Memahami praktik terbaik untuk penanganan pesan yang efisien, seperti menggunakan polling panjang, mengelola batas waktu visibilitas, dan memverifikasi metrik melalui dasbor pemantauan atau. AWS CLI Ikuti langkah-langkah praktis untuk menjaga antrian dan menangani pesan secara efektif sambil meminimalkan gangguan.

Mengedit antrian Amazon SQS menggunakan konsol

Anda dapat menggunakan konsol Amazon SQS untuk mengedit parameter konfigurasi antrian (kecuali jenis antrian) dan memodifikasi atau menghapus fitur sesuai kebutuhan.

Untuk mengedit antrian Amazon SQS (konsol)

1. Buka halaman Antrian konsol Amazon SQS.
2. Pilih antrian, lalu pilih Edit.
3. (Opsional) Di bawah Konfigurasi, perbarui parameter konfigurasi antrian.
4. (Opsional) Untuk memperbarui kebijakan akses, di bawah kebijakan Access, ubah kebijakan JSON.
5. (Opsional) Untuk memperbarui kebijakan izin pengaktifan ulang antrian huruf mati, perluas kebijakan Izinkan Remrive.
6. (Opsional) Untuk memperbarui atau menghapus enkripsi, perluas Enkripsi.
7. (Opsional) Untuk menambah, memperbarui, atau menghapus antrian huruf mati (yang memungkinkan Anda menerima pesan yang tidak terkirim), perluas antrian Dead-letter.
8. (Opsional) Untuk menambah, memperbarui, atau menghapus tag untuk antrian, perluas Tag.
9. Pilih Simpan.

    - Konsol menampilkan halaman Detail untuk antrian.

Menerima dan menghapus pesan di Amazon SQS

Setelah mengirim pesan ke antrean Amazon SQS, Anda dapat mengambil dan menghapusnya untuk memproses alur kerja aplikasi Anda. Proses ini memastikan penanganan pesan yang aman

dan andal. Topik ini memandu Anda untuk mengambil dan menghapus pesan menggunakan konsol Amazon SQS dan menjelaskan setelan kunci untuk mengoptimalkan operasi ini. Berikut ini adalah konsep kunci untuk menerima dan menghapus pesan:

1. Menerima pesan
   - Saat mengambil pesan dari antrean Amazon SQS, Anda tidak dapat menargetkan pesan tertentu. Sebagai gantinya, tentukan jumlah maksimum pesan yang akan diambil dalam satu permintaan (hingga 10).
   - Karena sifat terdistribusi Amazon SQS, mengambil dari antrian dengan beberapa pesan dapat mengembalikan respons kosong. Untuk mengurangi ini:
     - Gunakan polling panjang, yang menunggu sampai pesan tersedia atau waktu polling habis. Pendekatan ini mengurangi biaya pemungutan suara yang tidak perlu dan meningkatkan efisiensi.
     - Terbitkan ulang permintaan jika diperlukan.
2. Visibilitas dan penghapusan pesan
   - Pesan tidak dihapus secara otomatis setelah pengambilan. Fitur ini memastikan Anda dapat memproses ulang pesan jika terjadi kegagalan aplikasi atau gangguan jaringan.
   - Setelah diproses, Anda harus secara eksplisit mengirim permintaan hapus untuk menghapus pesan secara permanen. Tindakan ini menegaskan penanganan yang berhasil.
   - Pesan yang diambil menggunakan konsol Amazon SQS tetap terlihat untuk pengambilan ulang. Sesuaikan pengaturan batas waktu visibilitas untuk lingkungan otomatis untuk menyembunyikan sementara pesan dari konsumen lain saat sedang diproses.
3. Batas waktu visibilitas
   - Pengaturan ini menentukan berapa lama pesan tetap tersembunyi setelah pengambilan. Tetapkan batas waktu yang sesuai untuk memastikan pesan diproses hanya sekali dan untuk mencegah duplikasi selama pemrosesan terdistribusi.

Untuk menerima dan menghapus pesan menggunakan konsol

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pada halaman Antrian, pilih antrian yang ingin Anda terima pesan, lalu pilih Kirim dan terima pesan.
4. Pada halaman Kirim dan terima pesan, pilih Poll untuk pesan.

Amazon SQS menampilkan bilah kemajuan yang menunjukkan durasi polling. Pesan yang diambil akan muncul di bagian Pesan, yang menunjukkan:

- ID Pesan
- Tanggal terkirim
- Size
- Menerima hitungan

5. Untuk menghapus pesan, pilih yang ingin Anda hapus dan pilih Hapus.

   Konfirmasikan penghapusan di kotak dialog Hapus Pesan dengan memilih Hapus.

Untuk detail selengkapnya tentang operasi lanjutan, termasuk pengambilan dan penghapusan pesan berbasis API, lihat Panduan Referensi API Amazon SQS.

Mengonfirmasi bahwa antrian Amazon SQS kosong

Dalam kebanyakan kasus, Anda dapat menggunakan polling panjang untuk menentukan apakah antrian kosong. Dalam kasus yang jarang terjadi, Anda mungkin menerima tanggapan kosong bahkan ketika antrian masih berisi pesan, terutama jika Anda menetapkan nilai rendah untuk Menerima pesan waktu tunggu saat Anda membuat antrean. Bagian ini menjelaskan cara mengonfirmasi bahwa antrian kosong.

Untuk mengonfirmasi bahwa antrian kosong (konsol)

1. Hentikan semua produsen mengirim pesan.
2. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
3. Di panel navigasi, pilih Antrian.
4. Pada halaman Antrian, pilih antrian.
5. Pilih tab Pemantauan.
6. Di kanan atas dasbor Pemantauan, pilih panah bawah di sebelah simbol Refresh. Dari menu tarik-turun, pilih Segarkan otomatis. Biarkan interval Refresh pada 1 Menit.
7. Perhatikan dasbor berikut:

   - Perkiraan Jumlah Pesan Tertunda
   - Perkiraan Jumlah Pesan Tidak Terlihat

- Perkiraan Jumlah Pesan yang Terlihat

  Ketika semuanya menunjukkan `0` nilai selama beberapa menit, antrian kosong.

Untuk mengonfirmasi bahwa antrian kosong (AWS CLI, AWS API)

1. Hentikan semua produsen mengirim pesan.
2. Berulang kali menjalankan salah satu perintah berikut:
   - AWS CLI: `get-queue-attributes`
   - AWS API: `GetQueueAttributes`
3. Perhatikan metrik untuk atribut berikut:
   - `ApproximateNumberOfMessagesDelayed`
   - `ApproximateNumberOfMessagesNotVisible`
   - `ApproximateNumberOfMessagesVisible`

   Ketika semuanya `0` selama beberapa menit, antrian kosong.

Jika Anda mengandalkan CloudWatch metrik Amazon, pastikan Anda melihat beberapa titik data nol berturut-turut sebelum menganggap antrian itu kosong. Untuk informasi selengkapnya tentang CloudWatch metrik, lihatCloudWatch Metrik yang tersedia untuk Amazon SQS.

Menghapus antrian Amazon SQS

Jika Anda tidak lagi menggunakan antrean Amazon SQS dan tidak berencana untuk menggunakannya dalam waktu dekat, hapus antrean.

> **Tip**
>
> Jika Anda ingin memverifikasi bahwa antrian kosong sebelum Anda menghapusnya, lihatMengonfirmasi bahwa antrian Amazon SQS kosong.

Anda dapat menghapus antrian bahkan ketika itu tidak kosong. Untuk menghapus pesan dalam antrian tetapi bukan antrian itu sendiri, bersihkan antrean.

Untuk menghapus antrian (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pada halaman Antrian, pilih antrian yang akan dihapus.
4. Pilih Hapus.
5. Di kotak dialog Hapus antrian, konfirmasikan penghapusan dengan memasukkan. **`delete`**
6. Pilih Hapus.

Untuk menghapus antrian (AWS CLI dan API)

Pilih metode yang sesuai untuk menghapus antrian Anda berdasarkan kebutuhan Anda:

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

Membersihkan pesan dari antrian menggunakan konsol Amazon SQS

Untuk menjaga antrean Amazon SQS tetapi menghapus semua pesannya, Anda dapat membersihkan antrean. Ini akan menghapus semua pesan, termasuk yang saat ini tidak terlihat (dalam penerbangan). Proses pembersihan dapat memakan waktu hingga 60 detik, jadi tunggu 60 detik penuh terlepas dari ukuran antrian.

> **Important**
>
> Saat Anda membersihkan antrian, Anda tidak dapat mengambil pesan yang dihapus.

Untuk membersihkan antrian (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pada halaman Antrian, pilih antrian yang akan dibersihkan.
4. Dari Actions, pilih Purge.

5. Di kotak dialog Purge queue, konfirmasikan pembersihan dengan memasukkan **`purge`** dan memilih Purge.

   - Semua pesan dihapus dari antrian. Konsol menampilkan spanduk konfirmasi.

Antrian standar Amazon SQS

Amazon SQS menyediakan antrian standar sebagai tipe antrian default, mendukung jumlah panggilan API yang hampir tidak terbatas per detik untuk tindakan seperti,, dan. `SendMessage` `ReceiveMessage` `DeleteMessage` Antrian standar memastikan pengiriman at-least-once pesan, tetapi karena arsitektur yang sangat terdistribusi, lebih dari satu salinan pesan mungkin terkirim, dan pesan terkadang gagal. Meskipun demikian, antrian standar melakukan upaya terbaik untuk menjaga urutan pengiriman pesan.

Saat Anda mengirim pesan menggunakan`SendMessage`, Amazon SQS menyimpan pesan secara berlebihan di beberapa zona ketersediaan (AZs) sebelum mengakuinya. Redundansi ini memastikan bahwa tidak ada satu komputer, jaringan, atau kegagalan AZ yang dapat membuat pesan tidak dapat diakses.

Anda dapat membuat dan mengonfigurasi antrian menggunakan konsol Amazon SQS. Untuk petunjuk mendetail, lihat Membuat antrian standar menggunakan konsol Amazon SQS. Untuk contoh khusus Java, lihat. Contoh Amazon SQS Java SDK

Gunakan kasus untuk antrian standar

Antrian pesan standar cocok untuk berbagai skenario, selama aplikasi Anda dapat menangani pesan yang mungkin tiba lebih dari sekali atau rusak. Contohnya termasuk:

- Memisahkan permintaan pengguna langsung dari pekerjaan latar belakang intensif - Pengguna dapat mengunggah media saat sistem mengubah ukuran atau menyandikannya di latar belakang.
- Mengalokasikan tugas ke beberapa node pekerja — Misalnya, menangani permintaan validasi kartu kredit dengan volume tinggi.
- Batching messages for future processing — Menjadwalkan beberapa entri yang akan ditambahkan ke database di lain waktu.

Untuk informasi tentang kuota yang terkait dengan antrian standar, lihat. Kuota antrian standar Amazon SQS

Untuk praktik terbaik bekerja dengan antrian standar, lihat. Praktik terbaik Amazon SQS

Pengiriman Amazon at-least-once SQS

Amazon SQS menyimpan salinan pesan Anda di beberapa server untuk redundansi dan ketersediaan tinggi. Pada kesempatan yang jarang terjadi, salah satu server yang menyimpan salinan pesan mungkin tidak tersedia saat Anda menerima atau menghapus pesan.

Jika ini terjadi, salinan pesan tidak akan dihapus di server yang tidak tersedia, dan Anda mungkin mendapatkan salinan pesan itu lagi saat menerima pesan. Rancang aplikasi Anda menjadi idempoten (tidak boleh terpengaruh secara negatif saat memproses pesan yang sama lebih dari sekali).

Antrian Amazon SQS dan pengidentifikasi pesan

Topik ini menjelaskan pengidentifikasi antrian standar dan FIFO. Pengidentifikasi ini dapat membantu Anda menemukan dan memanipulasi antrian dan pesan tertentu.

Pengidentifikasi untuk antrian standar Amazon SQS

Untuk informasi selengkapnya tentang pengidentifikasi berikut, lihat Referensi API Layanan Antrian Sederhana Amazon.

Nama antrian dan URL

Saat membuat antrian baru, Anda harus menentukan nama antrian yang unik untuk AWS akun dan wilayah Anda. Amazon SQS menetapkan setiap antrian yang Anda buat pengenal yang disebut URL antrian yang menyertakan nama antrian dan komponen Amazon SQS lainnya. Kapan pun Anda ingin melakukan tindakan pada antrian, Anda memberikan URL antreannya.

Berikut ini adalah URL antrian untuk antrian bernama `MyQueue` dimiliki oleh pengguna dengan nomor akun AWS. `123456789012`

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

Anda dapat mengambil URL antrian secara terprogram dengan mencantumkan antrian Anda dan mengurai string yang mengikuti nomor akun. Untuk informasi selengkapnya, lihat `ListQueues`.

ID Pesan

Setiap pesan menerima ID pesan yang ditetapkan sistem yang dikembalikan Amazon SQS kepada Anda dalam respons. `SendMessage` Pengenal ini berguna untuk mengidentifikasi pesan. Panjang maksimum ID pesan adalah 100 karakter.

Pegangan tanda terima

Setiap kali Anda menerima pesan dari antrian, Anda menerima tanda terima untuk pesan tersebut. Pegangan ini dikaitkan dengan tindakan menerima pesan, bukan dengan pesan itu sendiri. Untuk menghapus pesan atau mengubah visibilitas pesan, Anda harus memberikan tanda terima (bukan ID pesan). Dengan demikian, Anda harus selalu menerima pesan sebelum Anda dapat menghapusnya (Anda tidak dapat memasukkan pesan ke dalam antrian dan kemudian mengingatnya). Panjang maksimum pegangan tanda terima adalah 1.024 karakter.

> **Important**
>
> Jika Anda menerima pesan lebih dari sekali, setiap kali Anda menerimanya, Anda mendapatkan pegangan tanda terima yang berbeda. Anda harus memberikan tanda terima yang paling baru diterima saat Anda meminta untuk menghapus pesan (jika tidak, pesan mungkin tidak dihapus).

Berikut ini adalah contoh pegangan tanda terima yang rusak di tiga baris.

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

Antrian Amazon SQS FIFO

Antrian FIFO (First-In-First-Out) memiliki semua kemampuan antrian standar, tetapi dirancang untuk meningkatkan pesan antar aplikasi ketika urutan operasi dan peristiwa sangat penting, atau di mana duplikat tidak dapat ditoleransi.

Fitur terpenting dari antrian FIFO adalah pengiriman FIFO (First-In-First-Out) dan pemrosesan tepat sekali:

- Urutan pengiriman dan penerimaan pesan dipertahankan dengan ketat dan pesan dikirimkan sekali dan tetap tidak tersedia sampai konsumen memproses dan menghapusnya.
- Duplikat tidak dimasukkan ke dalam antrian.

Selain itu, antrian FIFO mendukung grup pesan yang memungkinkan beberapa grup pesan yang dipesan dalam satu antrian. Tidak ada kuota untuk jumlah grup pesan dalam antrian FIFO.

Contoh situasi di mana Anda mungkin menggunakan antrian FIFO meliputi:

1. Sistem manajemen pesanan e-commerce di mana pesanan sangat penting
2. Mengintegrasikan dengan sistem pihak ketiga di mana acara perlu diproses secara berurutan
3. Memproses input yang dimasukkan pengguna dalam urutan yang dimasukkan
4. Komunikasi dan jaringan — Mengirim dan menerima data dan informasi dalam urutan yang sama
5. Sistem komputer — Memastikan bahwa perintah yang dimasukkan pengguna dijalankan dalam urutan yang benar
6. Lembaga pendidikan — Mencegah siswa mendaftar di kursus sebelum mendaftar untuk akun
7. Sistem tiket online — Di mana tiket didistribusikan berdasarkan first come first serve

> **Note**
>
> Antrian FIFO juga menyediakan pemrosesan yang tepat sekali, tetapi memiliki jumlah transaksi per detik (TPS) terbatas. Anda dapat menggunakan mode throughput tinggi Amazon SQS dengan antrean FIFO untuk meningkatkan batas transaksi. Untuk detail tentang penggunaan mode throughput tinggi, lihatThroughput tinggi untuk antrian FIFO di Amazon SQS. Untuk informasi tentang kuota throughput, lihat. the section called “Kuota pesan”

Antrian Amazon SQS FIFO tersedia di semua Wilayah tempat Amazon SQS tersedia.

Untuk informasi lebih lanjut tentang penggunaan antrian FIFO dengan pengurutan kompleks, lihat Memecahkan Tantangan Pemesanan Kompleks dengan Antrian FIFO Amazon SQS.

Untuk informasi tentang cara membuat dan mengonfigurasi antrian menggunakan konsol Amazon SQS, lihat. Membuat antrian standar menggunakan konsol Amazon SQS Untuk contoh Java, lihatContoh Amazon SQS Java SDK.

Untuk praktik terbaik untuk bekerja dengan antrian FIFO, lihat. Praktik terbaik Amazon SQS

Istilah kunci antrian Amazon SQS FIFO

Istilah kunci berikut dapat membantu Anda lebih memahami fungsionalitas antrian FIFO. Untuk informasi selengkapnya, lihat Referensi API Layanan Antrian Sederhana Amazon.

Klien

Klien Asinkron Buffered Amazon SQS saat ini tidak mendukung antrian FIFO.

ID deduplikasi pesan

Token yang digunakan dalam antrian Amazon SQS FIFO untuk mengidentifikasi pesan secara unik dan mencegah duplikasi. Jika beberapa pesan dengan ID deduplikasi yang sama dikirim dalam interval deduplikasi 5 menit, mereka diperlakukan sebagai duplikat, dan hanya satu salinan yang dikirim. Jika Anda tidak menentukan ID deduplikasi dan deduplikasi berbasis konten diaktifkan, Amazon SQS akan menghasilkan ID deduplikasi dengan melakukan hashing pada isi pesan. Mekanisme ini memastikan pengiriman tepat sekali dengan menghilangkan pesan duplikat dalam jangka waktu yang ditentukan.

> **Note**
>
> Amazon SQS terus melacak ID deduplikasi bahkan setelah pesan diterima dan dihapus.

ID grup pesan

Dalam antrian FIFO (First-In-First-Out), `MessageGroupId` adalah atribut yang mengatur pesan ke dalam grup yang berbeda. Pesan dalam grup pesan yang sama selalu diproses satu per satu, dalam urutan yang ketat, memastikan bahwa tidak ada dua pesan dari grup yang sama

yang diproses secara bersamaan. Dalam antrian standar, menggunakan `MessageGroupId` memungkinkan antrian yang adil. Jika pemesanan ketat diperlukan, gunakan antrian FIFO.

Menerima ID percobaan permintaan

ID percobaan permintaan terima adalah token unik yang digunakan untuk menghapus duplikasi `ReceiveMessage`panggilan di Amazon SQS.

Nomor urut

Jumlah besar dan tidak berurutan yang diberikan Amazon SQS untuk setiap pesan.

Layanan

Jika aplikasi Anda menggunakan beberapa AWS layanan, atau campuran layanan eksternal, penting untuk memahami fungsionalitas layanan mana yang tidak mendukung antrian FIFO. AWS

Beberapa AWS atau layanan eksternal yang mengirim pemberitahuan ke Amazon SQS mungkin tidak kompatibel dengan antrian FIFO, meskipun memungkinkan Anda untuk menetapkan antrian FIFO sebagai target.

Fitur AWS layanan berikut saat ini tidak kompatibel dengan antrian FIFO:

- Pemberitahuan Acara Amazon S3
- Kait Siklus Hidup Auto Scaling
- AWS IoT Tindakan Aturan
- AWS Lambda Antrian Surat Mati

Untuk informasi tentang kompatibilitas layanan lain dengan antrian FIFO, lihat dokumentasi layanan Anda.

Logika pengiriman antrian FIFO di Amazon SQS

Konsep berikut memperjelas bagaimana antrian Amazon SQS FIFO menangani pengiriman dan penerimaan pesan, terutama ketika berhadapan dengan pemesanan pesan dan grup pesan. IDs

Mengirim pesan

Antrian Amazon SQS FIFO mempertahankan urutan pesan menggunakan deduplikasi unik dan grup pesan. IDs IDs Topik ini menyoroti pentingnya grup pesan IDs untuk menjaga urutan yang ketat dalam grup dan menyoroti praktik terbaik untuk memastikan pengiriman pesan yang andal dan teratur di beberapa produsen.

1. Pelestarian pesanan
   - Ketika beberapa pesan dikirim berturut-turut ke antrian FIFO dengan deduplikasi pesan unik, IDs Amazon SQS menyimpannya dan mengakui transmisi mereka. Pesan-pesan ini kemudian diterima dan diproses dalam urutan yang tepat mereka dikirim.
2. ID grup pesan
   - Dalam antrian FIFO, pesan diurutkan berdasarkan ID grup pesan mereka. Jika beberapa produsen atau utas mengirim pesan dengan ID grup pesan yang sama, Amazon SQS memastikan pesan tersebut disimpan dan diproses sesuai urutan kedatangan.
   - Praktik terbaik: Untuk menjamin urutan pesan yang ketat di beberapa produsen, tetapkan ID grup pesan unik untuk semua pesan dari masing-masing produser.
3. Pemesanan per kelompok
   - Logika antrian FIFO berlaku berdasarkan ID grup per pesan:
     - Setiap ID grup pesan mewakili grup pesan yang berbeda dan terurut.
     - Dalam ID grup pesan, semua pesan dikirim dan diterima dalam urutan yang ketat.
     - Pesan dengan grup pesan yang berbeda IDs dapat tiba atau diproses rusak relatif terhadap satu sama lain.
   - Persyaratan - Anda harus mengaitkan ID grup pesan dengan setiap pesan. Jika pesan dikirim tanpa ID grup, tindakan gagal.
   - Skenario grup tunggal - Jika Anda mengharuskan semua pesan diproses dalam urutan yang ketat, gunakan ID grup pesan yang sama untuk setiap pesan.

Menerima pesan

Antrian FIFO Amazon SQS menangani pengambilan pesan, termasuk pemrosesan batch, jaminan pesanan FIFO, dan batasan saat meminta grup pesan tertentu. IDs Topik ini menjelaskan cara Amazon SQS mengambil pesan di dalam dan di seluruh grup pesan IDs sambil mempertahankan aturan pemesanan dan visibilitas yang ketat.

1. Pengambilan Batch
   - Saat menerima pesan dari antrian FIFO dengan beberapa grup pesan IDs, Amazon SQS:
     - Mencoba mengembalikan pesan sebanyak mungkin dengan ID grup pesan yang sama dalam satu panggilan.
     - Memungkinkan konsumen lain untuk memproses pesan dari grup pesan yang berbeda IDs secara bersamaan.

- Klarifikasi penting
  - Anda dapat menerima beberapa pesan dari ID grup pesan yang sama dalam satu batch (hingga 10 pesan dalam satu panggilan menggunakan `MaxNumberOfMessages` parameter).
  - Namun, Anda tidak dapat menerima pesan tambahan dari ID grup pesan yang sama dalam permintaan berikutnya hingga:
    - Pesan yang diterima saat ini dihapus, atau
    - Mereka menjadi terlihat lagi (misalnya, setelah batas waktu visibilitas berakhir).

2. Jaminan pesanan FIFO
   - Pesan yang diambil dalam batch mempertahankan urutan FIFO mereka di dalam grup.
   - Jika kurang dari 10 pesan tersedia untuk ID grup pesan yang sama, Amazon SQS dapat menyertakan pesan dari grup pesan lain IDs dalam kumpulan yang sama, tetapi setiap grup mempertahankan urutan FIFO.
3. Keterbatasan konsumen
   - Anda tidak dapat secara eksplisit meminta untuk menerima pesan dari ID grup pesan tertentu.

Mencoba lagi beberapa kali

Produsen dan konsumen dapat dengan aman mencoba kembali tindakan yang gagal di antrian Amazon SQS FIFO tanpa mengganggu urutan pesan atau memperkenalkan duplikat. Topik ini menyoroti bagaimana batas waktu deduplikasi IDs dan visibilitas memastikan integritas pesan selama percobaan ulang.

1. Produsen mencoba lagi
   - Jika `SendMessage`tindakan gagal, produser dapat mencoba lagi mengirim pesan beberapa kali dengan ID deduplikasi pesan yang sama.
   - Selama produsen menerima setidaknya satu pengakuan sebelum interval deduplikasi berakhir, coba lagi:
     - Jangan memperkenalkan pesan duplikat.
     - Jangan mengganggu pesanan pesan.
2. Percobaan ulang konsumen
   - Jika suatu `ReceiveMessage`tindakan gagal, konsumen dapat mencoba lagi sebanyak yang diperlukan menggunakan ID percobaan permintaan terima yang sama.

- Selama konsumen menerima setidaknya satu pengakuan sebelum batas waktu visibilitas berakhir, coba lagi:
  - Jangan mengganggu pesanan pesan.

Catatan tambahan tentang perilaku FIFO

Pelajari cara menangani batas waktu visibilitas, mengaktifkan pemrosesan paralel dengan beberapa IDs grup pesan, dan memastikan pemrosesan sekuensial yang ketat dalam skenario grup tunggal.

1. Menangani batas waktu visibilitas
   - Ketika pesan diambil tetapi tidak dihapus, pesan tetap tidak terlihat sampai batas waktu visibilitas berakhir.
   - Tidak ada pesan tambahan dari ID grup pesan yang sama yang dikembalikan hingga pesan pertama dihapus atau terlihat lagi.
2. Konkurensi dan pemrosesan paralel
   - Antrian FIFO memungkinkan pemrosesan paralel pesan di berbagai grup pesan. IDs
   - Untuk memaksimalkan konkurensi, rancang sistem Anda dengan beberapa grup pesan IDs untuk alur kerja independen.
3. Skenario kelompok tunggal
   - Untuk pemrosesan sekuensial yang ketat dari semua pesan dalam antrian FIFO, gunakan ID grup pesan tunggal untuk semua pesan dalam antrian.

Contoh untuk pemahaman yang lebih baik

Berikut ini adalah skenario praktis yang menggambarkan perilaku antrian FIFO di Amazon SQS.

1. Skenario 1: ID grup tunggal
   - Produser mengirim lima pesan dengan grup pesan yang sama ID Grup A.
   - Seorang konsumen menerima pesan-pesan ini dalam urutan FIFO. Sampai konsumen menghapus pesan ini atau batas waktu visibilitas berakhir, tidak ada pesan tambahan dari Grup A yang diterima.
2. Skenario 2: Beberapa grup IDs
   - Seorang produser mengirim lima pesan ke Grup A dan 5 ke Grup B.

- Konsumen 1 memproses pesan dari Grup A, sementara Konsumen 2 memproses pesan dari Grup B. Hal ini memungkinkan pemrosesan paralel dengan urutan ketat yang dipertahankan dalam setiap grup.

3. Skenario 3: Pengambilan Batch
   - Seorang produser mengirim tujuh pesan ke Grup A dan tiga ke Grup B.
   - Satu konsumen mengambil hingga 10 pesan. Jika antrian memungkinkan, itu mungkin kembali:
     - Tujuh pesan dari Grup A dan tiga dari Grup B (atau kurang jika lebih sedikit pesan yang tersedia dari satu grup).

Tepat sekali diproses di Amazon SQS

Tidak seperti antrian standar, antrian FIFO tidak memperkenalkan pesan duplikat. Antrian FIFO membantu Anda menghindari pengiriman duplikat ke antrian. Jika Anda mencoba lagi `SendMessage` tindakan dalam interval deduplikasi 5 menit, Amazon SQS tidak memasukkan duplikat apa pun ke dalam antrian.

Untuk mengonfigurasi deduplikasi, Anda harus melakukan salah satu hal berikut:

- Aktifkan deduplikasi berbasis konten. Ini menginstruksikan Amazon SQS untuk menggunakan hash SHA-256 untuk menghasilkan ID deduplikasi pesan menggunakan isi pesan — tetapi bukan atribut pesan. Untuk informasi selengkapnya, lihat dokumentasi tentang`CreateQueue`,`GetQueueAttributes`, dan `SetQueueAttributes` tindakan di Referensi API Layanan Antrian Sederhana Amazon.
- Berikan ID deduplikasi pesan secara eksplisit (atau lihat nomor urut) untuk pesan tersebut. Untuk informasi selengkapnya, lihat dokumentasi tentang`SendMessage`,`SendMessageBatch`, dan `ReceiveMessage` tindakan di Referensi API Layanan Antrian Sederhana Amazon.

Pindah dari antrian standar ke antrian FIFO di Amazon SQS

Jika aplikasi Anda yang ada menggunakan antrian standar dan Anda ingin memanfaatkan pemesanan atau fitur pemrosesan antrian FIFO yang tepat sekali, Anda perlu mengonfigurasi antrian dan aplikasi Anda dengan benar.

Pertimbangan utama

- Membuat antrian FIFO: Anda tidak dapat mengonversi antrian standar yang ada menjadi antrian FIFO. Anda harus membuat antrian FIFO baru untuk aplikasi Anda atau menghapus antrian standar yang ada dan membuatnya kembali sebagai antrian FIFO.
- Parameter Penundaan: Antrian FIFO tidak mendukung penundaan per pesan, hanya penundaan per antrian. Jika aplikasi Anda menetapkan `DelaySeconds` parameter pada setiap pesan, Anda harus memodifikasinya untuk mengatur `DelaySeconds` seluruh antrian sebagai gantinya.
- ID Grup Pesan: Berikan ID grup pesan untuk setiap pesan yang dikirim. ID ini memungkinkan pemrosesan pesan secara paralel sambil mempertahankan urutannya masing-masing. Gunakan dimensi bisnis granular untuk ID grup pesan agar lebih baik dengan antrian FIFO. Semakin banyak grup pesan tempat IDs Anda mendistribusikan pesan, semakin besar jumlah pesan yang tersedia untuk dikonsumsi.
- Mode Throughput Tinggi: Gunakan mode throughput tinggi yang direkomendasikan untuk antrian FIFO untuk mencapai peningkatan throughput. Untuk informasi selengkapnya tentang kuota pesan, lihatKuota pesan Amazon SQS.

Daftar periksa untuk pindah ke antrian FIFO

Sebelum mengirim pesan ke antrian FIFO, konfirmasikan hal berikut:

1. Konfigurasikan pengaturan penundaan
   - Ubah aplikasi Anda untuk menghapus penundaan per pesan.
   - Atur `DelaySeconds` parameter pada seluruh antrian.
2. Setel grup pesan IDs
   - Mengatur pesan ke dalam grup pesan dengan menentukan ID grup pesan berdasarkan dimensi bisnis.
   - Gunakan dimensi bisnis yang lebih terperinci untuk meningkatkan skalabilitas.
3. Menangani deduplikasi pesan
   - Jika aplikasi Anda tidak dapat mengirim pesan dengan badan pesan yang identik, berikan ID deduplikasi pesan unik untuk setiap pesan.
   - Jika aplikasi Anda mengirim pesan dengan badan pesan unik, aktifkan deduplikasi berbasis konten.
4. Konfigurasikan konsumen
   - Umumnya, tidak ada perubahan kode yang diperlukan untuk konsumen.

- Jika memproses pesan membutuhkan waktu lama dan batas waktu visibilitas ditetapkan tinggi, pertimbangkan untuk menambahkan ID percobaan permintaan terima ke setiap `ReceiveMessage` tindakan. Ini membantu mencoba lagi menerima upaya jika terjadi kegagalan jaringan dan mencegah antrian berhenti karena upaya penerimaan yang gagal.

Dengan mengikuti langkah-langkah ini, Anda dapat memastikan aplikasi Anda berfungsi dengan benar dengan antrian FIFO, memanfaatkan sepenuhnya pemesanan mereka dan fitur pemrosesan yang tepat sekali. Untuk informasi selengkapnya, lihat Referensi API Layanan Antrian Sederhana Amazon.

Antrian Amazon SQS FIFO dan perilaku konkurensi Lambda

Dengan menggunakan antrean FIFO (First-In-First-Out) dengan Lambda, Anda dapat memastikan pemrosesan pesan secara teratur dalam setiap grup pesan. Fungsi Lambda tidak akan menjalankan beberapa instance untuk grup pesan yang sama secara bersamaan, sehingga menjaga urutannya. Namun, ini dapat ditingkatkan untuk menangani beberapa grup pesan secara paralel, memastikan pemrosesan beban kerja antrian Anda secara efisien. Poin-poin berikut menjelaskan perilaku fungsi Lambda saat memproses pesan dari antrian FIFO Amazon SQS sehubungan dengan grup pesan: IDs

- Instance tunggal per grup pesan: Kapan saja, hanya satu instance Lambda yang akan memproses pesan dari ID grup pesan tertentu. Ini memastikan bahwa pesan dalam grup yang sama diproses secara berurutan, menjaga integritas urutan FIFO.
- Pemrosesan grup yang berbeda secara bersamaan: Lambda dapat memproses pesan secara bersamaan dari IDs grup pesan yang berbeda menggunakan beberapa instance. Ini berarti bahwa sementara satu instance fungsi Lambda menangani pesan dari satu ID grup pesan, instance lain dapat secara bersamaan menangani pesan dari grup pesan lain IDs, memanfaatkan kemampuan konkurensi Lambda untuk memproses beberapa grup secara paralel.

Pengelompokan pesan antrian FIFO

Antrian FIFO memastikan bahwa pesan diproses dalam urutan yang tepat saat dikirim. Mereka menggunakan ID grup pesan untuk mengelompokkan pesan yang harus diproses secara berurutan.

Pesan dalam grup pesan yang sama diproses secara berurutan, dan hanya satu pesan dari setiap grup yang diproses pada satu waktu untuk mempertahankan urutan ini.

Konkurensi Lambda dengan antrian FIFO

Setelah Anda membuat antrian, Anda dapat mengirim pesan ke sana.

Saat Anda menyiapkan fungsi Lambda untuk memproses pesan dari antrian FIFO Amazon SQS, Lambda menghormati jaminan pemesanan yang disediakan oleh antrian FIFO. Poin berikut menjelaskan perilaku fungsi Lambda dalam hal konkurensi dan penskalaan saat memproses pesan dari antrian FIFO Amazon SQS saat menggunakan grup pesan. IDs

- Konkurensi dalam grup pesan: Hanya satu instance Lambda yang memproses pesan untuk ID grup pesan tertentu pada satu waktu. Ini memastikan bahwa pesan dalam grup ditangani secara berurutan.
- Penskalaan dan beberapa grup pesan: Meskipun Lambda dapat meningkatkan skala untuk memproses pesan secara bersamaan, penskalaan ini terjadi di berbagai grup pesan. Jika Anda memiliki beberapa grup pesan, Lambda dapat memproses beberapa grup secara paralel, dengan setiap grup ditangani oleh instance Lambda terpisah.

Untuk informasi selengkapnya, lihat Penskalaan dan konkurensi di Lambda di Panduan Operator.AWS Lambda

Contoh kasus penggunaan

Misalkan antrian FIFO Anda menerima pesan dengan ID grup pesan yang sama, dan fungsi Lambda Anda memiliki batas konkurensi yang tinggi (hingga 1000).

Jika pesan dari ID grup 'A' sedang diproses dan pesan lain dari ID grup 'A' tiba, pesan kedua tidak akan memicu instance Lambda baru hingga pesan pertama diproses sepenuhnya.

Namun, jika pesan dari grup IDs 'A' dan 'B' tiba, kedua pesan dapat diproses secara bersamaan oleh instance Lambda yang terpisah.

Throughput tinggi untuk antrian FIFO di Amazon SQS

Antrian FIFO throughput tinggi di Amazon SQS secara efisien mengelola throughput pesan tinggi sambil mempertahankan urutan pesan yang ketat, memastikan keandalan dan skalabilitas untuk aplikasi yang memproses banyak pesan. Solusi ini sangat ideal untuk skenario yang menuntut throughput tinggi dan pengiriman pesan terpesan.

Antrian FIFO throughput tinggi Amazon SQS tidak diperlukan dalam skenario di mana urutan pesan yang ketat tidak penting dan di mana volume pesan masuk relatif rendah atau sporadis. Misalnya, jika Anda memiliki aplikasi skala kecil yang memproses pesan yang jarang atau tidak berurutan, kompleksitas dan biaya tambahan yang terkait dengan antrian FIFO throughput tinggi mungkin tidak dapat dibenarkan. Selain itu, jika aplikasi Anda tidak memerlukan kemampuan throughput yang ditingkatkan yang disediakan oleh antrian FIFO throughput tinggi, memilih antrian Amazon SQS standar mungkin lebih hemat biaya dan lebih mudah dikelola.

Untuk meningkatkan kapasitas permintaan dalam antrian FIFO throughput tinggi, disarankan untuk meningkatkan jumlah grup pesan. Untuk informasi selengkapnya tentang kuota pesan throughput tinggi, lihat kuota layanan Amazon SQS di Referensi Umum Amazon Web Services.

Untuk informasi kuota per antrian dan strategi distribusi data, lihat Kuota pesan Amazon SQS dan Partisi dan distribusi data untuk throughput tinggi untuk antrian SQS FIFO.

Kasus penggunaan untuk throughput tinggi untuk antrian Amazon SQS FIFO

Kasus penggunaan berikut menyoroti beragam aplikasi antrian FIFO throughput tinggi, menunjukkan efektivitasnya di seluruh industri dan skenario:

1. Real-time Pemrosesan data: Aplikasi yang menangani aliran data real-time, seperti pemrosesan peristiwa atau penyerapan data telemetri, dapat memperoleh manfaat dari antrian FIFO throughput tinggi untuk menangani masuknya pesan secara terus menerus sambil mempertahankan urutan mereka untuk analisis yang akurat.
2. E-commerce Pemrosesan pesanan: Dalam platform e-commerce di mana menjaga urutan transaksi pelanggan sangat penting, antrian FIFO throughput tinggi memastikan bahwa pesanan diproses secara berurutan dan tanpa penundaan, bahkan selama musim belanja puncak.
3. Layanan keuangan: Lem baga keuangan yang menangani perdagangan frekuensi tinggi atau data transaksional mengandalkan Antrian FIFO throughput tinggi untuk memproses data pasar dan transaksi dengan latensi minimal sambil mematuhi persyaratan peraturan yang ketat untuk pemesanan pesan.
4. Streaming media: Platform streaming dan layanan distribusi media memanfaatkan antrian FIFO throughput tinggi untuk mengelola pengiriman file media dan konten streaming, memastikan pengalaman pemutaran yang lancar bagi pengguna sambil mempertahankan urutan pengiriman konten yang benar.

Partisi dan distribusi data untuk throughput tinggi untuk antrian SQS FIFO

Amazon SQS menyimpan data antrian FIFO di partisi. Partisi adalah alokasi penyimpanan untuk antrian yang secara otomatis direplikasi di beberapa Zona Ketersediaan dalam Wil AWS ayah. Anda tidak mengelola partisi. Sebagai gantinya, Amazon SQS menangani manajemen partisi.

Untuk antrian FIFO, Amazon SQS mengubah jumlah partisi dalam antrian dalam situasi berikut:

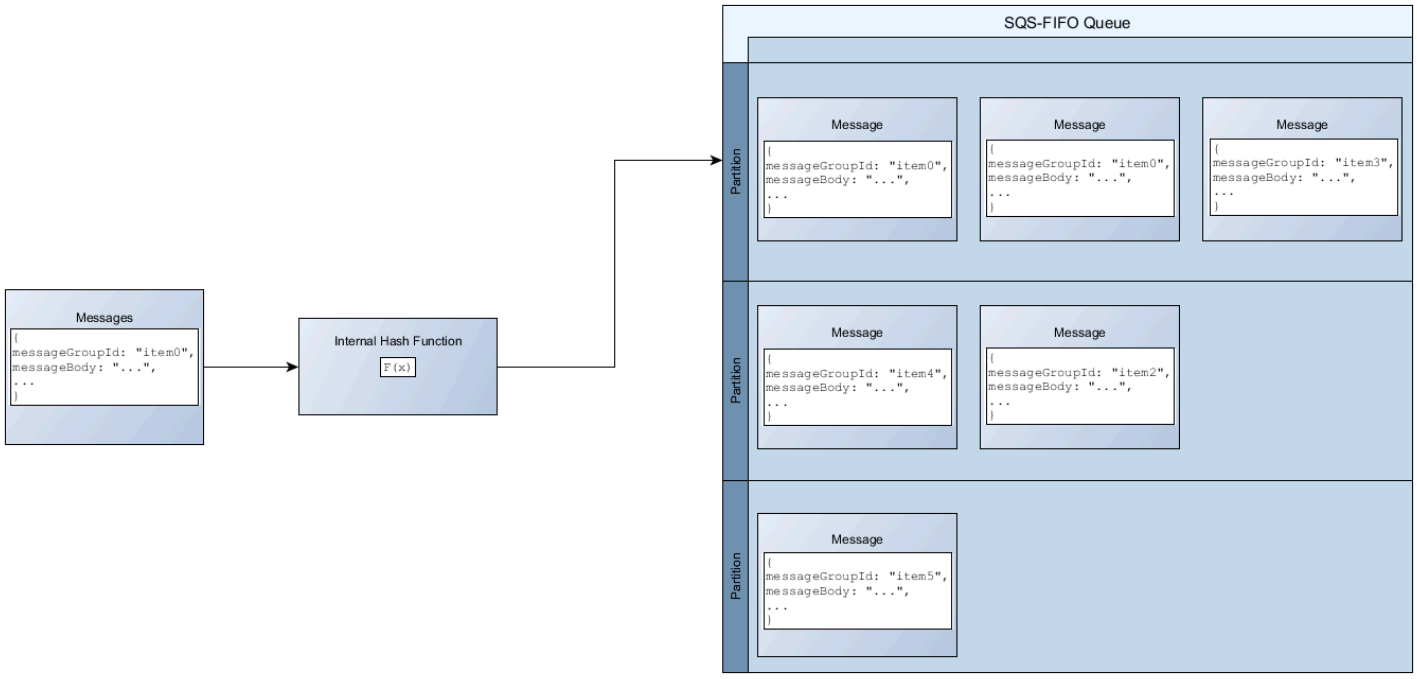
- Jika tingkat permintaan saat ini mendekati atau melebihi apa yang dapat didukung oleh partisi yang ada, partisi tambahan dialokasikan hingga antrian mencapai kuota regional. Untuk informasi tentang kuota, lihatKuota pesan Amazon SQS.
- Jika partisi saat ini memiliki pemanfaatan rendah, jumlah partisi dapat dikurangi.

Manajemen partisi terjadi secara otomatis di latar belakang dan transparan bagi aplikasi Anda. Antrian dan pesan Anda tersedia setiap saat.

Mendistribusikan data berdasarkan ID grup pesan

Untuk menambahkan pesan ke antrian FIFO, Amazon SQS menggunakan nilai ID grup pesan setiap pesan sebagai input ke fungsi hash internal. Nilai output dari fungsi hash menentukan partisi mana yang menyimpan pesan.

Diagram berikut menunjukkan antrian yang mencakup beberapa partisi. ID grup pesan antrian didasarkan pada nomor item. Amazon SQS menggunakan fungsi hash untuk menentukan tempat menyimpan item baru; dalam hal ini, ini didasarkan pada nilai hash string`item0`. Perhatikan bahwa item disimpan dalam urutan yang sama di mana mereka ditambahkan ke antrian. Lokasi setiap item ditentukan oleh nilai hash dari ID grup pesannya.



> **Note**
>
> Amazon SQS dioptimalkan untuk distribusi item yang seragam di seluruh partisi antrian FIFO, terlepas dari jumlah partisi. AWS menyarankan agar Anda menggunakan ID grup pesan yang dapat memiliki sejumlah besar nilai berbeda.

Mengoptimalkan pemanfaatan partisi

Setiap partisi mendukung hingga 3.000 pesan per detik dengan batching, atau hingga 300 pesan per detik untuk operasi kirim, terima, dan hapus di wilayah yang didukung. Untuk informasi selengkapnya tentang kuota pesan throughput tinggi, lihat kuota layanan Amazon SQS di Referensi Umum Amazon Web Services.

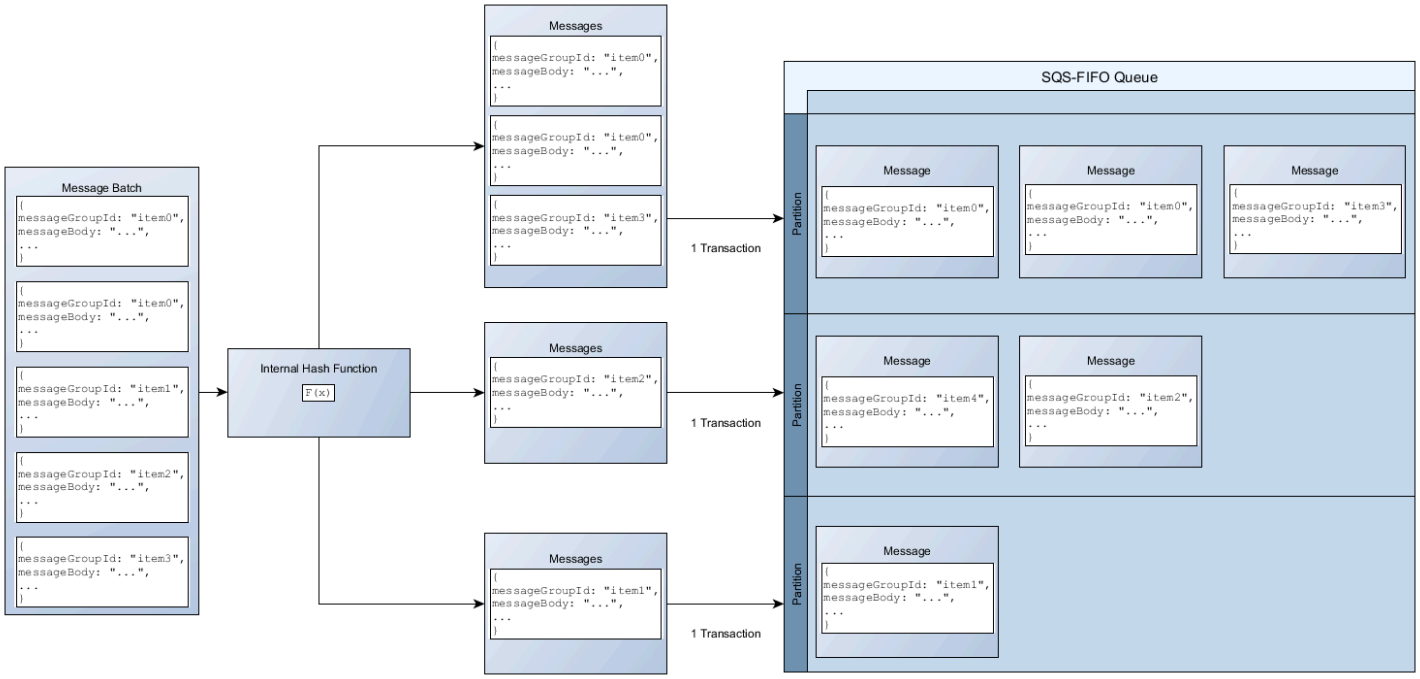
Saat menggunakan API batch, setiap pesan dirutekan berdasarkan proses yang dijelaskan diMendistribusikan data berdasarkan ID grup pesan. Pesan yang dialihkan ke partisi yang sama dikelompokkan dan diproses dalam satu transaksi.

Untuk mengoptimalkan pemanfaatan partisi untuk `SendMessageBatch` API, AWS merekomendasikan batch pesan dengan ID grup pesan yang sama bila memungkinkan.

Untuk mengoptimalkan pemanfaatan partisi untuk `ChangeMessageVisibilityBatch` API `DeleteMessageBatch` dan AWS , sarankan menggunakan `ReceiveMessage` permintaan dengan

`MaxNumberOfMessages` parameter yang disetel ke 10, dan menggabungkan tanda terima yang dikembalikan oleh satu permintaan. `ReceiveMessage`

Dalam contoh berikut, kumpulan pesan dengan berbagai ID grup pesan dikirim. Batch dibagi menjadi tiga kelompok, yang masing-masing diperhitungkan terhadap kuota untuk partisi.



> **Note**
>
> Amazon SQS hanya menjamin bahwa pesan dengan fungsi hash internal ID grup pesan yang sama dikelompokkan dalam permintaan batch. Bergantung pada output dari fungsi hash internal dan jumlah partisi, pesan dengan ID grup pesan yang berbeda mungkin dikelompokkan. Karena fungsi hash atau jumlah partisi dapat berubah sewaktu-waktu, pesan yang dikelompokkan pada satu titik mungkin tidak dikelompokkan nanti.

Mengaktifkan throughput tinggi untuk antrian FIFO di Amazon SQS

Anda dapat mengaktifkan throughput tinggi untuk antrian FIFO baru atau yang sudah ada. Fitur ini mencakup tiga opsi baru saat Anda membuat dan mengedit antrian FIFO:

- Aktifkan FIFO throughput tinggi - Membuat throughput yang lebih tinggi tersedia untuk pesan dalam antrian FIFO saat ini.

- Lingkup deduplikasi — Menentukan apakah deduplikasi terjadi pada tingkat antrian atau grup pesan.
- Batas throughput FIFO — Menentukan apakah kuota throughput pada pesan dalam antrian FIFO ditetapkan pada tingkat antrian atau grup pesan.

Untuk mengaktifkan throughput tinggi untuk antrian FIFO (konsol)

1. Mulai membuat atau mengedit antri an FIFO.
2. Saat menentukan opsi untuk antrian, pilih Aktifkan FIFO throughput tinggi.

   Mengaktifkan throughput tinggi untuk antrian FIFO menetapkan opsi terkait sebagai berikut:

   - Lingkup deduplikasi diatur ke Grup pesan, pengaturan yang diperlukan untuk menggunakan throughput tinggi untuk antrian FIFO.
   - Batas throughput FIFO diatur ke Per ID grup pesan, pengaturan yang diperlukan untuk menggunakan throughput tinggi untuk antrian FIFO.

   Jika Anda mengubah salah satu pengaturan yang diperlukan untuk menggunakan throughput tinggi untuk antrian FIFO, throughput normal berlaku untuk antrian, dan deduplikasi terjadi seperti yang ditentukan.
3. Lanjutkan menentukan semua opsi untuk antrian. Setelah selesai, pilih Buat antrean atau S impan.

Setelah membuat atau mengedit antrian FIFO, Anda dapat mengirim pesan ke sana dan menerima dan menghapus pesan, semuanya pada TPS yang lebih tinggi. Untuk kuota throughput tinggi, lihat Throughput pesan diKuota pesan Amazon SQS.

Antrian FIFO dan pengidentifikasi pesan di Amazon SQS

Bagian ini menjelaskan pengidentifikasi antrian FIFO. Pengidentifikasi ini dapat membantu Anda menemukan dan memanipulasi antrian dan pesan tertentu.

Pengidentifikasi untuk antrian FIFO di Amazon SQS

Untuk informasi selengkapnya tentang pengidentifikasi berikut, lihat Referensi API Layanan Antrian Sederhana Amazon.

Nama antrian dan URL

Saat membuat antrian baru, Anda harus menentukan nama antrian yang unik untuk AWS akun dan wilayah Anda. Amazon SQS menetapkan setiap antrian yang Anda buat pengenal yang disebut URL antrian yang menyertakan nama antrian dan komponen Amazon SQS lainnya. Kapan pun Anda ingin melakukan tindakan pada antrian, Anda memberikan URL antreannya.

Nama antrian FIFO harus diakhiri dengan akhiran. `.fifo` Akhiran dihitung terhadap kuota nama antrian 80 karakter. Untuk menentukan apakah antrian adalah FIFO, Anda dapat memeriksa apakah nama antrian diakhiri dengan akhiran.

Berikut ini adalah URL antrian untuk antrian FIFO bernama `MyQueue` dimiliki oleh pengguna dengan nomor akun AWS. `123456789012`

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

Anda dapat mengambil URL antrian secara terprogram dengan mencantumkan antrian Anda dan mengurai string yang mengikuti nomor akun. Untuk informasi selengkapnya, lihat `ListQueues`.

ID Pesan

Setiap pesan menerima ID pesan yang ditetapkan sistem yang dikembalikan Amazon SQS kepada Anda dalam respons. `SendMessage` Pengenal ini berguna untuk mengidentifikasi pesan. Panjang maksimum ID pesan adalah 100 karakter.

Pegangan tanda terima

Setiap kali Anda menerima pesan dari antrian, Anda menerima tanda terima untuk pesan tersebut. Pegangan ini dikaitkan dengan tindakan menerima pesan, bukan dengan pesan itu sendiri. Untuk menghapus pesan atau mengubah visibilitas pesan, Anda harus memberikan tanda terima (bukan ID pesan). Dengan demikian, Anda harus selalu menerima pesan sebelum Anda dapat menghapusnya (Anda tidak dapat memasukkan pesan ke dalam antrian dan kemudian mengingatnya). Panjang maksimum pegangan tanda terima adalah 1.024 karakter.

> **Important**
>
> Jika Anda menerima pesan lebih dari sekali, setiap kali Anda menerimanya, Anda mendapatkan pegangan tanda terima yang berbeda. Anda harus memberikan tanda terima

yang paling baru diterima saat Anda meminta untuk menghapus pesan (jika tidak, pesan mungkin tidak dihapus).

Berikut ini adalah contoh pegangan tanda terima (rusak di tiga baris).

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

Pengidentifikasi tambahan untuk antrian Amazon SQS FIFO

Untuk informasi selengkapnya tentang pengidentifikasi berikut, lihat Tepat sekali diproses di Amazon SQS dan Referensi API Layanan Antrian Sederhana Amazon.

ID deduplikasi pesan

Token yang digunakan dalam antrian Amazon SQS FIFO untuk mengidentifikasi pesan secara unik dan mencegah duplikasi. Jika beberapa pesan dengan ID deduplikasi yang sama dikirim dalam interval deduplikasi 5 menit, mereka diperlakukan sebagai duplikat, dan hanya satu salinan yang dikirim. Jika Anda tidak menentukan ID deduplikasi dan deduplikasi berbasis konten diaktifkan, Amazon SQS akan menghasilkan ID deduplikasi dengan melakukan hashing pada isi pesan. Mekanisme ini memastikan pengiriman tepat sekali dengan menghilangkan pesan duplikat dalam jangka waktu yang ditentukan.

ID grup pesan

`MessageGroupId`Ini adalah atribut yang hanya digunakan dalam antrian Amazon SQS FIFO (First-In-First-Out) untuk mengatur pesan ke dalam grup yang berbeda. Pesan dalam grup pesan yang sama selalu diproses satu per satu, dalam urutan yang ketat, memastikan bahwa tidak ada dua pesan dari grup yang sama yang diproses secara bersamaan. Antrian standar tidak digunakan `MessageGroupId` dan tidak memberikan jaminan pemesanan. Jika pemesanan ketat diperlukan, gunakan antrian FIFO sebagai gantinya.

Nomor urut

Jumlah besar dan tidak berurutan yang diberikan Amazon SQS untuk setiap pesan.

Kuota Amazon SQS

Topik ini menjelaskan kuota dan batasan untuk Amazon SQS FIFO dan antrian standar, yang merinci bagaimana pengaruhnya terhadap pembuatan antrian, konfigurasi, dan penanganan pesan. Pelajari tentang kendala seperti batas penyimpanan pesan, batas pesan dalam penerbangan, dan ambang batas throughput, serta strategi untuk memaksimalkan efisiensi melalui batching, optimasi panggilan API, dan polling panjang. Topik ini juga mencakup konvensi penamaan, aturan penandaan, dan metode untuk meminta peningkatan kuota untuk memenuhi beban kerja permintaan tinggi, memastikan manajemen antrian yang efektif dan kinerja yang optimal.

Kuota antrian Amazon SQS FIFO

Kuota Amazon SQS

Tabel berikut mencantumkan kuota yang terkait dengan antrian FIFO.

Kuota	Deskripsi
Tunda antrian	Penundaan default (minimum) untuk antrian adalah 0 detik. Maksimal 15 menit.
Antrian terdaftar	1.000 antrian per permintaan. `ListQueues`
Waktu tunggu polling yang panjang	Waktu tunggu polling maksimum yang panjang adalah 20 detik.
Grup pesan	Tidak ada kuota untuk jumlah grup pesan dalam antrian FIFO.
Pesan per antrian (backlog)	Jumlah pesan yang dapat disimpan antrian Amazon SQS tidak terbatas.
Pesan per antrian (dalam penerbangan)	Antrian FIFO mendukung maksimal 120.000 pesan dalam penerbangan (pesan yang diterima oleh konsumen tetapi belum dihapus). Jika batas ini tercapai, Amazon SQS tidak mengembalikan kesalahan, tetapi pemrosesan

Kuota	Deskripsi
	mungkin terpengaruh. Anda dapat meminta peningkatan di luar batas ini dengan menghubungi AWS Support.
Nama antrian	Nama antrian FIFO harus diakhiri dengan akhiran. `.fifo` Akhiran dihitung terhadap kuota nama antrian 80 karakter. Untuk menentukan apakah antrian adalah FIFO, Anda dapat memeriksa apakah nama antrian diakhiri dengan akhiran.
Tag antrian	Kami tidak menyarankan menambahkan lebih dari 50 tag ke antrian. Penandaan mendukung karakter Unicode di UTF-8.
	Tag `Key` diperlukan, tetapi tag `Value` adalah opsional.
	Tag `Key` dan tag `Value` peka huruf besar/kecil.
	Tag `Key` dan tag `Value` dapat menyertakan karakter alfanumerik Unicode di UTF-8 dan spasi putih. Karakter khusus berikut diperbolehkan: `_ . : / = + - @`
	Tag `Key` atau tidak `Value` boleh menyertakan awalan cadangan `aws:` (Anda tidak dapat menghapus kunci tag atau nilai dengan awalan ini).
	`Key`Panjang tag maksimum adalah 128 karakter Unicode di UTF-8. Tag tidak `Key` boleh kosong atau null.
	`Value`Panjang tag maksimum adalah 256 karakter Unicode di UTF-8. Tag `Value` mungkin kosong atau nol.
	Tindakan penandaan dibatasi hingga 30 TPS per. Akun AWS Jika aplikasi Anda membutuhkan throughput yang lebih tinggi, kirimkan permintaan.

Kuota antrian standar Amazon SQS

Tabel berikut mencantumkan kuota yang terkait dengan antrian standar.

Kuota	Deskripsi
Tunda antrian	Penundaan default (minimum) untuk antrian adalah 0 detik. Maksimal 15 menit.
Antrian terdaftar	1.000 antrian per permintaan. `ListQueues`
Waktu tunggu polling yang panjang	Waktu tunggu polling maksimum yang panjang adalah 20 detik.
Pesan per antrian (backlog)	Jumlah pesan yang dapat disimpan antrian Amazon SQS tidak terbatas.
Pesan per antrian (dalam penerbangan)	Untuk sebagian besar antrian standar (tergantung pada lalu lintas antrian dan backlog pesan), bisa ada maksimum sekitar 120.000 dalam pesan penerbangan (diterima dari antrian oleh konsumen, tetapi belum dihapus dari antrian). Jika Anda mencapai kuota ini saat menggunakan polling singkat, Amazon SQS mengembalikan `OverLimit` pesan kesalahan. Jika Anda menggunakan polling panjang, Amazon SQS tidak mengembalikan pesan kesalahan. Untuk menghindari mencapai kuota, Anda harus menghapus pesan dari antrian setelah diproses. Anda juga dapat meningkatkan jumlah antrian yang Anda gunakan untuk memproses pesan Anda. Untuk meminta peningkatan kuota, kirimkan permintaan dukungan.
Nama antrian	Nama antrian dapat memiliki hingga 80 karakter. Karakter berikut diterima: karakter alfanumerik, tanda hubung (-), dan garis bawah (). _

Kuota	Deskripsi
	Note Nama antrian peka huruf besar/kecil (misalnya, `Test-queue` dan antrian `test-queue` yang berbeda).
Tag antrian	Kami tidak menyarankan menambahkan lebih dari 50 tag ke antrian. Penandaan mendukung karakter Unicode di UTF-8.
	Tag `Key` diperlukan, tetapi tag `Value` adalah opsional.
	Tag `Key` dan tag `Value` peka huruf besar/kecil.
	Tag `Key` dan tag `Value` dapat menyertakan karakter alfanumerik Unicode di UTF-8 dan spasi putih. Karakter khusus berikut diperbolehkan: `_ . : / = + - @`
	Tag `Key` atau tidak `Value` boleh menyertakan awalan cadangan `aws:` (Anda tidak dapat menghapus kunci tag atau nilai dengan awalan ini).
	`Key`Panjang tag maksimum adalah 128 karakter Unicode di UTF-8. Tag tidak `Key` boleh kosong atau null.
	`Value`Panjang tag maksimum adalah 256 karakter Unicode di UTF-8. Tag `Value` mungkin kosong atau nol.
	Tindakan penandaan dibatasi hingga 30 TPS per. Akun AWS Jika aplikasi Anda membutuhkan throughput yang lebih tinggi, kirimkan permintaan.

Kuota pesan Amazon SQS

Tabel berikut mencantumkan kuota yang terkait dengan pesan.

Kuota	Deskripsi
ID pesan berkelompok	ID pesan batch dapat memiliki hingga 80 karakter. Karakter berikut diterima: karakter alfanumerik, tanda hubung (-), dan garis bawah (). _
Atribut pesan	Pesan dapat berisi hingga 10 atribut metadata.
Batch pesan	Permintaan batch pesan tunggal dapat mencakup maksimal 10 pesan. Untuk informasi selengkapnya, lihat Mengkonfigurasi Amazon SQSBuffered AsyncClient di bagian Tindakan batch Amazon SQS.
Isi pesan	Pesan hanya dapat menyertakan XML, JSON, dan teks yang tidak diformat. Karakter Unicode berikut diperbolehkan: #x9 #xA | #xD | #xD7FF | #x20 #xE000 ke #xFFFD | #x10000 ke #x10FFFF Setiap karakter yang tidak termasuk dalam daftar ini ditolak. Untuk informasi selengkapnya, lihat spesifikasi W3C untuk karakter.
ID grup pesan	MessageGroupId diperlukan untuk antrian FIFO. Jika Anda tidak memberikan MessageGroupId saat mengirim pesan ke antrian FIFO, tindakan gagal. Dalam antrian standar, penggunaan MessageGroupId memungkinkan antrian yang adil. Sebaiknya sertakan a MessageGroupId di semua pesan saat menggunakan antrian yang adil. Panj MessageGroupId angnya 128 karakter. Nilai yang valid: karakter alfanumerik dan tanda baca(!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~) .
Penyimpanan pesan	Secara default, pesan disimpan selama 4 hari. Minimal adalah 60 detik (1 menit). Maksimal adalah 1.209.600 detik (14 hari).
Throughput pesan	Antrian standar

Kuota	Deskripsi
	Antrian standar mendukung jumlah panggilan API yang sangat tinggi dan hampir tidak terbatas per detik, per tindakan (`SendMessage`, `ReceiveMessage`, atau `DeleteMessage`). Throughput tinggi ini membuatnya ideal untuk kasus penggunaan yang memerlukan pemrosesan pesan dalam jumlah besar dengan cepat, seperti streaming data real-time atau aplikasi skala besar. Sementara antrian standar berskala secara otomatis sesuai permintaan, penting untuk memantau pola penggunaan untuk memastikan kinerja yang optimal, terutama di wilayah dengan beban kerja yang lebih tinggi.

Kuota	Deskripsi
	Antrian FIFO • Setiap partisi dalam antrian FIFO dibatasi hingga 300 transaksi per detik, per tindakan API (`SendMessage` ,`ReceiveMessage` , dan`DeleteMessage`). Batas ini berlaku khusus untuk mode throughput non-tinggi. Dengan beralih ke mode throughput tinggi, Anda dapat melampaui batas default ini. Untuk mengaktifkan mode throughput tinggi, lihatMengaktifkan throughput tinggi untuk antrian FIFO di Amazon SQS. • Jika Anda menggunakan batching, antrian FIFO dengan throughput non-tinggi mendukung hingga 3.000 pesan per detik, per tindakan API (,`SendMessage` , `ReceiveMessage` dan). `DeleteMessage` 3.000 pesan per detik mewakili 300 panggilan API, masing-masing dengan batch 10 pesan. Throughput tinggi untuk antrian FIFO Batas Amazon SQS FIFO didasarkan pada jumlah permintaan API, bukan batas pesan. Untuk mode throughput tinggi, batas permintaan API ini adalah sebagai berikut: Batas throughput transaksi (panggilan Non-batching API) Batasan ini menentukan seberapa sering setiap operasi API (seperti SendMessage, ReceiveMessage, atau DeleteMessage) dapat dilakukan secara independen, memastikan kinerja sistem yang efisien dalam transaksi per detik yang diizinkan (TPS). Batas berikut didasarkan pada panggilan API non-batch:

Kuota	Deskripsi
	• AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia): Hingga 70.000 transaksi per detik (TPS). • AS Timur (Ohio) dan Eropa (Frankfurt): Hingga 19.000 TPS. • Asia Pasifik (Mumbai), Asia Pasifik (Singapura), Asia Pasifik (Sydney), Asia Pasifik (Tokyo), dan Eropa (Spanyol): Hingga 9.000 TPS. • Eropa (London) dan Amerika Selatan (São Paulo): Hingga 4.500 TPS. • Semua lainnya Wilayah AWS: Throughput default 2.400 TPS. Memaksimalkan throughput dengan batching Memproses beberapa pesan dalam satu panggilan API, yang secara signifikan meningkatkan efisiensi. Alih-alih menangani setiap pesan secara individual, batching memungkinkan Anda mengirim, menerima, atau menghapus hingga 10 pesan dalam satu permintaan API. Ini mengurangi jumlah total panggilan API, memungkin kan Anda memproses lebih banyak pesan per detik sambil tetap berada dalam batas transaksi (TPS) untuk wilayah tersebut, memaksimalkan throughput dan kinerja sistem. Untuk informasi selengkapnya, lihat Meningkat kan throughput menggunakan penskalaan horizontal dan batching aksi dengan Amazon SQS. Batas berikut didasarkan pada panggilan API batch: • AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia): Hingga 700.000 pesan per detik (10x batas non-batch 70.000 TPS).

Kuota	Deskripsi
	• AS Timur (Ohio) dan Eropa (Frankfurt): Hingga 190.000 pesan per detik. • Asia Pasifik (Mumbai), Asia Pasifik (Singapura), Asia Pasifik (Sydney), Asia Pasifik (Tokyo), dan Eropa (Spanyol): Hingga 90.000 pesan per detik. • Eropa (London) dan Amerika Selatan (São Paulo): Hingga 45.000 pesan per detik. • Semua lainnya Wilayah AWS: Hingga 24.000 pesan per detik. Mengoptimalkan throughput di luar batching Meskipun batching dapat sangat meningkatkan throughput, penting untuk mempertimbangkan strategi lain untuk mengoptimalkan kinerja FIFO: • Mendistribusikan pesan di beberapa ID grup pesan — Karena pesan dalam satu grup diproses secara berurutan, mendistribusikan beban kerja Anda di beberapa grup pesan memungkinkan paralelisme yang lebih baik dan throughput keseluruhan yang lebih tinggi. Untuk informasi selengkapnya, lihat Partisi dan distribusi data untuk throughput tinggi untuk antrian SQS FIFO. • Penggunaan panggilan API yang efisien — Minimalkan panggilan API yang tidak perlu, seperti perubahan visibilitas yang sering atau penghapusan pesan berulang, untuk mengoptimalkan penggunaan TPS yang tersedia dan meningkatkan efisiensi. • Gunakan penerimaan jajak pendapat panjang — Gunakan polling panjang dengan meny `WaitTimeSeconds` etel permintaan penerimaan Anda untuk mengurangi tanggapan kosong ketika tidak ada pesan yang tersedia, menurunkan panggilan API yang tidak

Kuota	Deskripsi
	perlu, dan memanfaatkan kuota TPS Anda dengan lebih baik. • Meminta peningkatan throughput — Jika aplikasi Anda memerlukan throughput yang lebih tinggi dari batas default, minta peningkatan menggunakan konsol Kuota Layanan. Ini mungkin diperlukan untuk beban kerja dengan permintaan tinggi atau di wilayah dengan batas default yang lebih rendah. Untuk mengaktifkan mode throughput tinggi, lihatMengaktifkan throughput tinggi untuk antrian FIFO di Amazon SQS.
Pengatur waktu pesan	Penundaan default (minimum) untuk pesan adalah 0 detik. Maksimal adalah 15 menit.
Ukuran pesan	Ukuran pesan minimum adalah 1 byte (1 karakter). Maksimal adalah 1.048.576 byte (1 MiB). Untuk mengirim pesan yang lebih besar dari 1 MiB, Anda dapat menggunakan Amazon SQS Extended Client Library untuk Java dan Amazon SQS Extended Client Library untuk Python. Pustaka ini memungkinkan Anda mengirim pesan Amazon SQS yang berisi referensi ke payload pesan di Amazon S3. Ukuran payload maksimum adalah 2 GB. **Note** Perpustakaan yang diperluas ini hanya berfungsi untuk klien sinkron.
Batas waktu visibilitas pesan	Batas waktu visibilitas default untuk pesan adalah 30 detik. Minimal adalah 0 detik. Maksimal adalah 12 jam.

Kuota	Deskripsi
Informasi kebijakan	Kuota maksimum adalah 8.192 byte, 20 pernyataan, 50 prinsipal, atau 10 kondisi. Lihat informasi yang lebih lengkap di Kuota kebijakan Amazon SQS.

Kuota kebijakan Amazon SQS

Tabel berikut mencantumkan kuota yang terkait dengan kebijakan.

Nama	Maksimum
Byte	8,192
Ketentuan	10
Pengguna utama	50
Pernyataan	20
Tindakan per pernyataan	7

Fitur dan kemampuan Amazon SQS

Topik ini menyediakan fitur yang umum digunakan di Amazon SQS untuk mengelola antrian pesan, mengoptimalkan kinerja, memastikan pengiriman pesan yang andal, dan menangani pemrosesan pesan secara efisien.

Menggunakan antrian huruf mati di Amazon SQS

Amazon SQS mendukung antrian huruf mati (DLQ), yang antrian sumber dapat menargetkan pesan yang tidak berhasil diproses. DLQ berguna untuk men-debug aplikasi Anda karena Anda dapat mengisolasi pesan yang tidak dikonsumsi untuk menentukan mengapa pemrosesan tidak berhasil. Antrian huruf mati harus sama Akun AWS dan Wilayah sebagai antrian sumber. Setelah pesan berada dalam antrean huruf mati, Anda dapat:

- Periksa log untuk pengecualian yang mungkin menyebabkan pesan dipindahkan ke antrian huruf mati.
- Menganalisis isi pesan yang dipindahkan ke antrian surat mati untuk mendiagnosis masalah aplikasi.
- Tentukan apakah Anda telah memberi konsumen Anda waktu yang cukup untuk memproses pesan.
- Pindahkan pesan dari antrian huruf mati menggunakan penggerak ulang antrian huruf mati.

Anda harus terlebih dahulu membuat antrian baru sebelum mengonfigurasinya sebagai antrian huruf mati. Untuk informasi tentang mengonfigurasi antrian huruf mati menggunakan konsol Amazon SQS, lihat. Konfigurasikan antrian huruf mati menggunakan konsol Amazon SQS Untuk bantuan dengan antrian surat mati, seperti cara mengonfigurasi alarm untuk setiap pesan yang dipindahkan ke antrian huruf mati, lihat. Membuat alarm untuk antrian huruf mati menggunakan Amazon CloudWatch

> **Note**
>
> Jangan gunakan antrian huruf mati dengan antrian FIFO jika Anda tidak ingin merusak urutan pesan atau operasi yang tepat. Misalnya, jangan gunakan antrian huruf mati dengan instruksi dalam Daftar Keputusan Edit (EDL) untuk rangkaian pengeditan video, di mana mengubah urutan pengeditan mengubah konteks pengeditan berikutnya.

Menggunakan kebijakan untuk antrian surat mati

Gunakan kebijakan redrive untuk menentukan. `maxReceiveCount` `maxReceiveCount`Ini adalah berapa kali konsumen dapat menerima pesan dari antrian sumber sebelum dipindahkan ke antrian surat mati. Misalnya, jika `maxReceiveCount` disetel ke nilai rendah seperti 1, satu kegagalan untuk menerima pesan akan menyebabkan pesan pindah ke antrian huruf mati. Untuk memastikan bahwa sistem Anda tahan terhadap kesalahan, atur cukup `maxReceiveCount` tinggi untuk memungkinkan percobaan ulang yang cukup.

Untuk antrian standar dengan kebijakan redrive yang `maxReceiveCount` lebih besar dari 3, jika pesan diterima 3 kali atau lebih tanpa dihapus, SQS memindahkannya ke bagian belakang antrian. `ApproximateAgeOfOldestMessage`Metrik kemudian mencerminkan usia pesan berikutnya yang belum melebihi ambang batas ini.

Kebijakan redrive allow menentukan antrian sumber mana yang dapat mengakses antrian huruf mati. Anda dapat memilih apakah akan mengizinkan semua antrian sumber, mengizinkan antrian sumber tertentu, atau menolak semua antrian sumber penggunaan antrian huruf mati. Default memungkinkan semua antrian sumber untuk menggunakan antrian huruf mati. Jika Anda memilih untuk mengizinkan antrian tertentu menggunakan `byQueue` opsi, Anda dapat menentukan hingga 10 antrian sumber menggunakan antrian sumber Amazon Resource Name (ARN). Jika Anda menentukan`denyAll`, antrian tidak dapat digunakan sebagai antrian huruf mati.

Memahami periode retensi pesan untuk antrian surat mati

Untuk antrian standar, kedaluwarsa pesan selalu didasarkan pada stempel waktu enqueue aslinya. Ketika pesan dipindahkan ke antrian huruf mati, stempel waktu enqueue tidak berubah. `ApproximateAgeOfOldestMessage`Metrik menunjukkan kapan pesan dipindahkan ke antrian huruf mati, bukan saat pesan awalnya dikirim. Misalnya, asumsikan bahwa pesan menghabiskan 1 hari dalam antrian asli sebelum dipindahkan ke antrian huruf mati. Jika periode retensi antrian surat mati adalah 4 hari, pesan akan dihapus dari antrian surat mati setelah 3 hari dan 3 hari. `ApproximateAgeOfOldestMessage` Dengan demikian, ini adalah praktik terbaik untuk selalu mengatur periode retensi antrian huruf mati menjadi lebih lama dari periode retensi antrian asli.

Untuk antrian FIFO, stempel waktu enqueue akan disetel ulang saat pesan dipindahkan ke antrian huruf mati. `ApproximateAgeOfOldestMessage`Metrik menunjukkan kapan pesan dipindahkan ke antrian huruf mati. Dalam contoh yang sama di atas, pesan dihapus dari antrian surat mati setelah empat hari dan empat hari. `ApproximateAgeOfOldestMessage`

Konfigurasikan antrian huruf mati menggunakan konsol Amazon SQS

Antrian surat mati (DLQ) adalah antrian yang menerima pesan yang tidak berhasil diproses dari antrian lain, yang dikenal sebagai antrian sumber. Amazon SQS tidak membuat antrian huruf mati secara otomatis. Anda harus terlebih dahulu membuat antrian sebelum menggunakannya sebagai antrian huruf mati. Saat mengonfigurasi DLQ, tipe antrian harus sesuai dengan tipe antrian sumber—antrian FIFO hanya dapat menggunakan DLQ FIFO, dan antrian standar hanya dapat menggunakan DLQ standar. Anda dapat mengonfigurasi antrian huruf mati saat membuat atau mengedit antrian. Untuk detail selengkapnya, lihat Menggunakan antrian huruf mati di Amazon SQS.

Untuk mengonfigurasi antrian huruf mati untuk antrian yang ada (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pilih antrian sumber (antrian yang akan mengirim pesan gagal ke antrian huruf mati), lalu pilih Edit.
4. Gulir ke bagian antrian Dead-letter dan beralih Enabled.
5. Di bawah Pengaturan antrian huruf mati, pilih Nama Sumber Daya Amazon (ARN) dari antrian yang ada yang ingin Anda gunakan sebagai antrean huruf mati.
6. Tetapkan nilai penerimaan Maksimum, yang menentukan berapa kali pesan dapat diterima sebelum dikirim ke antrian surat mati (rentang yang valid: 1 hingga 1.000).
7. Pilih Simpan.

Pelajari cara mengonfigurasi redrive antrian huruf mati di Amazon SQS

Gunakan redrive antrian huruf mati untuk memindahkan pesan yang tidak dikonsumsi dari antrian huruf mati ke tujuan lain untuk diproses. Secara default, redrive antrian huruf mati memindahkan pesan dari antrian huruf mati ke antrian sumber. Namun, Anda juga dapat mengonfigurasi antrian lain sebagai tujuan redrive jika kedua antrian adalah tipe yang sama. Misalnya, jika antrian huruf mati adalah antrian FIFO, antrian tujuan redrive harus berupa antrian FIFO juga. Selain itu, Anda dapat mengonfigurasi kecepatan redrive untuk mengatur kecepatan di mana Amazon SQS memindahkan pesan.

> **Note**
>
> Ketika pesan dipindahkan dari antrian FIFO ke DLQ FIFO, ID deduplikasi pesan asli akan diganti dengan ID pesan asli. Ini untuk memastikan bahwa deduplikasi DLQ tidak akan mencegah penyimpanan dua pesan independen yang kebetulan berbagi ID deduplikasi.

Antrian surat mati menggerakkan ulang pesan sesuai urutan penerimaannya, dimulai dengan pesan tertua. Namun, antrian tujuan menyerap pesan yang digerakkan ulang, serta pesan baru dari produsen lain, sesuai dengan urutan penerimaannya. Misalnya, jika produser mengirim pesan ke antrian FIFO sumber saat secara bersamaan menerima pesan yang digerakkan ulang dari antrian surat mati, pesan yang digerakkan ulang akan terjalin dengan pesan baru dari produser.

> **Note**
>
> Tugas redrive mengatur ulang periode retensi. Semua pesan yang digerakkan ulang dianggap pesan baru dengan pesan baru `messageID` dan ditetapkan ke pesan `enqueueTime` yang digerakkan ulang.

Mengonfigurasi redrive antrian huruf mati untuk antrean standar yang ada menggunakan Amazon SQS API

Anda dapat mengonfigurasi redrive antrian huruf mati menggunakan tindakan`StartMessageMoveTask`,`ListMessageMoveTasks`, dan API: `CancelMessageMoveTask`

Tindakan API	Deskripsi
`StartMessageMoveTask`	Memulai tugas asinkron untuk memindahkan pesan dari antrian sumber tertentu ke antrian tujuan tertentu.
`ListMessageMoveTasks`	Mendapat tugas pergerakan pesan terbaru (hingga 10) di bawah antrian sumber tertentu.

Tindakan API	Deskripsi
CancelMessageMoveTask	Membatalkan tugas pergerakan pesan tertentu. Pergerakan pesan hanya dapat dibatalkan ketika status saat ini sedang BERJALAN.

Mengonfigurasi redrive antrian huruf mati untuk antrean standar yang ada menggunakan konsol Amazon SQS

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pilih nama antrian yang telah Anda konfigurasikan sebagai antrian huruf mati.
4. Pilih Mulai DLQ redrive.
5. Di bawah konfigurasi Recrive, untuk tujuan Pesan, lakukan salah satu hal berikut:
   - Untuk mengarahkan ulang pesan ke antrean sumbernya, pilih Recrive to source queue (s).
   - Untuk mengarahkan ulang pesan ke antrian lain, pilih Recrive ke tujuan kustom. Kemudian, masukkan Nama Sumber Daya Amazon (ARN) dari antrian tujuan yang ada.
6. Di bawah Pengaturan kontrol kecepatan, pilih salah satu dari berikut ini:
   - Sistem dioptimalkan - Dorong ulang pesan antrian huruf mati dengan jumlah maksimum pesan per detik.
   - Kecepatan maks khusus - Dorong ulang pesan antrian huruf mati dengan kecepatan maksimum pesan per detik khusus. Tarif maksimum yang diizinkan adalah 500 pesan per detik.
     - Disarankan untuk memulai dengan nilai kecil untuk kecepatan maks Kustom dan memverifikasi bahwa antrian sumber tidak kewalahan dengan pesan. Dari sana, secara bertahap tingkatkan nilai kecepatan maks Kustom, terus memantau status antrian sumber.
7. Setelah Anda selesai mengonfigurasi redrive antrian huruf mati, pilih Recrive messages.

> **Important**
>
> Amazon SQS tidak mendukung pemfilteran dan modifikasi pesan saat mengarahkan ulang pesan dari antrian huruf mati.

Tugas redrive antrian huruf mati dapat berjalan maksimal 36 jam. Amazon SQS mendukung maksimal 100 tugas redrive aktif per akun.

8. Jika Anda ingin membatalkan tugas redrive pesan, pada halaman Detail untuk antrian Anda, pilih Batalkan drive ulang DLQ. Saat membatalkan redrive pesan yang sedang berlangsung, pesan apa pun yang telah berhasil dipindahkan ke antrian tujuan pindahnya akan tetap berada dalam antrean tujuan.

Mengkonfigurasi izin antrian untuk redrive antrian huruf mati

Anda dapat memberi pengguna akses ke tindakan antrean huruf mati tertentu dengan menambahkan izin ke kebijakan Anda. Izin minimum yang diperlukan untuk redrive antrian huruf mati adalah sebagai berikut:

Izin Minimum	Metode API yang diperlukan
Untuk memulai redrive pesan	• Tambahkan `sqs:StartMessageMoveTask` ,, `sqs:ReceiveMessage` `sqs:DeleteMessage` , dan `sqs:GetQueueAttributes` antrian huruf mati. Jika antrian huruf mati atau antrian sumber asli dienkripsi (juga dikenal sebagai antrian SSE), `kms:Decrypt` untuk kunci KMS apa pun yang telah digunakan untuk mengenkripsi pesan juga diperlukan. • Tambahkan antrian tujuan. `sqs:SendMessage` Jika antrian tujuan dienkripsi, `kms:GenerateDataKey` dan `kms:Decrypt` juga diperlukan.
Untuk membatalkan redrive pesan yang sedang berlangsung	• Tambahkan `sqs:CancelMessageMoveTask` ,, `sqs:ReceiveMessage` `sqs:DeleteMessage` , dan `sqs:GetQueueAttributes` antrian huruf mati. Jika antrian huruf mati dienkripsi (juga dikenal sebagai antrian SSE), juga diperlukan. `kms:Decrypt`
Untuk menampilkan status pemindahan pesan	• Tambahkan `sqs:ListMessageMoveTasks` dan `sqs:GetQueueAttributes` antrian huruf mati.

Untuk mengonfigurasi izin untuk pasangan antrian terenkripsi (antrian sumber dengan antrian huruf mati)

Gunakan langkah-langkah berikut untuk mengonfigurasi izin minimum untuk redrive antrian huruf mati (DLQ):

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Kebijakan.
3. Buat kebijakan baru dan tambahkan izin berikut. Lampirkan kebijakan ke pengguna IAM atau peran yang akan melakukan operasi redrive.

   - Izin untuk DLQ (antrian sumber):
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:GetQueueAttributes`
     - `sqs:ListDeadLetterSourceQueues`
     - Tentukan ARN Sumber Daya dari DLQ (antrian sumber) (misalnya, “arn:aws:sqs::: “). *<DLQ_region> <DLQ_accountId> <DLQ_name>*
   - Izin untuk antrian tujuan:
     - `sqs:SendMessage`
     - Tentukan `Resource` ARN antrian tujuan (misalnya, “arn:aws:sqs: “). *<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*
   - Izin untuk kunci KMS:
     - `kms:Decrypt`(Diperlukan untuk mendekripsi pesan di DLQ.)
     - `kms:GenerateDataKey`(Diperlukan untuk mengenkripsi pesan dalam antrian tujuan.)
       - `Resource` ARNs:
         - ARN dari kunci KMS digunakan untuk mengenkripsi pesan di DLQ (antrian sumber) (misalnya, “arn:aws:kms: ::key/ “). *<region> <accountId> <SourceQueueKeyId>*
         - ARN dari kunci KMS digunakan untuk mengenkripsi pesan dalam antrian tujuan (misalnya, “arn:aws:kms: ::key/ “). *<region> <accountId> <DestinationQueueKeyId>*

Kebijakan akses Anda harus menyerupai yang berikut:

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>"
            ]
        }
    ]
}
```

Untuk mengonfigurasi izin menggunakan pasangan antrian yang tidak terenkripsi (antrian sumber dengan antrian huruf mati)

Ikuti langkah-langkah ini untuk mengonfigurasi izin minimum yang diperlukan untuk menangani antrian huruf mati (DLQ) standar yang tidak terenkripsi. Izin minimum yang diperlukan adalah menerima, menghapus, dan mendapatkan atribut dari antrian huruf mati, dan mengirim atribut ke antrian sumber.

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Kebijakan.
3. Buat kebijakan baru dan tambahkan izin berikut. Lampirkan kebijakan ke pengguna IAM atau peran yang akan melakukan operasi redrive.

   - Izin untuk DLQ (antrian sumber):
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:ListDeadLetterSourceQueues`
     - Tentukan ARN Sumber Daya dari DLQ (antrian sumber) (misalnya, “arn:aws:sqs::: “). *<DLQ_region>* *<DLQ_accountId>* *<DLQ_name>*
   - Izin untuk antrian tujuan:
     - `sqs:SendMessage`
     - Tentukan `Resource` ARN antrian tujuan (misalnya, “arn:aws:sqs: “). *<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*

Kebijakan akses Anda harus menyerupai yang berikut:

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"
                }
            }
        }
    ]
}
```

Menggunakan redrive antrian huruf mati dengan kontrol akses titik akhir VPC

Ketika Anda membatasi akses antrian ke spesifik VPCs menggunakan `aws:sourceVpc` kondisi, Anda perlu membuat pengecualian untuk AWS layanan untuk mengaktifkan fungsionalitas redrive antrian huruf mati (DLQ). Ini karena layanan Amazon SQS beroperasi di luar VPC Anda saat memindahkan pesan.

Untuk mengizinkan operasi redrive DLQ, tambahkan `aws:CalledViaLast` kondisi ke kebijakan antrian Anda. Hal ini memungkinkan Amazon SQS untuk melakukan panggilan API atas nama Anda sambil mempertahankan pembatasan VPC untuk akses langsung.

Untuk mengizinkan akses terbatas VPC dan redrive DLQ:

1. Gunakan `aws:CalledViaLast` kondisi dalam kebijakan antrian Anda.
2. Menerapkan kebijakan ke antrian sumber dan DLQ
3. Pertahankan pembatasan VPC untuk akses langsung dari sumber lain

Berikut adalah contoh kebijakan yang menerapkan persyaratan ini:

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"
        }
      }
    }
  ]
}
```

- Ganti nilai placeholder dengan nilai aktual Anda
- Kebijakan ini menggunakan pernyataan “tolak” dengan kondisi, yang lebih aman daripada menggunakan pernyataan “izinkan”
- `StringNotEqualsIfExists`Operator menangani kasus di mana kunci kondisi mungkin tidak ada dalam konteks permintaan.

Atau, Anda dapat menggunakan tombol `aws:ViaAWSService` kondisi untuk mengizinkan akses berbasis layanan sambil mempertahankan batasan VPC. Kunci kondisi ini menunjukkan apakah permintaan berasal dari AWS layanan. Berikut adalah contoh kebijakan yang menggunakan `aws:ViaAWSService` alih-alih`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"
        }
      }
    }
  ]
}
```

BoolIfExists Operator dengan `aws:ViaAWSService` kondisi memastikan bahwa permintaan diizinkan ketika mereka datang dari layanan sambil mempertahankan pembatasan VPC untuk akses langsung. Ini bisa lebih mudah dipahami dan dipelihara, karena secara langsung memeriksa apakah permintaan dibuat oleh AWS layanan daripada memeriksa layanan mana yang melakukan panggilan terakhir.

Untuk informasi selengkapnya tentang kunci kondisi yang digunakan dalam IAM dan kebijakan sumber daya, lihat elemen kebijakan IAM JSON: Kondisi.

CloudTrail persyaratan pembaruan dan izin untuk redrive antrian surat mati Amazon SQS

Pada 8 Juni 2023, Amazon SQS memperkenalkan redrive antrian huruf mati (DLQ) untuk SDK dan (CLI). AWS AWS Command Line Interface Kemampuan ini merupakan tambahan untuk redrive DLQ yang sudah didukung untuk konsol. AWS Jika sebelumnya Anda telah menggunakan AWS konsol untuk mengarahkan ulang pesan antrian huruf mati, Anda mungkin terpengaruh oleh perubahan berikut:

CloudTrail penggantian nama acara

Pada 15 Oktober 2023, nama CloudTrail acara untuk redrive antrian huruf mati akan berubah di konsol Amazon SQS. Jika Anda telah menyetel alarm untuk CloudTrail acara ini, Anda harus memperbaruinya sekarang. Berikut ini adalah nama CloudTrail acara baru untuk DLQ redrive:

Nama acara sebelumnya	Nama acara baru
`CreateMoveTask`	`StartMessageMoveTask`
`CancelMoveTask`	`CancelMessageMoveTask`

Izin yang diperbarui

Termasuk dengan rilis SDK dan CLI, Amazon SQS juga telah memperbarui izin antrian untuk redrive DLQ untuk mematuhi praktik terbaik keamanan. Gunakan jenis izin antrian berikut untuk mengarahkan ulang pesan dari Anda. DLQs

1. Izin berbasis tindakan (pembaruan untuk tindakan API DLQ)
2. Izin kebijakan Amazon SQS terkelola
3. Kebijakan izin yang menggunakan sqs:* wildcard

> **Important**
>
> Untuk menggunakan redrive DLQ untuk SDK atau CLI, Anda harus memiliki kebijakan izin redrive DLQ yang cocok dengan salah satu opsi di atas.

Jika izin antrian Anda untuk drive ulang DLQ tidak cocok dengan salah satu opsi di atas, Anda harus memperbarui izin Anda sebelum 31 Agustus 2023. Antara sekarang dan 31 Agustus 2023, akun Anda akan dapat menggerakkan ulang pesan menggunakan izin yang Anda konfigurasikan menggunakan AWS konsol hanya di wilayah tempat Anda sebelumnya menggunakan drive ulang DLQ. Misalnya, Anda memiliki “Akun A” di us-east-1 dan eu-west-1. “Akun A” digunakan untuk menggerakkan ulang pesan di AWS konsol di us-east-1 sebelum 8 Juni 2023, tetapi tidak di eu-west-1. Antara 8 Juni 2023 dan 31 Agustus 2023, jika izin kebijakan “Akun A” tidak cocok dengan salah satu opsi di atas, itu hanya dapat digunakan untuk menggerakkan ulang pesan di konsol AWS di us-east-1, dan bukan di eu-west-1.

> **Important**
>
> Jika izin penggerak ulang DLQ Anda tidak cocok dengan salah satu opsi ini setelah 31 Agustus 2023, akun Anda tidak akan lagi dapat menggerakkan ulang pesan DLQ menggunakan konsol. AWS
> Namun, jika Anda menggunakan fitur redrive DLQ di AWS Konsol selama Agustus 2023, Anda memiliki ekstensi hingga 15 Oktober 2023 untuk mengadopsi izin baru sesuai dengan salah satu opsi ini.
> Untuk informasi selengkapnya, lihat the section called “Mengidentifikasi kebijakan yang terkena dampak”.

Berikut ini adalah contoh izin antrian untuk setiap opsi redrive DLQ. Saat menggunakan antrian terenkripsi sisi server (SSE), izin kunci yang sesuai diperlukan. AWS KMS

Berbasis aksi

JSON

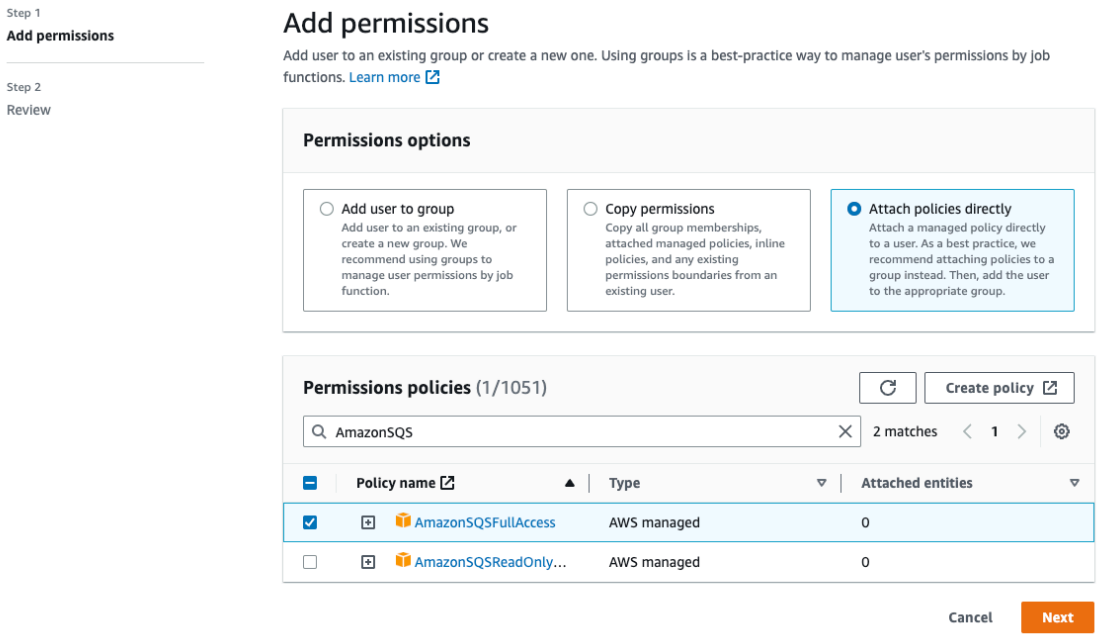
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:StartMessageMoveTask",
```

```
                "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>"
        }
    ]
}
```

Kebijakan terkelola

Kebijakan terkelola berikut berisi izin diperbarui yang diperlukan:

- Amazon SQSFull Access - Termasuk tugas redrive antrian huruf mati berikut: mulai, batalkan, dan daftar.
- Amazon SQSRead OnlyAccess - Menyediakan akses hanya-baca, dan menyertakan daftar tugas redrive antrian huruf mati.



Kebijakan Izin yang menggunakan sqs* wildcard

JSON

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

Mengidentifikasi kebijakan yang terkena dampak

Jika Anda menggunakan kebijakan terkelola pelanggan (CMPs), Anda dapat menggunakan AWS CloudTrail dan IAM untuk mengidentifikasi kebijakan yang dipengaruhi oleh pembaruan izin antrian.

> **Note**
>
> Jika Anda menggunakan `AmazonSQSFullAccess` dan`AmazonSQSReadOnlyAccess`, tidak ada tindakan lebih lanjut yang diperlukan.

1. Masuk ke AWS CloudTrail konsol.
2. Pada halaman Riwayat acara, di bawah Cari atribut, gunakan menu tarik-turun untuk memilih nama Acara. Kemudian, cari`CreateMoveTask`.
3. Pilih acara untuk membuka halaman Detail. Di bagian Catatan peristiwa, ambil `UserName` atau `RoleName` dari `userIdentity` ARN.
4. Masuk ke konsol IAM.
   - Untuk pengguna, pilih Pengguna. Pilih pengguna dengan yang `UserName` diidentifikasi pada langkah sebelumnya.
   - Untuk peran, pilih Peran. Cari pengguna dengan yang `RoleName` diidentifikasi pada langkah sebelumnya.
5. Pada halaman Detail, di bagian Izin, tinjau kebijakan apa pun dengan `sqs:` awalan di`Action`, atau tinjau kebijakan yang menentukan antrean Amazon SQS. `Resource`

Membuat alarm untuk antrian huruf mati menggunakan Amazon CloudWatch

Siapkan CloudWatch alarm untuk memantau pesan dalam antrian huruf mati menggunakan metrik. `ApproximateNumberOfMessagesVisible` Untuk petunjuk mendetail, lihat Membuat CloudWatch alarm untuk metrik Amazon SQS. Saat alarm dipicu, yang menunjukkan pesan telah dipindahkan ke antrian huruf mati, Anda dapat melakukan polling antrian untuk meninjau dan mengambilnya.

Metadata pesan untuk Amazon SQS

Gunakan atribut pesan untuk menambahkan metadata khusus ke pesan Amazon SQS untuk aplikasi Anda. Gunakan atribut sistem pesan untuk menyimpan metadata untuk integrasi dengan yang lain Layanan AWS, seperti AWS X-Ray.

Atribut pesan Amazon SQS

Amazon SQS memungkinkan Anda menyertakan metadata terstruktur (seperti stempel waktu, data geospasial, tanda tangan, dan pengidentifikasi) dengan pesan menggunakan atribut pesan. Setiap pesan dapat memiliki hingga 10 atribut. Atribut pesan bersifat opsional dan terpisah dari badan pesan (namun, mereka dikirim bersamanya). Konsumen Anda dapat menggunakan atribut pesan untuk menangani pesan dengan cara tertentu tanpa harus memproses isi pesan terlebih dahulu. Untuk informasi tentang mengirim pesan dengan atribut menggunakan konsol Amazon SQS, lihatMengirim pesan dengan atribut menggunakan Amazon SQS.

> **Note**
>
> Jangan bingung atribut pesan dengan atribut sistem pesan: Meskipun Anda dapat menggunakan atribut pesan untuk melampirkan metadata khusus ke pesan Amazon SQS untuk aplikasi Anda, Anda dapat menggunakan atribut sistem pesan untuk menyimpan metadata untuk AWS layanan lain, seperti AWS X-Ray.

Topik

- Komponen atribut pesan
- Jenis data atribut pesan
- Menghitung ringkasan pesan MD5 untuk atribut pesan

Komponen atribut pesan

> **Important**
>
> Semua komponen atribut pesan disertakan dalam batasan ukuran pesan 1 MiB.
> I `Name` si `TypeValue`,, dan isi pesan tidak boleh kosong atau nol.

Setiap atribut pesan terdiri dari komponen-komponen berikut:

- Nama — Nama atribut pesan dapat berisi karakter berikut: `A` -`Z`, a `0` -`z`, garis bawah (_)9, tanda hubung (`-`), dan titik (`.`). Pembatasan berikut berlaku:
  - Dapat mencapai 256 karakter
  - Tidak dapat memulai dengan `AWS.` atau `Amazon.` (atau variasi casing apa pun)
  - Peka huruf besar/kecil
  - Harus unik di antara semua nama atribut untuk pesan
  - Tidak boleh dimulai atau diakhiri dengan periode
  - Tidak boleh memiliki periode dalam urutan
- Jenis — Jenis data atribut pesan. Jenis yang didukung termasuk`String`,`Number`, dan`Binary`. Anda juga dapat menambahkan informasi khusus untuk jenis data apa pun. Tipe data memiliki batasan yang sama dengan isi pesan (untuk informasi selengkapnya, lihat `SendMessage` di Referensi API Layanan Antrian Sederhana Amazon). Selain itu, batasan berikut berlaku:
  - Dapat mencapai 256 karakter
  - Peka huruf besar/kecil
- Nilai — Nilai atribut pesan. Untuk `String` tipe data, nilai atribut memiliki batasan yang sama dengan badan pesan.

Jenis data atribut pesan

Jenis data atribut pesan menginstruksikan Amazon SQS cara menangani nilai atribut pesan yang sesuai. Misalnya, jika jenisnya adalah`Number`, Amazon SQS memvalidasi nilai numerik.

Amazon SQS mendukung tipe data logis`String`,`Number`, dan `Binary` dengan label tipe data kustom opsional dengan format *`.custom-data-type`*

- String — `String` atribut dapat menyimpan teks Unicode menggunakan karakter XML yang valid.

- Ang ka — `Number` atribut dapat menyimpan nilai numerik positif atau negatif. Sebuah angka dapat memiliki presisi hingga 38 digit, dan bisa antara 10^-128 dan 10^+126.

> **Note**
>
> Amazon SQS menghapus nol di depan dan di belakang.

- Biner — Atribut biner dapat menyimpan data biner apa pun seperti data terkompresi, data terenkripsi, atau gambar.
- Kustom — Untuk membuat tipe data kustom, tambahkan label tipe khusus ke jenis data apa pun. Contoh:
  - `Number.byte`,`Number.short`,`Number.int`, dan `Number.float` dapat membantu membedakan antara jenis angka.
  - `Binary.gif`dan `Binary.png` dapat membantu membedakan antara jenis file.

> **Note**
>
> Amazon SQS tidak menafsirkan, memvalidasi, atau menggunakan data yang ditambahkan. Label tipe khusus memiliki batasan yang sama dengan badan pesan.

Menghitung ringkasan pesan MD5 untuk atribut pesan

Jika Anda menggunakan AWS SDK untuk Java, Anda dapat melewati bagian ini. Kel `MessageMD5ChecksumHandler` as SDK untuk Java mendukung ringkasan pesan MD5 untuk atribut pesan Amazon SQS.

Jika menggunakan Query API atau salah satu AWS SDK yang tidak mendukung ringkasan pesan MD5 untuk atribut pesan Amazon SQS, Anda harus menggunakan pedoman berikut untuk melakukan perhitungan ringkasan pesan MD5.

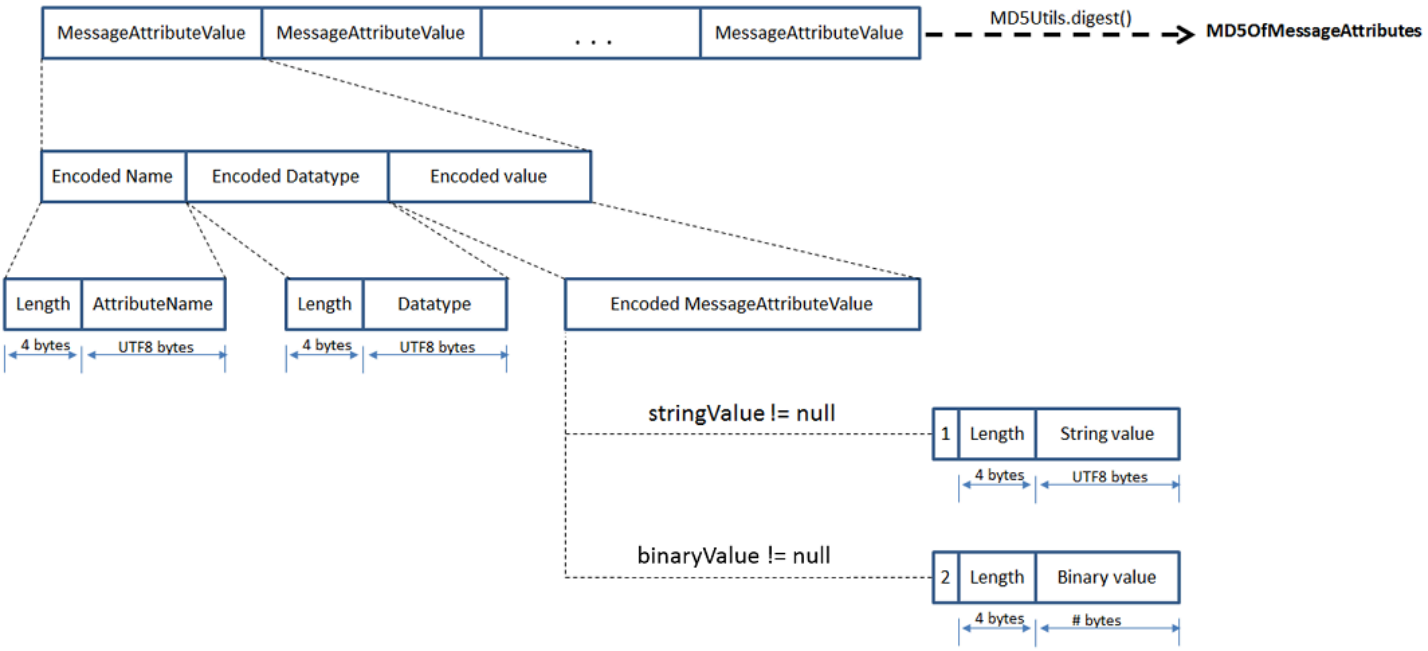
> **Note**
>
> Selalu sertakan sufiks tipe data kustom dalam perhitungan ringkasan pesan MD5.

Gambaran umum

Berikut ini adalah ikhtisar algoritma perhitungan ringkasan pesan MD5:

1. Urutkan semua atribut pesan berdasarkan nama dalam urutan menaik.
2. Enkode bagian individu dari setiap atribut (`Name`,`Type`, dan`Value`) ke dalam buffer.
3. Hitung ringkasan pesan dari seluruh buffer.

Diagram berikut menunjukkan pengkodean ringkasan pesan MD5 untuk atribut pesan tunggal:



Untuk menyandikan satu atribut pesan Amazon SQS

1. Enkode nama: panjang (4 byte) dan UTF-8 byte nama.
2. Enkode tipe data: panjang (4 byte) dan UTF-8 byte dari tipe data.
3. Enkode jenis transportasi (`String`atau`Binary`) dari nilai (1 byte).

> **Note**
>
> Jenis data logis `String` dan `Number` menggunakan tipe `String` transport.
> Tipe data logis `Binary` menggunakan tipe `Binary` transport.

   a. Untuk jenis `String` transportasi, enkode 1.

b. Untuk jenis `Binary` transportasi, enkode 2.

4. Enkode nilai atribut.

   a. Untuk jenis `String` transportasi, kodekan nilai atribut: panjang (4 byte) dan UTF-8 byte nilai.

   b. Untuk jenis `Binary` transportasi, kodekan nilai atribut: panjang (4 byte) dan byte mentah dari nilai.

Atribut sistem pesan Amazon SQS

Meskipun Anda dapat menggunakan atribut pesan untuk melampirkan metadata khusus ke pesan Amazon SQS untuk aplikasi Anda, Anda dapat menggunakan atribut sistem pesan untuk menyimpan metadata untuk AWS layanan lain, seperti AWS X-Ray. Untuk informasi selengkapnya, lihat parameter `MessageSystemAttribute` permintaan tindakan `SendMessage` dan `SendMessageBatch` API, `AWSTraceHeader` atribut tindakan `ReceiveMessage` API, dan tipe `MessageSystemAttributeValue` data di Referensi API Layanan Antrian Sederhana Amazon.

Atribut sistem pesan terstruktur persis seperti atribut pesan, dengan pengecualian berikut:

- Saat ini, satu-satunya atribut sistem pesan yang didukung adalah`AWSTraceHeader`. Jenisnya harus `String` dan nilainya harus berupa string header AWS X-Ray jejak yang diformat dengan benar.
- Ukuran atribut sistem pesan tidak diperhitungkan dalam ukuran total pesan.

Sumber daya yang diperlukan untuk memproses pesan Amazon SQS

Amazon SQS memberikan perkiraan perkiraan jumlah pesan yang tertunda, terlihat, dan tidak terlihat dalam antrian untuk membantu Anda menilai sumber daya yang diperlukan untuk pemrosesan. Untuk informasi selengkapnya tentang visibilitas, lihatBatas waktu visibilitas Amazon SQS.

> **Note**
>
> Untuk beberapa metrik, hasilnya adalah perkiraan karena arsitektur terdistribusi Amazon SQS. Dalam kebanyakan kasus, hitungan harus mendekati jumlah sebenarnya dari pesan dalam antrian.

Tabel berikut mencantumkan nama atribut yang akan digunakan dengan `GetQueueAttributes` tindakan:

Tugas	Nama atribut
Dapatkan perkiraan jumlah pesan yang tersedia untuk diambil dari antrian.	`ApproximateNumberOfMessages Visible`
Dapatkan perkiraan jumlah pesan dalam antrian yang tertunda dan tidak tersedia untuk dibaca segera. Ini dapat terjadi ketika antrian dikonfigurasi sebagai antrian penundaan atau ketika pesan telah dikirim dengan parameter penundaan.	`ApproximateNumberOfMessages Delayed`
Dapatkan perkiraan jumlah pesan yang sedang dalam penerbangan. Pesan dianggap dalam penerbangan jika telah dikirim ke klien tetapi belum dihapus atau belum mencapai akhir jendela visibilitas mereka.	`ApproximateNumberOfMessages NotVisible`

Pagination antrian daftar Amazon SQS

Metode `listQueues`dan `listDeadLetterQueues`API mendukung kontrol pagination opsional. Secara default, metode API ini mengembalikan hingga 1000 antrian dalam pesan respons. Anda dapat mengatur `MaxResults` parameter untuk mengembalikan hasil yang lebih sedikit di setiap respons.

Tetapkan parameter `MaxResults` dalam `listDeadLetterQueues` permintaan `listQueues` atau untuk menentukan jumlah maksimum hasil yang akan dikembalikan dalam respons. Jika Anda tidak

mengaturMaxResults, respon mencakup maksimal 1.000 hasil dan NextToken nilai dalam respon adalah nol.

Jika Anda mengaturMaxResults, respons menyertakan nilai untuk NextToken jika ada hasil tambahan untuk ditampilkan. Gunakan NextToken sebagai parameter dalam permintaan Anda berikutnya listQueues untuk menerima halaman hasil berikutnya. Jika tidak ada hasil tambahan untuk ditampilkan, NextToken nilai dalam respons adalah nol.

Tag alokasi biaya Amazon SQS

Untuk mengatur dan mengidentifikasi antrian Amazon SQS untuk alokasi biaya, Anda dapat menambahkan tag metadata yang mengidentifikasi tujuan, pemilik, atau lingkungan antrian. Ini sangat berguna ketika Anda memiliki banyak antrian. Untuk mengonfigurasi tag menggunakan konsol Amazon SQS, lihat the section called “Mengkonfigurasi tag untuk antrian”

Anda dapat menggunakan tag alokasi biaya untuk mengatur AWS tagihan Anda untuk mencerminkan struktur biaya Anda sendiri. Untuk melakukan ini, daftar untuk mendapatkan Akun AWS tagihan Anda untuk menyertakan kunci tag dan nilai. Untuk informasi selengkapnya, lihat Menyiapkan Laporan Alokasi Biaya Bulanan di Panduan Pengguna AWS Billing .

Setiap tag terdiri dari pasangan kunci-nilai yang Anda tentukan. Misalnya, Anda dapat dengan mudah mengidentifikasi antrian produksi dan pengujian jika Anda menandai antrian Anda sebagai berikut:

Antrean	Key	Nilai
MyQueueA	QueueType	Production
MyQueueB	QueueType	Testing

Note

Saat Anda menggunakan tag antrian, ingatlah pedoman berikut:

- Kami tidak menyarankan menambahkan lebih dari 50 tag ke antrian. Penandaan mendukung karakter Unicode di UTF-8.
- Tag tidak memiliki arti semantik. Amazon SQS menafsirkan tag sebagai string karakter.
- Tag peka huruf besar/kecil.
- Tag baru dengan kunci yang identik dengan tag yang ada menimpa tag yang ada.

- Tindakan penandaan dibatasi hingga 30 TPS per. Akun AWS Jika aplikasi Anda membutuhkan throughput yang lebih tinggi, kirimkan permintaan.

Untuk daftar lengkap pembatasan tag, lihatKuota antrian standar Amazon SQS.

Polling pendek dan panjang Amazon SQS

Amazon SQS menawarkan opsi polling pendek dan panjang untuk menerima pesan dari antrian. Pertimbangkan persyaratan aplikasi Anda untuk responsivitas dan efisiensi biaya saat memilih di antara dua opsi polling ini:

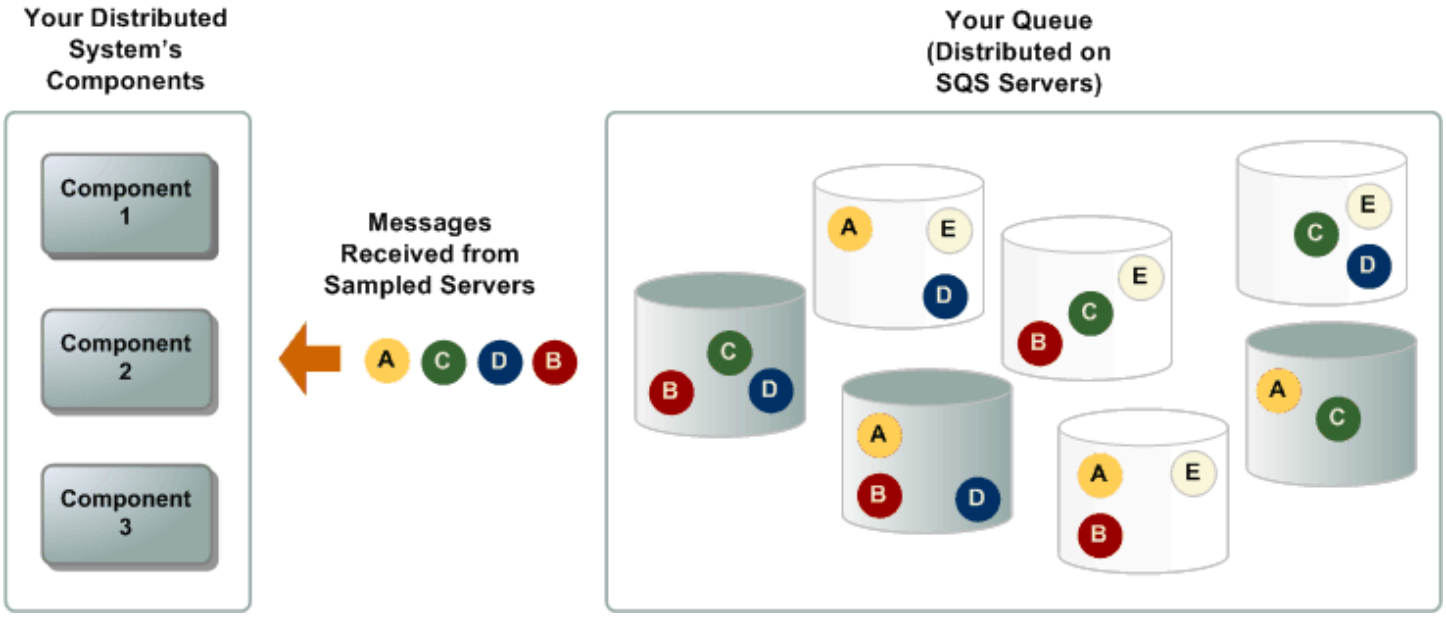
- Polling pendek (default) — Per `ReceiveMessage` mintaan meminta subset server (berdasarkan distribusi acak tertimbang) untuk menemukan pesan yang tersedia dan mengirimkan respons langsung, bahkan jika tidak ada pesan yang ditemukan.
- Polling panjang — `ReceiveMessage` menanyakan semua server untuk pesan, mengirim respons setelah setidaknya satu pesan tersedia, hingga maksimum yang ditentukan. Tanggapan kosong dikirim hanya jika waktu tunggu polling berakhir. Opsi ini dapat mengurangi jumlah tanggapan kosong dan berpotensi menurunkan biaya.

Bagian berikut menjelaskan rincian polling pendek dan polling panjang.

Mengkonsumsi pesan menggunakan polling pendek

Saat Anda menggunakan pesan dari antrian (FIFO atau standar) menggunakan polling pendek, Amazon SQS mengambil sampel subset servernya (berdasarkan distribusi acak tertimbang) dan mengembalikan pesan hanya dari server tersebut. Dengan demikian, `ReceiveMessage` permintaan tertentu mungkin tidak mengembalikan semua pesan Anda. Namun, jika Anda memiliki kurang dari 1.000 pesan dalam antrian Anda, permintaan berikutnya akan mengembalikan pesan Anda. Jika Anda terus mengonsumsi dari antrian Anda, Amazon SQS mengambil sampel semua servernya, dan Anda menerima semua pesan Anda.

Diagram berikut menunjukkan perilaku polling singkat pesan yang dikembalikan dari antrian standar setelah salah satu komponen sistem Anda membuat permintaan penerimaan. Amazon SQS mengambil sampel beberapa servernya (berwarna abu-abu) dan mengembalikan pesan A, C, D, dan B dari server ini. Pesan E tidak dikembalikan untuk permintaan ini, tetapi dikembalikan untuk permintaan berikutnya.



Mengkonsumsi pesan menggunakan polling panjang

Ketika waktu tunggu untuk tindakan `ReceiveMessage` API lebih besar dari 0, polling panjang berlaku. Waktu tunggu polling yang panjang maksimum adalah 20 detik. Polling panjang membantu mengurangi biaya penggunaan Amazon SQS dengan mengurangi jumlah tanggapan kosong (ketika tidak ada pesan yang tersedia untuk `ReceiveMessage` permintaan) dan respons kosong palsu (ketika pesan tersedia tetapi tidak disertakan dalam respons). Untuk informasi tentang mengaktifkan polling panjang untuk antrian baru atau yang sudah ada menggunakan konsol Amazon SQS, lihat. Mengkonfigurasi parameter antrian menggunakan konsol Amazon SQS Untuk praktik terbaik, lihat Mengatur polling panjang di Amazon SQS.

Pemungutan suara panjang menawarkan manfaat berikut:

- Kurangi tanggapan kosong dengan mengizinkan Amazon SQS menunggu hingga pesan tersedia dalam antrian sebelum mengirim respons. Kecuali waktu koneksi habis, respons terhadap `ReceiveMessage` permintaan berisi setidaknya satu pesan yang tersedia, hingga jumlah maksimum pesan yang ditentukan dalam `ReceiveMessage` tindakan. Dalam kasus yang jarang terjadi, Anda mungkin menerima respons kosong bahkan ketika antrian masih berisi pesan, terutama jika Anda menentukan nilai rendah untuk `ReceiveMessageWaitTimeSeconds` parameter.
- Kurangi respons kosong palsu dengan menanyakan semuanya—bukan subkumpulan dari — server Amazon SQS.
- Kembalikan pesan segera setelah tersedia.

Untuk informasi tentang cara mengonfirmasi bahwa antrian kosong, lihatMengonfirmasi bahwa antrian Amazon SQS kosong.

Perbedaan antara polling panjang dan pendek

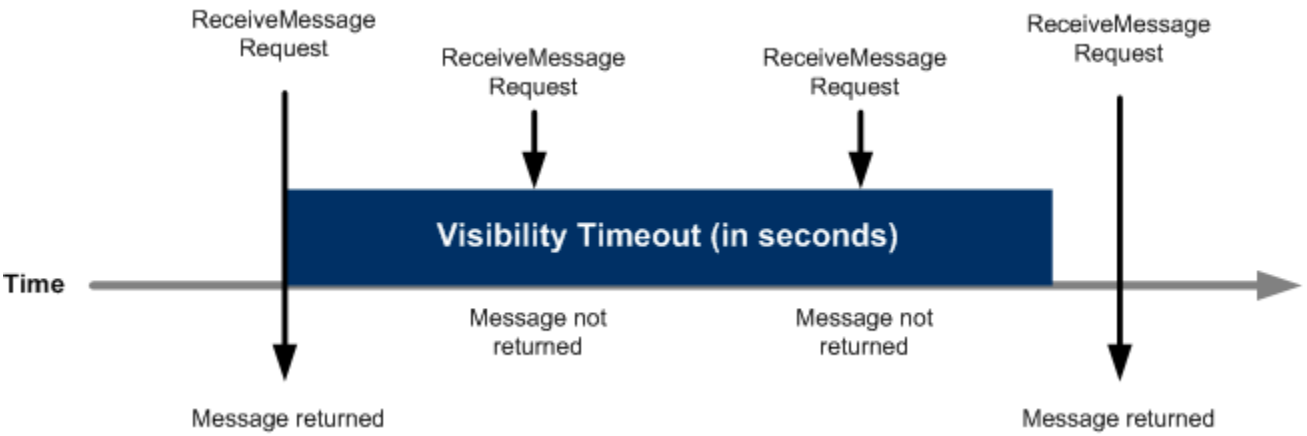
Polling pendek terjadi ketika `WaitTimeSeconds` parameter `ReceiveMessage` permintaan diatur `0` dalam salah satu dari dua cara:

- Pang `ReceiveMessage` gilan diatur `WaitTimeSeconds` ke`0`.
- Pang `ReceiveMessage` gilan tidak disetel`WaitTimeSeconds`, tetapi atribut antrean `ReceiveMessageWaitTimeSeconds` disetel ke`0`.

Batas waktu visibilitas Amazon SQS

Ketika Anda menerima pesan dari antrian Amazon SQS, pesan itu tetap dalam antrian tetapi menjadi sementara tidak terlihat oleh konsumen lain. Tembus pandang ini dikendalikan oleh batas waktu visibilitas, yang memastikan bahwa konsumen lain tidak dapat memproses pesan yang sama saat Anda mengerjakannya. Amazon SQS menawarkan dua opsi untuk menghapus pesan setelah diproses:

- Penghapusan manual — Anda secara eksplisit menghapus pesan menggunakan `DeleteMessage` tindakan.
- Penghapusan otomatis - Didukung dalam AWS SDK tertentu, pesan dihapus secara otomatis setelah pemrosesan berhasil, menyederhanakan alur kerja.



Kasus penggunaan batas waktu visibilitas

Kelola tugas yang berjalan lama — Gunakan batas waktu visibilitas untuk menangani tugas yang memerlukan waktu pemrosesan yang diperpanjang. Tetapkan batas waktu visibilitas yang sesuai untuk pesan yang memerlukan waktu pemrosesan yang diperpanjang. Ini memastikan bahwa konsumen lain tidak menerima pesan yang sama saat sedang diproses, mencegah pekerjaan duplikat dan menjaga efisiensi sistem.

Menerapkan mekanisme coba ulang — Perpanjang batas waktu visibilitas secara terprogram untuk tugas yang gagal diselesaikan dalam batas waktu awal. Jika tugas gagal diselesaikan dalam batas waktu visibilitas awal, Anda dapat memperpanjang batas waktu secara terprogram. Ini memungkinkan sistem Anda untuk mencoba lagi memproses pesan tanpa terlihat oleh konsumen lain, meningkatkan toleransi kesalahan dan keandalan. Gabungkan dengan Dead-Letter Antrian (DLQ) untuk mengelola kegagalan persisten.

Koordinasi sistem terdistribusi — Gunakan batas waktu visibilitas SQS untuk mengoordinasikan tugas di seluruh sistem terdistribusi. Tetapkan batas waktu visibilitas yang selaras dengan waktu pemrosesan yang diharapkan untuk komponen yang berbeda. Ini membantu menjaga konsistensi dan mencegah kondisi balapan dalam arsitektur yang kompleks dan terdistribusi.

Mengoptimalkan pemanfaatan sumber daya — Sesuaikan batas waktu visibilitas SQS untuk mengoptimalkan pemanfaatan sumber daya dalam aplikasi Anda. Dengan menetapkan batas waktu yang tepat, Anda dapat memastikan bahwa pesan diproses secara efisien tanpa mengikat sumber daya yang tidak perlu. Hal ini mengarah pada kinerja sistem secara keseluruhan dan efektivitas biaya yang lebih baik.

Mengatur dan menyesuaikan batas waktu visibilitas

Waktu tunggu visibilitas dimulai segera setelah pesan dikirimkan kepada Anda. Selama periode ini, Anda diharapkan untuk memproses dan menghapus pesan. Jika Anda tidak menghapusnya sebelum batas waktu berakhir, pesan akan terlihat lagi dalam antrian dan dapat diambil oleh konsumen lain. Batas waktu visibilitas default untuk antrian adalah 30 detik, tetapi Anda dapat menyesuaikannya agar sesuai dengan waktu yang dibutuhkan aplikasi Anda untuk memproses dan menghapus pesan. Anda juga dapat mengatur batas waktu visibilitas tertentu untuk pesan individual tanpa mengubah pengaturan keseluruhan antrian. Gunakan `ChangeMessageVisibility` tindakan untuk memperpanjang atau mempersingkat batas waktu secara terprogram sesuai kebutuhan.

Dalam pesan penerbangan dan kuota

Di Amazon SQS, pesan dalam penerbangan adalah pesan yang telah diterima oleh konsumen tetapi belum dihapus. Untuk antrian standar, ada batas sekitar 120.000 pesan dalam penerbangan, tergantung pada lalu lintas antrian dan backlog pesan. Jika Anda mencapai batas ini, Amazon SQS mengembalikan `OverLimit` kesalahan, menunjukkan bahwa tidak ada pesan tambahan yang dapat diterima sampai beberapa pesan dalam penerbangan dihapus. Untuk antrian FIFO, batas bergantung pada grup pesan aktif.

- Saat menggunakan polling pendek — Jika batas ini tercapai saat menggunakan polling pendek, Amazon SQS akan mengembalikan `OverLimit` kesalahan, menunjukkan bahwa tidak ada pesan tambahan yang dapat diterima sampai beberapa pesan dalam penerbangan dihapus.
- Saat menggunakan polling panjang — Jika Anda menggunakan polling panjang, Amazon SQS tidak mengembalikan kesalahan saat batas pesan dalam penerbangan tercapai. Sebaliknya, ia tidak akan mengembalikan pesan baru sampai jumlah pesan dalam penerbangan turun di bawah batas.

Untuk mengelola pesan dalam penerbangan secara efektif:

1. Penghapusan cepat — Hapus pesan (secara manual atau otomatis) setelah diproses untuk mengurangi jumlah dalam penerbangan.
2. Pantau dengan CloudWatch — Atur alarm untuk jumlah dalam penerbangan yang tinggi untuk mencegah mencapai batas.
3. Mendistribusikan beban — Jika Anda memproses pesan dengan volume tinggi, gunakan antrian atau konsumen tambahan untuk menyeimbangkan beban dan menghindari kemacetan.
4. Minta kenaikan kuota — Kirim permintaan ke AWS Dukungan jika batas yang lebih tinggi diperlukan.

Memahami batas waktu visibilitas dalam antrian standar dan FIFO

Dalam antrian standar dan FIFO (First-In-First-Out), batas waktu visibilitas membantu mencegah beberapa konsumen memproses pesan yang sama secara bersamaan. Namun, karena model pengiriman Amazon SQS setidaknya sekali, tidak ada jaminan mutlak bahwa pesan tidak akan dikirimkan lebih dari sekali selama periode batas waktu visibilitas.

- Antrian standar — Waktu tunggu visibilitas dalam antrian standar mencegah beberapa konsumen memproses pesan yang sama pada saat yang sama. Namun, karena model pengiriman paling sedikit sekali, Amazon SQS tidak menjamin bahwa pesan tidak akan dikirimkan lebih dari sekali dalam periode batas waktu visibilitas.
- Antrian FIFO — Untuk antrian FIFO, pesan dengan ID grup pesan yang sama diproses dalam urutan yang ketat. Ketika pesan dengan ID grup pesan sedang dalam penerbangan, pesan berikutnya dalam grup tersebut tidak tersedia sampai pesan dalam penerbangan dihapus atau batas waktu visibilitas berakhir. Namun, ini tidak "mengunci" grup tanpa batas — setiap pesan diproses secara berurutan, dan hanya ketika setiap pesan dihapus atau terlihat lagi pesan berikutnya dalam grup tersebut akan tersedia untuk konsumen. Pendekatan ini memastikan pemrosesan yang teratur dalam grup tanpa mengunci grup tanpa perlu mengunci grup agar tidak mengirimkan pesan.

Penanganan kegagalan

Jika Anda tidak memproses dan menghapus pesan sebelum batas waktu visibilitas habis—karena kesalahan aplikasi, crash, atau masalah konektivitas—pesan akan terlihat lagi dalam antrian. Kemudian dapat diambil oleh konsumen yang sama atau yang berbeda untuk upaya pemrosesan lain. Ini memastikan bahwa pesan tidak hilang bahkan jika pemrosesan awal gagal. Namun, mengatur batas waktu visibilitas terlalu tinggi dapat menunda kemunculan kembali pesan yang belum diproses, berpotensi memperlambat percobaan ulang. Sangat penting untuk menetapkan batas waktu visibilitas yang sesuai berdasarkan waktu pemrosesan yang diharapkan untuk penanganan pesan tepat waktu.

Mengubah dan mengakhiri batas waktu visibilitas

Anda dapat mengubah atau menghentikan batas waktu visibilitas menggunakan `ChangeMessageVisibility` tindakan:

- Mengubah batas waktu — Sesuaikan batas waktu visibilitas secara dinamis menggunakan. `ChangeMessageVisibility` Ini memungkinkan Anda untuk memperpanjang atau mengurangi durasi batas waktu agar sesuai dengan kebutuhan pemrosesan.
- Mengakhiri batas waktu — Jika Anda memutuskan untuk tidak memproses pesan yang diterima, hentikan batas waktu visibilitasnya dengan menyetel `VisibilityTimeout` ke 0 detik melalui tindakan. `ChangeMessageVisibility` Ini segera membuat pesan tersedia untuk diproses oleh konsumen lain.

Praktik terbaik

Gunakan praktik terbaik berikut untuk mengelola batas waktu visibilitas di Amazon SQS, termasuk menyetel, menyesuaikan, dan memperpanjang batas waktu, serta menangani pesan yang belum diproses menggunakan An Dead-Letter trian (DLQ).

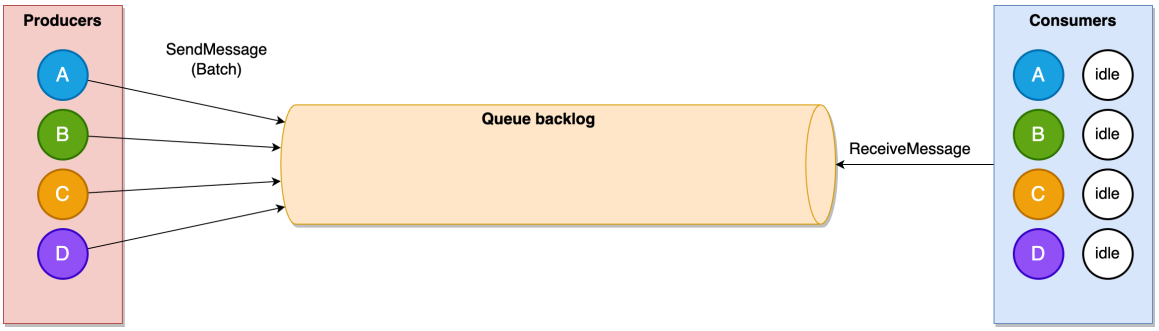
- Mengatur dan menyesuaikan batas waktu. Mulailah dengan menyetel batas waktu visibilitas agar sesuai dengan waktu maksimum yang biasanya dibutuhkan aplikasi Anda untuk memproses dan menghapus pesan. Jika Anda tidak yakin tentang waktu pemrosesan yang tepat, mulailah dengan batas waktu yang lebih pendek (misalnya, 2 menit) dan perpanjang seperlunya. Terapkan mekanisme detak jantung untuk memperpanjang batas waktu visibilitas secara berkala, memastikan pesan tetap tidak terlihat sampai pemrosesan selesai. Ini meminimalkan penundaan dalam memproses ulang pesan yang tidak ditangani dan mencegah visibilitas dini.
- Memperpanjang batas waktu dan menangani batas 12 Jam. Jika waktu pemrosesan bervariasi atau mungkin melebihi batas waktu yang ditetapkan sebelumnya, gunakan `ChangeMessageVisibility` tindakan untuk memperpanjang batas waktu visibilitas saat memproses pesan. Perlu diingat bahwa batas waktu visibilitas memiliki batas maksimum 12 jam sejak pesan pertama kali diterima. Memperpanjang batas waktu tidak mengatur ulang batas 12 jam ini. Jika pemrosesan Anda membutuhkan lebih banyak waktu daripada batas ini, pertimbangkan untuk menggunakan AWS Step Functions atau memecah tugas menjadi langkah-langkah yang lebih kecil.
- Menangani pesan yang belum diproses. Untuk mengelola pesan yang gagal dalam beberapa upaya pemrosesan, konfigurasikan An Dead-Letter trian (DLQ). Ini memastikan bahwa pesan yang tidak dapat diproses setelah beberapa kali percobaan ulang ditangkap secara terpisah untuk analisis atau penanganan lebih lanjut, mencegahnya beredar berulang kali di antrian utama.

Antrian adil Amazon SQS

Antrian adil Amazon SQS secara otomatis mengurangi dampak tetangga yang bising dalam antrian multi-penyewa yang berisi pesan dari beberapa entitas logis, seperti pelanggan, aplikasi klien, atau jenis pesan. Dalam lingkungan antrian bersama ini, satu metrik kinerja penting adalah waktu tinggal, yang mengukur total waktu yang dihabiskan pesan dalam antrian dari kedatangan hingga pemrosesan. Ketika satu penyewa membuat backlog dalam antrian dengan menerbitkan lebih banyak pesan daripada yang dapat ditangani sistem, antrian yang adil meminimalkan dampak pada waktu tinggal bagi penyewa lain.

Keadaan mantap

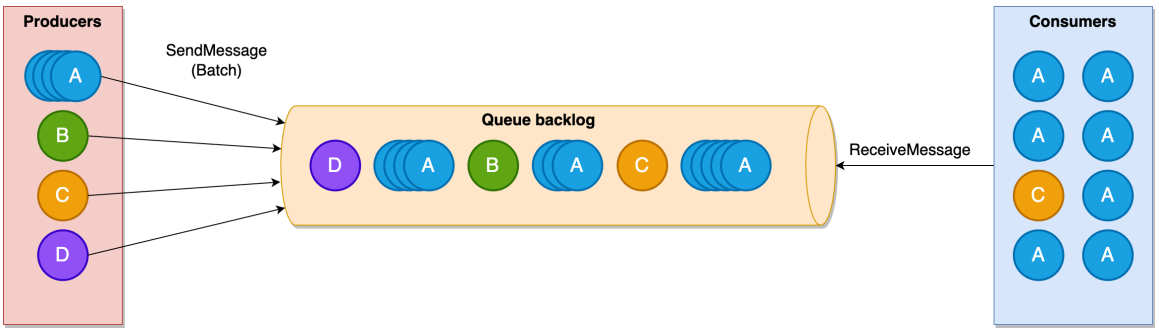
Diagram berikut menggambarkan antrian multi-tenant yang berisi pesan dari empat penyewa yang berbeda (berlabel A, B, C, dan D). Antrian beroperasi dalam keadaan mapan, dan tidak ada backlog pesan karena konsumen menerima pesan segera setelah muncul dalam antrian. Semua penyewa mengalami waktu tinggal yang rendah. Tidak semua kapasitas konsumen dimanfaatkan sepenuhnya dalam kondisi mapan ini.



Dampak tetangga yang bising

Dampak tetangga yang bising terjadi ketika satu penyewa dalam antrian multi-penyewa membuat backlog, meningkatkan waktu tinggal pesan untuk semua penyewa lainnya. Penyewa dapat menjadi tetangga yang berisik dengan mengirimkan volume pesan yang lebih besar daripada penyewa lain, atau ketika konsumen membutuhkan waktu lebih lama untuk memproses pesan dari penyewa tertentu.

Diagram ini menggambarkan bagaimana peningkatan lalu lintas dari Tenant A menciptakan backlog dalam antrian. Konsumen sibuk memproses pesan hanya dari Penyewa A, sementara pesan dari penyewa lain menunggu di backlog, yang mengarah ke waktu tinggal yang lebih tinggi untuk semua penyewa.

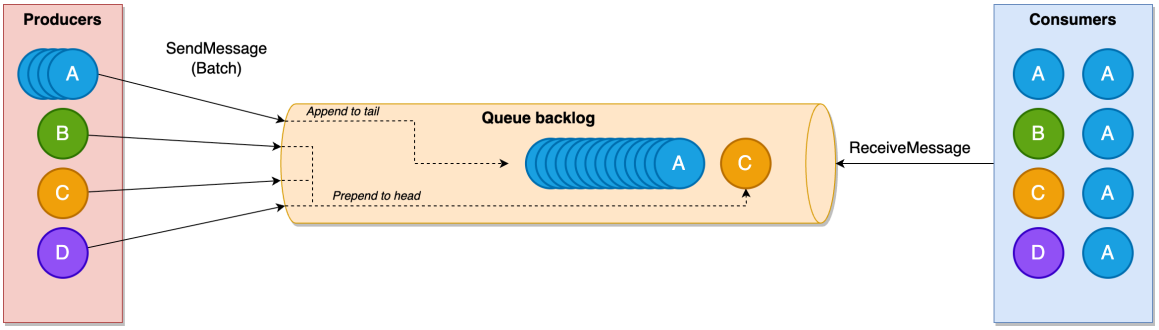


Mitigasi dengan antrian yang adil

Amazon SQS mendeteksi tetangga yang berisik dengan memantau distribusi pesan di antara penyewa selama pemrosesan (status “dalam penerbangan”). Ketika penyewa memiliki jumlah pesan dalam penerbangan yang tidak proporsional dibandingkan dengan yang lain, Amazon SQS

mengidentifikasi penyewa sebagai tetangga yang berisik dan memprioritaskan pengiriman pesan untuk penyewa lain. Pendekatan ini mengurangi dampak waktu tinggal bagi penyewa lainnya.

Diagram ini menggambarkan bagaimana antrian adil Amazon SQS mengatasi masalah tetangga yang bising. Ketika satu penyewa (Tenant A) menjadi berisik, Amazon SQS memprioritaskan pengembalian pesan dari penyewa lain (B, C, dan D). Prioritas ini membantu mempertahankan waktu tinggal yang rendah untuk penyewa yang tenang Penyewa B, C, dan D, sementara waktu tinggal untuk pesan Tenant A dinaikkan hingga backlog antrian dikonsumsi tanpa memengaruhi penyewa lain.



> **Note**
>
> Amazon SQS tidak membatasi tingkat konsumsi per penyewa. Hal ini memungkinkan konsumen untuk menerima pesan dari penyewa tetangga yang berisik ketika ada kapasitas konsumen dan antrian tidak memiliki pesan lain untuk dikembalikan. Seperti antrian standar Amazon SQS, antrian yang adil memungkinkan throughput yang hampir tidak terbatas, dan tidak ada batasan jumlah penyewa yang dapat Anda miliki dalam antrian Anda.

Untuk detail tentang cara Amazon SQS mendeteksi tetangga yang berisik dan mengelola pesanan pengiriman pesan, lihat Cara kerja antrian adil Amazon SQS.

Cara menggunakan antrian yang adil

Untuk mengaktifkan antrian yang adil, produsen pesan harus menambahkan pengenal penyewa dengan menyetel pesan keluar: `MessageGroupId`

```
// Send message with tenant identifier
SendMessageRequest request = new SendMessageRequest()
    .withQueueUrl(queueUrl)
    .withMessageBody(messageBody)
```

```
    .withMessageGroupId("tenant-123"); // Tenant identifier
sqs.sendMessage(request);
```

Kemampuan keadilan akan diterapkan secara otomatis di semua antrian standar Amazon SQS untuk pesan dengan properti. MessageGroupId Itu tidak memerlukan perubahan apa pun dalam kode konsumen, tidak berdampak pada latensi API, dan tidak disertai dengan batasan throughput apa pun.

> **Note**
>
> `MessageGroupId`pada antrian standar dengan antrian yang adil tidak memiliki perilaku yang sama seperti `MessageGroupId` pada antrian FIFO. Pada antrian standar, hanya `MessageGroupId` digunakan sebagai pengenal penyewa untuk antrian yang adil dan tidak memberlakukan pemesanan pesan. Untuk detail `MessageGroupId` tentang antrian FIFO, lihat Menggunakan ID grup pesan dengan Antrian FIFO Amazon SQS.

Kapan harus menggunakan antrian yang adil

Pertimbangkan untuk menggunakan antrian yang adil ketika semua hal berikut berlaku untuk antrian Anda:

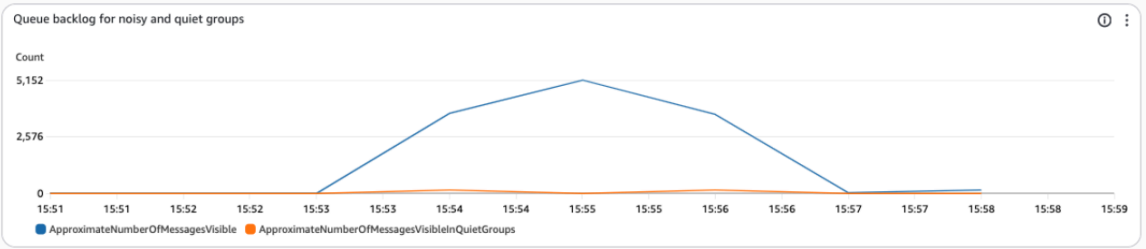
- Antriannya multi-tenant. Antrian membawa pesan milik beberapa entitas logis, seperti pelanggan, aplikasi klien, atau jenis permintaan, dan Anda dapat mengidentifikasi setiap entitas dengan `MessageGroupId` file.
- Antriannya adalah throughput tinggi. Pada throughput rendah, ledakan satu penyewa jarang menciptakan backlog yang mempengaruhi penyewa lain. Pada throughput tinggi, penskalaan berlebihan armada konsumen untuk menyerap setiap ledakan tidak praktis, dan bahkan dengan auto-scaling, penundaan sebelum konsumen baru online dapat membiarkan backlog terbentuk.
- Waktu tinggal adalah bagian dari kualitas layanan aplikasi Anda. Antrian yang adil melindungi penyewa yang tenang dari waktu tinggal yang ditinggikan yang disebabkan oleh tetangga yang berisik. Jika aplikasi Anda tidak sensitif terhadap waktu tinggal, antrian pameran perlindungan tetangga yang berisik mungkin tidak diperlukan.

Metrik antrian CloudWatch yang adil

Amazon SQS menyediakan CloudWatch metrik tambahan untuk membantu Anda memantau mitigasi dampak tetangga yang bising. Sebagai contoh, Anda dapat membandingkan

`Approximate..InQuietGroups` metrik dengan metrik tingkat antrian standar. Selama lonjakan lalu lintas untuk penyewa tertentu, metrik tingkat antrian umum mungkin mengungkapkan peningkatan backlog atau usia pesan yang lebih tua. Namun, melihat kelompok yang tenang secara terpisah, Anda dapat mengidentifikasi bahwa sebagian besar grup pesan atau penyewa yang tidak berisik tidak terpengaruh.

Di bawah ini Anda dapat menemukan contoh di mana metrik backlog antrian standar (ApproximateNumberOfMessagesVisible) meningkat karena penyewa yang berisik sementara backlog untuk penyewa yang tidak berisik () tetap rendah. ApproximateNumberOfMessagesVisibleInQuietGroups



Untuk daftar lengkap metrik Amazon SQS dan deskripsinya, lihat CloudWatch metrik CloudWatch untuk Amazon SQS.

Cara kerja antrian adil Amazon SQS

Antrian adil Amazon SQS secara otomatis mengurangi dampak tetangga yang bising dalam antrian multi-penyewa dengan mencegah penyewa tunggal memperlambat pemrosesan pesan untuk penyewa lain. Ketika antrian yang adil mendeteksi bahwa satu penyewa menggunakan bagian yang tidak proporsional dari kapasitas konsumen antrian, mereka memprioritaskan pengiriman pesan dari penyewa yang tenang.

Identifikasi penyewa

Antrian yang adil mengidentifikasi penyewa berdasarkan `MessageGroupId` properti pada setiap pesan. Pesan yang berbagi yang sama `MessageGroupId` milik penyewa yang sama. Antrian yang adil dapat berisi campuran pesan dengan dan tanpa file. `MessageGroupId` Setiap pesan tanpa a `MessageGroupId` diperlakukan sebagai milik penyewa yang berbeda.

Sebaiknya setel a `MessageGroupId` pada setiap pesan yang Anda kirim ke antrian yang adil, menggunakan pengenal yang memetakan ke entitas nyata di sistem Anda, seperti ID pelanggan, ID aplikasi klien, atau jenis permintaan.

Cara kerja deteksi

Amazon SQS mendeteksi tetangga yang berisik menggunakan dua ukuran:

- Concurrency share — pesan dalam penerbangan penyewa sebagai sebagian kecil dari semua pesan dalam penerbangan dalam antrian. Ini mendeteksi penyewa dengan banyak pesan yang diproses sekaligus, baik karena volume pengiriman yang tinggi, pemrosesan yang lambat, atau keduanya.
- Pembagian waktu pemrosesan — bagian terakhir penyewa dari total waktu pemrosesan konsumen. Ini mendeteksi penyewa yang pesannya sedikit jumlahnya tetapi lambat untuk diproses, sehingga penyewa menempati waktu konsumen yang signifikan tanpa memiliki banyak pesan dalam penerbangan sekaligus.

Penyewa ditandai sebagai tetangga yang berisik ketika salah satu dari berikut ini benar:

- Berbagi konkurensi: penyewa menyumbang lebih dari 10% pesan dalam penerbangan dalam antrian dan memiliki setidaknya 30 pesannya sendiri dalam penerbangan.
- Pembagian waktu pemrosesan: pangsa penyewa baru-baru ini dari total waktu pemrosesan konsumen melebihi 10%.

Karena Amazon SQS adalah sistem terdistribusi, ambang batas ini merupakan perkiraan dan deteksi mungkin tidak diaktifkan pada nilai persis yang dijelaskan.

Bagaimana pesanan pengiriman pesan berubah

Setelah penyewa terdeteksi sebagai tetangga yang berisik, Amazon SQS memprioritaskan pengiriman pesan dari penyewa yang tenang. Contoh konsumen yang melakukan polling untuk pesan menerima pesan dari penyewa yang tenang setiap kali pesan penyewa tenang tersedia. Prioritas ini berlanjut sampai pangsa konkurensi tetangga yang berisik dan pangsa waktu pemrosesan turun ke level yang sebanding dengan penyewa yang tenang.

Pesan tetangga yang berisik tidak dijatuhkan atau dibatasi. Waktu tinggal mereka meningkat sementara pesan penyewa yang tenang diprioritaskan, dan kembali normal setelah backlog mereka hilang. Ketika tidak ada pesan penyewa tenang yang menunggu, contoh konsumen menerima pesan tetangga yang berisik seperti biasa.

Jika beberapa penyewa terdeteksi sebagai tetangga yang berisik secara bersamaan, contoh konsumen menerima pesan dari penyewa yang berisik dengan pesan dalam penerbangan paling

sedikit terlebih dahulu. Ini membantu menyeimbangkan waktu pemrosesan di seluruh penyewa yang berisik.

Ketika tetangga yang berisik menjadi pendiam

Penyewa tidak lagi diperlakukan sebagai tetangga yang berisik ketika salah satu dari berikut ini benar:

- Backlog penyewa telah sepenuhnya dikonsumsi.
- Tidak ada pesan dari penyewa yang telah terbang selama 5 menit.

Setelah itu, pesan penyewa tidak lagi diprioritaskan dan diperlakukan sama seperti pesan dari penyewa tenang lainnya.

Pertimbangan kapasitas konsumen

Agar ukuran pembagian konkurensi berfungsi secara efektif, konsumen Anda perlu memproses pesan yang cukup secara bersamaan sehingga bagian satu penyewa dari pesan dalam penerbangan dapat menonjol. Pertimbangkan ini saat mengukur armada konsumen Anda.

Saat menggunakan Lambda sebagai konsumen melalui pemetaan sumber peristiwa, jumlah pesan dalam penerbangan bergantung pada konkurensi fungsi Lambda dan ukuran batch. Evaluasi pengaturan ini bersama-sama saat mengukur konsumen Anda.

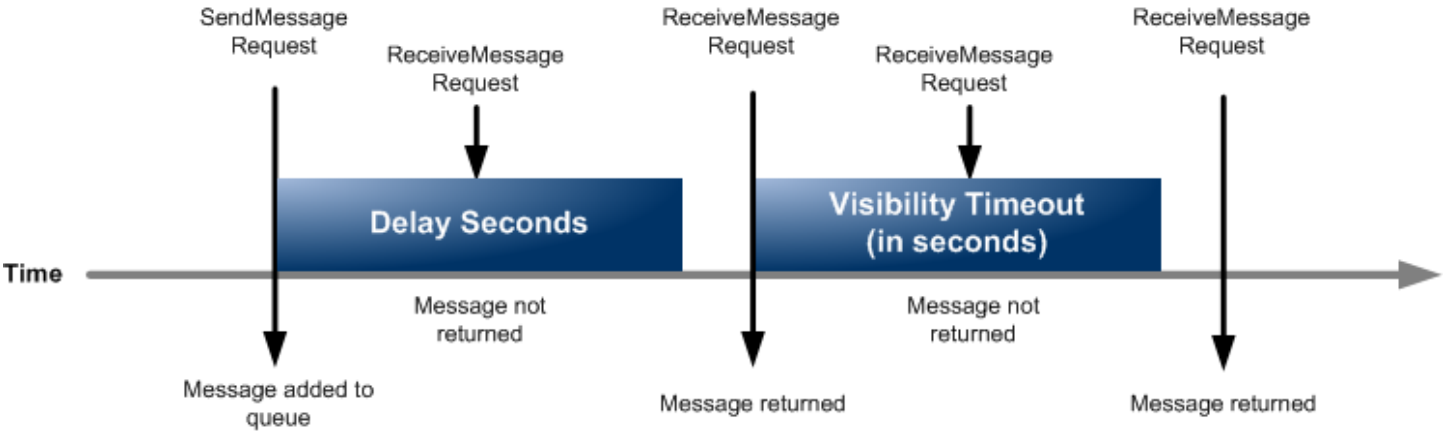
Ketika jumlah pesan dalam penerbangan terlalu rendah untuk memicu ambang pembagian konkurensi, ukuran pembagian waktu pemrosesan masih dapat mendeteksi tetangga yang berisik. Namun, antrian yang adil bekerja paling baik ketika konsumen memproses pesan yang cukup secara bersamaan untuk kedua tindakan yang akan dievaluasi.

Antrian penundaan Amazon SQS

Antrian tunda memungkinkan Anda menunda pengiriman pesan baru ke konsumen selama beberapa detik, misalnya, ketika aplikasi konsumen Anda membutuhkan waktu tambahan untuk memproses pesan. Jika Anda membuat antrean penundaan, pesan apa pun yang Anda kirim ke antrian tetap tidak terlihat oleh konsumen selama periode penundaan. Penundaan default (minimum) untuk antrian adalah 0 detik. Maksimal adalah 15 menit. Untuk informasi tentang mengonfigurasi antrean penundaan menggunakan konsol, lihatMengkonfigurasi parameter antrian menggunakan konsol Amazon SQS.

> **Note**
>
> Untuk antrian standar, pengaturan penundaan per antrian tidak bersifat retroaktif — mengubah pengaturan tidak mempengaruhi penundaan pesan yang sudah ada dalam antrian.
> Untuk antrian FIFO, pengaturan penundaan per antrian bersifat retroaktif — mengubah pengaturan memengaruhi penundaan pesan yang sudah ada dalam antrian.

Antrian penundaan mirip dengan batas waktu visibilitas karena kedua fitur membuat pesan tidak tersedia bagi konsumen untuk jangka waktu tertentu. Perbedaan antara keduanya adalah bahwa, untuk antrean penundaan, pesan disembunyikan ketika pertama kali ditambahkan ke antrian, sedangkan untuk batas waktu visibilitas pesan disembunyikan hanya setelah dikonsumsi dari antrian. Diagram berikut menggambarkan hubungan antara antrean penundaan dan batas waktu visibilitas.



Opsi penjadwalan yang diperpanjang

Meskipun antrian penundaan Amazon SQS dan pengatur waktu pesan memungkinkan penjadwalan pengiriman pesan hingga 15 menit di masa mendatang, Anda mungkin memerlukan kemampuan penjadwalan yang lebih fleksibel. Dalam kasus seperti itu, pertimbangkan untuk menggunakan EventBridge Penjadwal, yang memungkinkan Anda menjadwalkan miliaran tindakan API satu kali atau berulang tanpa batasan waktu. EventBridge Penjadwal adalah solusi yang disarankan untuk kasus penggunaan penjadwalan pesan lanjutan.

Untuk mengatur detik penundaan pada pesan individual, bukan pada seluruh antrian, gunakan pengatur waktu pesan untuk mengizinkan Amazon SQS menggunakan nilai pengatur waktu pesan alih-alih `DelaySeconds` nilai antrian `DelaySeconds` penundaan. EventBridge Scheduler juga mendukung penjadwalan pesan individual.

Antrian sementara Amazon SQS

Antrian sementara membantu Anda menghemat waktu pengembangan dan biaya penerapan saat menggunakan pola pesan umum seperti respons permintaan. Anda dapat menggunakan Klien Antrian Sementara untuk membuat antrian sementara dengan throughput tinggi, hemat biaya, dan dikelola aplikasi.

Klien memetakan beberapa antrian sementara — antrian yang dikelola aplikasi yang dibuat sesuai permintaan untuk proses tertentu — ke dalam satu antrian Amazon SQS secara otomatis. Hal ini memungkinkan aplikasi Anda membuat lebih sedikit panggilan API dan memiliki throughput yang lebih tinggi ketika lalu lintas ke setiap antrian sementara rendah. Ketika antrian sementara tidak lagi digunakan, klien membersihkan antrian sementara secara otomatis, bahkan jika beberapa proses yang menggunakan klien tidak dimatikan dengan bersih.

Berikut ini adalah manfaat antrian sementara:

- Mereka berfungsi sebagai saluran komunikasi ringan untuk utas atau proses tertentu.
- Mereka dapat dibuat dan dihapus tanpa menimbulkan biaya tambahan.
- Mereka API-compatible dengan antrian Amazon SQS statis (normal). Ini berarti bahwa kode yang ada yang mengirim dan menerima pesan dapat mengirim pesan ke dan menerima pesan dari antrian virtual.

Antrian virtual

Antrian virtual adalah struktur data lokal yang dibuat oleh Klien Antrian Sementara. Antrian virtual memungkinkan Anda menggabungkan beberapa tujuan dengan lalu lintas rendah ke dalam satu antrian Amazon SQS. Untuk praktik terbaik, lihat Menghindari penggunaan kembali ID grup pesan yang sama dengan antrian virtual.

> **Note**
>
> - Membuat antrian virtual hanya menciptakan struktur data sementara bagi konsumen untuk menerima pesan. Karena antrian virtual tidak membuat panggilan API ke Amazon SQS, antrian virtual tidak dikenakan biaya.
> - Kuota TPS berlaku untuk semua antrian virtual di satu antrian host. Untuk informasi selengkapnya, lihat Kuota pesan Amazon SQS.

Kelas pem `AmazonSQSVirtualQueuesClient` bungkus menambahkan dukungan untuk atribut yang terkait dengan antrian virtual. Untuk membuat antrian virtual, Anda harus memang `CreateQueue` gil tindakan API menggunakan `HostQueueURL` atribut. Atribut ini menentukan antrian yang ada yang menghosting antrian virtual.

URL antrian virtual adalah dalam format berikut.

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

Ketika produsen memanggil tindakan `SendMessageBatch` API `SendMessage` atau pada URL antrian virtual, Klien Antrian Sementara melakukan hal berikut:

1. Mengekstrak nama antrian virtual.
2. Melampirkan nama antrian virtual sebagai atribut pesan tambahan.
3. Mengirim pesan ke antrian host.

Sementara produser mengirim pesan, thread latar belakang mensurvei antrian host dan mengirim pesan yang diterima ke antrian virtual sesuai dengan atribut pesan yang sesuai.

Sementara konsumen memanggil tindakan `ReceiveMessage` API pada URL antrian virtual, Klien Antrian Sementara memblokir panggilan secara lokal sampai thread latar belakang mengirimkan pesan ke antrian virtual. (Proses ini mirip dengan prefetching pesan di Buffered Asynchronous Client: tindakan API tunggal dapat memberikan pesan hingga 10 antrian virtual.) Menghapus antrian virtual menghapus sumber daya sisi klien tanpa memanggil Amazon SQS itu sendiri.

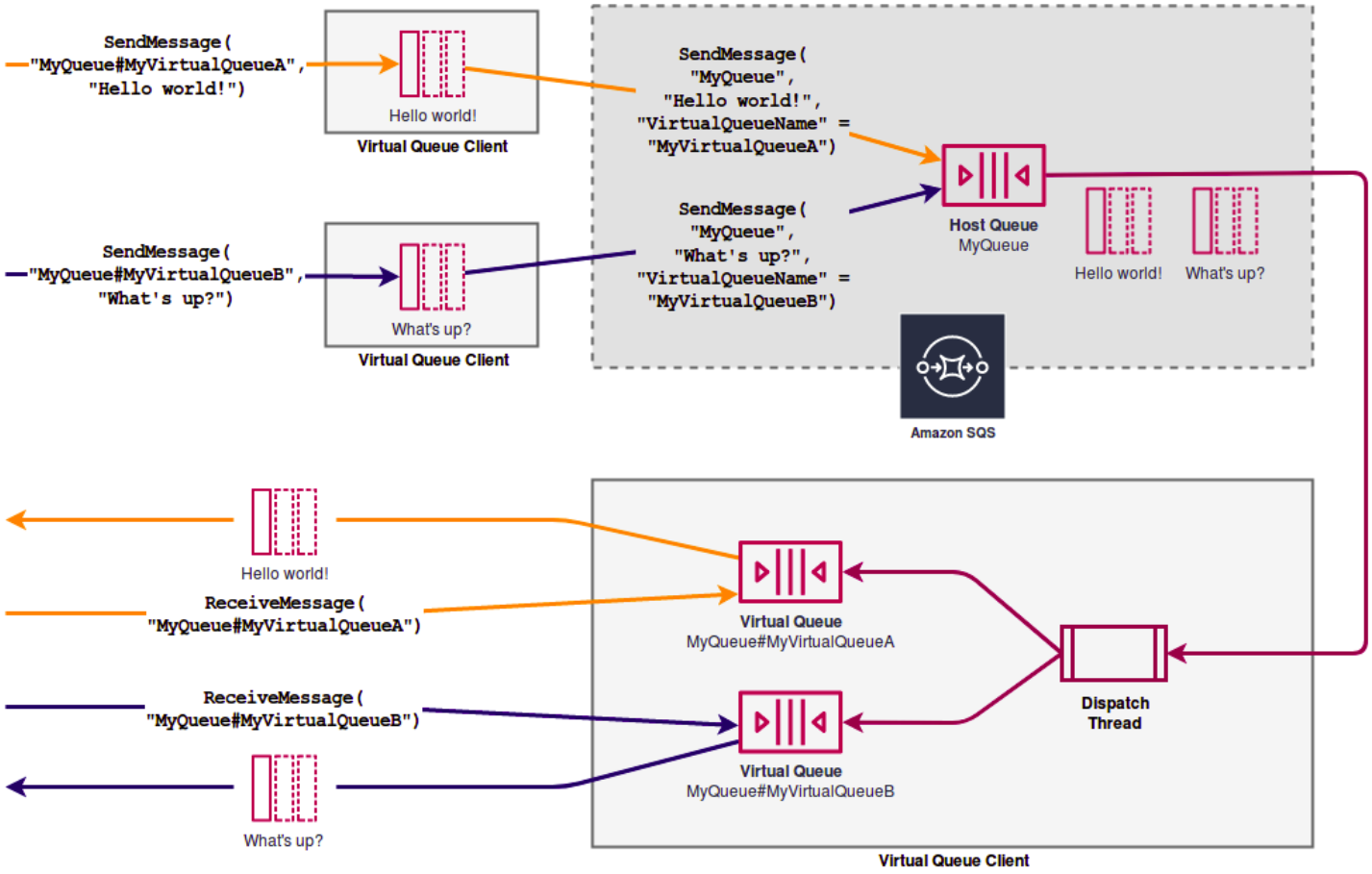
`AmazonSQSTemporaryQueuesClient`Kelas mengubah semua antrian yang dibuatnya menjadi antrian sementara secara otomatis. Ini juga membuat antrian host dengan atribut antrian yang sama secara otomatis, sesuai permintaan. Nama-nama antrian ini berbagi awalan umum yang dapat dikonfigurasi (secara default,`__RequesterClientQueues__`) yang mengidentifikasinya sebagai antrian sementara. Hal ini memungkinkan klien untuk bertindak sebagai pengganti drop-in yang mengoptimalkan kode yang ada yang membuat dan menghapus antrian. Klien juga menyertakan `AmazonSQSResponder` antarmuka `AmazonSQSRequester` dan yang memungkinkan komunikasi dua arah antara antrian.

Request-response pola pesan (antrian virtual)

Kasus penggunaan yang paling umum untuk antrian sementara adalah pola pesan per mintaan-respons, di mana pemohon membuat antrian sementara untuk menerima setiap pesan respons.

Untuk menghindari pembuatan antrian Amazon SQS untuk setiap pesan respons, Klien Antrian Sementara memungkinkan Anda membuat dan menghapus beberapa antrian sementara tanpa melakukan panggilan API Amazon SQS. Untuk informasi selengkapnya, lihat Menerapkan sistem permintaan-respons.

Diagram berikut menunjukkan konfigurasi umum menggunakan pola ini.



Contoh skenario: Memproses permintaan login

Skenario contoh berikut menunjukkan bagaimana Anda dapat menggunakan `AmazonSQSResponder` antarmuka `AmazonSQSRequester` dan untuk memproses permintaan login pengguna.

Di sisi klien

```
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();
    }
}
```

Mengirim permintaan login melakukan hal berikut:

1. Membuat antrian sementara.
2. Melampirkan URL antrian sementara ke pesan sebagai atribut.
3. Mengirim pesan.
4. Menerima respons dari antrian sementara.
5. Menghapus antrian sementara.
6. Mengembalikan respon.

Di sisi server

Contoh berikut mengasumsikan bahwa, setelah konstruksi, sebuah thread dibuat untuk melakukan polling antrian dan memang `handleLoginRequest()` gil metode untuk setiap pesan. Selain itu, `doLogin()` adalah metode yang diasumsikan.

```
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));
    }
}
```

Membersihkan antrian

Untuk memastikan bahwa Amazon SQS mengklaim kembali sumber daya dalam memori yang digunakan oleh antrian virtual, saat aplikasi Anda tidak lagi membutuhkan Klien Antrian Sementara, aplikasi harus memanggil metode tersebut. `shutdown()` Anda juga dapat menggunakan `shutdown()` metode `AmazonSQSRequester` antarmuka.

Klien Antrian Sementara juga menyediakan cara untuk menghilangkan antrian host yatim piatu. Untuk setiap antrian yang menerima panggilan API selama periode waktu tertentu (secara default, lima menit), klien menggunakan tindakan `TagQueue` API untuk menandai antrian yang tetap digunakan.

> **Note**
>
> Setiap tindakan API yang dilakukan pada antrian menandainya sebagai non-idle, termasuk `ReceiveMessage` tindakan yang tidak mengembalikan pesan.

Utas latar belakang menggunakan tindakan `ListQueues` dan `ListTags` API untuk memeriksa semua antrian dengan awalan yang dikonfigurasi, menghapus antrian apa pun yang belum diberi tag setidaknya selama lima menit. Dengan cara ini, jika satu klien tidak menutup dengan bersih, klien aktif lainnya membersihkan setelahnya. Untuk mengurangi duplikasi pekerjaan, semua klien dengan awalan yang sama berkomunikasi melalui antrian kerja internal bersama yang dinamai menurut awalan.

Pengatur waktu pesan Amazon SQS

Pengatur waktu pesan memungkinkan Anda mengatur periode tembus pandang awal untuk pesan saat ditambahkan ke antrian. Misalnya, jika Anda mengirim pesan dengan timer 45 detik, itu tetap tersembunyi dari konsumen selama 45 detik pertama. Penundaan default (minimum) untuk pesan adalah 0 detik. Maksimal 15 menit. Untuk informasi tentang mengirim pesan dengan timer menggunakan konsol, lihatMengirim pesan menggunakan antrian standar.

> **Note**
>
> Antrian FIFO tidak mendukung pengatur waktu pada pesan individual.

Untuk mengatur periode penundaan pada seluruh antrian, bukan pada pesan individual, gunakan antrian penundaan. Pengaturan pengatur waktu pesan untuk pesan individual akan mengganti `DelaySeconds` nilai apa pun pada antrean penundaan Amazon SQS.

Opsi penjadwalan yang diperluas

Meskipun antrian penundaan Amazon SQS dan pengatur waktu pesan memungkinkan penjadwalan pengiriman pesan hingga 15 menit di masa mendatang, Anda mungkin memerlukan kemampuan penjadwalan yang lebih fleksibel. Dalam kasus seperti itu, pertimbangkan untuk menggunakan EventBridge Scheduler, yang memungkinkan Anda menjadwalkan miliaran tindakan API satu kali atau berulang tanpa batasan waktu. EventBridge Scheduler adalah solusi yang direkomendasikan untuk kasus penggunaan penjadwalan pesan lanjutan.

Mengakses EventBridge Pipa Amazon melalui konsol Amazon SQS

Amazon EventBridge Pipes menghubungkan sumber ke target. Pipa dimaksudkan untuk point-to-point integrasi antara sumber dan target yang didukung, dengan dukungan untuk transformasi dan pengayaan lanjutan. EventBridge Pipes menyediakan cara yang sangat skalabel untuk menghubungkan antrian Amazon SQS Anda AWS ke layanan seperti Step Functions, Amazon SQS, dan API Gateway, serta aplikasi perangkat lunak pihak ketiga sebagai layanan (SaaS) seperti Salesforce.

Untuk menyiapkan pipa, Anda memilih sumber, menambahkan pemfilteran opsional, menentukan pengayaan opsional, dan memilih target untuk data peristiwa.

Pada halaman detail untuk antrian Amazon SQS, Anda dapat melihat pipa yang menggunakan antrian tersebut sebagai sumbernya. Dari sana, Anda juga bisa:

- Luncurkan EventBridge konsol untuk melihat detail pipa.
- Luncurkan EventBridge konsol untuk membuat pipa baru dengan antrian sebagai sumbernya.

Untuk informasi selengkapnya tentang mengonfigurasi antrian Amazon SQS sebagai sumber pipa, lihat antrian Amazon SQS sebagai sumber di Panduan Pengguna Amazon. EventBridge Untuk informasi lebih lanjut tentang EventBridge Pipa secara umum, lihat EventBridge Pipa.

Untuk mengakses EventBridge pipa untuk antrian Amazon SQS tertentu

1. Buka halaman Antrian konsol Amazon SQS.
2. Pilih antrian.
3. Pada halaman detail antrian, pilih tab EventBridge Pipes.

   Tab EventBridge Pipes menyertakan daftar pipa yang saat ini dikonfigurasi untuk menggunakan antrian yang dipilih sebagai sumber, termasuk:

   - nama pipa
   - status saat ini
   - target pipa
   - ketika pipa terakhir dimodifikasi
4. Lihat lebih banyak detail pipa atau buat pipa baru, jika diinginkan:

   - Untuk mengakses detail lebih lanjut tentang pipa:

Pilih nama pipa.

Ini meluncurkan halaman detail Pipe EventBridge konsol.

- Untuk membuat pipa baru:

  Pilih Connect Amazon SQS antrian ke pipa.

  Ini meluncurkan halaman Create pipe EventBridge konsol, dengan antrean Amazon SQS ditentukan sebagai sumber pipa. Untuk informasi selengkapnya, lihat Membuat EventBridge pipa di Panduan EventBridge Pengguna Amazon.

  > **Important**
  >
  > Pesan pada antrean Amazon SQS dibaca oleh satu pipa dan kemudian dihapus dari antrian setelah diproses, apakah pesan tersebut cocok dengan filter yang dapat Anda konfigurasikan untuk pipa itu atau tidak. Lanjutkan dengan hati-hati saat mengonfigurasi beberapa pipa untuk menggunakan antrian yang sama dengan sumbernya.

Mengelola pesan Amazon SQS besar dengan Extended Client Library dan Amazon Simple Storage Service

Gunakan Amazon SQS Extended Client Library untuk Java dan Amazon SQS Extended Client Library untuk Python untuk mengirim pesan besar, terutama untuk muatan antara 256 KB dan 2 GB. Pustaka ini menyimpan payload pesan dalam bucket Amazon S3 dan mengirim referensi ke objek yang disimpan dalam antrean Amazon SQS.

> **Note**
>
> Amazon SQS Extended Client Libraries kompatibel dengan antrian standar dan FIFO.

Mengelola pesan Amazon SQS besar menggunakan Java dan Amazon S3

Gunakan Amazon SQS Extended Client Library untuk Java dengan Amazon S3 untuk mengelola pesan Amazon SQS yang besar, terutama untuk muatan mulai dari 256 KB hingga 2 GB. Pustaka

menyimpan payload pesan dalam bucket Amazon S3 dan mengirimkan pesan yang berisi referensi ke objek yang disimpan dalam antrean Amazon SQS.

Dengan Amazon SQS Extended Client Library untuk Java, Anda dapat:

- Tentukan apakah pesan selalu disimpan di Amazon S3 atau hanya jika ukuran pesan melebihi 256 KB
- Mengirim pesan yang mereferensikan objek pesan tunggal yang disimpan dalam bucket S3
- Ambil objek pesan dari bucket Amazon S3
- Hapus objek pesan dari bucket Amazon S3

Prasyarat

Contoh berikut menggunakan AWS Java SDK. Untuk menginstal dan menyiapkan SDK, lihat Mengatur AWS SDK for Java di Panduan AWS SDK untuk Java Pengembang.

Sebelum Anda menjalankan kode contoh, konfigurasikan AWS kredensil Anda. Untuk informasi selengkapnya, lihat Menyiapkan AWS Kredensial dan Wilayah untuk Pengembangan di Panduan AWS SDK untuk Java Pengembang.

SDK for Java dan Amazon SQS Extended Client Library untuk Java memerlukan J2SE Development Kit 8.0 atau yang lebih baru.

> **Note**
>
> Anda dapat menggunakan Amazon SQS Extended Client Library for Java untuk mengelola pesan Amazon SQS menggunakan Amazon S3 hanya dengan file. AWS SDK untuk Java Anda tidak dapat melakukan ini dengan AWS CLI, konsol Amazon SQS, Amazon SQS HTTP API, atau yang lainnya. AWS SDKs

AWS SDK for Java 2.x Contoh: Menggunakan Amazon S3 untuk mengelola pesan Amazon SQS besar

Contoh SDK for SDK for Java 2.x berikut menggunakan Extended Client Library for Java untuk bekerja dengan pesan besar. Dalam konstruktor, kode berikut:

- Membuat bucket Amazon S3 dengan nama acak

- Membuat antrian SQS yang dimulai dengan `MyQueue`
- Membungkus klien Java SDK Amazon S3 standar dalam instance `AmazonSQSExtendedClient`

Dalam `sendAnReceiveMessage` metode ini, contoh mengirimkan pesan acak yang disimpan dalam bucket Amazon S3 karena lebih dari 256 KB (ukuran pesan maksimum standar). Akhirnya, metode mengambil pesan dan menampilkan informasi tentangnya ke konsol.

Anda dapat melihat contoh lengkapnya dihttps://github.com/awsdocs/aws-doc-sdk-examples/blob/94d1b24df12deda0f4fd91433b8231fed6d18b85/javav2/example_code/sqs/src/main/java/com/example/sqs/SqsExtendedClientExample.java#L1.

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

             import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.BucketLifecycleConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ExpirationStatus;
import software.amazon.awssdk.services.s3.model.LifecycleExpiration;
import software.amazon.awssdk.services.s3.model.LifecycleRule;
import software.amazon.awssdk.services.s3.model.LifecycleRuleFilter;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsResponse;
```

```
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.PutBucketLifecycleConfigurationRequest;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

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


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

    public static void main(String[] args) {

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

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

```
                .status(ExpirationStatus.ENABLED)
                .build();
        final BucketLifecycleConfiguration lifecycleConfig =
 BucketLifecycleConfiguration.builder()
                .rules(lifeCycleRule)
                .build();

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

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

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

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

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

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

        // Send the message
        final SendMessageRequest sendMessageRequest = SendMessageRequest.builder()
                .queueUrl(myQueueUrl)
                .messageBody(myLongString)
                .build();
        sqsExtended.sendMessage(sendMessageRequest);
```

```
        System.out.println("Sent the message.");

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

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

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

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


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

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

    }

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

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

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

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

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

Anda dapat menggunakan Apache Maven untuk mengonfigurasi dan membangun Amazon SQS Extended Client untuk proyek Java Anda, atau untuk membangun SDK itu sendiri. Tentukan modul individual dari SDK yang Anda gunakan dalam aplikasi Anda.

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

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

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

```
</dependencies>
```

Mengelola pesan Amazon SQS besar menggunakan Python dan Amazon S3

Gunakan Amazon SQS Amazon SQS Extended Client Library untuk Python dengan Amazon S3 untuk mengelola pesan Amazon SQS besar, terutama untuk muatan antara 256 KB dan 2 GB. Pustaka menyimpan payload pesan dalam bucket Amazon S3 dan mengirimkan pesan yang berisi referensi ke objek yang disimpan dalam antrean Amazon SQS.

Dengan Amazon SQS Extended Client Library untuk Python, Anda dapat:

- Tentukan apakah muatan selalu disimpan di Amazon S3, atau hanya disimpan di Amazon S3 jika ukuran muatan melebihi 256 KB
- Kirim pesan yang mereferensikan satu objek pesan yang disimpan di bucket Amazon S3
- Ambil objek payload yang sesuai dari bucket Amazon S3
- Hapus objek payload yang sesuai dari bucket Amazon S3

Prasyarat

Berikut ini adalah prasyarat untuk menggunakan Amazon SQS Extended Client Library untuk Python:

- AWS Akun dengan kredensi yang diperlukan. Untuk membuat AWS akun, navigasikan ke AWS halaman beranda, lalu pilih Buat AWS Akun. Ikuti petunjuk online. Untuk informasi tentang kredensial, lihat Kredensional.
- AWS SDK: Contoh pada halaman ini menggunakan AWS Python SDK Boto3. Untuk menginstal dan menyiapkan SDK, lihat dokumentasi SDK untuk Python di AWS SDK for Python Developer Guide AWS
- Python 3.x (atau yang lebih baru) dan. `pip`
- Perpustakaan Klien Diperpanjang Amazon SQS untuk Python, tersedia dari PyPI

> **Note**
>
> Anda dapat menggunakan Amazon SQS Extended Client Library untuk Python untuk mengelola pesan Amazon SQS menggunakan Amazon S3 hanya dengan SDK untuk Python.

AWS Anda tidak dapat melakukan ini dengan AWS CLI, konsol Amazon SQS, Amazon SQS HTTP API, atau yang lainnya. AWS SDKs

Mengkonfigurasi penyimpanan pesan

Amazon SQS Extended Client menggunakan atribut pesan berikut untuk mengonfigurasi opsi penyimpanan pesan Amazon S3:

- `large_payload_support`: Nama bucket Amazon S3 untuk menyimpan pesan besar.
- `always_through_s3`: Jika`True`, maka semua pesan disimpan di Amazon S3. Jika`False`, pesan yang lebih kecil dari 256 KB tidak akan diserialisasikan ke bucket s3. Nilai default-nya `False`.
- `use_legacy_attribute`: Jika`True`, semua pesan yang dipublikasikan menggunakan atribut pesan cadangan Legacy (`SQSLargePayloadSize`), bukan atribut pesan cadangan saat ini (`ExtendedPayloadSize`).

Mengelola pesan Amazon SQS besar dengan Extended Client Library untuk Python

Contoh berikut membuat bucket Amazon S3 dengan nama acak. Kemudian membuat antrian Amazon SQS bernama `MyQueue` dan mengirim pesan yang disimpan dalam bucket S3 dan lebih dari 256 KB ke antrian. Akhirnya, kode mengambil pesan, mengembalikan informasi tentangnya, dan kemudian menghapus pesan, antrian, dan ember.

```
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
```

Mengonfigurasi antrian Amazon SQS menggunakan konsol Amazon SQS

Gunakan konsol Amazon SQS untuk mengonfigurasi dan mengelola antrian dan fitur Amazon SQS. Anda juga dapat:

- Aktifkan enkripsi sisi server untuk keamanan yang ditingkatkan.
- Kaitkan antrian surat mati untuk menangani pesan yang belum diproses.
- Siapkan pemicu untuk menjalankan fungsi Lambda untuk pemrosesan berbasis peristiwa.

Kontrol akses berbasis atribut untuk Amazon SQS

Apa itu ABAC?

Attribute-based access control (ABAC) adalah proses otorisasi yang mendefinisikan izin berdasarkan tag yang dilampirkan ke pengguna dan sumber daya. AWS ABAC menyediakan kontrol akses yang terperinci dan fleksibel berdasarkan atribut dan nilai, mengurangi risiko keamanan yang terkait dengan kebijakan berbasis peran yang dikonfigurasi ulang, dan memusatkan audit dan manajemen kebijakan akses. Untuk detail selengkapnya tentang ABAC, lihat Untuk apa ABAC AWS di Panduan Pengguna IAM.

Amazon SQS mendukung ABAC dengan memungkinkan Anda mengontrol akses ke antrian Amazon SQS berdasarkan tag dan alias yang terkait dengan antrian Amazon SQS. Kunci kondisi tag dan alias yang mengaktifkan ABAC di Amazon SQS memberi otorisasi kepada prinsipal IAM untuk menggunakan antrian Amazon SQS tanpa mengedit kebijakan atau mengelola hibah. Untuk mempelajari selengkapnya tentang kunci kondisi ABAC, lihat Kunci kondisi untuk Amazon SQS di Referensi Otorisasi Layanan.

Dengan ABAC, Anda dapat menggunakan tag untuk mengonfigurasi izin dan kebijakan akses IAM untuk antrian Amazon SQS, yang membantu Anda menskalakan pengelolaan izin. Anda dapat membuat kebijakan izin tunggal di IAM menggunakan tag yang ditambahkan ke setiap peran bisnis—tanpa harus memperbarui kebijakan setiap kali menambahkan sumber daya baru. Anda juga dapat melampirkan tag ke prinsipal IAM untuk membuat kebijakan ABAC. Anda dapat mendesain kebijakan ABAC untuk mengizinkan operasi Amazon SQS saat tag pada peran pengguna IAM yang membuat panggilan cocok dengan tag antrean Amazon SQS. Untuk mempelajari lebih lanjut tentang menandai AWS, lihat Strategi AWS Penandaan dan. Tag alokasi biaya Amazon SQS

> **Note**
>
> ABAC untuk Amazon SQS saat ini tersedia di AWS semua Wilayah Komersil tempat Amazon SQS tersedia, dengan pengecualian berikut:
>
> - Asia Pasifik (Hyderabad)
> - Asia Pacific (Melbourne)
> - Eropa (Spanyol)
> - Europe (Zurich)

Mengapa saya harus menggunakan ABAC di Amazon SQS?

Berikut adalah beberapa manfaat menggunakan ABAC di Amazon SQS:

- ABAC untuk Amazon SQS memerlukan kebijakan izin yang lebih sedikit. Anda tidak perlu membuat kebijakan yang berbeda untuk fungsi pekerjaan yang berbeda. Anda dapat menggunakan tag sumber daya dan permintaan yang berlaku untuk lebih dari satu antrian, yang mengurangi overhead operasional.
- Gunakan ABAC untuk menskalakan tim dengan cepat. Izin untuk sumber daya baru secara otomatis diberikan berdasarkan tag ketika sumber daya diberi tag dengan tepat selama pembuatannya.
- Gunakan izin pada prinsipal IAM untuk membatasi akses sumber daya. Anda dapat membuat tag untuk prinsipal IAM dan menggunakannya untuk membatasi akses ke tindakan tertentu yang cocok dengan tag pada prinsipal IAM. Ini membantu Anda mengotomatiskan proses pemberian izin permintaan.
- Lacak siapa yang mengakses sumber daya Anda. Anda dapat menentukan identitas sesi dengan melihat atribut pengguna di AWS CloudTrail.

Topik

- Penandaan untuk kontrol akses di Amazon SQS
- Membuat pengguna IAM dan antrian Amazon SQS
- Menguji kontrol akses berbasis atribut di Amazon SQS

Penandaan untuk kontrol akses di Amazon SQS

Berikut ini adalah contoh penggunaan tag untuk kontrol akses di Amazon SQS. Kebijakan IAM membatasi pengguna IAM untuk semua tindakan Amazon SQS untuk semua antrian yang menyertakan tag sumber daya dengan lingkungan kunci dan produksi nilai. Untuk informasi selengkapnya, lihat Kontrol akses berbasis atribut dengan tag dan Organizations AWS.

JSON

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

Membuat pengguna IAM dan antrian Amazon SQS

Contoh berikut menjelaskan cara membuat kebijakan ABAC untuk mengontrol akses ke Amazon SQS menggunakan Konsol Manajemen AWS dan. CloudFormation

Menggunakan Konsol Manajemen AWS

Buat pengguna IAM

1. Masuk ke Konsol Manajemen AWS dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Pengguna dari panel navigasi kiri.
3. Pilih Tambah Pengguna dan masukkan nama di kotak teks Nama pengguna.

4. Pilih tombol Akses - kotak Akses terprogram dan pilih Berikutnya:Izin.
5. Pilih Selanjutnya: Tag.
6. Tambahkan kunci tag sebagai `environment` dan nilai tag sebagai`beta`.
7. Pilih Berikutnya:Tinjau dan kemudian pilih Buat pengguna.
8. Salin dan simpan ID kunci akses dan kunci akses rahasia di lokasi yang aman.

Tambahkan izin pengguna IAM

1. Pilih pengguna IAM yang Anda buat.
2. Pilih Tambahkan kebijakan inline.
3. Pada tab JSON, tempel kebijakan berikut:
4. Pilih Tinjau kebijakan.
5. Pilih Buat kebijakan.

Menggunakan AWS CloudFormation

Gunakan contoh CloudFormation template berikut untuk membuat pengguna IAM dengan kebijakan inline yang dilampirkan dan antrean 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"
```

Menguji kontrol akses berbasis atribut di Amazon SQS

Contoh berikut menunjukkan cara menguji kontrol akses berbasis atribut di Amazon SQS.

Buat antrian dengan kunci tag disetel ke lingkungan dan nilai tag disetel ke prod

Jalankan perintah AWS CLI ini untuk menguji pembuatan antrian dengan kunci tag disetel ke lingkungan dan nilai tag disetel ke prod. Jika Anda tidak memiliki AWS CLI, Anda dapat mengunduh dan mengonfigurasinya untuk mesin Anda.

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

Anda menerima `AccessDenied` kesalahan dari titik akhir Amazon SQS:

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

Ini karena nilai tag pada pengguna IAM tidak cocok dengan tag yang diteruskan dalam panggilan `CreateQueue`API. Ingat bahwa kami menerapkan tag ke pengguna IAM dengan kunci yang disetel ke `environment` dan nilai yang disetel ke`beta`.

Buat antrian dengan kunci tag disetel ke lingkungan dan nilai tag disetel ke beta

Jalankan perintah CLI ini untuk menguji pembuatan antrian dengan kunci tag disetel ke `environment` dan nilai tag disetel ke. `beta`

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

Anda menerima pesan yang mengonfirmasi keberhasilan pembuatan antrian, mirip dengan yang di bawah ini.

```
{
```

```
"QueueUrl": "<queueUrl>“
}
```

Mengirim pesan ke antrian

Jalankan perintah CLI ini untuk menguji pengiriman pesan ke antrian.

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

Respons menunjukkan pengiriman pesan yang berhasil ke antrean Amazon SQS. Izin pengguna IAM memungkinkan Anda mengirim pesan ke antrian yang memiliki `beta` tag. Tanggapan termasuk `MD5OfMessageBody` dan `MessageId` berisi pesan.

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

Mengkonfigurasi parameter antrian menggunakan konsol Amazon SQS

Saat membuat atau mengedit antrian, Anda dapat mengonfigurasi parameter berikut:

- Batas waktu visibilitas — Lamanya waktu pesan yang diterima dari antrian (oleh satu konsumen) tidak akan terlihat oleh konsumen pesan lainnya. Untuk informasi selengkapnya, lihat Batas waktu visibilitas.

  > **Note**
  >
  > Menggunakan konsol untuk mengonfigurasi batas waktu visibilitas mengonfigurasi nilai batas waktu untuk semua pesan dalam antrian. Untuk mengonfigurasi batas waktu untuk satu atau beberapa pesan, Anda harus menggunakan salah satu pesan. AWS SDKs

- Periode penyimpanan pesan — Jumlah waktu Amazon SQS menyimpan pesan yang tetap berada dalam antrean. Secara default, antrian menyimpan pesan selama empat hari. Anda dapat mengonfigurasi antrian untuk menyimpan pesan hingga 14 hari. Untuk informasi selengkapnya, lihat Periode penyimpanan pesan.

- Penundaan pengiriman - Jumlah waktu yang akan ditunda Amazon SQS sebelum mengirimkan pesan yang ditambahkan ke antrian. Untuk informasi selengkapnya, lihat Penundaan pengiriman.
- Ukuran pesan maksimum - Ukuran pesan maksimum untuk antrian ini. Untuk informasi selengkapnya, lihat Ukuran pesan maksimum.
- Menerima waktu tunggu pesan — Jumlah waktu maksimum Amazon SQS menunggu pesan tersedia setelah antrian mendapat permintaan terima. Untuk informasi selengkapnya, lihat Polling pendek dan panjang Amazon SQS.
- Aktifkan deduplikasi berbasis konten — Amazon SQS dapat secara otomatis membuat deduplikasi IDs berdasarkan isi pesan. Untuk informasi selengkapnya, lihat Antrian Amazon SQS FIFO.
- Aktifkan throughput tinggi FIFO — Gunakan untuk mengaktifkan throughput tinggi untuk pesan dalam antrian. Memilih opsi ini mengubah opsi terkait (cakupan Deduplikasi dan batas throughput FIFO) ke pengaturan yang diperlukan untuk mengaktifkan throughput tinggi untuk antrian FIFO. Untuk informasi selengkapnya, lihat Throughput tinggi untuk antrian FIFO di Amazon SQS dan Kuota pesan Amazon SQS.
- Kebijakan redrive allow: mendefinisikan antrian sumber mana yang dapat menggunakan antrian ini sebagai antrian huruf mati. Untuk informasi selengkapnya, lihat Menggunakan antrian huruf mati di Amazon SQS.

Untuk mengonfigurasi parameter antrian untuk antrian yang ada (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian. Pilih antrian dan pilih Edit.
3. Gulir ke bagian Konfigurasi.
4. Untuk batas waktu Visibilitas, masukkan durasi dan unit. Kisarannya adalah 0 detik hingga 12 jam. Nilai default-nya adalah 30 detik.
5. Untuk periode penyimpanan Pesan, masukkan durasi dan unit. Kisarannya adalah 1 menit hingga 14 hari. Nilai defaultnya adalah 4 hari.
6. Untuk antrian standar, masukkan nilai untuk Menerima waktu tunggu pesan. Kisarannya adalah 0 hingga 20 detik. Nilai defaultnya adalah 0 detik, yang menetapkan polling pendek. Setiap nilai bukan nol menetapkan polling panjang.
7. Untuk keterlambatan Pengiriman, masukkan durasi dan unit. Kisarannya adalah 0 detik hingga 15 menit. Nilai defaultnya adalah 0 detik.

8. Untuk Ukuran pesan maksimum, masukkan nilai. Kisarannya dari 1 KiB hingga 1024 KiB. Nilai defaultnya adalah 1024 KiB.
9. Untuk antrian FIFO, pilih Aktifkan deduplikasi berbasis konten untuk mengaktifkan deduplikasi berbasis konten. Pengaturan default dinonaktifkan.
10. (Opsional) Untuk antrian FIFO untuk mengaktifkan throughput yang lebih tinggi untuk mengirim dan menerima pesan dalam antrian, pilih Aktifkan FIFO throughput tinggi.

    Memilih opsi ini mengubah opsi terkait (cakupan Deduplikasi dan batas throughput FIFO) ke pengaturan yang diperlukan untuk mengaktifkan throughput tinggi untuk antrian FIFO. Jika Anda mengubah salah satu pengaturan yang diperlukan untuk menggunakan FIFO throughput tinggi, throughput normal berlaku untuk antrian, dan deduplikasi terjadi seperti yang ditentukan. Untuk informasi selengkapnya, lihat Throughput tinggi untuk antrian FIFO di Amazon SQS dan Kuota pesan Amazon SQS.
11. Untuk kebijakan Izinkan Remrive, pilih Diaktifkan. Pilih dari berikut ini: Izinkan semua (default), Dengan antrian atau Tolak semua. Saat memilih Dengan antrian, tentukan daftar hingga 10 antrian sumber berdasarkan Nama Sumber Daya Amazon (ARN).
12. Setelah Anda selesai mengonfigurasi parameter antrian, pilih Simpan.

Mengonfigurasi kebijakan akses di Amazon SQS

Saat mengedit antrian, Anda dapat mengonfigurasi kebijakan aksesnya untuk mengontrol siapa yang dapat berinteraksi dengannya.

- Kebijakan akses menentukan akun, pengguna, dan peran mana yang memiliki izin untuk mengakses antrian.
- Ini menentukan tindakan yang diizinkan, seperti `SendMessage`, `ReceiveMessage`, atau `DeleteMessage`.
- Secara default, hanya pemilik antrian yang memiliki izin untuk mengirim dan menerima pesan.

Untuk mengonfigurasi kebijakan akses untuk antrean yang ada (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pilih antrian dan pilih Edit.
4. Gulir ke bagian Kebijakan akses.

5. Edit pernyataan kebijakan akses di kotak input. Untuk informasi lebih lanjut tentang pernyataan kebijakan akses, lihatIdentitas dan manajemen akses di Amazon SQS.
6. Setelah selesai mengonfigurasi kebijakan akses, pilih Simpan.

Mengonfigurasi enkripsi sisi server untuk antrian menggunakan kunci enkripsi yang dikelola SQS

Selain opsi enkripsi sisi server (SSE) terkelola Amazon SQS default, Amazon SQS managed SSE (SSE-SQS) memungkinkan Anda membuat enkripsi sisi server terkelola khusus yang menggunakan kunci enkripsi yang dikelola SQS untuk melindungi data sensitif yang dikirim melalui antrian pesan. Dengan SSE-SQS, Anda tidak perlu membuat dan mengelola kunci enkripsi, atau memodifikasi kode Anda untuk mengenkripsi data Anda. SSE-SQS memungkinkan Anda mengirimkan data dengan aman dan membantu Anda memenuhi kepatuhan enkripsi yang ketat dan persyaratan peraturan tanpa biaya tambahan.

SSE-SQS melindungi data saat istirahat menggunakan enkripsi Advanced Encryption Standard (AES-256) 256-bit. SSE mengenkripsi pesan segera setelah Amazon SQS menerimanya. Amazon SQS menyimpan pesan dalam bentuk terenkripsi dan mendekripsi hanya saat mengirimnya ke konsumen resmi.

> **Note**
>
> - Opsi SSE default hanya efektif ketika Anda membuat antrian tanpa menentukan atribut enkripsi.
> - Amazon SQS memungkinkan Anda mematikan semua enkripsi antrian. Oleh karena itu, mematikan KMS-SSE, tidak akan secara otomatis mengaktifkan SQS-SSE. Jika Anda ingin mengaktifkan SQS-SSE setelah mematikan KMS-SSE, Anda harus menambahkan perubahan atribut dalam permintaan.

Untuk mengkonfigurasi enkripsi SSE-SQS untuk antrian (konsol)

> **Note**
>
> Setiap antrian baru yang dibuat menggunakan titik akhir HTTP (non-TLS) tidak akan mengaktifkan enkripsi SSE-SQS secara default. Ini adalah praktik terbaik keamanan untuk membuat antrian Amazon SQS menggunakan titik akhir HTTPS atau Signature Version 4.

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pilih antrian, lalu pilih Edit.
4. Perluas Enkripsi.
5. Untuk enkripsi sisi Server, pilih Diaktifkan (default).

   > **Note**
   >
   > Dengan SSE diaktifkan, anonim `SendMessage` dan `ReceiveMessage` permintaan ke antrian terenkripsi akan ditolak. Praktik terbaik keamanan Amazon SQS merekomendasikan agar tidak menggunakan permintaan anonim. Jika Anda ingin mengirim permintaan anonim ke antrian Amazon SQS, pastikan untuk menonaktifkan SSE.

6. Pilih kunci Amazon SQS (SSE-SQS). Tidak ada biaya tambahan untuk menggunakan opsi ini.
7. Pilih Simpan.

Mengonfigurasi enkripsi sisi server untuk antrian menggunakan konsol Amazon SQS

Untuk melindungi data dalam pesan antrian, Amazon SQS mengaktifkan enkripsi sisi server (SSE) secara default untuk semua antrian yang baru dibuat. Amazon SQS terintegrasi dengan Amazon Web Services Key Management Service (Amazon Web Services KMS) untuk mengelola kunci KMS untuk enkripsi sisi server (SSE). Untuk informasi tentang menggunakan SSE, lihatEnkripsi saat istirahat di Amazon SQS.

Kunci KMS yang Anda tetapkan ke antrian Anda harus memiliki kebijakan kunci yang menyertakan izin untuk semua kepala sekolah yang berwenang untuk menggunakan antrian. Untuk selengkapnya, lihat Manajemen Kunci.

Jika Anda bukan pemilik kunci KMS, atau jika Anda masuk dengan akun yang tidak memiliki `kms:ListAliases` dan `kms:DescribeKey` izin, Anda tidak akan dapat melihat informasi tentang kunci KMS di konsol Amazon SQS. Mintalah pemilik kunci KMS untuk memberi Anda izin ini. Untuk informasi selengkapnya, lihat Manajemen Kunci.

Saat Anda membuat atau mengedit antrian, Anda dapat mengonfigurasi SSE-KMS.

Untuk mengkonfigurasi SSE-KMS untuk antrian yang ada (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pilih antrian, lalu pilih Edit.
4. Perluas Enkripsi.
5. Untuk enkripsi sisi Server, pilih Diaktifkan (default).

   > **Note**
   >
   > Dengan SSE diaktifkan, anonim `SendMessage` dan `ReceiveMessage` permintaan ke antrian terenkripsi akan ditolak. Praktik terbaik keamanan Amazon SQS merekomendasikan agar tidak menggunakan permintaan anonim. Jika Anda ingin mengirim permintaan anonim ke antrian Amazon SQS, pastikan untuk menonaktifkan SSE.

6. Pilih AWS Kunci Layanan Manajemen Kunci (SSE-KMS).

   Konsol menampilkan Deskripsi, Akun, dan ARN kunci KMS dari kunci KMS.
7. Tentukan ID kunci KMS untuk antrian. Untuk informasi selengkapnya, lihat Istilah kunci.

   a. Pilih opsi Pilih alias kunci KMS.

   b. Kunci default adalah kunci KMS terkelola Amazon Web Services untuk Amazon SQS. Untuk menggunakan kunci ini, pilih dari daftar kunci KMS.

   c. Untuk menggunakan kunci KMS kustom dari akun Amazon Web Services Anda, pilih dari daftar kunci KMS. Untuk petunjuk cara membuat kunci KMS kustom, lihat Membuat Kunci di Panduan Pengembang Layanan Manajemen Kunci Amazon Web Services.

   d. Untuk menggunakan kunci KMS kustom yang tidak ada dalam daftar, atau kunci KMS kustom dari akun Amazon Web Services lainnya, pilih Masukkan alias kunci KMS dan masukkan kunci KMS Nama Sumber Daya Amazon (ARN).

8. (Opsional) Untuk periode penggunaan kembali kunci data, tentukan nilai antara 1 menit dan 24 jam. Default adalah 5 menit. Untuk informasi selengkapnya, lihat Memahami periode penggunaan kembali kunci data.
9. Ketika Anda selesai mengkonfigurasi SSE-KMS, pilih Simpan.

Mengonfigurasi tag alokasi biaya untuk antrian menggunakan konsol Amazon SQS

Untuk mengatur dan mengidentifikasi antrian Amazon SQS Anda, Anda dapat menambahkan tag alokasi biaya. Untuk informasi selengkapnya, lihat Tag alokasi biaya Amazon SQS.

- Tab Tagging pada halaman Detail menampilkan tag antrian.
- Anda dapat menambahkan atau memodifikasi tag saat membuat atau mengedit antrian.

Untuk mengonfigurasi tag untuk antrian yang ada (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pilih antrian dan pilih Edit.
4. Gulir ke bagian Tag.
5. Menambahkan, memodifikasi, atau menghapus tag antrian:

   a. Untuk menambahkan tag, pilih Tambahkan tag baru, masukkan Kunci dan Nilai, lalu pilih Tambahkan tag baru.

   b. Untuk memperbarui tag, ubah Kunci dan Nilainya.

   c. Untuk menghapus tag, pilih Hapus di samping pasangan nilai kunci-nya.

6. Setelah Anda selesai mengonfigurasi tag, pilih Simpan.

Berlangganan antrian ke topik Amazon SNS menggunakan konsol Amazon SQS

Anda dapat berlangganan satu atau beberapa antrian Amazon SQS ke topik Amazon SNS. Saat Anda memublikasikan pesan ke suatu topik, Amazon SNS mengirimkan pesan ke setiap antrian berlangganan. Amazon SQS mengelola langganan dan menangani izin yang diperlukan. Untuk informasi selengkapnya tentang Amazon SNS, lihat Apa itu Amazon SNS? di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Saat Anda berlangganan antrian Amazon SQS ke topik Amazon SNS, Amazon SNS menggunakan HTTPS untuk meneruskan pesan ke Amazon SQS. Untuk informasi tentang menggunakan Amazon SNS dengan antrian Amazon SQS terenkripsi, lihat. Konfigurasikan izin KMS untuk AWS layanan

> **Important**
>
> Amazon SQS mendukung maksimal 20 pernyataan untuk setiap kebijakan akses. Berlangganan topik Amazon SNS menambahkan satu pernyataan seperti itu. Melebihi jumlah ini akan mengakibatkan pengiriman langganan topik gagal.

Untuk berlangganan antrian ke topik Amazon SNS (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Dari daftar antrian, pilih antrian untuk berlangganan topik Amazon SNS.
4. Dari Tindakan, pilih Berlangganan topik Amazon SNS.
5. Dari Tentukan topik Amazon SNS yang tersedia untuk menu antrian ini, pilih topik Amazon SNS untuk antrian Anda.

   Jika topik SNS tidak terdaftar, pilih Masukkan ARN topik Amazon SNS, lalu masukkan nama sumber daya Amazon (ARN) topik tersebut.
6. Pilih Simpan.
7. Untuk memverifikasi langganan, publikasikan pesan ke topik dan lihat pesan dalam antrian. Untuk informasi selengkapnya, lihat Penerbitan pesan Amazon SNS di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Langganan lintas akun

Jika antrean Amazon SQS dan topik Amazon SNS Anda berbeda Akun AWS, izin tambahan diperlukan.

Pemilik topik (Akun A)

Ubah kebijakan akses topik Amazon SNS untuk memungkinkan Akun AWS antrean Amazon SQS berlangganan. Contoh pernyataan kebijakan:

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

Kebijakan ini memungkinkan akun `111122223333` untuk berlangganan`MyTopic`.

Pemilik antrian (Akun B)

Ubah kebijakan akses antrian Amazon SQS untuk mengizinkan topik Amazon SNS mengirim pesan. Contoh pernyataan kebijakan:

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

Kebijakan ini memungkinkan `MyTopic` untuk mengirim pesan ke`MyQueue`.

Langganan lintas wilayah

Untuk berlangganan topik Amazon SNS yang berbeda Wilayah AWS, pastikan bahwa:

- Kebijakan akses topik Amazon SNS memungkinkan langganan lintas wilayah.

- Kebijakan akses antrian Amazon SQS mengizinkan topik Amazon SNS untuk mengirim pesan di seluruh wilayah.

Untuk informasi selengkapnya, Mengirim pesan Amazon SNS ke antrean AWS Lambda atau fungsi Amazon SQS di Wilayah lain di Panduan Pengembang Layanan Pemberitahuan Sederhana Amazon.

Mengonfigurasi antrian Amazon SQS untuk memicu fungsi AWS Lambda

Anda dapat menggunakan fungsi Lambda untuk memproses pesan dari antrean Amazon SQS. Lambda melakukan polling antrian dan memanggil fungsi Anda secara serempak, meneruskan sekumpulan pesan sebagai acara.

Mengkonfigurasi batas waktu visibilitas

Setel batas waktu visibilitas antrian setidaknya enam kali batas waktu fungsi. Ini memastikan Lambda memiliki cukup waktu untuk mencoba lagi jika suatu fungsi dibatasi saat memproses batch sebelumnya.

Menggunakan antrian dead-letter (DLQ)

Tentukan antrian huruf mati untuk menangkap pesan yang gagal diproses oleh fungsi Lambda.

Menangani beberapa antrian dan fungsi

Fungsi Lambda dapat memproses beberapa antrian dengan membuat sumber peristiwa terpisah untuk setiap antrian. Anda juga dapat mengaitkan beberapa fungsi Lambda dengan antrian yang sama.

Izin untuk antrian terenkripsi

Jika Anda mengaitkan antrian terenkripsi dengan fungsi Lambda tetapi Lambda tidak melakukan polling untuk pesan, tambahkan `kms:Decrypt` izin ke peran eksekusi Lambda Anda.

Pembatasan

Fungsi antrian dan Lambda harus sama. Wilayah AWS

Antrian terenkripsi yang menggunakan kunci default (kunci KMS AWS terkelola untuk Amazon SQS) tidak dapat menjalankan fungsi Lambda secara berbeda. Akun AWS

Untuk detail implementasi, lihat Menggunakan AWS Lambda Amazon SQS di Panduan AWS Lambda Pengembang.

Prasyarat

Untuk mengonfigurasi pemicu fungsi Lambda, Anda harus memenuhi persyaratan berikut:

- Jika Anda menggunakan pengguna, peran Amazon SQS Anda harus menyertakan izin berikut:
  - `lambda:CreateEventSourceMapping`
  - `lambda:ListEventSourceMappings`
  - `lambda:ListFunctions`
- Peran eksekusi Lambda harus menyertakan izin berikut:
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `sqs:ReceiveMessage`
- Jika Anda mengaitkan antrian terenkripsi dengan fungsi Lambda, tambahkan izin `kms:Decrypt` ke peran eksekusi Lambda.

Untuk informasi selengkapnya, lihat Ikhtisar pengelolaan akses di Amazon SQS.

Untuk mengonfigurasi antrian untuk memicu fungsi Lambda (konsol)

1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pada halaman Antrian, pilih antrian untuk dikonfigurasi.
4. Pada halaman antrian, pilih tab pemicu Lambda.
5. Pada halaman pemicu Lambda, pilih pemicu Lambda.

   Jika daftar tidak menyertakan pemicu Lambda yang Anda butuhkan, pilih Konfigurasi pemicu fungsi Lambda. Masukkan Nama Sumber Daya Amazon (ARN) dari fungsi Lambda atau pilih sumber daya yang ada. Lalu, pilih Simpan.
6. Pilih Simpan. Konsol menyimpan konfigurasi dan menampilkan halaman Detail untuk antrian.

   Pada halaman Detail, tab pemicu Lambda menampilkan fungsi Lambda dan statusnya. Dibutuhkan sekitar 1 menit agar fungsi Lambda dikaitkan dengan antrian Anda.

7. Untuk memverifikasi hasil konfigurasi, kirim pesan ke antrian Anda, lalu lihat fungsi Lambda yang dipicu di konsol Lambda.

Mengotomatiskan pemberitahuan dari AWS layanan ke Amazon SQS menggunakan Amazon EventBridge

Amazon EventBridge memungkinkan Anda untuk mengotomatisasi Layanan AWS dan menanggapi peristiwa, seperti masalah aplikasi atau perubahan sumber daya, dalam waktu dekat.

- Anda dapat membuat aturan untuk memfilter peristiwa tertentu dan menentukan tindakan otomatis saat aturan cocok dengan peristiwa.
- EventBridge mendukung beberapa target, termasuk standar Amazon SQS dan antrian FIFO, yang menerima acara dalam format JSON.

Untuk informasi selengkapnya, lihat EventBridge Target Amazon di Panduan EventBridge Pengguna Amazon.

Mengirim pesan dengan atribut menggunakan Amazon SQS

Untuk antrian standar dan FIFO, Anda dapat menyertakan metadata terstruktur ke pesan, termasuk stempel waktu, data geospasial, tanda tangan, dan pengidentifikasi. Untuk informasi selengkapnya, lihat Atribut pesan Amazon SQS.

Untuk mengirim pesan dengan atribut ke antrian menggunakan konsol Amazon SQS

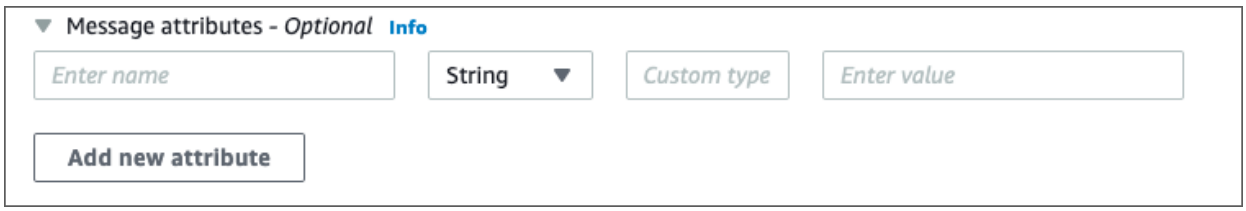
1. Buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Di panel navigasi, pilih Antrian.
3. Pada halaman Antrian, pilih antrian.
4. Pilih Kirim dan terima pesan.
5. Masukkan parameter atribut pesan.

    a. Di kotak teks nama, masukkan nama unik hingga 256 karakter.

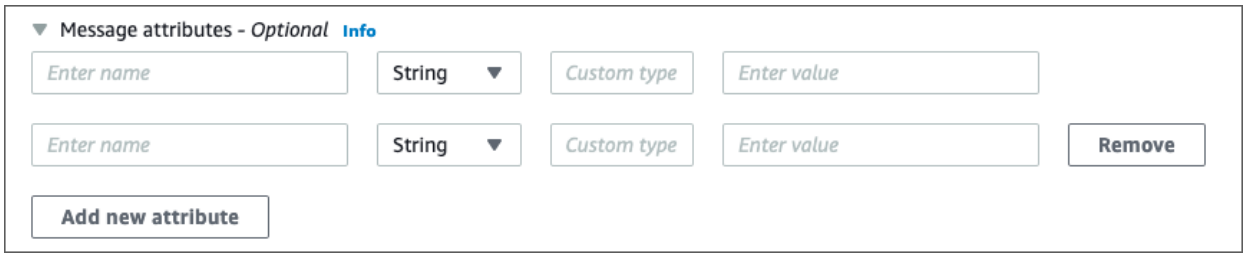
    b. Untuk jenis atribut, pilih String, Number, atau Binary.

    c. (Opsional) Masukkan tipe data khusus. Misalnya, Anda dapat menambahkan**`byte`**,**`int`**, atau **`float`** sebagai tipe data khusus untuk Nomor.

d. Di kotak teks nilai, masukkan nilai atribut pesan.



6. Untuk menambahkan atribut pesan lain., pilih Tambahkan atribut baru.



7. Anda dapat mengubah nilai atribut kapan saja sebelum mengirim pesan.
8. Untuk menghapus atribut, pilih Hapus. Untuk menghapus atribut pertama, tutup atribut Pesan.
9. Setelah selesai menambahkan atribut ke pesan, pilih Kirim pesan. Pesan Anda dikirim dan konsol menampilkan pesan sukses. Untuk melihat informasi tentang atribut pesan dari pesan terkirim, pilih Lihat detail. Pilih Selesai untuk menutup kotak dialog Rincian pesan.

Praktik terbaik Amazon SQS

Amazon SQS mengelola dan memproses antrian pesan, memungkinkan berbagai bagian aplikasi untuk bertukar pesan secara andal dan dalam skala besar. Topik ini mencakup praktik terbaik operasional utama, termasuk menggunakan polling panjang untuk mengurangi respons kosong, menerapkan antrian huruf mati untuk menangani kesalahan pemrosesan, dan mengoptimalkan izin antrian untuk keamanan.

Topik

- Penanganan kesalahan Amazon SQS dan pesan bermasalah
- Deduplikasi dan pengelompokan pesan Amazon SQS
- Pemrosesan dan pengaturan waktu pesan Amazon SQS

Penanganan kesalahan Amazon SQS dan pesan bermasalah

Topik ini memberikan petunjuk terperinci tentang mengelola dan mengurangi kesalahan di Amazon SQS, termasuk teknik untuk menangani kesalahan permintaan, menangkap pesan bermasalah, dan mengonfigurasi retensi antrian huruf mati untuk memastikan keandalan pesan.

Topik

- Menangani kesalahan permintaan di Amazon SQS
- Menangkap pesan bermasalah di Amazon SQS
- Mengatur retensi antrian huruf mati di Amazon SQS

Menangani kesalahan permintaan di Amazon SQS

Untuk menangani kesalahan permintaan, gunakan salah satu strategi berikut:

- Jika Anda menggunakan AWS SDK, Anda sudah memiliki logika coba ulang dan backoff otomatis yang Anda inginkan. Untuk informasi selengkapnya, lihat Error Retries dan Exponential Backoff di. AWSReferensi Umum Amazon Web Services
- Jika Anda tidak menggunakan fitur AWS SDK untuk coba lagi dan backoff, izinkan jeda (misalnya, 200 ms) sebelum mencoba kembali ReceiveMessagetindakan setelah tidak menerima pesan, batas waktu, atau pesan kesalahan dari Amazon SQS. Untuk penggunaan selanjutnya

`ReceiveMessage` yang memberikan hasil yang sama, biarkan jeda yang lebih lama (misalnya, 400 ms).

Menangkap pesan bermasalah di Amazon SQS

Untuk menangkap semua pesan yang tidak dapat diproses, dan untuk mengumpulkan CloudWatch metrik yang akurat, konfigurasikan antrian huruf mati.

- Kebijakan redrive mengalihkan pesan ke antrean huruf mati setelah antrian sumber gagal memproses pesan beberapa kali tertentu.
- Menggunakan antrian surat mati mengurangi jumlah pesan dan mengurangi kemungkinan mengekspos Anda ke pesan pil racun (pesan yang diterima tetapi tidak dapat diproses).
- Termasuk pesan pil racun dalam antrian dapat mendistorsi `ApproximateAgeOfOldestMessage` CloudWatch metrik dengan memberikan usia yang salah dari pesan pil racun. Mengkonfigurasi antrian huruf mati membantu menghindari alarm palsu saat menggunakan metrik ini.

Mengatur retensi antrian huruf mati di Amazon SQS

Untuk antrian standar, kedaluwarsa pesan selalu didasarkan pada stempel waktu enqueue aslinya. Ketika pesan dipindahkan ke antrian huruf mati, stempel waktu enqueue tidak berubah. `ApproximateAgeOfOldestMessage`Metrik menunjukkan kapan pesan dipindahkan ke antrian huruf mati, bukan saat pesan awalnya dikirim. Misalnya, asumsikan bahwa pesan menghabiskan 1 hari dalam antrian asli sebelum dipindahkan ke antrian huruf mati. Jika periode retensi antrian surat mati adalah 4 hari, pesan akan dihapus dari antrian surat mati setelah 3 hari dan 3 hari. `ApproximateAgeOfOldestMessage` Dengan demikian, ini adalah praktik terbaik untuk selalu mengatur periode retensi antrian huruf mati menjadi lebih lama dari periode retensi antrian asli.

Untuk antrian FIFO, stempel waktu enqueue akan disetel ulang saat pesan dipindahkan ke antrian huruf mati. `ApproximateAgeOfOldestMessage`Metrik menunjukkan kapan pesan dipindahkan ke antrian huruf mati. Dalam contoh yang sama di atas, pesan dihapus dari antrian huruf mati setelah 4 hari dan `ApproximateAgeOfOldestMessage` adalah 4 hari.

Deduplikasi dan pengelompokan pesan Amazon SQS

Topik ini memberikan praktik terbaik untuk memastikan pemrosesan pesan yang konsisten di Amazon SQS. Ini menjelaskan cara menggunakan:

- `MessageDeduplicationId`untuk mencegah duplikat pesan dalam antrian FIFO.
- `MessageGroupId`untuk mengelola pemesanan pesan dalam grup pesan yang berbeda.

Topik

- Menghindari pemrosesan pesan yang tidak konsisten di Amazon SQS
- Menggunakan ID deduplikasi pesan di Amazon SQS
- Menggunakan ID grup pesan dengan Amazon SQS FIFO Queues
- Menggunakan ID percobaan permintaan terima Amazon SQS

Menghindari pemrosesan pesan yang tidak konsisten di Amazon SQS

Karena Amazon SQS adalah sistem terdistribusi, adalah mungkin bagi konsumen untuk tidak menerima pesan bahkan ketika Amazon SQS menandai pesan sebagai terkirim saat berhasil kembali dari `ReceiveMessage`panggilan metode API. Dalam hal ini, Amazon SQS mencatat pesan yang dikirimkan setidaknya sekali, meskipun konsumen belum pernah menerimanya. Karena tidak ada upaya tambahan untuk mengirimkan pesan dalam kondisi ini, kami tidak menyarankan menyetel jumlah penerimaan maksimum menjadi 1 untuk antrian huruf mati.

Menggunakan ID deduplikasi pesan di Amazon SQS

`MessageDeduplicationId`adalah token yang hanya digunakan dalam antrian Amazon SQS FIFO untuk mencegah pengiriman pesan duplikat. Ini memastikan bahwa dalam jendela deduplikasi 5 menit, hanya satu contoh pesan dengan ID deduplikasi yang sama diproses dan dikirim.

Jika Amazon SQS telah menerima pesan dengan ID deduplikasi tertentu, pesan berikutnya dengan ID yang sama akan diakui tetapi tidak dikirimkan ke konsumen.

> **Note**
>
> Amazon SQS terus melacak ID deduplikasi bahkan setelah pesan diterima dan dihapus.

Topik

- Kapan memberikan ID deduplikasi pesan di Amazon SQS
- Mengaktifkan deduplikasi untuk sistem produsen/konsumen tunggal di Amazon SQS

- Skenario pemulihan pemadaman di Amazon SQS
- Mengonfigurasi batas waktu visibilitas di Amazon SQS

Kapan memberikan ID deduplikasi pesan di Amazon SQS

Produser harus menentukan ID deduplikasi pesan dalam skenario berikut:

- Saat mengirim badan pesan identik yang harus diperlakukan sebagai unik.
- Saat mengirim pesan dengan konten yang sama tetapi atribut pesan yang berbeda, memastikan setiap pesan diproses secara terpisah.
- Saat mengirim pesan dengan konten yang berbeda (misalnya, penghitung coba lagi di badan pesan) tetapi mengharuskan Amazon SQS untuk mengenalinya sebagai duplikat.

Mengaktifkan deduplikasi untuk sistem produsen/konsumen tunggal di Amazon SQS

Jika Anda memiliki satu produsen dan satu konsumen, dan pesannya unik karena menyertakan ID pesan khusus aplikasi di badan, ikuti praktik terbaik berikut:

- Aktifkan deduplikasi berbasis konten untuk antrian (setiap pesan Anda memiliki badan yang unik). Produser dapat menghilangkan ID deduplikasi pesan.
- Saat deduplikasi berbasis konten diaktifkan untuk antrean Amazon SQS FIFO, dan pesan dikirim dengan ID deduplikasi, ID deduplikasi akan mengganti ID deduplikasi berbasis konten yang dihasilkan. `SendMessage`
- Meskipun konsumen tidak diharuskan untuk memberikan ID percobaan permintaan terima untuk setiap permintaan, ini adalah praktik terbaik karena memungkinkan urutan coba gagal untuk dijalankan lebih cepat.
- Anda dapat mencoba lagi mengirim atau menerima permintaan karena tidak mengganggu urutan pesan dalam antrian FIFO.

Skenario pemulihan pemadaman di Amazon SQS

Proses deduplikasi dalam antrian FIFO sensitif terhadap waktu. Saat merancang aplikasi Anda, pastikan bahwa produsen dan konsumen dapat pulih dari pemadaman klien atau jaringan tanpa menimbulkan duplikat atau kegagalan pemrosesan.

Pertimbangan produsen

- Amazon SQS memberlakukan jendela deduplikasi 5 menit.
- Jika produser mencoba ulang `SendMessage`permintaan setelah 5 menit, Amazon SQS memperlakukannya sebagai pesan baru, berpotensi membuat duplikat.

Pertimbangan konsumen

- Jika konsumen gagal memproses pesan sebelum batas waktu visibilitas berakhir, konsumen lain dapat menerima dan memprosesnya, yang mengarah ke pemrosesan duplikat.
- Sesuaikan batas waktu visibilitas berdasarkan waktu pemrosesan aplikasi Anda.
- Gunakan `ChangeMessageVisibility`API untuk memperpanjang batas waktu saat pesan masih diproses.
- Jika pesan berulang kali gagal diproses, arahkan ke antrian huruf mati (DLQ) alih-alih membiarkannya diproses ulang tanpa batas waktu.

- Produsen harus menyadari interval deduplikasi antrian. Amazon SQS memiliki interval deduplikasi 5 menit. Mencoba kembali `SendMessage` permintaan setelah interval deduplikasi berakhir dapat memperkenalkan pesan duplikat ke dalam antrian. Misalnya, perangkat seluler di dalam mobil mengirim pesan yang pesanannya penting. Jika mobil kehilangan konektivitas seluler untuk jangka waktu tertentu sebelum menerima pengakuan, mencoba kembali permintaan setelah mendapatkan kembali konektivitas seluler dapat membuat duplikat.
- Konsumen harus memiliki batas waktu visibilitas yang meminimalkan risiko tidak dapat memproses pesan sebelum batas waktu visibilitas berakhir. Anda dapat memperpanjang batas waktu visibilitas saat pesan sedang diproses dengan memanggil tindakan. `ChangeMessageVisibility` Namun, jika batas waktu visibilitas berakhir, konsumen lain dapat segera mulai memproses pesan, menyebabkan pesan diproses beberapa kali. Untuk menghindari skenario ini, konfigurasikan antrian huruf mati.

Mengonfigurasi batas waktu visibilitas di Amazon SQS

Untuk memastikan pemrosesan pesan yang andal, atur batas waktu visibilitas menjadi lebih lama dari batas waktu baca AWS SDK. Ini berlaku saat menggunakan `ReceiveMessage`API dengan polling pendek dan polling panjang. Batas waktu visibilitas yang lebih lama mencegah pesan tersedia bagi konsumen lain sebelum permintaan asli selesai, mengurangi risiko pemrosesan duplikat.

Menggunakan ID grup pesan dengan Amazon SQS FIFO Queues

Dalam antrian FIFO (First-In-First-Out), `MessageGroupId`adalah atribut yang mengatur pesan ke dalam grup yang berbeda. Pesan dalam grup pesan yang sama selalu diproses satu per satu, dalam urutan yang ketat, memastikan bahwa tidak ada dua pesan dari grup yang sama yang diproses secara bersamaan. Dalam antrian standar, menggunakan `MessageGroupId` memungkinkan antrian yang adil. Jika pemesanan ketat diperlukan, gunakan antrian FIFO.

Topik

- Menginterleaving beberapa grup pesan yang dipesan di Amazon SQS
- Mencegah pemrosesan duplikat dalam sistem multi-produsen/konsumen di Amazon SQS
- Hindari backlog pesan besar dengan ID grup pesan yang sama di Amazon SQS
- Hindari menggunakan kembali ID grup pesan yang sama dengan antrian virtual di Amazon SQS

Menginterleaving beberapa grup pesan yang dipesan di Amazon SQS

Untuk menginterleave beberapa grup pesan yang diurutkan dalam satu antrian FIFO, tetapkan a `MessageGroupId`ke setiap grup (misalnya, data sesi untuk pengguna yang berbeda). Hal ini memungkinkan beberapa konsumen untuk membaca dari antrian secara bersamaan sambil memastikan bahwa pesan dalam grup yang sama diproses secara berurutan.

Ketika pesan dengan spesifik `MessageGroupId` sedang diproses dan tidak terlihat, tidak ada konsumen lain yang dapat memproses pesan dari grup yang sama hingga batas waktu visibilitas berakhir atau pesan dihapus.

Mencegah pemrosesan duplikat dalam sistem multi-produsen/konsumen di Amazon SQS

Dalam sistem throughput tinggi, latensi rendah di mana pemesanan pesan bukan prioritas, produsen dapat menetapkan unik untuk setiap pesan. `MessageGroupId` Ini memastikan bahwa antrian Amazon SQS FIFO menghilangkan duplikat, bahkan dalam pengaturan multi-produsen/multi-konsumen. Meskipun pendekatan ini mencegah pesan duplikat, itu tidak menjamin pemesanan pesan karena setiap pesan diperlakukan sebagai grup independennya sendiri.

Dalam sistem apa pun dengan banyak produsen dan konsumen, selalu ada risiko pengiriman duplikat. Jika konsumen gagal memproses pesan sebelum batas waktu visibilitas berakhir, Amazon

SQS membuat pesan tersebut tersedia kembali, berpotensi memungkinkan konsumen lain untuk mengambilnya. Untuk mengurangi hal ini, pastikan pengaturan batas waktu pemberitahuan dan visibilitas pesan yang tepat berdasarkan waktu pemrosesan.

Hindari backlog pesan besar dengan ID grup pesan yang sama di Amazon SQS

Antrian FIFO mendukung maksimal 120.000 pesan dalam penerbangan (pesan yang diterima oleh konsumen tetapi belum dihapus). Jika batas ini tercapai, Amazon SQS tidak mengembalikan kesalahan, tetapi pemrosesan mungkin terpengaruh. Anda dapat meminta peningkatan di luar batas ini dengan menghubungi AWS Support.

Antrian FIFO memindai 120.000 pesan pertama untuk menentukan grup pesan yang tersedia. Jika backlog besar terbentuk dalam satu grup pesan, pesan dari grup lain yang dikirim nanti akan tetap diblokir hingga backlog diproses.

> **Note**
>
> Backlog pesan dapat terjadi ketika konsumen berulang kali gagal memproses pesan. Ini bisa disebabkan oleh masalah konten pesan atau kegagalan sisi konsumen. Untuk mencegah penundaan pemrosesan pesan, konfigurasikan antrian huruf mati untuk memindahkan pesan yang belum diproses setelah beberapa upaya gagal. Ini memastikan bahwa pesan lain dalam grup pesan yang sama dapat diproses, mencegah kemacetan sistem.

Hindari menggunakan kembali ID grup pesan yang sama dengan antrian virtual di Amazon SQS

Saat menggunakan antrian virtual dengan antrian host bersama, hindari menggunakan kembali antrian virtual yang berbeda `MessageGroupId`. Jika beberapa antrian virtual berbagi antrian host yang sama dan berisi pesan yang sama`MessageGroupId`, pesan tersebut dapat memblokir satu sama lain, mencegah pemrosesan yang efisien. Untuk memastikan pemrosesan pesan yang lancar, tetapkan `MessageGroupId` nilai unik untuk pesan dalam antrian virtual yang berbeda.

Menggunakan ID percobaan permintaan terima Amazon SQS

ID percobaan permintaan terima adalah token unik yang digunakan untuk menghapus duplikasi `ReceiveMessage`panggilan di Amazon SQS. Selama pemadaman jaringan atau masalah konektivitas antara aplikasi Anda dan Amazon SQS, praktik terbaik adalah:

- Berikan ID percobaan permintaan terima saat melakukan `ReceiveMessage` panggilan.
- Coba lagi menggunakan ID percobaan permintaan terima yang sama jika operasi gagal.

Pemrosesan dan pengaturan waktu pesan Amazon SQS

Topik ini memberikan panduan komprehensif tentang mengoptimalkan kecepatan dan efisiensi penanganan pesan di Amazon SQS, termasuk strategi untuk pemrosesan pesan tepat waktu, memilih mode polling terbaik, dan mengonfigurasi polling panjang untuk meningkatkan kinerja.

Topik

- Memproses pesan tepat waktu di Amazon SQS
- Mengatur polling panjang di Amazon SQS
- Menggunakan mode polling yang sesuai di Amazon SQS

Memproses pesan tepat waktu di Amazon SQS

Pengaturan batas waktu visibilitas tergantung pada berapa lama waktu yang dibutuhkan aplikasi Anda untuk memproses dan menghapus pesan. Misalnya, jika aplikasi Anda memerlukan 10 detik untuk memproses pesan dan Anda mengatur batas waktu visibilitas menjadi 15 menit, Anda harus menunggu waktu yang relatif lama untuk mencoba memproses pesan lagi jika upaya pemrosesan sebelumnya gagal. Atau, jika aplikasi Anda memerlukan 10 detik untuk memproses pesan tetapi Anda menetapkan batas waktu visibilitas hanya 2 detik, pesan duplikat diterima oleh konsumen lain saat konsumen asli masih mengerjakan pesan.

Untuk memastikan bahwa ada cukup waktu untuk memproses pesan, gunakan salah satu strategi berikut:

- Jika Anda tahu (atau dapat memperkirakan secara wajar) berapa lama waktu yang dibutuhkan untuk memproses pesan, perpanjang batas waktu visibilitas pesan ke waktu maksimum yang diperlukan untuk memproses dan menghapus pesan. Untuk informasi selengkapnya, lihat Mengonfigurasi Batas Waktu Visibilitas.
- Jika Anda tidak tahu berapa lama waktu yang dibutuhkan untuk memproses pesan, buat detak jantung untuk proses konsumen Anda: Tentukan batas waktu visibilitas awal (misalnya, 2 menit) dan lalu—selama konsumen Anda masih mengerjakan pesan tersebut—terus perpanjang batas waktu visibilitas sebanyak 2 menit setiap menit.

> **Important**
>
> Batas waktu visibilitas maksimum adalah 12 jam dari waktu Amazon SQS menerima permintaan. `ReceiveMessage` Memperpanjang batas waktu visibilitas tidak mengatur ulang maksimum 12 jam.
> Selain itu, Anda mungkin tidak dapat mengatur batas waktu pada pesan individual ke 12 jam penuh (misalnya 43.200 detik) sejak `ReceiveMessage` permintaan memulai pengatur waktu. Misalnya, jika Anda menerima pesan dan segera mengatur maksimum 12 jam dengan mengirim `ChangeMessageVisibility` panggilan `VisibilityTimeout` sama dengan 43.200 detik, kemungkinan akan gagal. Namun, menggunakan nilai 43.195 detik akan berfungsi kecuali ada penundaan yang signifikan antara meminta pesan melalui `ReceiveMessage` dan memperbarui batas waktu visibilitas. Jika konsumen Anda membutuhkan waktu lebih dari 12 jam, pertimbangkan untuk menggunakan Step Functions.

Mengatur polling panjang di Amazon SQS

Ketika waktu tunggu untuk tindakan `ReceiveMessage` API lebih besar dari 0, polling panjang berlaku. Waktu tunggu polling maksimum yang panjang adalah 20 detik. Polling panjang membantu mengurangi biaya penggunaan Amazon SQS dengan mengurangi jumlah respons kosong (bila tidak ada pesan yang tersedia untuk `ReceiveMessage` permintaan) dan respons kosong palsu (saat pesan tersedia tetapi tidak disertakan dalam respons). Untuk informasi selengkapnya, lihat Polling pendek dan panjang Amazon SQS.

Untuk pemrosesan pesan yang optimal, gunakan strategi berikut:

- Dalam kebanyakan kasus, Anda dapat mengatur waktu `ReceiveMessage` tunggu menjadi 20 detik. Jika 20 detik terlalu lama untuk aplikasi Anda, tetapkan waktu `ReceiveMessage` tunggu yang lebih pendek (minimum 1 detik). Jika Anda tidak menggunakan AWS SDK untuk mengakses Amazon SQS, atau jika Anda mengonfigurasi SDK agar memiliki waktu tunggu AWS yang lebih singkat, Anda mungkin harus memodifikasi klien Amazon SQS untuk mengizinkan permintaan yang lebih lama atau menggunakan waktu tunggu yang lebih singkat untuk polling yang lama.
- Jika Anda menerapkan polling panjang untuk beberapa antrian, gunakan satu utas untuk setiap antrian, bukan satu utas untuk semua antrian. Menggunakan satu utas untuk setiap antrian memungkinkan aplikasi Anda memproses pesan di setiap antrian saat tersedia, sementara

menggunakan satu utas untuk polling beberapa antrian dapat menyebabkan aplikasi Anda tidak dapat memproses pesan yang tersedia di antrian lain saat aplikasi menunggu (hingga 20 detik) untuk antrian yang tidak memiliki pesan yang tersedia.

> **Important**
>
> Untuk menghindari kesalahan HTTP, pastikan batas waktu respons HTTP untuk `ReceiveMessage` permintaan lebih panjang dari `WaitTimeSeconds` parameter. Untuk informasi selengkapnya, lihat ReceiveMessage.

Menggunakan mode polling yang sesuai di Amazon SQS

- Polling panjang memungkinkan Anda mengkonsumsi pesan dari antrean Amazon SQS segera setelah tersedia.
  - Untuk mengurangi biaya penggunaan Amazon SQS dan mengurangi jumlah penerima kosong ke antrian kosong (tanggapan terhadap `ReceiveMessage` tindakan yang tidak menampilkan pesan), aktifkan polling panjang. Untuk informasi selengkapnya, lihat Polling Panjang Amazon SQS.
  - Untuk meningkatkan efisiensi saat melakukan polling untuk beberapa utas dengan beberapa penerima, kurangi jumlah utas.
  - Jajak pendapat panjang lebih disukai daripada polling pendek dalam banyak kasus.
- Polling singkat segera mengembalikan respons, meskipun antrian Amazon SQS yang disurvei kosong.
  - Untuk memenuhi persyaratan aplikasi yang mengharapkan tanggapan segera terhadap `ReceiveMessage` permintaan, gunakan jajak pendapat singkat.
  - Jajak pendapat pendek ditagih sama dengan long polling.

Contoh Amazon SQS Java SDK

AWS SDK untuk Java Ini memungkinkan Anda membangun aplikasi Java yang berinteraksi dengan Amazon SQS dan lainnya. Layanan AWS

- Untuk menginstal dan menyiapkan SDK, lihat Memulai di Panduan AWS SDK for Java 2.x Pengembang.
- Untuk operasi antrean dasar—seperti membuat antrean atau mengirim pesan—lihat Bekerja dengan Antrian Pesan Amazon SQS di Panduan Pengembang.AWS SDK for Java 2.x
- Panduan ini juga mencakup contoh fitur Amazon SQS tambahan, seperti:
  - Menggunakan enkripsi sisi server dengan antrian Amazon SQS
  - Mengkonfigurasi tag untuk antrian Amazon SQS
  - Mengirim atribut pesan ke antrian Amazon SQS

Menggunakan enkripsi sisi server dengan antrian Amazon SQS

Gunakan AWS SDK untuk Java untuk menambahkan enkripsi sisi server (SSE) ke antrean Amazon SQS. Setiap antrian menggunakan kunci AWS Key Management Service (AWS KMS) KMS untuk menghasilkan kunci enkripsi data. Contoh ini menggunakan kunci KMS AWS terkelola untuk Amazon SQS.

Untuk informasi selengkapnya tentang penggunaan SSE dan peran kunci KMS, lihat. Enkripsi saat istirahat di Amazon SQS

Menambahkan SSE ke antrian yang ada

Untuk mengaktifkan enkripsi sisi server untuk antrian yang ada, gunakan `SetQueueAttributes` metode untuk mengatur atribut. `KmsMasterKeyId`

Contoh kode berikut menetapkan AWS KMS key sebagai kunci KMS AWS terkelola untuk Amazon SQS. Contoh ini juga mengatur periode AWS KMS key penggunaan kembali menjadi 140 detik.

Sebelum Anda menjalankan kode contoh, pastikan Anda telah menetapkan AWS kredensialnya. Untuk informasi selengkapnya, lihat Menyiapkan AWS Kredensial dan Wilayah untuk Pengembangan di Panduan AWS SDK for Java 2.x Pengembang.

```
    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();
        }
    }
```

Menonaktifkan SSE untuk antrian

Untuk menonaktifkan enkripsi sisi server untuk antrian yang ada, setel `KmsMasterKeyId` atribut ke string kosong menggunakan metode. `SetQueueAttributes`

> **Important**
>
> `null` bukanlah nilai yang valid untuk `KmsMasterKeyId`.

Membuat antrian dengan SSE

Untuk mengaktifkan SSE saat Anda membuat antrian, tambahkan `KmsMasterKeyId` atribut ke metode `CreateQueue` API.

Contoh berikut membuat antrian baru dengan SSE diaktifkan. Antrian menggunakan kunci KMS AWS terkelola untuk Amazon SQS. Contoh ini juga mengatur periode AWS KMS key penggunaan kembali menjadi 160 detik.

Sebelum Anda menjalankan kode contoh, pastikan Anda telah menetapkan AWS kredensialnya. Untuk informasi selengkapnya, lihat Menyiapkan AWS Kredensial dan Wilayah untuk Pengembangan di Panduan AWS SDK for Java 2.x Pengembang.

```
// 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);
```

Mengambil atribut SSE

Untuk informasi tentang mengambil atribut antrian, lihat Contoh di Referensi API Layanan Antrian Sederhana Amazon.

Untuk mengambil ID kunci KMS atau periode penggunaan kembali kunci data untuk antrian tertentu, jalankan `GetQueueAttributes` metode dan ambil nilai dan. `KmsMasterKeyId` `KmsDataKeyReusePeriodSeconds`

Mengkonfigurasi tag untuk antrian Amazon SQS

Gunakan tag alokasi biaya untuk membantu mengatur dan mengidentifikasi antrian Amazon SQS Anda. Contoh berikut menunjukkan cara mengkonfigurasi tag menggunakan AWS SDK untuk Java. Untuk informasi selengkapnya, lihat Tag alokasi biaya Amazon SQS.

Sebelum Anda menjalankan kode contoh, pastikan Anda telah menetapkan AWS kredensialnya. Untuk informasi selengkapnya, lihat Menyiapkan AWS Kredensial dan Wilayah untuk Pengembangan di Panduan AWS SDK for Java 2.x Pengembang.

Mencantumkan tanda

Untuk membuat daftar tag untuk antrian, gunakan `ListQueueTags` metode ini.

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

Menambahkan atau memperbarui tag

Untuk menambahkan atau memperbarui nilai tag untuk antrian, gunakan `TagQueue` metode ini.

```
 // 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);
```

Menghapus tanda

Untuk menghapus satu atau beberapa tag dari antrian, gunakan `UntagQueue` metode ini. Contoh berikut menghapus `Accounting ID` tag.

```

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

```
        .build();

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

Mengirim atribut pesan ke antrian Amazon SQS

Anda dapat menyertakan metadata terstruktur (seperti stempel waktu, data geospasial, tanda tangan, dan pengenal) dengan pesan menggunakan atribut pesan. Untuk informasi selengkapnya, lihat Atribut pesan Amazon SQS.

Sebelum Anda menjalankan kode contoh, pastikan Anda telah menetapkan AWS kredensialnya. Untuk informasi selengkapnya, lihat Mengatur AWS Kredensial dan Wilayah untuk Pengembangan di Panduan AWS SDK for Java 2.x Pengembang.

Mendefinisikan atribut

Untuk menentukan atribut untuk pesan, tambahkan kode berikut, yang menggunakan tipe `MessageAttributeValue` data. Untuk informasi selengkapnya, lihat Komponen atribut pesan dan Jenis data atribut pesan.

AWS SDK untuk Java Secara otomatis menghitung checksum isi pesan dan atribut pesan dan membandingkannya dengan data yang dikembalikan Amazon SQS. Untuk informasi selengkapnya, lihat Panduan AWS SDK for Java 2.x Pengembang dan Menghitung ringkasan pesan MD5 untuk atribut pesan untuk bahasa pemrograman lainnya.

String

Contoh ini mendefinisikan `String` atribut bernama `Name` dengan nilai`Jane`.

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

Number

Contoh ini mendefinisikan `Number` atribut bernama `AccurateWeight` dengan nilai`230.0000000000000001`.

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

Binary

Contoh ini mendefinisikan `Binary` atribut bernama `ByteArray` dengan nilai array 10-byte yang tidak diinisialisasi.

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

String (custom)

Contoh ini mendefinisikan atribut kustom `String.EmployeeId` bernama `EmployeeId` dengan nilai`ABC123456`.

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

Number (custom)

Contoh ini mendefinisikan atribut kustom `Number.AccountId` bernama `AccountId` dengan nilai`000123456`.

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

> **Note**
>
> Karena tipe data dasar adalah`Number`, `ReceiveMessage` metode kembali`123456`.

Binary (custom)

Contoh ini mendefinisikan atribut kustom `Binary.JPEG` bernama `ApplicationIcon` dengan nilai array 10-byte yang tidak diinisialisasi.

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

Mengirim pesan dengan atribut

Contoh ini menambahkan atribut ke `SendMessageRequest` sebelum mengirim pesan.

```
// 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

Jika Anda mengirim pesan ke antrian First-In-First-Out (FIFO), pastikan `sendMessage` metode dijalankan setelah Anda memberikan ID grup pesan.
Jika Anda menggunakan `SendMessageBatch` metode, bukan`SendMessage`, Anda harus menentukan atribut pesan untuk setiap pesan dalam batch.

Menggunakan API dengan Amazon SQS

Topik ini memberikan informasi tentang membangun endpoint Amazon SQS, membuat permintaan API kueri menggunakan metode GET dan POST, dan menggunakan tindakan API batch. Untuk informasi terperinci tentang tindakan Amazon SQS —termasuk parameter, kesalahan, contoh, dan tipe data, lihat Referensi API Layanan Antrian Sederhana Amazon.

Untuk mengakses Amazon SQS menggunakan berbagai bahasa pemrograman, Anda juga dapat menggunakan AWS SDK, yang berisi fungsionalitas otomatis berikut:

- Secara kriptografi menandatangani permintaan layanan Anda
- Mencoba kembali permintaan
- Menangani respons kesalahan

Untuk informasi selengkapnya, lihat the section called “Bekerja dengan AWS SDK”.

Untuk informasi alat baris perintah, lihat bagian Amazon SQS di Referensi AWS CLI Perintah dan Referensi Alat AWS untuk PowerShell Cmdlet.

Amazon SQS API dengan AWS protokol JSON

Amazon SQS menggunakan protokol AWS JSON sebagai mekanisme transportasi untuk semua Amazon SQS API pada versi SDK yang ditentukan.AWS AWS Protokol JSON menyediakan throughput yang lebih tinggi, latensi yang lebih rendah, dan komunikasi aplikasi-ke-aplikasi yang lebih cepat. AWS Protokol JSON lebih efisien dalam serialization/deserialization permintaan dan tanggapan jika dibandingkan dengan protokol AWS kueri. Jika Anda masih lebih suka menggunakan protokol AWS kueri dengan SQS API, lihat Bahasa apa yang didukung untuk protokol AWS JSON yang digunakan di Amazon APIs SQS? versi AWS SDK yang mendukung protokol kueri Amazon AWS SQS.

Amazon SQS menggunakan protokol AWS JSON untuk berkomunikasi antara klien AWS SDK (misalnya, Java, Python, Golang,) dan JavaScript server Amazon SQS. Permintaan HTTP dari operasi Amazon SQS API menerima input berformat JSON. Operasi Amazon SQS dijalankan, dan respons eksekusi dikirim kembali ke klien SDK dalam format JSON. Dibandingkan dengan AWS query, AWS JSON lebih sederhana, lebih cepat, dan lebih efisien untuk mengangkut data antara klien dan server.

- AWS Protokol JSON bertindak sebagai mediator antara klien dan server Amazon SQS.

- Server tidak memahami bahasa pemrograman di mana operasi Amazon SQS dibuat, tetapi memahami protokol AWS JSON.
- Protokol AWS JSON menggunakan serialisasi (konversi objek ke format JSON) dan de-serialisasi (konversi format JSON ke objek) antara klien Amazon SQS dan server.

Untuk informasi selengkapnya tentang protokol AWS JSON dengan Amazon SQS, lihat. Protokol JSON Amazon SQS AWS FAQs

AWS Protokol JSON tersedia pada versi AWS SDK yang ditentukan. Untuk meninjau versi SDK dan tanggal rilis di seluruh varian bahasa, lihat AWS matriks dukungan versi SDK dan Alat di Panduan Referensi AWS SDK dan Alat

Membuat permintaan API kueri menggunakan protokol AWS JSON di Amazon SQS

Topik ini menjelaskan cara membuat endpoint Amazon SQS, membuat permintaan POST, dan menafsirkan tanggapan.

> **Note**
>
> AWS Protokol JSON didukung untuk sebagian besar varian bahasa. Untuk daftar lengkap varian bahasa yang didukung, lihatBahasa apa yang didukung untuk protokol AWS JSON yang digunakan di Amazon APIs SQS?.

Membangun titik akhir

Untuk bekerja dengan antrian Amazon SQS, Anda harus membuat titik akhir. Untuk informasi tentang titik akhir Amazon SQS, lihat halaman berikut di: Referensi Umum Amazon Web

- Titik akhir regional
- Titik akhir dan kuota Layanan Antrian Sederhana Amazon

Setiap titik akhir Amazon SQS bersifat independen. Misalnya, jika dua antrian diberi nama MyQueuedan satu memiliki titik akhir `sqs.us-east-2.amazonaws.com` sementara yang lain memiliki titik akhir`sqs.eu-west-2.amazonaws.com`, kedua antrian tidak berbagi data apa pun satu sama lain.

Berikut ini adalah contoh dari endpoint yang membuat permintaan untuk membuat antrian.

```
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**
>
> Nama antrian dan antrian peka huruf URLs besar/kecil.
> Struktur *AUTHPARAMS* tergantung pada tanda tangan permintaan API. Untuk informasi selengkapnya, lihat Menandatangani Permintaan AWS API di Referensi Umum Amazon Web Services.

Membuat permintaan POST

Permintaan Amazon SQS POST mengirimkan parameter kueri sebagai formulir di badan permintaan HTTP.

Berikut ini adalah contoh dari header HTTP dengan `X-Amz-Target` set keAmazonSQS.<operationName>, dan header HTTP dengan `Content-Type` set keapplication/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"
}
```

Permintaan HTTP POST ini mengirimkan pesan ke antrian Amazon SQS.

> **Note**
>
> Baik header HTTP `X-Amz-Target` dan `Content-Type` diperlukan.
> Klien HTTP Anda mungkin menambahkan item lain ke permintaan HTTP, sesuai dengan versi HTTP klien.

Menafsirkan tanggapan Amazon SQS JSON API

Saat Anda mengirim permintaan ke Amazon SQS, ia mengembalikan respons JSON dengan hasilnya. Struktur respons bergantung pada tindakan API yang Anda gunakan.

Untuk memahami detail tanggapan ini, lihat:

- Tindakan API spesifik dalam tindakan API di Referensi API Layanan Antrian Sederhana Amazon
- Sebuah Protokol JSON Amazon SQS AWS FAQs

Struktur respons JSON yang sukses

Jika permintaan berhasil, elemen respons utama adalah`x-amzn-RequestId`, yang berisi Universal Unique Identifier (UUID) permintaan, serta bidang respons tambahan lainnya. Misalnya, `CreateQueue`respons berikut berisi `QueueUrl` bidang, yang, pada gilirannya, berisi URL antrian yang dibuat.

```
HTTP/1.1 200 OK
x-amzn-RequestId: <requestId>
Content-Length: <PayloadSizeBytes>
Date: <Date>
Content-Type: application/x-amz-json-1.0
```

```
{
    "QueueUrl":"https://sqs.us-east-1.amazonaws.com/111122223333/MyQueue"
}
```

Struktur respons kesalahan JSON

Jika permintaan tidak berhasil, Amazon SQS mengembalikan respons utama, termasuk header HTTP dan isi.

Di header HTTP, `x-amzn-RequestId` berisi UUID permintaan. `x-amzn-query-error`berisi dua informasi: jenis kesalahan, dan apakah kesalahan itu adalah kesalahan produsen atau konsumen.

Di badan respons, `"__type"` menunjukkan detail kesalahan lainnya, dan `Message` menunjukkan kondisi kesalahan dalam format yang dapat dibaca.

Berikut ini adalah contoh respon kesalahan dalam format 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."
}
```

Protokol JSON Amazon SQS AWS FAQs

Topik ini mencakup pertanyaan umum tentang penggunaan protokol AWS JSON dengan Amazon SQS.

Apa itu protokol AWS JSON, dan apa bedanya dengan permintaan dan tanggapan Amazon SQS API yang ada?

JSON adalah salah satu metode pengkabelan yang paling banyak digunakan dan diterima untuk komunikasi antara sistem heterogen. Amazon SQS menggunakan JSON sebagai media untuk berkomunikasi antara klien AWS SDK (misalnya, Java, Python, Golang,) JavaScript dan server Amazon SQS. Permintaan HTTP dari operasi Amazon SQS API menerima masukan dalam bentuk

JSON. Operasi Amazon SQS dijalankan dan respons eksekusi dibagikan kembali ke klien SDK dalam bentuk JSON. Dibandingkan dengan AWS query, JSON lebih efisien dalam mengangkut data antara klien dan server.

- Protokol Amazon SQS AWS JSON bertindak sebagai mediator antara klien dan server Amazon SQS.
- Server tidak memahami bahasa pemrograman di mana operasi Amazon SQS dibuat, tetapi memahami protokol AWS JSON.
- Protokol Amazon SQS AWS JSON menggunakan serialisasi (konversi objek ke format JSON) dan deserialisasi (konversi format JSON ke objek) antara klien Amazon SQS dan server.

Bagaimana cara memulai protokol AWS JSON untuk Amazon SQS?

Untuk memulai dengan versi AWS SDK terbaru untuk mencapai pengiriman pesan yang lebih cepat untuk Amazon SQS, tingkatkan SDK AWS Anda ke versi yang ditentukan atau versi berikutnya. Untuk mempelajari lebih lanjut tentang klien SDK, lihat kolom Panduan pada tabel di bawah ini.

Berikut ini adalah daftar versi SDK di seluruh varian bahasa untuk protokol AWS JSON untuk digunakan dengan Amazon SQS: APIs

Bahasa	Repositori klien SDK	Versi klien SDK yang diperlukan	Panduan
C++	aws/ aws-sdk-cpp	1.11.98	AWS SDK for C++
Golang 1.x	aws/ aws-sdk-go	v1.47.7	AWS SDK for Go
Golang 2.x	aws/ 2 aws-sdk-go-v	v1.28.0	AWS SDK for Go V2
Java 1.x	aws/ aws-sdk-java	1.12.585	AWS SDK for Java
Java 2.x	aws/ 2 aws-sdk-java-v	2.21.19	AWS SDK for Java
JavaScript v2.x	aws/ aws-sdk-js	JavaScript pada AWS	

Bahasa	Repositori klien SDK	Versi klien SDK yang diperlukan	Panduan
JavaScript v3.x	aws/ 3 aws-sdk-js-v	v3.447.0	JavaScript pada AWS
.NET	aws/ aws-sdk-net	3.7.681.0	AWS SDK for .NET
PHP	aws/ aws-sdk-php	3.285.2	AWS SDK for PHP
Python-Boto3	boto/boto3	1.28.82	AWS SDK untuk Python (Boto3)
Python-botocore	boto/botocore	1.31.82	AWS SDK untuk Python (Boto3)
awscli	AWS CLI	1.29.82	AWS Antarmuka Baris Perintah
Ruby	aws/ aws-sdk-ruby	1.67.0	AWS SDK for Ruby

Apa risiko mengaktifkan protokol JSON untuk beban kerja Amazon SQS saya?

Jika Anda menggunakan implementasi khusus AWS SDK atau kombinasi klien kustom dan AWS SDK untuk berinteraksi dengan Amazon SQS yang AWS menghasilkan respons berbasis Kueri (alias berbasis XML), mungkin tidak kompatibel dengan protokol JSON. AWS Jika Anda mengalami masalah, hubungi AWS Support.

Bagaimana jika saya sudah menggunakan versi AWS SDK terbaru, tetapi solusi open source saya tidak mendukung JSON?

Anda harus mengubah versi SDK Anda ke versi sebelumnya yang Anda gunakan. Lihat Bagaimana cara memulai protokol AWS JSON untuk Amazon SQS? untuk informasi lebih lanjut. AWS Versi SDK yang tercantum dalam Bagaimana cara memulai protokol AWS JSON untuk Amazon SQS? menggunakan protokol kawat JSON untuk Amazon SQS. APIs Jika Anda mengubah AWS SDK ke versi sebelumnya, Amazon APIs SQS Anda akan menggunakan AWS kueri.

Bahasa apa yang didukung untuk protokol AWS JSON yang digunakan di Amazon APIs SQS?

Amazon SQS mendukung semua varian bahasa yang umumnya AWS SDKs tersedia (GA). Saat ini, kami tidak mendukung Kotlin, Rust, atau Swift. Untuk mempelajari lebih lanjut tentang varian bahasa lain, lihat Alat untuk Dibangun AWS.

Wilayah apa yang didukung untuk protokol AWS JSON yang digunakan di Amazon SQS APIs

Amazon SQS mendukung protokol AWS JSON di semua AWS wilayah tempat Amazon SQS tersedia.

Peningkatan latensi apa yang dapat saya harapkan saat memutakhirkan ke versi AWS SDK yang ditentukan untuk Amazon SQS menggunakan protokol JSON? AWS

AWS Protokol JSON lebih efisien dalam serialisasi dan deserialisasi permintaan dan tanggapan jika dibandingkan dengan protokol kueri. AWS Berdasarkan pengujian AWS kinerja untuk muatan pesan 5 KB, protokol JSON untuk Amazon SQS end-to-end mengurangi latensi pemrosesan pesan hingga 23%, dan mengurangi penggunaan CPU dan memori sisi klien aplikasi.

Apakah protokol AWS kueri akan tidak digunakan lagi?

AWS protokol kueri akan terus didukung. Anda dapat terus menggunakan protokol AWS kueri selama versi AWS SDK Anda disetel versi sebelumnya selain yang tercantum di Bagaimana cara memulai dengan protokol AWS JSON untuk Amazon SQS.

Di mana saya dapat menemukan informasi lebih lanjut tentang protokol AWS JSON?

Anda dapat menemukan informasi lebih lanjut tentang protokol JSON di protokol AWS JSON 1.0 di dokumentasi Smithy. Untuk selengkapnya tentang permintaan Amazon SQS API menggunakan protokol AWS JSON, lihat. Membuat permintaan API kueri menggunakan protokol AWS JSON di Amazon SQS

Membuat permintaan API kueri menggunakan protokol AWS kueri di Amazon SQS

Topik ini menjelaskan cara membuat endpoint Amazon SQS, membuat permintaan GET dan POST, dan menafsirkan tanggapan.

Membangun titik akhir

Agar dapat bekerja dengan antrian Amazon SQS, Anda harus membuat titik akhir. Untuk informasi tentang titik akhir Amazon SQS, lihat halaman berikut di: Referensi Umum Amazon Web

- Titik akhir regional
- Titik akhir dan kuota Layanan Antrian Sederhana Amazon

Setiap titik akhir Amazon SQS bersifat independen. Misalnya, jika dua antrian diberi nama MyQueuedan satu memiliki titik akhir `sqs.us-east-2.amazonaws.com` sementara yang lain memiliki titik akhir`sqs.eu-west-2.amazonaws.com`, kedua antrian tidak berbagi data apa pun satu sama lain.

Berikut ini adalah contoh dari endpoint yang membuat permintaan untuk membuat antrian.

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

> **Note**
>
> Nama antrian dan antrian peka huruf URLs besar/kecil.
> Struktur *AUTHPARAMS* tergantung pada tanda tangan permintaan API. Untuk informasi selengkapnya, lihat Menandatangani Permintaan AWS API di Referensi Umum Amazon Web Services.

Membuat permintaan GET

Permintaan Amazon SQS GET disusun sebagai URL yang terdiri dari berikut ini:

- Endpoint — Sumber daya tempat permintaan bertindak (nama antrian dan URL), misalnya: `https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`

- Tindakan — Tindakan yang ingin Anda lakukan di titik akhir. Tanda tanya (?) memisahkan titik akhir dari tindakan, misalnya: `?Action=SendMessage&MessageBody=Your%20Message%20Text`
- Parameter — Parameter permintaan apa pun. Setiap parameter dipisahkan oleh ampersand (&), misalnya: `&Version=2012-11-05&`*`AUTHPARAMS`*

Berikut ini adalah contoh permintaan GET yang mengirim pesan ke antrian Amazon SQS.

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

> **Note**
>
> Nama antrian dan antrian peka huruf URLs besar/kecil.
> Karena permintaan GET adalah URLs, Anda harus mengkodekan URL semua nilai parameter. Karena spasi tidak diizinkan masuk URLs, setiap spasi dikodekan URL sebagai. `%20` Contoh lainnya tidak dikodekan URL untuk membuatnya lebih mudah dibaca.

Membuat permintaan POST

Permintaan Amazon SQS POST mengirimkan parameter kueri sebagai formulir di badan permintaan HTTP.

Berikut ini adalah contoh header HTTP dengan `Content-Type` set ke`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
```

Header diikuti oleh permintaan `form-urlencoded` GET yang mengirim pesan ke antrian Amazon SQS. Setiap parameter dipisahkan oleh ampersand ()&.

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

```
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> Hanya header `Content-Type` HTTP yang diperlukan. Sama seperti untuk permintaan GET.
> *AUTHPARAMS*
> Klien HTTP Anda mungkin menambahkan item lain ke permintaan HTTP, sesuai dengan versi HTTP klien.

Menafsirkan tanggapan Amazon SQS XMLAPI

Saat Anda mengirim permintaan ke Amazon SQS, permintaan tersebut akan mengembalikan respons XMLyang berisi hasil permintaan. Untuk memahami struktur dan detail tanggapan ini, lihat tindakan API tertentu di Referensi API Layanan Antrian Sederhana Amazon.

Struktur respons XHTML yang sukses

Jika permintaan berhasil, elemen respons utama dinamai setelah tindakan, dengan `Response` ditambahkan (misalnya,*`ActionName`*`Response`).

Elemen ini berisi elemen turunan berikut:

- ***`ActionName`*`Result`**— Berisi elemen khusus tindakan. Misalnya, `CreateQueueResult` elemen berisi `QueueUrl` elemen yang, pada gilirannya, berisi URL antrian yang dibuat.
- **`ResponseMetadata`**— Berisi `RequestId` yang, pada gilirannya, berisi Universal Unique Identifier (UUID) dari permintaan.

Berikut ini adalah contoh respon sukses dalam format 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>
```

Struktur respon kesalahan XML

Jika permintaan tidak berhasil, Amazon SQS selalu mengembalikan elemen respons utama. `ErrorResponse` Elemen ini berisi elemen `Error` dan elemen `RequestId`.

`Error`Elemen berisi elemen anak berikut:

- **`Type`**- Menentukan apakah kesalahan adalah produsen atau kesalahan konsumen.
- **`Code`**- Menentukan jenis kesalahan.
- **`Message`**- Menentukan kondisi kesalahan dalam format yang dapat dibaca.
- **`Detail`**- (Opsional) Menentukan rincian tambahan tentang kesalahan.

`RequestId`Elemen berisi UUID permintaan.

Berikut ini adalah contoh respon kesalahan dalam format 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>
```

Mengotentikasi permintaan untuk Amazon SQS

Otentikasi adalah proses mengidentifikasi dan memverifikasi pihak yang mengirim permintaan. Selama tahap pertama otentikasi, AWS memverifikasi identitas produsen dan apakah produsen terdaftar untuk digunakan AWS (untuk informasi lebih lanjut, lihatMenyiapkan Amazon SQS). Selanjutnya AWS , mematuhi prosedur berikut:

1. Produser (pengirim) memperoleh kredensi yang diperlukan.

2. Produsen mengirimkan permintaan dan kredensi kepada konsumen (penerima).
3. Konsumen menggunakan kredensi untuk memverifikasi apakah produsen mengirim permintaan.
4. Salah satu hal berikut terjadi:
   - Jika otentikasi berhasil, konsumen memproses permintaan.
   - Jika otentikasi gagal, konsumen menolak permintaan dan mengembalikan kesalahan.

Proses otentikasi dasar dengan HMAC-SHA

Saat mengakses Amazon SQS menggunakan Query API, Anda harus memberikan item berikut untuk mengotentikasi permintaan Anda:

- ID AWS Kunci Akses yang mengidentifikasi Anda Akun AWS, yang digunakan AWS untuk mencari Kunci Akses Rahasia Anda.
- Tanda tangan HMAC-SHA permintaan, dihitung menggunakan Kunci Akses Rahasia Anda (rahasia bersama yang hanya diketahui oleh Anda dan AWS—untuk informasi selengkapnya, lihat RFC 2104). AWS SDK menangani proses penandatanganan; namun, jika Anda mengirimkan permintaan kueri melalui HTTP atau HTTPS, Anda harus menyertakan tanda tangan di setiap permintaan kueri.

  1. Dapatkan Kunci Penandatanganan Versi 4 Tanda Tangan. Untuk informasi selengkapnya, lihat Meng urunkan Kunci Penandatanganan dengan Java.

     > **Note**
     >
     > Amazon SQS mendukung Signature Version 4, yang memberikan peningkatan SHA256-based keamanan dan kinerja dibandingkan versi sebelumnya. Saat Anda membuat aplikasi baru yang menggunakan Amazon SQS, gunakan Signature Version 4.

  2. Base64-encode tanda tangan permintaan. Contoh kode Java berikut melakukan ini:

```
package amazon.webservices.common;

// Define common routines for encoding data in AWS requests.
public class Encoding {

    /* Perform base64 encoding of input bytes.
```

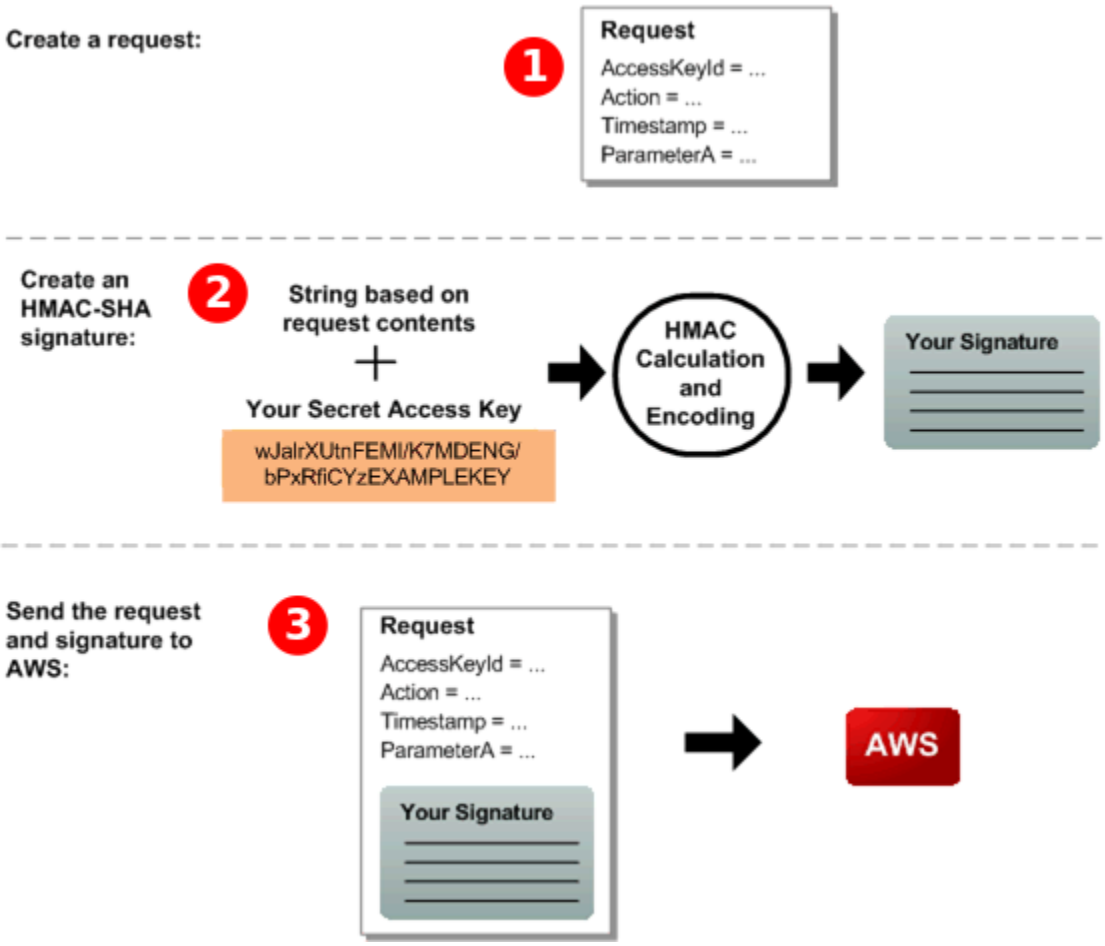
```
     * rawData is the array of bytes to be encoded.
     * return is the base64-encoded string representation of rawData.
     */
    public static String EncodeBase64(byte[] rawData) {
        return Base64.encodeBytes(rawData);
    }
}
```

- S tempel waktu (atau kedaluwarsa) permintaan. Stempel waktu yang Anda gunakan dalam permintaan harus berupa `dateTime` objek, dengan tanggal lengkap, termasuk jam, menit, dan detik. Sebagai contoh: `2007-01-31T23:59:59Z` Meskipun ini tidak diperlukan, sebaiknya berikan objek menggunakan zona waktu Waktu Universal Terkoordinasi (Greenwich Mean Time).

> **Note**
>
> Pastikan waktu server Anda diatur dengan benar. Jika Anda menentukan stempel waktu (bukan kedaluwarsa), permintaan secara otomatis kedaluwarsa 15 menit setelah waktu yang ditentukan (AWS tidak memproses permintaan dengan stempel waktu lebih dari 15 menit lebih awal dari waktu saat ini di server). AWS
> Jika Anda menggunakan.NET, Anda tidak boleh mengirim cap waktu yang terlalu spesifik (karena interpretasi yang berbeda tentang bagaimana presisi waktu ekstra harus dihilangkan). Dalam hal ini, Anda harus secara manual membangun `dateTime` objek dengan presisi tidak lebih dari satu milidetik.

Bagian 1: Permintaan dari pengguna

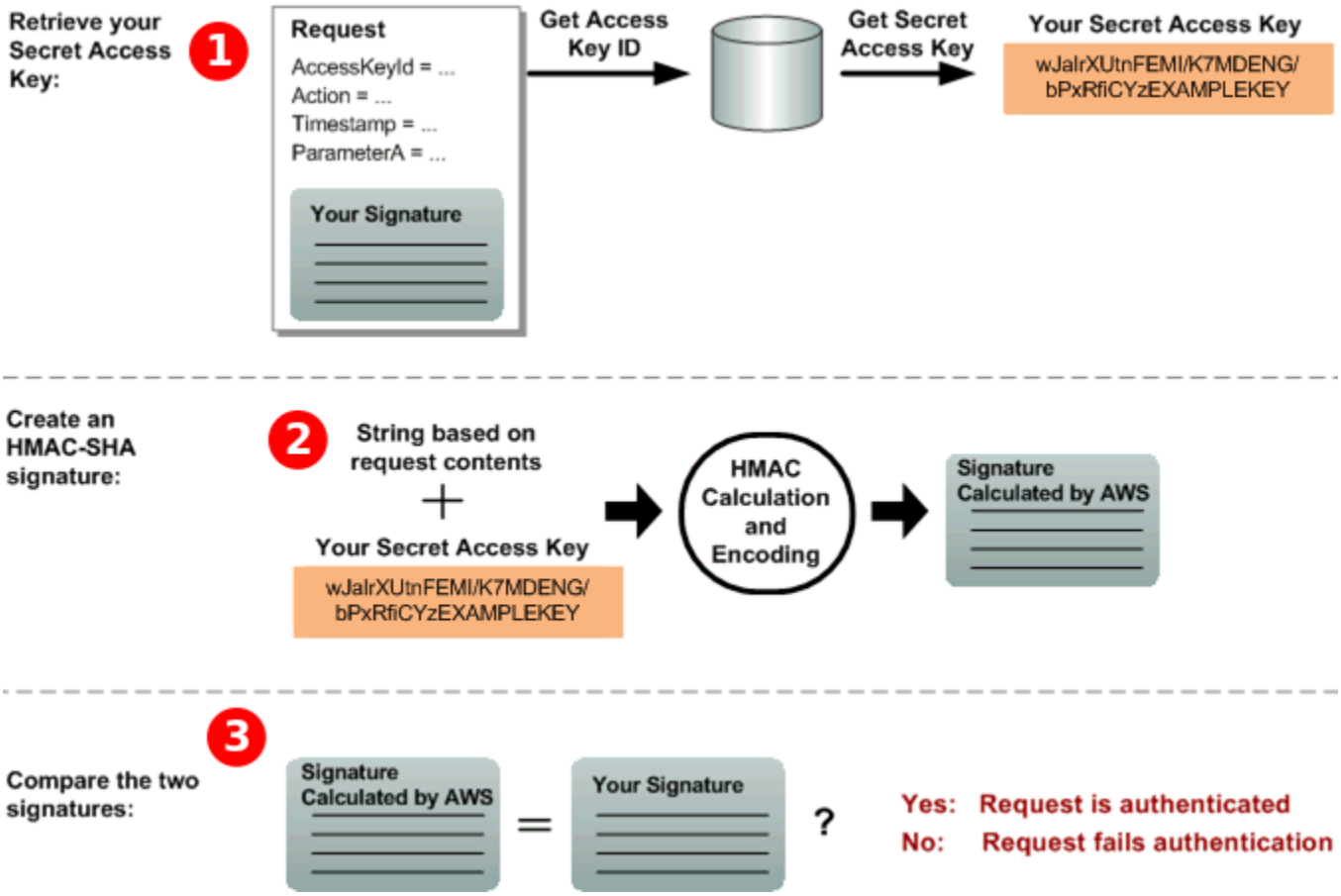
Berikut ini adalah proses yang harus Anda ikuti untuk mengotentikasi AWS permintaan menggunakan tanda tangan HMAC-SHA permintaan.



1. Buat permintaan ke AWS.
2. Hitung tanda tangan kode otentikasi pesan hash kunci (HMAC-SHA) menggunakan Kunci Akses Rahasia Anda.
3. Sertakan tanda tangan dan ID Kunci Akses Anda dalam permintaan, lalu kirim permintaan ke AWS.

Bagian 2: Tanggapan dari AWS

AWS memulai proses berikut sebagai tanggapan.



1. AWS menggunakan ID Kunci Akses untuk mencari Kunci Akses Rahasia Anda.
2. AWS menghasilkan tanda tangan dari data permintaan dan Kunci Akses Rahasia, menggunakan algoritma yang sama yang Anda gunakan untuk menghitung tanda tangan yang Anda kirim dalam permintaan.
3. Salah satu hal berikut terjadi:
   - Jika tanda tangan yang AWS dihasilkan cocok dengan yang Anda kirim dalam permintaan, AWS anggap permintaan tersebut otentik.
   - Jika perbandingan gagal, permintaan dibuang, dan AWS mengembalikan kesalahan.

Tindakan batch Amazon SQS

Amazon SQS menyediakan tindakan batch untuk membantu Anda mengurangi biaya dan memanipulasi hingga 10 pesan dengan satu tindakan. Tindakan batch ini meliputi:

- `SendMessageBatch`

- `DeleteMessageBatch`
- `ChangeMessageVisibilityBatch`

Dengan menggunakan tindakan batch, Anda dapat melakukan beberapa operasi dalam satu panggilan API, yang membantu mengoptimalkan kinerja dan mengurangi biaya. Anda dapat memanfaatkan fungsionalitas batch menggunakan API kueri atau AWS SDK apa pun yang mendukung tindakan batch Amazon SQS.

Detail Penting

- Batas Ukuran Pesan: Ukuran total semua pesan yang dikirim dalam satu `SendMessageBatch` panggilan tidak boleh melebihi 1.048.576 byte (1 MiB)
- Izin: Anda tidak dapat menetapkan izin secara eksplisit untuk`SendMessageBatch`,, atau. `DeleteMessageBatch` `ChangeMessageVisibilityBatch` Sebagai gantinya, menyetel izin untuk `SendMessage`,, `DeleteMessage`atau `ChangeMessageVisibility` menetapkan izin untuk versi batch tindakan yang sesuai.
- Dukungan Konsol: Konsol Amazon SQS tidak mendukung tindakan batch. Anda harus menggunakan API kueri atau AWS SDK untuk melakukan operasi batch.

Tindakan pesan batching

Untuk lebih mengoptimalkan biaya dan efisiensi, pertimbangkan praktik terbaik berikut untuk tindakan pengelompokan pesan:

- Tindakan API Batch: Gunakan tindakan tindakan API batch Amazon SQS untuk mengirim, menerima, dan menghapus pesan, serta mengubah batas waktu visibilitas pesan untuk beberapa pesan dengan satu tindakan. Ini mengurangi jumlah panggilan API dan biaya terkait.
- Buffering Sisi Klien dan Polling Panjang: Gabungkan buffering sisi klien dengan batch permintaan dengan menggunakan polling panjang bersama dengan klien asinkron buffer yang disertakan dengan. AWS SDK untuk Java Pendekatan ini membantu meminimalkan jumlah permintaan dan mengoptimalkan penanganan pesan dalam jumlah besar.

> **Note**
>
> Klien Asinkron Buffered Amazon SQS saat ini tidak mendukung antrian FIFO.

Mengaktifkan buffering sisi klien dan batching permintaan dengan Amazon SQS

AWS SDK untuk JavaTermasuk `AmazonSQSBufferedAsyncClient` yang mengakses Amazon SQS. Klien ini memungkinkan batching permintaan sederhana menggunakan buffering sisi klien. Panggilan yang dilakukan dari klien pertama kali di-buffer dan kemudian dikirim sebagai permintaan batch ke Amazon SQS.

Buffering sisi klien memungkinkan hingga 10 permintaan untuk di-buffer dan dikirim sebagai permintaan batch, mengurangi biaya penggunaan Amazon SQS dan mengurangi jumlah permintaan yang dikirim. `AmazonSQSBufferedAsyncClient`buffer panggilan sinkron dan asinkron. Permintaan batch dan dukungan untuk polling panjang juga dapat membantu meningkatkan throughput. Untuk informasi selengkapnya, lihat Meningkatkan throughput menggunakan penskalaan horizontal dan batching aksi dengan Amazon SQS.

Karena `AmazonSQSBufferedAsyncClient` mengimplementasikan antarmuka yang sama seperti`AmazonSQSAsyncClient`, migrasi dari `AmazonSQSAsyncClient` ke `AmazonSQSBufferedAsyncClient` biasanya hanya memerlukan sedikit perubahan pada kode Anda yang ada.

> **Note**
>
> Klien Asinkron Buffered Amazon SQS saat ini tidak mendukung antrian FIFO.

Menggunakan Amazon SQSBuffered AsyncClient

Sebelum memulai, selesaikan langkah-langkah di Menyiapkan Amazon SQS.

AWS SDK for Java 1.x

Untuk AWS SDK for Java 1.x, Anda dapat membuat yang `AmazonSQSBufferedAsyncClient` baru berdasarkan contoh berikut:

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
```

```
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

Setelah Anda membuat yang baruAmazonSQSBufferedAsyncClient, Anda dapat menggunakannya untuk mengirim beberapa permintaan ke Amazon SQS (seperti yang Anda bisa denganAmazonSQSAsyncClient), misalnya:

```
final CreateQueueRequest createRequest = new
 CreateQueueRequest().withQueueName("MyQueue");

final CreateQueueResult res = bufferedSqs.createQueue(createRequest);

final SendMessageRequest request = new SendMessageRequest();
final String body = "Your message text" + System.currentTimeMillis();
request.setMessageBody( body );
request.setQueueUrl(res.getQueueUrl());

final Future<SendMessageResult> sendResult = bufferedSqs.sendMessageAsync(request);

final ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
final ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

Mengkonfigurasi Amazon SQSBuffered AsyncClient

AmazonSQSBufferedAsyncClienttelah dikonfigurasi sebelumnya dengan pengaturan yang berfungsi untuk sebagian besar kasus penggunaan. Anda dapat mengkonfigurasi lebih lanjutAmazonSQSBufferedAsyncClient, misalnya:

1. Buat instance QueueBufferConfig kelas dengan parameter konfigurasi yang diperlukan.
2. Berikan instance ke AmazonSQSBufferedAsyncClient konstruktor.

```
// 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 parameter konfigurasi

Parameter	Nilai default	Deskripsi
`longPoll`	`true`	Ketika `longPoll` diatur ke`true`, `AmazonSQSBufferedAsyncClient` mencoba untuk menggunakan polling panjang ketika mengkonsumsi pesan.
`longPollWaitTimeoutSeconds`	20 s	Jumlah waktu maksimum (dalam detik) yang diblokir `ReceiveMessage` panggilan di server, menunggu pesan muncul dalam antrian sebelum kembali dengan hasil penerimaan kosong. **Note** Ketika polling panjang dinonaktifkan, pengaturan ini tidak berpengaruh.
`maxBatchOpenMs`	200 ms	Jumlah waktu maksimum (dalam milidetik) panggilan keluar menunggu panggilan lain yang dengannya ia mengumpulkan pesan dari jenis yang sama.

Parameter	Nilai default	Deskripsi
		Semakin tinggi pengaturan, semakin sedikit batch yang diperlukan untuk melakukan jumlah pekerjaan yang sama (namun, panggilan pertama dalam batch harus menghabiskan waktu lebih lama menunggu). Saat Anda menyetel parameter ini`0`, permintaa n yang dikirimkan tidak menunggu permintaan lain, yang secara efektif menonakti fkan batching.
`maxBatchSize`	10 permintaan per batch	Jumlah maksimum pesan yang dikumpulkan bersama dalam satu permintaan. Semakin tinggi pengaturan, semakin sedikit batch yang diperlukan untuk melakukan jumlah permintaan yang sama. **Note** 10 permintaan per batch adalah nilai maksimum yang diizinkan untuk Amazon SQS.

Parameter	Nilai default	Deskripsi
`maxBatchSizeBytes`	1 MiB	Ukuran maksimum kumpulan pesan, dalam byte, yang coba dikirim klien ke Amazon SQS. **Note** 1 MiB adalah nilai maksimum yang diizinkan untuk Amazon SQS.

Parameter	Nilai default	Deskripsi
`maxDoneReceiveBatches`	10 batch	Jumlah maksimum batch penerima yang `AmazonSQSBufferedAsyncClient` mengambil dan menyimpan sisi klien. Semakin tinggi pengaturan, semakin banyak permintaan penerimaan yang dapat dipenuhi tanpa harus melakukan panggilan ke Amazon SQS (namun, semakin banyak pesan yang diambil sebelumnya, semakin lama mereka tetap berada di buffer, menyebabkan batas waktu visibilitas mereka sendiri kedaluwarsa). **Note** `0`menunjukkan bahwa semua pesan pra-pengambilan dinonaktifkan dan pesan hanya dikonsumsi sesuai permintaan.

Parameter	Nilai default	Deskripsi
`maxInflightOutboundBatches`	5 batch	Jumlah maksimum batch keluar aktif yang dapat diproses secara bersamaan. Semakin tinggi pengaturan, batch keluar yang lebih cepat dapat dikirim (tunduk pada kuota seperti CPU atau bandwidth) dan semakin banyak thread yang dikonsumsi oleh. `AmazonSQSBufferedAsyncClient`

Parameter	Nilai default	Deskripsi
`maxInflightReceiveBatches`	10 batch	Jumlah maksimum batch penerima aktif yang dapat diproses pada saat yang bersamaan. Semakin tinggi pengaturan, semakin banyak pesan yang dapat diterima (tunduk pada kuota seperti CPU atau bandwidth), dan semakin banyak utas yang dikonsumsi `AmazonSQSBufferedAsyncClient` . **Note** `0`menunjukkan bahwa semua pesan pra-pengambilan dinonaktifkan dan pesan hanya dikonsumsi sesuai permintaan.

Parameter	Nilai default	Deskripsi
`visibilityTimeoutSeconds`	-1	Ketika parameter ini disetel ke nilai positif, bukan nol, batas waktu visibilitas yang ditetapkan di sini akan mengganti batas waktu visibilitas yang ditetapkan pada antrian dari mana pesan dikonsumsi. **Note** `-1`menunjukkan bahwa pengaturan default dipilih untuk antrian. Anda tidak dapat mengatur batas waktu visibilitas ke. `0`

AWS SDK for Java 2.x

Untuk AWS SDK for Java 2.x, Anda dapat membuat yang `SqsAsyncBatchManager` baru berdasarkan contoh berikut:

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

Setelah Anda membuat yang baru`SqsAsyncBatchManager`, Anda dapat menggunakannya untuk mengirim beberapa permintaan ke Amazon SQS (seperti yang Anda bisa dengan`SqsAsyncClient`), misalnya:

```
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);
```

Mengkonfigurasi SqsAsyncBatchManager

`SqsAsyncBatchManager`telah dikonfigurasi sebelumnya dengan pengaturan yang berfungsi untuk sebagian besar kasus penggunaan. Anda dapat mengkonfigurasi lebih lanjut`SqsAsyncBatchManager`, misalnya:

Membuat konfigurasi khusus melaluiSqsAsyncBatchManager.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();
```

Parameter **`BatchOverrideConfiguration`**

Parameter	Nilai default	Deskripsi
`maxBatchSize`	10 permintaan per batch	Jumlah maksimum pesan yang dikumpulkan bersama dalam satu permintaan. Semakin tinggi pengaturan, semakin sedikit batch yang diperlukan untuk melakukan jumlah permintaan yang sama. **Note** Nilai maksimum yang diizinkan untuk Amazon SQS adalah 10 permintaan per batch.
`sendRequestFrequency`	200 ms	Jumlah waktu maksimum (dalam milidetik) panggilan keluar menunggu panggilan lain yang dengannya ia

Parameter	Nilai default	Deskripsi
		mengumpulkan pesan dari jenis yang sama. Semakin tinggi pengaturan, semakin sedikit batch yang diperlukan untuk melakukan jumlah pekerjaan yang sama (namun, panggilan pertama dalam batch harus menghabiskan waktu lebih lama menunggu). Saat Anda menyetel parameter ini`0`, permintaa n yang dikirimkan tidak menunggu permintaan lain, yang secara efektif menonakti fkan batching.

Parameter	Nilai default	Deskripsi
`receiveMessageVisibilityTimeout`	-1	Ketika parameter ini disetel ke nilai positif, bukan nol, batas waktu visibilitas yang ditetapkan di sini akan mengganti batas waktu visibilitas yang ditetapkan pada antrian dari mana pesan dikonsumsi. **Note** 1menunjukkan bahwa pengaturan default dipilih untuk antrian. Anda tidak dapat mengatur batas waktu visibilitas ke. `0`
`receiveMessageMinWaitDuration`	50 ms	Jumlah waktu minimal (dalam milidetik) `receiveMessage` panggilan menunggu pesan yang tersedia diambil. Semakin tinggi pengaturan, semakin sedikit batch yang diperlukan untuk melakukan jumlah permintaan yang sama.

Meningkatkan throughput menggunakan penskalaan horizontal dan batching aksi dengan Amazon SQS

Amazon SQS mendukung perpesanan throughput tinggi. Untuk detail tentang batas throughput, lihat. Kuota pesan Amazon SQS

Untuk memaksimalkan throughput:

- Skalakan produsen dan konsumen secara horizontal dengan menambahkan lebih banyak contoh masing-masing.
- Gunakan batch tindakan untuk mengirim atau menerima beberapa pesan dalam satu permintaan, sehingga mengurangi overhead panggilan API.

Penskalaan horizontal

Karena Anda mengakses Amazon SQS melalui protokol permintaan-respons HTTP, latensi permintaan (interval antara memulai permintaan dan menerima respons) membatasi throughput yang dapat Anda capai dari satu utas menggunakan satu koneksi. Misalnya, jika latensi dari klien berbasis Amazon EC2 ke Amazon SQS di wilayah yang sama rata-rata 20 ms, throughput maksimum dari satu utas melalui satu koneksi rata-rata 50 TPS.

Penskalaan horizontal melibatkan peningkatan jumlah produsen pesan (yang membuat `SendMessage` permintaan) dan konsumen (yang membuat `ReceiveMessage` dan `DeleteMessage` meminta) untuk meningkatkan throughput antrian Anda secara keseluruhan. Anda dapat menskalakan secara horizontal dengan tiga cara:

- Meningkatkan jumlah thread per klien
- Tambahkan lebih banyak klien
- Tingkatkan jumlah thread per klien dan tambahkan lebih banyak klien

Ketika Anda menambahkan lebih banyak klien, Anda mencapai keuntungan linier pada dasarnya dalam throughput antrian. Misalnya, jika Anda menggandakan jumlah klien, Anda juga menggandakan throughput.

Aksi batching

Batching melakukan lebih banyak pekerjaan selama setiap perjalanan pulang pergi ke layanan (misalnya, ketika Anda mengirim beberapa pesan dengan satu `SendMessageBatch` permintaan). Tindakan batch Amazon SQS adalah`SendMessageBatch`,`DeleteMessageBatch`, dan. `ChangeMessageVisibilityBatch` Untuk memanfaatkan batching tanpa mengubah produsen atau konsumen, Anda dapat menggunakan Amazon SQS Buffered Asynchronous Client.

> **Note**
>
> Karena `ReceiveMessage` dapat memproses 10 pesan sekaligus, tidak ada `ReceiveMessageBatch` tindakan.

Batching mendistribusikan latensi tindakan batch melalui beberapa pesan dalam permintaan batch, daripada menerima seluruh latensi untuk satu pesan (misalnya, permintaan). `SendMessage` Karena setiap perjalanan pulang pergi membawa lebih banyak pekerjaan, permintaan batch membuat penggunaan thread dan koneksi lebih efisien, meningkatkan throughput.

Anda dapat menggabungkan batching dengan penskalaan horizontal untuk menyediakan throughput dengan thread, koneksi, dan permintaan yang lebih sedikit daripada permintaan pesan individual. Anda dapat menggunakan tindakan Amazon SQS batch untuk mengirim, menerima, atau menghapus hingga 10 pesan sekaligus. Karena Amazon SQS mengenakan biaya berdasarkan permintaan, batching dapat secara substansional mengurangi biaya Anda.

Batching dapat menimbulkan beberapa kerumitan untuk aplikasi Anda (misalnya, aplikasi Anda harus mengumpulkan pesan sebelum mengirimnya, atau terkadang harus menunggu lebih lama untuk respons). Namun, batching masih bisa efektif dalam kasus-kasus berikut:

- Aplikasi Anda menghasilkan banyak pesan dalam waktu singkat, sehingga penundaan tidak pernah terlalu lama.
- Konsumen pesan mengambil pesan dari antrian atas kebijakannya sendiri, tidak seperti produsen pesan biasa yang perlu mengirim pesan sebagai respons terhadap peristiwa yang tidak mereka kendalikan.

> **Important**
>
> Permintaan batch mungkin berhasil meskipun pesan individual dalam batch gagal. Setelah permintaan batch, selalu periksa kegagalan pesan individual dan coba lagi tindakan jika perlu.

Contoh Java yang berfungsi untuk permintaan operasi tunggal dan batch

Prasyarat

Tambahkan`aws-java-sdk-sqs.jar`,`aws-java-sdk-ec2.jar`, dan `commons-logging.jar` paket ke jalur kelas build Java Anda. Contoh berikut menampilkan dependensi ini dalam file `pom.xml` proyek 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.jawa

Contoh kode Java berikut mengimplementasikan pola produsen-konsumen sederhana. Thread utama memunculkan sejumlah thread produsen dan konsumen yang memproses pesan 1 KB untuk waktu tertentu. Contoh ini mencakup produsen dan konsumen yang membuat permintaan operasi tunggal dan mereka yang membuat permintaan 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.
            }
        }
    }
}
```

Memantau metrik volume dari contoh yang dijalankan

Amazon SQS secara otomatis menghasilkan metrik volume untuk pesan yang dikirim, diterima, dan dihapus. Anda dapat mengakses metrik tersebut dan lainnya melalui tab Monitoring untuk antrian Anda atau di CloudWatch konsol.

> **Note**
>
> Metrik dapat memakan waktu hingga 15 menit setelah antrian mulai tersedia.

Menggunakan Amazon SQS dengan AWS SDK

AWS kit pengembangan perangkat lunak (SDK) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan developer untuk membangun aplikasi dalam bahasa pilihan mereka.

Dokumentasi SDK	Contoh kode
AWS SDK untuk C++	AWS SDK untuk C++ contoh kode
AWS CLI	AWS CLI contoh kode
AWS SDK untuk Go	AWS SDK untuk Go contoh kode
AWS SDK untuk Java	AWS SDK untuk Java contoh kode
AWS SDK untuk JavaScript	AWS SDK untuk JavaScript contoh kode

Dokumentasi SDK	Contoh kode
AWS SDK untuk Kotlin	AWS SDK untuk Kotlin contoh kode
AWS SDK untuk .NET	AWS SDK untuk .NET contoh kode
AWS SDK untuk PHP	AWS SDK untuk PHP contoh kode
Alat AWS untuk PowerShell	Alat AWS untuk PowerShell contoh kode
AWS SDK untuk Python v4	AWS Contoh kode SDK untuk Python v4
AWS SDK untuk Python (Boto3)	AWS SDK untuk Python (Boto3) contoh kode
AWS SDK untuk Ruby	AWS SDK untuk Ruby contoh kode
AWS SDK for Rust	AWS SDK for Rust contoh kode
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP contoh kode
AWS SDK for Swift	AWS SDK for Swift contoh kode

Ketersediaan contoh

Tidak dapat menemukan apa yang Anda butuhkan? Minta contoh kode menggunakan tautan Berikan umpan balik di bagian bawah halaman ini.

Menggunakan JMS dengan Amazon SQS

Amazon SQS Java Messaging Library adalah antarmuka Java Message Service (JMS) untuk Amazon SQS yang memungkinkan Anda memanfaatkan Amazon SQS dalam aplikasi yang sudah menggunakan JMS. Antarmuka memungkinkan Anda menggunakan Amazon SQS sebagai penyedia JMS dengan perubahan kode minimal. Bersama dengan AWS SDK untuk Java, Amazon SQS Java Messaging Library memungkinkan Anda membuat koneksi dan sesi JMS, serta produsen dan konsumen yang mengirim dan menerima pesan ke dan dari antrian Amazon SQS.

Pustaka mendukung pengiriman dan penerimaan pesan ke antrian (point-to-pointmodel JMS) sesuai dengan spesifikasi JMS 1.1. Pustaka mendukung pengiriman pesan teks, byte, atau objek secara sinkron ke antrian Amazon SQS. Pustaka juga mendukung penerimaan objek secara sinkron atau asinkron.

Untuk informasi tentang fitur Amazon SQS Java Messaging Library yang mendukung spesifikasi JMS 1.1, lihat Amazon SQS mendukung implementasi JMS 1.1 dan Amazon SQS. FAQs

Prasyarat untuk bekerja dengan JMS dan Amazon SQS

Sebelum Anda mulai, Anda harus memiliki prasyarat berikut:

- SDK for Java

  Ada dua cara untuk menyertakan SDK for Java dalam proyek Anda:

  - Unduh dan instal SDK for Java.
  - Gunakan Maven untuk mendapatkan Amazon SQS Java Messaging Library.

  > **Note**
  >
  > SDK for Java disertakan sebagai dependensi.
  > SDK for Java dan Amazon SQS Extended Client Library untuk Java memerlukan J2SE Development Kit 8.0 atau yang lebih baru.

  Untuk informasi tentang mengunduh SDK for Java, lihat SDK for Java.

- Perpustakaan Pesan Amazon SQS Java

Jika Anda tidak menggunakan Maven, Anda harus menambahkan `amazon-sqs-java-messaging-lib.jar` paket ke jalur kelas Java. Untuk informasi tentang mengunduh pustaka, lihat Amazon SQS Java Messaging Library.

> **Note**
>
> Amazon SQS Java Messaging Library mencakup dukungan untuk Maven dan Spring Framework.
> Untuk contoh kode yang menggunakan Maven, Spring Framework, dan Amazon SQS Java Messaging Library, lihat. Contoh Java yang berfungsi untuk menggunakan JMS dengan antrian standar Amazon SQS
>
> ```
> <dependency>
>   <groupId>com.amazonaws</groupId>
>   <artifactId>amazon-sqs-java-messaging-lib</artifactId>
>   <version>1.0.4</version>
>   <type>jar</type>
> </dependency>
> ```

- Antrian Amazon SQS

  Buat antrian menggunakan Konsol Manajemen AWS for Amazon SQS, API, atau klien Amazon SQS `CreateQueue` yang dibungkus yang disertakan dalam Amazon SQS Java Messaging Library.

  - Untuk informasi tentang membuat antrian dengan Amazon SQS menggunakan API atau `CreateQueue` API, lihat Membuat Antrian. Konsol Manajemen AWS
  - Untuk informasi tentang menggunakan Amazon SQS Java Messaging Library, lihat. Menggunakan Amazon SQS Java Messaging Library

Menggunakan Amazon SQS Java Messaging Library

Untuk mulai menggunakan Java Message Service (JMS) dengan Amazon SQS, gunakan contoh kode di bagian ini. Bagian berikut menunjukkan cara membuat koneksi JMS dan sesi, dan cara mengirim dan menerima pesan.

Objek klien Amazon SQS yang dibungkus yang disertakan dalam Perpustakaan Pesan Java Amazon SQS memeriksa apakah antrian Amazon SQS ada. Jika antrian tidak ada, klien membuatnya.

Membuat koneksi JMS

Sebelum Anda mulai, lihat prasyarat di. Prasyarat untuk bekerja dengan JMS dan Amazon SQS

1. Buat pabrik koneksi dan panggil `createConnection` metode melawan pabrik.

```
// Create a new connection factory with all defaults (credentials and region) set
 automatically
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.defaultClient()
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

   `SQSConnection`Kelas meluas. `javax.jms.Connection` Bersama dengan metode koneksi standar JMS, `SQSConnection` menawarkan metode tambahan, seperti `getAmazonSQSClient` dan`getWrappedAmazonSQSClient`. Kedua metode memungkinkan Anda melakukan operasi administratif yang tidak termasuk dalam spesifikasi JMS, seperti membuat antrian baru. Namun, `getWrappedAmazonSQSClient` metode ini juga menyediakan versi terbungkus dari klien Amazon SQS yang digunakan oleh koneksi saat ini. Pembungkus mengubah setiap pengecualian dari klien menjadi`JMSException`, memungkinkannya untuk lebih mudah digunakan oleh kode yang ada yang mengharapkan `JMSException` kejadian.

2. Anda dapat menggunakan objek klien yang dikembalikan dari `getAmazonSQSClient` dan `getWrappedAmazonSQSClient` untuk melakukan operasi administratif yang tidak termasuk dalam spesifikasi JMS (misalnya, Anda dapat membuat antrean Amazon SQS).

   Jika Anda memiliki kode yang mengharapkan pengecualian JMS, maka Anda harus menggunakan: `getWrappedAmazonSQSClient`

   - Jika Anda menggunakan`getWrappedAmazonSQSClient`, objek klien yang dikembalikan mengubah semua pengecualian menjadi pengecualian JMS.
   - Jika Anda menggunakan`getAmazonSQSClient`, pengecualian adalah semua pengecualian Amazon SQS.

Membuat antrian Amazon SQS

Objek klien yang dibungkus memeriksa apakah antrian Amazon SQS ada.

Jika antrian tidak ada, klien membuatnya. Jika antrian memang ada, fungsi tidak mengembalikan apa pun. Untuk informasi selengkapnya, lihat bagian “Buat antrian jika diperlukan” pada TextMessageSender.jawa contoh.

Untuk membuat antrian standar

```
// 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");
}
```

Untuk membuat antrian 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**
>
> Nama antrian FIFO harus diakhiri dengan akhiran. `.fifo`
> Untuk informasi selengkapnya tentang `ContentBasedDeduplication` atribut, lihatTepat sekali diproses di Amazon SQS.

Mengirim pesan secara sinkron

1. Saat koneksi dan antrean Amazon SQS yang mendasarinya sudah siap, buat sesi JMS yang tidak ditransaksikan dengan mode. `AUTO_ACKNOWLEDGE`

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. Untuk mengirim pesan teks ke antrian, buat identitas antrian JMS dan produser pesan.

```
// 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. Buat pesan teks dan kirimkan ke antrian.

   - Untuk mengirim pesan ke antrian standar, Anda tidak perlu mengatur parameter tambahan apa pun.

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

   - Untuk mengirim pesan ke antrian FIFO, Anda harus mengatur ID grup pesan. Anda juga dapat mengatur ID deduplikasi pesan. Untuk informasi selengkapnya, lihat Istilah kunci antrian Amazon SQS FIFO.

```
// 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"));
```

Menerima pesan secara sinkron

1. Untuk menerima pesan, buat konsumen untuk antrian yang sama dan panggil metode. `start`

   Anda dapat memanggil `start` metode pada koneksi kapan saja. Namun, konsumen tidak mulai menerima pesan sampai Anda memanggilnya.

   ```
   // Create a consumer for the 'MyQueue'
   MessageConsumer consumer = session.createConsumer(queue);
   // Start receiving incoming messages
   connection.start();
   ```

2. Panggil `receive` metode pada konsumen dengan batas waktu diatur ke 1 detik, dan kemudian cetak konten pesan yang diterima.

   - Setelah menerima pesan dari antrian standar, Anda dapat mengakses konten pesan.

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

   - Setelah menerima pesan dari antrian FIFO, Anda dapat mengakses konten pesan dan atribut pesan khusus FIFO lainnya, seperti ID grup pesan, ID deduplikasi pesan, dan nomor urut. Untuk informasi selengkapnya, lihat Istilah kunci antrian Amazon SQS FIFO.

     ```
     // 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. Tutup koneksi dan sesi.

```
// Close the connection (and the session).
connection.close();
```

Output akan terlihat serupa dengan yang berikut ini:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

> **Note**
>
> Anda dapat menggunakan Spring Framework untuk menginisialisasi objek-objek ini. Untuk informasi tambahan, lihat `SpringExampleConfiguration.xmlSpringExample.java`,, dan kelas pembantu lainnya di `ExampleConfiguration.java` dan `ExampleCommon.java` di Contoh Java yang berfungsi untuk menggunakan JMS dengan antrian standar Amazon SQS bagian.

Untuk contoh lengkap mengirim dan menerima objek, lihat TextMessageSender.jawa danSyncMessageReceiver.jawa.

Menerima pesan secara asinkron

Dalam contoh diMenggunakan Amazon SQS Java Messaging Library, pesan dikirim ke `MyQueue` dan diterima secara serempak.

Contoh berikut menunjukkan cara menerima pesan secara asinkron melalui pendengar.

1. Menerapkan `MessageListener` antarmuka.

```
class MyListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to
 screen.
            System.out.println("Received: " + ((TextMessage) message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

onMessageMetode `MessageListener` antarmuka dipanggil ketika Anda menerima pesan. Dalam implementasi listener ini, teks yang disimpan dalam pesan dicetak.

2. Alih-alih secara eksplisit memanggil `receive` metode pada konsumen, atur pendengar pesan konsumen ke instance implementasi. `MyListener` Utas utama menunggu satu detik.

```
// 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);
```

Langkah-langkah lainnya identik dengan yang ada di Menggunakan Amazon SQS Java Messaging Library contoh. Untuk contoh lengkap konsumen asinkron, lihat di. `AsyncMessageReceiver.java` Contoh Java yang berfungsi untuk menggunakan JMS dengan antrian standar Amazon SQS

Output untuk contoh ini terlihat mirip dengan yang berikut:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
```

```
Received: Hello World!
```

Menggunakan mode pengakuan klien

Contoh dalam `AUTO_ACKNOWLEDGE` mode Menggunakan Amazon SQS Java Messaging Library penggunaan di mana setiap pesan yang diterima diakui secara otomatis (dan karenanya dihapus dari antrean Amazon SQS yang mendasarinya).

1. Untuk secara eksplisit mengakui pesan setelah diproses, Anda harus membuat sesi dengan mode. `CLIENT_ACKNOWLEDGE`

   ```
   // Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
   Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
   ```

2. Ketika pesan diterima, tampilkan dan kemudian secara eksplisit mengakuinya.

   ```
   // 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**
   >
   > Dalam mode ini, ketika pesan diakui, semua pesan yang diterima sebelum pesan ini secara implisit diakui juga. Misalnya, jika 10 pesan diterima, dan hanya pesan ke-10 yang diakui (dalam urutan pesan diterima), maka semua dari sembilan pesan sebelumnya juga diakui.

Langkah-langkah lainnya identik dengan yang ada di Menggunakan Amazon SQS Java Messaging Library contoh. Untuk contoh lengkap konsumen sinkron dengan mode pengakuan klien, lihat `SyncMessageReceiverClientAcknowledge.java` di. Contoh Java yang berfungsi untuk menggunakan JMS dengan antrian standar Amazon SQS

Output untuk contoh ini terlihat mirip dengan yang berikut:

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
```

```
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

Menggunakan mode pengakuan tidak berurutan

Saat menggunakan `CLIENT_ACKNOWLEDGE` mode, semua pesan yang diterima sebelum pesan yang diakui secara eksplisit diakui secara otomatis. Untuk informasi selengkapnya, lihat Menggunakan mode pengakuan klien.

Amazon SQS Java Messaging Library menyediakan mode pengakuan lain. Saat menggunakan `UNORDERED_ACKNOWLEDGE` mode, semua pesan yang diterima harus diakui secara individual dan eksplisit oleh klien, terlepas dari pesanan penerimaan mereka. Untuk melakukan ini, buat sesi dengan `UNORDERED_ACKNOWLEDGE` mode.

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

Langkah-langkah yang tersisa identik dengan yang ada di Menggunakan mode pengakuan klien contoh. Untuk contoh lengkap konsumen sinkron dengan `UNORDERED_ACKNOWLEDGE` mode, lihat`SyncMessageReceiverUnorderedAcknowledge.java`.

Dalam contoh ini, outputnya terlihat mirip dengan yang berikut:

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Menggunakan Java Message Service dengan klien Amazon SQS lainnya

Menggunakan Amazon SQS Java Message Service (JMS) Client dengan AWS SDK membatasi ukuran pesan Amazon SQS hingga 256 KB. Namun, Anda dapat membuat penyedia JMS menggunakan klien Amazon SQS apa pun. Misalnya, Anda dapat menggunakan Klien JMS dengan Amazon SQS Extended Client Library for Java untuk mengirim pesan Amazon SQS yang berisi referensi ke payload pesan (hingga 2 GB) di Amazon S3. Untuk informasi selengkapnya, lihat Mengelola pesan Amazon SQS besar menggunakan Java dan Amazon S3.

Contoh kode Java berikut menciptakan penyedia JMS untuk Extended Client Library.

Lihat prasyarat Prasyarat untuk bekerja dengan JMS dan Amazon SQS sebelum menguji contoh ini.

```
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);
```

Contoh kode Java berikut membuat pabrik koneksi:

```
// 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();
```

Contoh Java yang berfungsi untuk menggunakan JMS dengan antrian standar Amazon SQS

Contoh kode berikut menunjukkan cara menggunakan Java Message Service (JMS) dengan antrian standar Amazon SQS. Untuk informasi lebih lanjut tentang bekerja dengan antrian FIFO, lihatUntuk membuat antrian FIFO,, dan. Mengirim pesan secara sinkron Menerima pesan secara sinkron (Menerima pesan secara sinkron sama untuk antrian standar dan FIFO. Namun, pesan dalam antrian FIFO berisi lebih banyak atribut.)

Lihat prasyarat Prasyarat untuk bekerja dengan JMS dan Amazon SQS sebelum menguji contoh berikut.

ExampleConfiguration.jawa

Contoh kode Java SDK v 1.x berikut menetapkan nama antrian default, wilayah, dan kredenal yang akan digunakan dengan contoh Java lainnya.

```
/*
 * 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.jawa

Contoh kode Java berikut menciptakan produser pesan teks.

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

public class TextMessageSender {
    public static void main(String args[]) throws JMSException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("TextMessageSender", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());
```

```
        // Create the session
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer =
 session.createProducer( session.createQueue( config.getQueueName() ) );

        sendMessages(session, producer);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static void sendMessages( Session session, MessageProducer producer ) {
        BufferedReader inputReader = new BufferedReader(
            new InputStreamReader( System.in, Charset.defaultCharset() ) );

        try {
            String input;
            while( true ) {
                System.out.print( "Enter message to send (leave empty to exit): " );
                input = inputReader.readLine();
                if( input == null || input.equals("" ) ) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
                System.out.println( "Send message " + message.getJMSMessageID() );
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
            System.err.println( "Failed reading input: " + e.getMessage() );
        } catch (JMSException e) {
            System.err.println( "Failed sending message: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

SyncMessageReceiver.jawa

Contoh kode Java berikut menciptakan konsumen pesan sinkron.

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

public class SyncMessageReceiver {
public static void main(String args[]) throws JMSException {
    ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

    ExampleCommon.setupLogging();

    // Create the connection factory based on the config
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                    .withRegion(config.getRegion().getName())
                    .withCredentials(config.getCredentialsProvider())
            );

    // Create the connection
    SQSConnection connection = connectionFactory.createConnection();

    // Create the queue if needed
    ExampleCommon.ensureQueueExists(connection, config.getQueueName());

    // Create the session
    Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
    MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

    connection.start();
```

```
    receiveMessages(session, consumer);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println( "Connection closed" );
}

private static void receiveMessages( Session session, MessageConsumer consumer ) {
    try {
        while( true ) {
            System.out.println( "Waiting for messages");
            // Wait 1 minute for a message
            Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        }
    } catch (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
```

AsyncMessageReceiver.jawa

Contoh kode Java berikut menciptakan konsumen pesan asinkron.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
```

```
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class AsyncMessageReceiver {
    public static void main(String args[]) throws JMSException, InterruptedException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

        // No messages are processed until this is called
        connection.start();

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
```

```
    }


    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();

        public void waitForOneMinuteOfSilence() throws InterruptedException {
            for(;;) {
                long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
                long remainingTillOneMinuteOfSilence =
                    TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
                if( remainingTillOneMinuteOfSilence < 0 ) {
                    break;
                }
                TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
            }
        }


        @Override
        public void onMessage(Message message) {
            try {
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message " +
 message.getJMSMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSException e) {
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}
```

SyncMessageReceiverClientAcknowledge.jawa

Contoh kode Java berikut menciptakan konsumen sinkron dengan modus mengakui klien.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
```

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

/**
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also
 acknowledged.
 *
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

        // Setup logging for the example
```

```
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with client acknowledge mode
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));
```

```
        // Attempt to receive another message and acknowledge it. This results in
receiving no messages since
        // we have acknowledged the second message. Although we didn't explicitly
acknowledge the first message,
        // in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the
explicitly acknowledged message
        // are also acknowledged. Therefore, we have implicitly acknowledged the first
message.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
(TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
"Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSException {
        // Receive a message
```

```
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

SyncMessageReceiverUnorderedAcknowledge.jawa

Contoh kode Java berikut menciptakan konsumen sinkron dengan modus mengakui tidak berurutan.

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

/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for
 received messages. This example
 * complements the example given in {@link SyncMessageReceiverClientAcknowledge} for
 CLIENT_ACKNOWLEDGE mode.
 *
```

```
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that the first message received in the
 prior attempt is returned again
 * for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
 explicitly acknowledged no matter what
 * the order they're received.
 *
 * This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverClientAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with unordered acknowledge mode
```

```
        Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving the first message since
        // we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE
mode, all the messages must
        // be explicitly acknowledged.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
```

```
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
 messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
 "Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

SpringExampleConfiguration.xml.xl

Contoh kode XHTML berikut adalah file konfigurasi kacang untukSpringExample.jawa.

```
<!--
    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.jawa

Contoh kode Java berikut menggunakan file konfigurasi bean untuk menginisialisasi objek Anda.

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

public class SpringExample {
    public static void main(String args[]) throws JMSException {
        if( args.length != 1 || !args[0].endsWith(".xml")) {
            System.err.println( "Usage: " + SpringExample.class.getName() + " <spring
 config.xml>" );
            System.exit(1);
```

```
        }

        File springFile = new File( args[0] );
        if( !springFile.exists() || !springFile.canRead() ) {
            System.err.println( "File " + args[0] + " doesn't exist or isn't
readable.");
            System.exit(2);
        }

        ExampleCommon.setupLogging();

        FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext( "file://" +
springFile.getAbsolutePath() );

        Connection connection;
        try {
            connection = context.getBean(Connection.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the JMS connection to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        String queueName;
        try {
            queueName = context.getBean("QueueName", String.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the name of the queue to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        if( connection instanceof SQSConnection ) {
            ExampleCommon.ensureQueueExists( (SQSConnection) connection, queueName );
        }

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
session.createConsumer( session.createQueue( queueName) );
```

```
        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                    break;
                }
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message" );
            }
        } catch (JMSException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

ExampleCommon.jawa

Contoh kode Java berikut memeriksa apakah antrian Amazon SQS ada dan kemudian membuatnya jika tidak. Ini juga termasuk contoh kode logging.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
```

```
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is
 run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName)
 throws JMSException {
        AmazonSQSMessagingClientWrapper client =
 connection.getWrappedAmazonSQSClient();

        /**
         * In most cases, you can do this with just a createQueue call, but
 GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
 queue
         * already exists. Also many users and roles have permission to call
 GetQueueUrl
         * but don't have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: ");
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ){
```

```
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up to 256k so
 that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
            System.out.println( "\t" +  Base64.encodeAsString( bytes ) );
        } else if( message instanceof ObjectMessage ) {
            ObjectMessage objMessage = (ObjectMessage) message;
            System.out.println( "\t" + objMessage.getObject() );
        }
    }
}
```

Amazon SQS mendukung implementasi JMS 1.1

Amazon SQS Java Messaging Library mendukung implementasi JMS 1.1 berikut. Untuk informasi selengkapnya tentang fitur dan kemampuan yang didukung dari Amazon SQS Java Messaging Library, lihat FAQ Amazon SQS.

Antarmuka umum yang didukung

- `Connection`
- `ConnectionFactory`
- `Destination`
- `Session`
- `MessageConsumer`
- `MessageProducer`

Jenis pesan yang didukung

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

Mode pengakuan pesan yang didukung

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

> **Note**
>
> `UNORDERED_ACKNOWLEDGE`Mode ini bukan bagian dari spesifikasi JMS 1.1. Mode ini membantu Amazon SQS mengizinkan klien JMS untuk secara eksplisit mengakui pesan.

Header yang ditentukan JMS dan properti yang dicadangkan

Untuk mengirim pesan

Saat mengirim pesan, Anda dapat mengatur header dan properti berikut untuk setiap pesan:

- `JMSXGroupID`(diperlukan untuk antrian FIFO, tidak diperbolehkan untuk antrian standar)
- `JMS_SQS_DeduplicationId`(opsional untuk antrian FIFO, tidak diperbolehkan untuk antrian standar)

Setelah Anda mengirim pesan, Amazon SQS menetapkan header dan properti berikut untuk setiap pesan:

- `JMSMessageID`
- `JMS_SQS_SequenceNumber`(hanya untuk antrian FIFO)

Untuk menerima pesan

Saat Anda menerima pesan, Amazon SQS menetapkan header dan properti berikut untuk setiap pesan:

- `JMSDestination`
- `JMSMessageID`

- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID`(hanya untuk antrian FIFO)
- `JMS_SQS_DeduplicationId`(hanya untuk antrian FIFO)
- `JMS_SQS_SequenceNumber`(hanya untuk antrian FIFO)

Tutorial Amazon SQS

Topik ini menyediakan tutorial untuk membantu Anda menjelajahi fitur dan fungsionalitas Amazon SQS.

Tutorial

- Membuat antrian Amazon SQS menggunakan CloudFormation
- Tutorial: Mengirim pesan ke antrian Amazon SQS dari Amazon Virtual Private Cloud

Membuat antrian Amazon SQS menggunakan CloudFormation

Gunakan CloudFormation konsol bersama dengan template JSON atau YAML untuk membuat antrian Amazon SQS. Untuk detail selengkapnya, lihat Bek CloudFormation erja dengan Template dan Sumber `AWS::SQS::Queue` Daya di Panduan AWS CloudFormation Pengguna.

Untuk menggunakan CloudFormation untuk membuat antrian Amazon SQS.

1. Salin kode JSON berikut ke file bernama `MyQueue.json`. Untuk membuat antrian standar, hilangkan `ContentBasedDeduplication` properti `FifoQueue` dan. Untuk informasi selengkapnya tentang deduplikasi berbasis konten, lihat. Tepat sekali diproses di Amazon SQS

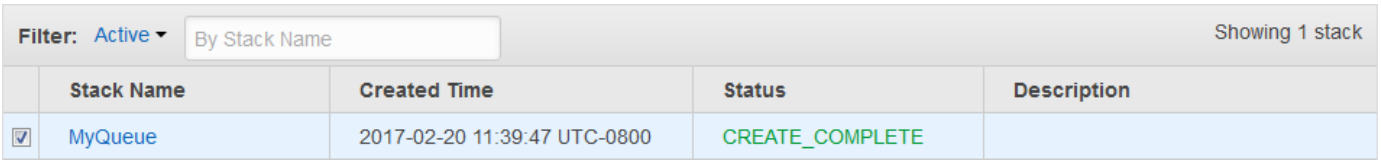
   > **Note**
   >
   > Nama antrian FIFO harus diakhiri dengan akhiran `.fifo`.

```
{
   "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. Masuk ke CloudFormation konsol, lalu pilih Buat Stack.
3. Pada panel T entukan Template, pilih Unggah file template, pilih `MyQueue.json` file Anda, lalu pilih Ber ikutnya.
4. Pada halaman T entukan Detail, ketik `MyQueue` untuk Nama Tumpukan, lalu pilih Ber ikutnya.
5. Pada halaman Opsi, pilih Selanjutnya.
6. Pada halaman Tinjau, pilih Buat.

   CloudFormation mulai membuat `MyQueue` tumpukan dan menampilkan status CREATE_IN_PROGRESS. Saat proses selesai, CloudFormation menampilkan status CREATE_COMPLETE.



7. (Opsional) Untuk menampilkan nama, URL, dan ARN antrian, pilih nama tumpukan dan kemudian pada halaman berikutnya memperluas bagian Output.

Tutorial: Mengirim pesan ke antrian Amazon SQS dari Amazon Virtual Private Cloud

Tutorial ini menunjukkan cara mengirim pesan ke antrian Amazon SQS melalui jaringan pribadi yang aman. Jaringan meliputi:

- VPC yang berisi instans Amazon EC2.
- Titik akhir VPC antarmuka, yang memungkinkan instans Amazon EC2 terhubung ke Amazon SQS tanpa menggunakan internet publik.

Bahkan dalam jaringan yang sepenuhnya pribadi, Anda dapat terhubung ke instans Amazon EC2 dan mengirim pesan ke antrian Amazon SQS. Untuk informasi selengkapnya, lihat Titik akhir Amazon Virtual Private Cloud untuk Amazon SQS.

Important

- Anda dapat menggunakan Amazon Virtual Private Cloud hanya dengan titik akhir HTTPS Amazon SQS.
- Saat mengonfigurasi Amazon SQS untuk mengirim pesan dari Amazon VPC, Anda harus mengaktifkan DNS pribadi dan menentukan titik akhir dalam format `sqs.us-east-2.amazonaws.com` atau untuk titik akhir tumpukan ganda. `sqs.us-east-2.api.aws`
- Amazon SQS juga mendukung endpoint FIPS melalui PrivateLink penggunaan layanan endpoint. `com.amazonaws.region.sqs-fips` Anda dapat terhubung ke titik akhir FIPS dalam format. `sqs-fips.region.amazonaws.com`
- Saat menggunakan titik akhir dual-stack di Amazon Virtual Private Cloud, permintaan akan dikirim menggunakan dan. IPv4 IPv6

- DNS pribadi tidak mendukung titik akhir lama seperti atau. `queue.amazonaws.com` *`us-east-2`*`.queue.amazonaws.com`

Langkah 1: Buat pasangan kunci Amazon EC2

Sebuah key pair memungkinkan Anda terhubung ke instans Amazon EC2. Ini terdiri dari kunci publik yang mengenkripsi informasi login Anda dan kunci pribadi yang mendekripsi itu.

1. Masuk ke konsol Amazon EC2.
2. Pada menu navigasi, di bawah Jaringan & Keamanan, pilih Pasangan Kunci.
3. Pilih Buat pasangan kunci.
4. Dalam Buat Pasangan Kunci kotak dialog, untuk Nama pasangan kunci, masukkan`SQS-VPCE-Tutorial-Key-Pair`, lalu pilih Buat.
5. Browser Anda mengunduh file kunci pribadi `SQS-VPCE-Tutorial-Key-Pair.pem` secara otomatis.

   > **Important**
   >
   > Simpan file ini di tempat yang aman. EC2 tidak menghasilkan `.pem` file untuk key pair yang sama untuk kedua kalinya.

6. Untuk memungkinkan klien SSH terhubung ke instans EC2 Anda, atur izin untuk file kunci pribadi Anda sehingga hanya pengguna Anda yang dapat memiliki izin membaca untuk itu, misalnya:

   ```
   chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
   ```

Langkah 2: Buat AWS sumber daya

Untuk menyiapkan infrastruktur yang diperlukan, Anda harus menggunakan CloudFormation templat, yang merupakan cetak biru untuk membuat tumpukan yang terdiri dari AWS sumber daya, seperti instans Amazon EC2 dan antrian Amazon SQS.

Tumpukan untuk tutorial ini mencakup sumber daya berikut:

- VPC dan sumber daya jaringan terkait, termasuk subnet, grup keamanan, gateway internet, dan tabel rute

- Instans Amazon EC2 diluncurkan ke subnet VPC
- Sebuah Antrean Amazon SQS

1. Unduh CloudFormation template yang dinamai `SQS-VPCE-Tutorial-CloudFormation.yaml`dariGitHub.
2. Masuk ke konsol CloudFormation.
3. Pilih Buat Tumpukan.
4. Pada halaman Pilih Template, pilih Unggah templat ke Amazon S3, pilih **`SQS-VPCE-SQS-Tutorial-CloudFormation.yaml`** file, lalu pilih Berikutnya.
5. Di halaman Tentukan Detail, lakukan hal berikut:
   a. Untuk Nama tumpukan, masukkan `SQS-VPCE-Tutorial-Stack`.
   b. Untuk KeyName, pilih SQS-VPCE-Tutorial-Key-Pair.
   c. Pilih Berikutnya.
6. Pada halaman Opsi, pilih Selanjutnya.
7. Pada halaman Tinjauan, di bagian Kemampuan, pilih Saya mengakui yang AWS CloudFormation mungkin membuat sumber daya IAM dengan nama khusus. , dan kemudian pilih Buat.

CloudFormation mulai membuat tumpukan dan menampilkan status CREATE_IN_PROGRESS. Saat proses selesai, CloudFormation menampilkan status CREATE_COMPLETE.

Langkah 3: Konfirmasikan bahwa instans EC2 Anda tidak dapat diakses publik

CloudFormation Template Anda meluncurkan instans EC2 yang diberi nama `SQS-VPCE-Tutorial-EC2-Instance` ke dalam VPC Anda. Instans EC2 ini tidak mengizinkan lalu lintas keluar dan tidak dapat mengirim pesan ke Amazon SQS. Untuk memverifikasi ini, Anda harus terhubung ke instance, mencoba menyambung ke titik akhir publik, dan kemudian mencoba mengirim pesan kepada Amazon SQS.

1. Masuk ke konsol Amazon EC2.
2. Pada menu navigasi, di bawah Instans, pilih Instans.
3. Pilih SQS-VPCE -. Tutorial-EC2Instance

4. Salin nama host di bawah DNS Publik, misalnya, ec2-203-0-113-0.us-west-2.compute.amazonaws.com.
5. Dari direktori yang berisi key pair yang Anda buat sebelumnya, sambungkan ke instance menggunakan perintah berikut, misalnya:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. Cobalah untuk terhubung ke titik akhir publik apa pun, misalnya:

```
ping amazon.com
```

   Upaya koneksi gagal, seperti yang diharapkan.

7. Masuk ke konsol Amazon SQS.
8. Dari daftar antrian, pilih antrian yang dibuat oleh CloudFormation template Anda, misalnya, VPCE-SQS-Tutorial-Stack - -1 IJK. CFQueue ABCDEFGH2
9. Pada tabel Detail, salin URL, misalnya, https://sqs.us-east-2.amazonaws.com/123456789012/.
10. Dari instans EC2 Anda, coba publikasikan pesan ke antrian menggunakan perintah berikut, misalnya:

```
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."
```

   Upaya pengiriman gagal, seperti yang diharapkan.

> **Important**
>
> Kemudian, saat Anda membuat titik akhir VPC untuk Amazon SQS, upaya pengiriman Anda akan berhasil.

Langkah 4: Buat titik akhir Amazon VPC untuk Amazon SQS

Untuk menghubungkan VPC Anda ke Amazon SQS, Anda harus menentukan titik akhir VPC antarmuka. Setelah menambahkan titik akhir, Anda dapat menggunakan Amazon SQS API dari

instans EC2 di VPC Anda. Ini memungkinkan Anda untuk mengirim pesan ke antrian dalam AWS jaringan tanpa melintasi internet publik.

> **Note**
>
> Instans EC2 masih belum memiliki akses ke AWS layanan dan titik akhir lain di internet.

1. Masuk ke Amazon konsol Amazon VPC.
2. Pada menu navigasi, pilih Endpoints.
3. Pilih Buat Titik Akhir.
4. Pada halaman Buat Titik Akhir, untuk Nama Layanan, pilih nama layanan untuk Amazon SQS.

   > **Note**
   >
   > Nama layanan bervariasi berdasarkan AWS Wilayah saat ini. Misalnya, jika Anda berada di AS Timur (Ohio), nama layanannya adalah com.amazonaws. ***`us-east-2`***.sqs.

5. Untuk VPC, pilih SQS-VPCE-Tutorial-VPC.
6. Untuk Subnet, pilih subnet yang Subnetnya ID berisi SQS-VPCE-Tutorial-Subnet.
7. Untuk grup Keamanan, pilih Pilih grup keamanan, lalu pilih grup keamanan yang Nama grupnya berisi SQS VPCE Tutorial Security Group.
8. Pilih Buat titik akhir.

   Titik akhir VPC antarmuka dibuat dan ID-nya ditampilkan, misalnya, vpce-0ab1cdef2ghi3j456k.
9. Pilih Tutup.

   Konsol Amazon VPC membuka halaman Titik akhir.

Amazon VPC mulai membuat titik akhir dan menampilkan status yang tertunda. Ketika proses selesai, Amazon VPC menampilkan status yang tersedia.

Langkah 5: Kirim pesan ke antrian Amazon SQS Anda

Sekarang VPC Anda menyertakan titik akhir untuk Amazon SQS, Anda dapat terhubung ke instans EC2 dan mengirim pesan ke antrian Anda.

1. Sambungkan kembali ke instans EC2 Anda, misalnya:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

2. Cobalah untuk mempublikasikan pesan ke antrian lagi menggunakan perintah berikut, misalnya:

```
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."
```

   Upaya pengiriman berhasil dan MD5 intisari badan pesan dan ID pesan ditampilkan, misalnya:

```
{
 "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
 "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

Untuk informasi tentang menerima dan menghapus pesan dari antrian yang dibuat oleh CloudFormation template Anda (misalnya, VPCE-SQS-Tutorial-Stack - -1 IJK), lihat. CFQueue ABCDEFGH2 Menerima dan menghapus pesan di Amazon SQS

Untuk informasi tentang menghapus sumber daya Anda, lihat berikut ini:

- Menghapus Titik Akhir VPC di Panduan Pengguna Amazon VPC
- Menghapus antrian Amazon SQS
- Menghentikan Instans Anda di Panduan Pengguna Amazon EC2
- Menghapus VPC Anda di Panduan Pengguna Amazon VPC
- Menghapus Tumpukan di CloudFormation Konsol di AWS CloudFormation Panduan Pengguna
- Menghapus Pasangan Kunci Anda di Panduan Pengguna Amazon EC2

Contoh kode untuk Amazon SQS menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan Amazon SQS dengan AWS perangkat pengembangan perangkat lunak (SDK).

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda bagaimana menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau dikombinasikan dengan yang lain Layanan AWS.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Contoh kode

- Contoh dasar untuk Amazon SQS menggunakan AWS SDKs
  - Halo Amazon SQS
  - Tindakan untuk Amazon SQS menggunakan AWS SDKs
    - Gunakan AddPermission dengan CLI
    - Gunakan ChangeMessageVisibility dengan AWS SDK atau CLI
    - Gunakan ChangeMessageVisibilityBatch dengan CLI
    - Gunakan CreateQueue dengan AWS SDK atau CLI
    - Gunakan DeleteMessage dengan AWS SDK atau CLI
    - Gunakan DeleteMessageBatch dengan AWS SDK atau CLI
    - Gunakan DeleteQueue dengan AWS SDK atau CLI
    - Gunakan GetQueueAttributes dengan AWS SDK atau CLI
    - Gunakan GetQueueUrl dengan AWS SDK atau CLI
    - Gunakan ListDeadLetterSourceQueues dengan CLI
    - Gunakan ListQueues dengan AWS SDK atau CLI
    - Gunakan PurgeQueue dengan CLI
    - Gunakan ReceiveMessage dengan AWS SDK atau CLI

    - Gunakan RemovePermission dengan CLI
    - Gunakan SendMessage dengan AWS SDK atau CLI
    - Gunakan SendMessageBatch dengan AWS SDK atau CLI
    - Gunakan SetQueueAttributes dengan AWS SDK atau CLI
- Skenario untuk Amazon SQS menggunakan AWS SDKs
    - Buat aplikasi web yang mengirim dan mengambil pesan dengan menggunakan Amazon SQS
    - Membuat aplikasi messenger dengan Step Functions
    - Membuat aplikasi penjelajah Amazon Textract
    - Membuat dan memublikasikan ke topik FIFO Amazon SNS menggunakan SDK AWS
    - Mendeteksi orang dan objek dalam video dengan Amazon Rekognition menggunakan SDK AWS
    - Mengelola pesan Amazon SQS besar menggunakan Amazon S3 dengan SDK AWS
    - Menerima dan memproses notifikasi peristiwa Amazon S3 dengan menggunakan SDK AWS
    - Mempublikasikan pesan Amazon SNS ke antrian Amazon SQS menggunakan SDK AWS
    - Mengirim dan menerima kumpulan pesan dengan Amazon SQS menggunakan SDK AWS
    - Gunakan AWS Message Processing Framework untuk.NET untuk mempublikasikan dan menerima pesan Amazon SQS
    - Gunakan Amazon SQS Java Messaging Library untuk bekerja dengan antarmuka Java Message Service (JMS) untuk Amazon SQS
    - Bekerja dengan tag antrian dan Amazon SQS menggunakan SDK AWS
- Contoh tanpa server untuk Amazon SQS
    - Memanggil fungsi Lambda dari pemicu Amazon SQS
    - Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS

Contoh dasar untuk Amazon SQS menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menggunakan dasar-dasar Amazon Simple Queue Service dengan AWS SDKs.

Contoh

- Halo Amazon SQS
- Tindakan untuk Amazon SQS menggunakan AWS SDKs
    - Gunakan AddPermission dengan CLI

- Gunakan ChangeMessageVisibility dengan AWS SDK atau CLI
- Gunakan ChangeMessageVisibilityBatch dengan CLI
- Gunakan CreateQueue dengan AWS SDK atau CLI
- Gunakan DeleteMessage dengan AWS SDK atau CLI
- Gunakan DeleteMessageBatch dengan AWS SDK atau CLI
- Gunakan DeleteQueue dengan AWS SDK atau CLI
- Gunakan GetQueueAttributes dengan AWS SDK atau CLI
- Gunakan GetQueueUrl dengan AWS SDK atau CLI
- Gunakan ListDeadLetterSourceQueues dengan CLI
- Gunakan ListQueues dengan AWS SDK atau CLI
- Gunakan PurgeQueue dengan CLI
- Gunakan ReceiveMessage dengan AWS SDK atau CLI
- Gunakan RemovePermission dengan CLI
- Gunakan SendMessage dengan AWS SDK atau CLI
- Gunakan SendMessageBatch dengan AWS SDK atau CLI
- Gunakan SetQueueAttributes dengan AWS SDK atau CLI

Halo Amazon SQS

Contoh kode berikut menunjukkan cara memulai menggunakan Amazon SQS.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
        }
    }
}
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kode untuk 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})
```

Kode untuk file sumber 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;
}
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk C++ API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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)
   }
  }
}
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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);
        }
    }
}
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Inisialisasi klien Amazon SQS dan daftar antrian.

```
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"));
};
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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")
            }
    }
}
```

- Untuk detail API, lihat ListQueuesdi AWS SDK untuk referensi API Kotlin.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

`Package.swift`File.

```
import PackageDescription

let package = Package(
    name: "sqs-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
```

```
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sqs-basics",
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

Kode sumber 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)
        }
    }
}
```

- Untuk detail API, lihat referensi ListQueues AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Tindakan untuk Amazon SQS menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara melakukan tindakan Amazon SQS individual dengan. AWS SDKs Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan instruksi untuk mengatur dan menjalankan kode.

Kutipan ini memanggil Amazon SQS API dan merupakan kutipan kode dari program yang lebih besar yang harus dijalankan dalam konteks. Anda dapat melihat tindakan dalam konteks diSkenario untuk Amazon SQS menggunakan AWS SDKs.

Contoh berikut hanya mencakup tindakan yang paling umum digunakan. Untuk daftar lengkapnya, lihat Referensi API Layanan Antrian Sederhana Amazon.

Contoh

- Gunakan AddPermission dengan CLI
- Gunakan ChangeMessageVisibility dengan AWS SDK atau CLI
- Gunakan ChangeMessageVisibilityBatch dengan CLI
- Gunakan CreateQueue dengan AWS SDK atau CLI
- Gunakan DeleteMessage dengan AWS SDK atau CLI
- Gunakan DeleteMessageBatch dengan AWS SDK atau CLI

- Gunakan DeleteQueue dengan AWS SDK atau CLI
- Gunakan GetQueueAttributes dengan AWS SDK atau CLI
- Gunakan GetQueueUrl dengan AWS SDK atau CLI
- Gunakan ListDeadLetterSourceQueues dengan CLI
- Gunakan ListQueues dengan AWS SDK atau CLI
- Gunakan PurgeQueue dengan CLI
- Gunakan ReceiveMessage dengan AWS SDK atau CLI
- Gunakan RemovePermission dengan CLI
- Gunakan SendMessage dengan AWS SDK atau CLI
- Gunakan SendMessageBatch dengan AWS SDK atau CLI
- Gunakan SetQueueAttributes dengan AWS SDK atau CLI

Gunakan **`AddPermission`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan`AddPermission`.

CLI

AWS CLI

Untuk menambahkan izin ke antrian

Contoh ini memungkinkan AWS akun yang ditentukan untuk mengirim pesan ke antrian yang ditentukan.

Perintah:

```
aws sqs add-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue --aws-
account-ids 12345EXAMPLE --actions SendMessage
```

Output:

```
None.
```

- Untuk detail API, lihat AddPermissiondi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini memungkinkan yang ditentukan Akun AWS untuk mengirim pesan dari antrian yang ditentukan.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat AddPermissiondi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini memungkinkan yang ditentukan Akun AWS untuk mengirim pesan dari antrian yang ditentukan.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat AddPermissiondi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ChangeMessageVisibility`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`ChangeMessageVisibility`.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
}
```

- Untuk detail API, lihat ChangeMessageVisibilitydi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk mengubah visibilitas batas waktu pesan

Contoh ini mengubah visibilitas batas waktu pesan yang ditentukan menjadi 10 jam (10 jam* 60 menit * 60 detik).

Perintah:

```
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.
```

- Untuk detail API, lihat ChangeMessageVisibilitydi Referensi AWS CLI Perintah.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Terima pesan Amazon SQS dan ubah visibilitas batas waktunya.

```
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;
};
```

- Untuk detail API, lihat ChangeMessageVisibilitydi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Terima pesan Amazon SQS dan ubah visibilitas batas waktunya.

```
// 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");
    }
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat ChangeMessageVisibilitydi Referensi AWS SDK untuk JavaScript API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengubah batas waktu visibilitas untuk pesan dengan pegangan tanda terima yang ditentukan dalam antrian yang ditentukan menjadi 10 jam (10 jam* 60 menit * 60 detik = 36000 detik).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Untuk detail API, lihat ChangeMessageVisibilitydi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengubah batas waktu visibilitas untuk pesan dengan pegangan tanda terima yang ditentukan dalam antrian yang ditentukan menjadi 10 jam (10 jam* 60 menit * 60 detik = 36000 detik).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Untuk detail API, lihat ChangeMessageVisibilitydi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat ChangeMessageVisibilitydi Referensi AWS SDK untuk Ruby API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ChangeMessageVisibilityBatch`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan`ChangeMessageVisibilityBatch`.

CLI

AWS CLI

Untuk mengubah visibilitas batas waktu beberapa pesan sebagai batch

Contoh ini mengubah visibilitas batas waktu 2 pesan yang ditentukan menjadi 10 jam (10 jam* 60 menit * 60 detik).

Perintah:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

Berkas masukan (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"
    }
  ]
}
```

- Untuk detail API, lihat ChangeMessageVisibilityBatchdi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengubah batas waktu visibilitas untuk 2 pesan dengan tanda terima yang ditentukan dalam antrian yang ditentukan. Batas waktu visibilitas pesan pertama diubah menjadi 10 jam (10 jam* 60 menit * 60 detik = 36000 detik). Batas waktu visibilitas pesan kedua diubah menjadi 5 jam (5 jam * 60 menit * 60 detik = 18000 detik).

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

Output:

```
Failed    Successful
```

```
------    ----------

{}        {Request2, Request1}
```

- Untuk detail API, lihat ChangeMessageVisibilityBatchdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengubah batas waktu visibilitas untuk 2 pesan dengan tanda terima yang ditentukan dalam antrian yang ditentukan. Batas waktu visibilitas pesan pertama diubah menjadi 10 jam (10 jam* 60 menit * 60 detik = 36000 detik). Batas waktu visibilitas pesan kedua diubah menjadi 5 jam (5 jam * 60 menit * 60 detik = 18000 detik).

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

Output:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- Untuk detail API, lihat ChangeMessageVisibilityBatchdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`CreateQueue`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`CreateQueue`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Publikasikan pesan ke antrian
- Mengirim dan menerima batch pesan
- Gunakan Amazon SQS Java Messaging Library untuk bekerja dengan antarmuka JMS

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat antrian dengan nama tertentu.

```
    /// <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;
    }
```

Buat antrian Amazon SQS dan kirim pesan ke sana.

```
    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;
        }
    }
```

- Untuk detail API, lihat CreateQueuedi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
}
```

- Untuk detail API, lihat CreateQueuedi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk membuat antrian

Contoh ini membuat antrian dengan nama yang ditentukan, menetapkan periode penyimpanan pesan menjadi 3 hari (3 hari* 24 jam* 60 menit * 60 detik), dan mengatur antrian surat mati antrian ke antrian yang ditentukan dengan jumlah penerimaan maksimum 1.000 pesan.

Perintah:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

File masukan (buat-antrean.json):

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}
```

Output:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
```

```
}
```

- Untuk detail API, lihat CreateQueuedi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
}
```

- Untuk detail API, lihat CreateQueuedi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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 (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);
        }
    }
}
```

- Untuk detail API, lihat CreateQueuedi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat antrian standar 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;
};
```

Buat antrian Amazon SQS dengan polling panjang.

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat CreateQueuedi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat antrian standar 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);
  }
});
```

Buat antrean Amazon SQS yang menunggu pesan tiba.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat CreateQueuedi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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 getQueueUrlResponse.queueUrl.toString()
    }
}
```

- Untuk detail API, lihat CreateQueuedi AWS SDK untuk referensi API Kotlin.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini membuat antrian dengan nama yang ditentukan.

```
New-SQSQueue -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat CreateQueuedi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini membuat antrian dengan nama yang ditentukan.

```
New-SQSQueue -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat CreateQueuedi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat CreateQueuedi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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__
```

- Untuk detail API, lihat CreateQueuedi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat antrian standar Amazon SQS.

```
    TRY.
        oo_result = lo_sqs->createqueue( iv_queuename = iv_queue_name ).        "
 oo_result is returned for testing purposes. "
        MESSAGE 'SQS queue created.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedeldrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
 queue with the same name.' TYPE 'E'.
      CATCH /aws1/cx_sqsqueuenameexists.
```

```
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
    ENDTRY.
```

Buat antrean Amazon SQS yang menunggu pesan tiba.

```
    TRY.
        DATA lt_attributes TYPE /aws1/cl_sqsqueueattrmap_w=>tt_queueattributemap.
        DATA ls_attribute TYPE /aws1/
cl_sqsqueueattrmap_w=>ts_queueattributemap_maprow.
        ls_attribute-key = 'ReceiveMessageWaitTimeSeconds'.               " Time
 in seconds for long polling, such as how long the call waits for a message to
 arrive in the queue before returning. "
        ls_attribute-value = NEW /aws1/cl_sqsqueueattrmap_w( iv_value =
 iv_wait_time ).
        INSERT ls_attribute INTO TABLE lt_attributes.
        oo_result = lo_sqs->createqueue(                  " oo_result is returned
 for testing purposes. "
                iv_queuename = iv_queue_name
                it_attributes = lt_attributes ).
        MESSAGE 'SQS queue created.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedeldrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
 queue with the same name.' TYPE 'E'.
      CATCH /aws1/cx_sqsqueuenameexists.
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat CreateQueuedi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
        }
```

- Untuk detail API, lihat referensi CreateQueue AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteMessage`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DeleteMessage`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Mengirim dan menerima batch pesan

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menerima pesan dari antrian Amazon SQS dan kemudian menghapus pesan.

```
        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;
        }
    }
```

- Untuk detail API, lihat DeleteMessagedi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
}
```

- Untuk detail API, lihat DeleteMessagedi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menghapus pesan

Contoh ini menghapus pesan yang ditentukan.

Perintah:

```
aws sqs delete-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

Output:

```
None.
```

- Untuk detail API, lihat DeleteMessagedi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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);
        }
```

- Untuk detail API, lihat DeleteMessagedi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menerima dan menghapus pesan 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,
        })),
      }),
    );
  }
};
```

- Untuk detail API, lihat DeleteMessagedi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menerima dan menghapus pesan 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);
      }
    });
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DeleteMessagedi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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!")
    }
}
```

- Untuk detail API, lihat DeleteMessagedi AWS SDK untuk referensi API Kotlin.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus pesan dengan pegangan tanda terima yang ditentukan dari antrian yang ditentukan.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Untuk detail API, lihat DeleteMessagedi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus pesan dengan pegangan tanda terima yang ditentukan dari antrian yang ditentukan.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Untuk detail API, lihat DeleteMessagedi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat DeleteMessagedi AWS SDK for Python (Boto3) Referensi API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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.
```

- Untuk detail API, lihat DeleteMessagedi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteMessageBatch`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DeleteMessageBatch`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Memproses pemberitahuan acara S3
- Publikasikan pesan ke antrian
- Mengirim dan menerima batch pesan

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
    }
```

- Untuk detail API, lihat DeleteMessageBatchdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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;
            }
```

- Untuk detail API, lihat DeleteMessageBatchdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menghapus beberapa pesan sebagai batch

Contoh ini menghapus pesan yang ditentukan.

Perintah:

```
aws sqs delete-message-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-
batch.json
```

Berkas masukan (delete-message-batch.json):

```
[
```

```
  {
        "Id": "FirstMessage",
        "ReceiptHandle": "AQEB1mgl...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

Output:

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- Untuk detail API, lihat DeleteMessageBatchdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
}
```

- Untuk detail API, lihat DeleteMessageBatchdi Referensi AWS SDK untuk Go API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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,
        })),
      }),
    );
  }
};
```

- Untuk detail API, lihat DeleteMessageBatchdi Referensi AWS SDK untuk JavaScript API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus 2 pesan dengan tanda terima yang ditentukan dari antrian yang ditentukan.

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

Output:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- Untuk detail API, lihat DeleteMessageBatchdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus 2 pesan dengan tanda terima yang ditentukan dari antrian yang ditentukan.

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

Output:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- Untuk detail API, lihat DeleteMessageBatchdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat DeleteMessageBatchdi AWS SDK for Python (Boto3) Referensi API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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.
```

- Untuk detail API, lihat DeleteMessageBatchdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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.")
        }
```

- Untuk detail API, lihat referensi DeleteMessageBatch AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteQueue`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`DeleteQueue`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Publikasikan pesan ke antrian
- Mengirim dan menerima batch pesan
- Gunakan Amazon SQS Java Messaging Library untuk bekerja dengan antarmuka JMS

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Hapus antrian dengan menggunakan URL-nya.

```
/// <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;
}
```

- Untuk detail API, lihat DeleteQueuedi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
}
```

- Untuk detail API, lihat DeleteQueuedi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menghapus antrian

Contoh ini menghapus antrian yang ditentukan.

Perintah:

```
aws sqs delete-queue --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

Output:

```
None.
```

- Untuk detail API, lihat DeleteQueuedi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
}
```

- Untuk detail API, lihat DeleteQueuedi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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);
        }
    }
}
```

- Untuk detail API, lihat DeleteQueuedi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Hapus antrian 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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DeleteQueuedi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Hapus antrian 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat DeleteQueuedi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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!")
    }
}
```

- Untuk detail API, lihat DeleteQueuedi AWS SDK untuk referensi API Kotlin.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus antrian yang ditentukan.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Untuk detail API, lihat DeleteQueuedi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus antrian yang ditentukan.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Untuk detail API, lihat DeleteQueuedi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat DeleteQueuedi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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)
```

- Untuk detail API, lihat DeleteQueuedi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
        MESSAGE 'SQS queue deleted' TYPE 'I'.
    ENDTRY.
```

- Untuk detail API, lihat DeleteQueuedi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
        }
```

- Untuk detail API, lihat referensi DeleteQueue AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetQueueAttributes`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`GetQueueAttributes`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Memproses pemberitahuan acara S3

- Publikasikan pesan ke antrian

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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;
    }
```

- Untuk detail API, lihat GetQueueAttributesdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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;
```

```
        }
```

- Untuk detail API, lihat GetQueueAttributesdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk mendapatkan atribut antrian

Contoh ini mendapatkan semua atribut antrian yang ditentukan.

Perintah:

```
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"
  }
}
```

Contoh ini hanya mendapatkan ukuran pesan maksimum antrian yang ditentukan dan atribut batas waktu visibilitas.

Perintah:

```
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"
  }
}
```

- Untuk detail API, lihat GetQueueAttributesdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
}
```

- Untuk detail API, lihat GetQueueAttributesdi Referensi AWS SDK untuk Go API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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;
};
```

- Untuk detail API, lihat GetQueueAttributesdi Referensi AWS SDK untuk JavaScript API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mencantumkan semua atribut untuk antrian yang ditentukan.

```
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":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                      arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                      [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

Contoh 2: Contoh ini mencantumkan secara terpisah hanya atribut yang ditentukan untuk antrian yang ditentukan.

```
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":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                      arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                           : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- Untuk detail API, lihat GetQueueAttributesdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mencantumkan semua atribut untuk antrian yang ditentukan.

```
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":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                      arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                      [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

Contoh 2: Contoh ini mencantumkan secara terpisah hanya atribut yang ditentukan untuk antrian yang ditentukan.

```
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":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                        arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- Untuk detail API, lihat GetQueueAttributesdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat GetQueueAttributesdi AWS SDK for Python (Boto3) Referensi API.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
            }
        }
```

- Untuk detail API, lihat referensi GetQueueAttributes AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetQueueUrl`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`GetQueueUrl`.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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.");
            }
        }
    }
```

- Untuk detail API, lihat GetQueueUrldi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
}
```

- Untuk detail API, lihat GetQueueUrldi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk mendapatkan URL antrian

Contoh ini mendapatkan URL antrian yang ditentukan.

Perintah:

```
aws sqs get-queue-url --queue-name MyQueue
```

Output:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
```

```
}
```

- Untuk detail API, lihat GetQueueUrldi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
            GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

- Untuk detail API, lihat GetQueueUrldi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Dapatkan URL untuk antrean 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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat GetQueueUrldi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Dapatkan URL untuk antrean 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat GetQueueUrldi Referensi AWS SDK untuk JavaScript API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mencantumkan URL antrian dengan nama yang ditentukan.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat GetQueueUrldi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mencantumkan URL antrian dengan nama yang ditentukan.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat GetQueueUrldi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat GetQueueUrldi AWS SDK for Python (Boto3) Referensi API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_sqs->getqueueurl( iv_queuename = iv_queue_name ).        "
oo_result is returned for testing purposes. "
        MESSAGE 'Queue URL retrieved.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedoesnotexist.
        MESSAGE 'The requested queue does not exist.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat GetQueueUrldi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListDeadLetterSourceQueues`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan`ListDeadLetterSourceQueues`.

CLI

AWS CLI

Untuk membuat daftar antrian sumber surat mati

Contoh ini mencantumkan antrian yang terkait dengan antrian sumber huruf mati yang ditentukan.

Perintah:

```
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"
  ]
}
```

- Untuk detail API, lihat ListDeadLetterSourceQueuesdi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mencantumkan antrian apa pun yang bergantung pada antrian yang ditentukan sebagai antrian surat mati mereka. URLs

```
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
```

- Untuk detail API, lihat ListDeadLetterSourceQueuesdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mencantumkan antrian apa pun yang bergantung pada antrian yang ditentukan sebagai antrian surat mati mereka. URLs

```
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
```

- Untuk detail API, lihat ListDeadLetterSourceQueuesdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListQueues`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`ListQueues`.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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;
}
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk membuat daftar antrian

Contoh ini mencantumkan semua antrian.

Perintah:

```
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"
  ]
}
```

Contoh ini hanya mencantumkan antrian yang dimulai dengan “My”.

Perintah:

```
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"
  ]
}
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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)
  }
 }
}
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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);
        }
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat daftar antrian Amazon SQS Anda.

```
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;
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat daftar antrian Amazon SQS Anda.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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)
        }
    }
```

```
}
```

- Untuk detail API, lihat ListQueuesdi AWS SDK untuk referensi API Kotlin.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mencantumkan semua antrian.

```
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
```

Contoh 2: Contoh ini mencantumkan antrian apa pun yang dimulai dengan nama yang ditentukan.

```
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
```

- Untuk detail API, lihat ListQueuesdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mencantumkan semua antrian.

```
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
```

Contoh 2: Contoh ini mencantumkan antrian apa pun yang dimulai dengan nama yang ditentukan.

```
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
```

- Untuk detail API, lihat ListQueuesdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat ListQueuesdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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']
  )

  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
```

- Untuk detail API, lihat ListQueuesdi Referensi AWS SDK untuk Ruby API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Ambil antrean Amazon SQS pertama yang terdaftar di Wilayah.

```
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())
}
```

- Untuk detail API, lihat ListQueuesreferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_sqs->listqueues( ).        " oo_result is returned for
 testing purposes. "
```

```
        MESSAGE 'Retrieved list of queues.' TYPE 'I'.
    ENDTRY.
```

- Untuk detail API, lihat ListQueuesdi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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)
            }
        }
```

- Untuk detail API, lihat referensi ListQueues AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`PurgeQueue`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan`PurgeQueue`.

CLI

AWS CLI

Untuk membersihkan antrian

Contoh ini menghapus semua pesan dalam antrian yang ditentukan.

Perintah:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue
```

Output:

```
None.
```

- Untuk detail API, lihat PurgeQueuedi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus semua pesan dari antrian yang ditentukan.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat PurgeQueuedi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus semua pesan dari antrian yang ditentukan.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat PurgeQueuedi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan `ReceiveMessage` dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`ReceiveMessage`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Kelola pesan besar menggunakan S3
- Memproses pemberitahuan acara S3
- Publikasikan pesan ke antrian
- Mengirim dan menerima batch pesan

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menerima pesan dari antrian dengan menggunakan URL-nya.

```
    /// <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;
    }
```

Menerima pesan dari antrian Amazon SQS, lalu hapus pesan.

```
        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;
        }
    }
```

- Untuk detail API, lihat ReceiveMessagedi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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 =
 sqsClient.ReceiveMessage(
            request);
    if (outcome.IsSuccess()) {

        const Aws::Vector<Aws::SQS::Model::Message> &messages =
                outcome.GetResult().GetMessages();
        if (!messages.empty()) {
            const Aws::SQS::Model::Message &message = messages[0];
            std::cout << "Received message:" << std::endl;
            std::cout << "  MessageId: " << message.GetMessageId() << std::endl;
            std::cout << "  ReceiptHandle: " << message.GetReceiptHandle() <<
 std::endl;
            std::cout << "  Body: " << message.GetBody() << std::endl <<
 std::endl;
        }
        else {
            std::cout << "No messages received from queue " << queueUrl <<
                    std::endl;

        }
    }
    else {
        std::cerr << "Error receiving message from queue " << queueUrl << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- Untuk detail API, lihat ReceiveMessagedi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk menerima pesan

Contoh ini menerima hingga 10 pesan yang tersedia, mengembalikan semua atribut yang tersedia.

Perintah:

```
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": "AIDAIAZKMSNQ7TEXAMPLE",
        "ApproximateReceiveCount": "5",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        },
        "City": {
          "DataType": "String",
          "StringValue": "Any City"
        }
      }
    }
  ]
}
```

Contoh ini menerima pesan berikutnya yang tersedia, hanya mengembalikan SentTimestamp atribut SenderId dan serta atribut PostalCode pesan.

Perintah:

```
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": "AIDAIAZKMSNQ7TEXAMPLE",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        }
      }
    }
  ]
}
```

- Untuk detail API, lihat ReceiveMessagedi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
}
```

- Untuk detail API, lihat ReceiveMessagedi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        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;
```

- Untuk detail API, lihat ReceiveMessagedi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menerima pesan dari antrian 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,
        })),
```

```
      }),
    );
  }
};
```

Menerima pesan dari antrian Amazon SQS menggunakan dukungan polling panjang.

```
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;
};
```

- Untuk detail API, lihat ReceiveMessagedi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menerima pesan dari antrian Amazon SQS menggunakan dukungan polling panjang.

```
// 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat ReceiveMessagedi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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)
        }
    }
}
```

- Untuk detail API, lihat ReceiveMessagedi AWS SDK untuk referensi API Kotlin.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mencantumkan informasi hingga 10 pesan berikutnya yang akan diterima untuk antrian yang ditentukan. Informasi akan berisi nilai untuk atribut pesan yang ditentukan, jika ada.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- Untuk detail API, lihat ReceiveMessagedi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mencantumkan informasi hingga 10 pesan berikutnya yang akan diterima untuk antrian yang ditentukan. Informasi akan berisi nilai untuk atribut pesan yang ditentukan, jika ada.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- Untuk detail API, lihat ReceiveMessagedi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat ReceiveMessagedi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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__
```

- Untuk detail API, lihat ReceiveMessagedi Referensi AWS SDK untuk Ruby API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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(())
}
```

- Untuk detail API, lihat ReceiveMessagereferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Menerima pesan dari antrian 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.
```

Menerima pesan dari antrian Amazon SQS menggunakan dukungan polling panjang.

```
   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.
```

- Untuk detail API, lihat ReceiveMessagedi AWS SDK untuk referensi SAP ABAP API.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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("---")
        }
```

- Untuk detail API, lihat referensi ReceiveMessage AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`RemovePermission`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan`RemovePermission`.

CLI

AWS CLI

Untuk menghapus izin

Contoh ini menghapus izin dengan label yang ditentukan dari antrian yang ditentukan.

Perintah:

```
aws sqs remove-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue
```

Output:

```
None.
```

- Untuk detail API, lihat RemovePermissiondi Referensi AWS CLI Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus pengaturan izin dengan label yang ditentukan dari antrian yang ditentukan.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat RemovePermissiondi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus pengaturan izin dengan label yang ditentukan dari antrian yang ditentukan.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat RemovePermissiondi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **SendMessage** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`SendMessage`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Kelola pesan besar menggunakan S3
- Mengirim dan menerima batch pesan

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat antrian Amazon SQS dan kirim pesan ke sana.

```
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;
        }
    }
```

- Untuk detail API, lihat SendMessagedi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
}
```

- Untuk detail API, lihat SendMessagedi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk mengirim pesan

Contoh ini mengirimkan pesan dengan isi pesan tertentu, periode tunda, dan atribut pesan, ke antrian yang ditentukan.

Perintah:

```
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 masukan (kirim-message.json):

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
```

```
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

Output:

```
{
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- Untuk detail API, lihat SendMessagedi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Dua contoh `SendMessage` operasi berikut:

- Kirim pesan dengan tubuh dan penundaan
- Mengirim pesan dengan atribut isi dan pesan

Kirim pesan dengan tubuh dan penundaan.

```
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);
        }
    }
}
```

Kirim pesan dengan atribut tubuh dan pesan.

```
    /**
     * <p>This method demonstrates how to add message attributes to a message.
     * Each attribute must specify a name, value, and data type. You use a Java
 Map to supply the attributes. The map's
     * key is the attribute name, and you specify the map's entry value using a
 builder that includes the attribute
     * value and data type.</p>
     *
     * <p>The data type must start with one of "String", "Number" or "Binary".
 You can optionally
     * define a custom extension by using a "." and your extension.</p>
     *
     * <p>The SQS Developer Guide provides more information on @see <a
     * href="https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-message-metadata.html#sqs-message-attributes">message
     * attributes</a>.</p>
     *
     * @param thumbailPath Filesystem path of the image.
     * @param queueUrl     URL of the SQS queue.
     */
```

```
   static void sendMessageWithAttributes(Path thumbailPath, String queueUrl) {
       Map<String, MessageAttributeValue> messageAttributeMap;
       try {
           messageAttributeMap = Map.of(
                   "Name", MessageAttributeValue.builder()
                           .stringValue("Jane Doe")
                           .dataType("String").build(),
                   "Age", MessageAttributeValue.builder()
                           .stringValue("42")
                           .dataType("Number.int").build(),
                   "Image", MessageAttributeValue.builder()

.binaryValue(SdkBytes.fromByteArray(Files.readAllBytes(thumbailPath)))
                           .dataType("Binary.jpg").build()
           );
       } catch (IOException e) {
           LOGGER.error("An I/O exception occurred reading thumbnail image: {}",
e.getMessage(), e);
           throw new RuntimeException(e);
       }

       SendMessageRequest request = SendMessageRequest.builder()
               .queueUrl(queueUrl)
               .messageBody("Hello SQS")
               .messageAttributes(messageAttributeMap)
               .build();
       try {
           SendMessageResponse sendMessageResponse =
SQS_CLIENT.sendMessage(request);
           LOGGER.info("Message ID: {}", sendMessageResponse.messageId());
       } catch (SqsException e) {
           LOGGER.error("Exception occurred sending message: {}",
e.getMessage(), e);
           throw new RuntimeException(e);
       }
   }
```

- Untuk detail API, lihat SendMessagedi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan ke antrian 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;
```

```
};
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat SendMessagedi Referensi AWS SDK untuk JavaScript API.

SDK untuk JavaScript (v2)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kirim pesan ke antrian 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);
  }
});
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK untuk JavaScript.
- Untuk detail API, lihat SendMessagedi Referensi AWS SDK untuk JavaScript API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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.")
    }
}
```

- Untuk detail API, lihat SendMessagedi AWS SDK untuk referensi API Kotlin.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengirimkan pesan dengan atribut dan badan pesan yang ditentukan ke antrian yang ditentukan dengan pengiriman pesan tertunda selama 10 detik.

```
$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   51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- Untuk detail API, lihat SendMessagedi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengirimkan pesan dengan atribut dan badan pesan yang ditentukan ke antrian yang ditentukan dengan pengiriman pesan tertunda selama 10 detik.

```
$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   51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- Untuk detail API, lihat SendMessagedi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat SendMessagedi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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__
```

- Untuk detail API, lihat SendMessagedi Referensi AWS SDK untuk Ruby API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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(())
}
```

- Untuk detail API, lihat SendMessagereferensi AWS SDK for Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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.
```

- Untuk detail API, lihat SendMessagedi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`SendMessageBatch`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`SendMessageBatch`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Mengirim dan menerima batch pesan

CLI

AWS CLI

Untuk mengirim beberapa pesan sebagai batch

Contoh ini mengirimkan 2 pesan dengan badan pesan tertentu, periode tunda, dan atribut pesan, ke antrian yang ditentukan.

Perintah:

```
aws sqs send-message-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-
batch.json
```

Berkas masukan (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"
    }
  ]
}
```

- Untuk detail API, lihat SendMessageBatchdi Referensi AWS CLI Perintah.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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);
```

- Untuk detail API, lihat SendMessageBatchdi Referensi AWS SDK for Java 2.x API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengirimkan 2 pesan dengan atribut tertentu dan badan pesan ke antrian yang ditentukan. Pengiriman ditunda selama 15 detik untuk pesan pertama dan 10 detik untuk pesan kedua.

```
$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}
```

- Untuk detail API, lihat SendMessageBatchdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengirimkan 2 pesan dengan atribut tertentu dan badan pesan ke antrian yang ditentukan. Pengiriman ditunda selama 15 detik untuk pesan pertama dan 10 detik untuk pesan kedua.

```
$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}
```

- Untuk detail API, lihat SendMessageBatchdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat SendMessageBatchdi AWS SDK for Python (Boto3) Referensi API.

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat SendMessageBatchdi Referensi AWS SDK untuk Ruby API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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_sqsemptybatchrequest.
        MESSAGE 'The batch request does not contain any entries.' TYPE 'E'.
      CATCH /aws1/cx_sqsinvbatchentryid.
        MESSAGE 'The ID of a batch entry in a batch request is not valid.' TYPE
'E'.
      CATCH /aws1/cx_sqstoomanyentriesin00.
        MESSAGE 'The batch request contains more entries than allowed.' TYPE 'E'.
      CATCH /aws1/cx_sqsunsupportedop.
        MESSAGE 'Operation not supported.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat SendMessageBatchdi AWS SDK untuk referensi SAP ABAP API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`SetQueueAttributes`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan`SetQueueAttributes`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Publikasikan pesan ke antrian

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Tetapkan atribut kebijakan antrian untuk topik.

```
    /// <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;
    }
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi AWS SDK untuk .NET API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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();
}
```

Konfigurasikan antrian huruf mati.

```
        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();
}
```

Konfigurasikan antrian Amazon SQS untuk menggunakan polling panjang.

```
        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();
}
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi AWS SDK untuk C++ API.

CLI

AWS CLI

Untuk mengatur atribut antrian

Contoh ini menetapkan antrian yang ditentukan ke penundaan pengiriman 10 detik, ukuran pesan maksimum 128 KB (128 KB* 1.024 byte), periode penyimpanan pesan 3 hari (3 hari * 24 jam * 60 menit * 60 detik), waktu tunggu pesan terima 20 detik, dan batas waktu visibilitas default 60 detik. Contoh ini juga mengaitkan antrian surat mati yang ditentukan dengan jumlah penerimaan maksimum 1.000 pesan.

Perintah:

```
aws sqs set-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-attributes.json
```

Berkas masukan (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.
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi AWS CLI Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi AWS SDK untuk Go API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Konfigurasikan Amazon SQS untuk menggunakan enkripsi sisi server (SSE) menggunakan kunci KMS kustom.

```
    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();
        }
    }
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi AWS SDK for Java 2.x API.

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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;
};
```

Konfigurasikan antrian Amazon SQS untuk menggunakan polling panjang.

```
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;
};
```

Konfigurasikan antrian huruf mati.

```
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;
};
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi AWS SDK untuk JavaScript API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menunjukkan cara menyetel kebijakan berlangganan antrian ke topik SNS. Ketika pesan dipublikasikan ke topik, pesan dikirim ke antrian berlangganan.

```
# 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 }
```

Contoh 2: Contoh ini menetapkan atribut tertentu untuk antrian yang ditentukan.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi Alat AWS untuk PowerShell Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menunjukkan cara menyetel kebijakan berlangganan antrian ke topik SNS. Ketika pesan dipublikasikan ke topik, pesan dikirim ke antrian berlangganan.

```
# 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 }
```

Contoh 2: Contoh ini menetapkan atribut tertentu untuk antrian yang ditentukan.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Untuk detail API, lihat SetQueueAttributesdi Referensi Alat AWS untuk PowerShell Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Tetapkan atribut kebijakan antrian untuk topik.

```
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
```

- Untuk detail API, lihat SetQueueAttributesdi AWS SDK for Python (Boto3) Referensi API.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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.")
        }
```

- Untuk detail API, lihat referensi SetQueueAttributes AWSSDK untuk Swift API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Skenario untuk Amazon SQS menggunakan AWS SDKs

Contoh kode berikut menunjukkan cara menerapkan skenario umum di Amazon SQS dengan. AWS SDKs Skenario ini menunjukkan kepada Anda cara menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam Amazon SQS atau digabungkan dengan yang lain. Layanan AWS Setiap skenario menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode.

Skenario menargetkan tingkat pengalaman menengah untuk membantu Anda memahami tindakan layanan dalam konteks.

Contoh

- Buat aplikasi web yang mengirim dan mengambil pesan dengan menggunakan Amazon SQS
- Membuat aplikasi messenger dengan Step Functions

- Membuat aplikasi penjelajah Amazon Textract
- Membuat dan memublikasikan ke topik FIFO Amazon SNS menggunakan SDK AWS
- Mendeteksi orang dan objek dalam video dengan Amazon Rekognition menggunakan SDK AWS
- Mengelola pesan Amazon SQS besar menggunakan Amazon S3 dengan SDK AWS
- Menerima dan memproses notifikasi peristiwa Amazon S3 dengan menggunakan SDK AWS
- Mempublikasikan pesan Amazon SNS ke antrian Amazon SQS menggunakan SDK AWS
- Mengirim dan menerima kumpulan pesan dengan Amazon SQS menggunakan SDK AWS
- Gunakan AWS Message Processing Framework untuk.NET untuk mempublikasikan dan menerima pesan Amazon SQS
- Gunakan Amazon SQS Java Messaging Library untuk bekerja dengan antarmuka Java Message Service (JMS) untuk Amazon SQS
- Bekerja dengan tag antrian dan Amazon SQS menggunakan SDK AWS

Buat aplikasi web yang mengirim dan mengambil pesan dengan menggunakan Amazon SQS

Contoh kode berikut menunjukkan cara membuat aplikasi perpesanan dengan menggunakan Amazon SQS.

Java

SDK untuk Java 2.x

Menunjukkan cara menggunakan Amazon SQS API untuk mengembangkan Spring REST API yang mengirim dan mengambil pesan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK untuk Kotlin

Menunjukkan cara menggunakan Amazon SQS API untuk mengembangkan Spring REST API yang mengirim dan mengambil pesan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Amazon SQS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi messenger dengan Step Functions

Contoh kode berikut menunjukkan cara membuat aplikasi AWS Step Functions messenger yang mengambil catatan pesan dari tabel database.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) with AWS Step Functions untuk membuat aplikasi messenger yang mengambil catatan pesan dari tabel Amazon DynamoDB dan mengirimkannya dengan Amazon Simple Queue Service (Amazon SQS). Mesin state terintegrasi dengan AWS Lambda fungsi untuk memindai database untuk pesan yang tidak terkirim.

- Buat mesin status yang mengambil dan memperbarui catatan pesan dari tabel Amazon DynamoDB.
- Perbarui definisi mesin status untuk mengirim pesan ke Amazon Simple Queue Service (Amazon SQS).
- Mulai dan hentikan berjalannya mesin status.

- Terhubung ke Lambda, DynamoDB, dan Amazon SQS dari mesin status menggunakan integrasi layanan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi penjelajah Amazon Textract

Contoh kode berikut ini menunjukkan cara menjelajahi output Amazon Textract melalui aplikasi interaktif.

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan aplikasi AWS SDK untuk JavaScript untuk membangun aplikasi React yang menggunakan Amazon Textract untuk mengekstrak data dari gambar dokumen dan menampilkannya di halaman web interaktif. Contoh ini berjalan di peramban web dan memerlukan identitas Amazon Cognito yang diautentikasi sebagai kredensialnya. Contoh ini menggunakan Amazon Simple Storage Service (Amazon S3) untuk penyimpanan, dan untuk notifikasi, contoh ini mengambil polling antrean Amazon Simple Queue Service (Amazon SQS) yang berlangganan topik Amazon Simple Notification Service (Amazon SNS).

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Identitas Amazon Cognito

- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan Amazon Textract untuk mendeteksi elemen teks, formulir, dan tabel dalam gambar dokumen. AWS SDK untuk Python (Boto3) Gambar input dan output Amazon Textract ditampilkan dalam aplikasi Tkinter yang memungkinkan Anda menjelajahi elemen yang terdeteksi.

- Kirim gambar dokumen ke Amazon Textract dan jelajahi output elemen yang terdeteksi.
- Kirim gambar langsung ke Amazon Textract atau melalui bucket Amazon Simple Storage Service (Amazon S3).
- Gunakan asinkron APIs untuk memulai pekerjaan yang menerbitkan pemberitahuan ke topik Simple Notification Service Amazon (Amazon SNS) saat pekerjaan selesai.
- Lakukan polling pada antrean Amazon Simple Queue Service (Amazon SQS) untuk mendapatkan pesan penyelesaian tugas dan tampilkan hasilnya.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Identitas Amazon Cognito
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat dan memublikasikan ke topik FIFO Amazon SNS menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat dan mempublikasikan ke topik FIFO Amazon SNS.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini

- membuat topik Amazon SNS FIFO, dua antrian FIFO Amazon SQS, dan satu antrian Standar.
- berlangganan antrian ke topik dan menerbitkan pesan ke topik tersebut.

Tes memverifikasi penerimaan pesan ke setiap antrian. Contoh lengkap juga menunjukkan penambahan kebijakan akses dan menghapus sumber daya di akhir.

```
public class PriceUpdateExample {
    public final static SnsClient snsClient = SnsClient.create();
    public final static SqsClient sqsClient = SqsClient.create();

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName>
 <analyticsQueueName>\n\n" +
            "Where:\n" +
            "   fifoTopicName - The name of the FIFO topic that you want to
 create. \n\n" +
            "   wholesaleQueueARN - The name of a SQS FIFO queue that will be
 created for the wholesale consumer. \n\n"
            +
            "   retailQueueARN - The name of a SQS FIFO queue that will created
 for the retail consumer. \n\n" +
```

```
            "   analyticsQueueARN - The name of a SQS standard queue that will be
created for the analytics consumer. \n\n";
        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        final String fifoTopicName = args[0];
        final String wholeSaleQueueName = args[1];
        final String retailQueueName = args[2];
        final String analyticsQueueName = args[3];

        // For convenience, the QueueData class holds metadata about a queue:
ARN, URL,
        // name and type.
        List<QueueData> queues = List.of(
            new QueueData(wholeSaleQueueName, QueueType.FIFO),
            new QueueData(retailQueueName, QueueType.FIFO),
            new QueueData(analyticsQueueName, QueueType.Standard));

        // Create queues.
        createQueues(queues);

        // Create a topic.
        String topicARN = createFIFOTopic(fifoTopicName);

        // Subscribe each queue to the topic.
        subscribeQueues(queues, topicARN);

        // Allow the newly created topic to send messages to the queues.
        addAccessPolicyToQueuesFINAL(queues, topicARN);

        // Publish a sample price update message with payload.
        publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}",
"Consumables");

        // Clean up resources.
        deleteSubscriptions(queues);
        deleteQueues(queues);
        deleteTopic(topicARN);
    }

    public static String createFIFOTopic(String topicName) {
        try {
```

```
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = Map.of(
                "FifoTopic", "true",
                "ContentBasedDeduplication", "false",
                "FifoThroughputScope", "MessageGroup");

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            String topicArn = response.topicArn();
            System.out.println("The topic ARN is" + topicArn);

            return topicArn;

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static void subscribeQueues(List<QueueData> queues, String topicARN) {
        queues.forEach(queue -> {
            SubscribeRequest subscribeRequest = SubscribeRequest.builder()
                .topicArn(topicARN)
                .endpoint(queue.queueARN)
                .protocol("sqs")
                .build();

            // Subscribe to the endpoint by using the SNS service client.
            // Only Amazon SQS queues can receive notifications from an Amazon
SNS FIFO
            // topic.
            SubscribeResponse subscribeResponse =
snsClient.subscribe(subscribeRequest);
            System.out.println("The queue [" + queue.queueARN + "] subscribed to
the topic [" + topicARN + "]");
            queue.subscriptionARN = subscribeResponse.subscriptionArn();
        });
    }
```

```
   public static void publishPriceUpdate(String topicArn, String payload, String
groupId) {

       try {
           // Create and publish a message that updates the wholesale price.
           String subject = "Price Update";
           String dedupId = UUID.randomUUID().toString();
           String attributeName = "business";
           String attributeValue = "wholesale";

           MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
               .dataType("String")
               .stringValue(attributeValue)
               .build();

           Map<String, MessageAttributeValue> attributes = new HashMap<>();
           attributes.put(attributeName, msgAttValue);
           PublishRequest pubRequest = PublishRequest.builder()
               .topicArn(topicArn)
               .subject(subject)
               .message(payload)
               .messageGroupId(groupId)
               .messageDeduplicationId(dedupId)
               .messageAttributes(attributes)
               .build();

           final PublishResponse response = snsClient.publish(pubRequest);
           System.out.println(response.messageId());
           System.out.println(response.sequenceNumber());
           System.out.println("Message was published to " + topicArn);

       } catch (SnsException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - CreateTopic
  - Publikasikan
  - Berlangganan

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat topik Amazon SNS FIFO, berlangganan Amazon SQS FIFO dan antrian standar ke topik tersebut, dan publikasikan pesan ke topik tersebut.

```
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
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - CreateTopic
  - Publikasikan
  - Berlangganan

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat topik FIFO, berlangganan antrian Amazon SQS FIFO ke topik tersebut, dan publikasikan pesan ke topik Amazon SNS.

```
    " Creates a FIFO topic. "
    DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>tt_topicattributesmap.
    DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>ts_topicattributesmap_maprow.
    ls_tpc_attributes-key = 'FifoTopic'.
    ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsmap_w( iv_value =
 'true' ).
    INSERT ls_tpc_attributes INTO TABLE lt_tpc_attributes.

    TRY.
        DATA(lo_create_result) = lo_sns->createtopic(
               iv_name = iv_topic_name
               it_attributes = lt_tpc_attributes ).
        DATA(lv_topic_arn) = lo_create_result->get_topicarn( ).
        ov_topic_arn = lv_topic_arn.                                    "
 ov_topic_arn is returned for testing purposes. "
        MESSAGE 'FIFO topic created' TYPE 'I'.
      CATCH /aws1/cx_snstopiclimitexcdex.
        MESSAGE 'Unable to create more topics. You have reached the maximum
 number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
 SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
 to an SNS FIFO topic. "
    TRY.
        DATA(lo_subscribe_result) = lo_sns->subscribe(
               iv_topicarn = lv_topic_arn
               iv_protocol = 'sqs'
               iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.                      "
 ov_subscription_arn is returned for testing purposes. "
        MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
      CATCH /aws1/cx_snssubscriptionlmte00.
```

```
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
 number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
 'String'

 iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.

        DATA(lo_result) = lo_sns->publish(
             iv_topicarn = lv_topic_arn
             iv_message = 'The price of your mobile plan has been increased from
 $19 to $23'
             iv_subject = 'Changes to mobile plan'
             iv_messagegroupid = 'Update-2'
             iv_messagededuplicationid = 'Update-2.1'
             it_messageattributes = lt_msg_attributes ).
        ov_message_id = lo_result->get_messageid( ).                    "
 ov_message_id is returned for testing purposes. "
        MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
      CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.
```

- Untuk mengetahui hal detail mengenai API, silakan lihat topik-topik berikut di referensi API AWS SDK untuk ABAP SAP.
  - CreateTopic
  - Publikasikan
  - Berlangganan

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Mendeteksi orang dan objek dalam video dengan Amazon Rekognition menggunakan SDK AWS

Contoh kode berikut menunjukkan cara mendeteksi orang dan objek dalam video dengan Amazon Rekognition.

Java

SDK untuk Java 2.x

Menunjukkan cara menggunakan Amazon Rekognition Java API untuk membuat aplikasi guna mendeteksi wajah dan objek di video yang berada di bucket Amazon Simple Storage Service (Amazon S3). Aplikasi ini mengirimkan notifikasi email kepada admin beserta hasilnya menggunakan Amazon Simple Email Service (Amazon SES).

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK untuk Python (Boto3)

Gunakan Amazon Rekognition untuk mendeteksi wajah, objek, dan orang dalam video dengan memulai tugas deteksi asinkron. Contoh ini juga mengonfigurasi Amazon Rekognition untuk memberi tahu topik Amazon Simple Notification Service (Amazon SNS) saat pekerjaan selesai dan berlangganan antrian Amazon Simple Queue Service (Amazon SQS) ke topik tersebut.

Ketika antrian menerima pesan tentang pekerjaan, pekerjaan diambil dan hasilnya adalah output.

Contoh ini paling baik dilihat di GitHub. Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Mengelola pesan Amazon SQS besar menggunakan Amazon S3 dengan SDK AWS

Contoh kode berikut menunjukkan cara menggunakan Amazon SQS Extended Client Library untuk bekerja dengan pesan Amazon SQS besar.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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;
```

```
        }
    }
```

- Untuk informasi selengkapnya, silakan lihat Panduan Developer AWS SDK for Java 2.x.
- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - CreateBucket
  - PutBucketLifecycleConfiguration
  - ReceiveMessage
  - SendMessage

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menerima dan memproses notifikasi peristiwa Amazon S3 dengan menggunakan SDK AWS

Contoh kode berikut menunjukkan cara bekerja dengan pemberitahuan acara S3 dengan cara berorientasi objek.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh ini menunjukkan cara memproses acara notifikasi S3 dengan menggunakan 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 <a href="https://sdk.amazonaws.com/java/api/latest/software/amazon/
awssdk/eventnotifications/s3/model/package-summary.html">S3EventNotification
 API</a>.
    * <p>
    * To use S3 event notification serialization/deserialization to objects, add
 the following
    * dependency to your Maven pom.xml file.
    * <dependency>
    * <groupId>software.amazon.awssdk</groupId>
    * <artifactId>s3-event-notifications</artifactId>
    * <version><LATEST></version>
    * </dependency>
    * <p>
    * The S3 event notification API became available with version 2.25.11 of the
 Java SDK.
    * <p>
    * This example shows the use of the API with AWS SQS, but it can be used to
 process S3 event notifications
    * in AWS SNS or AWS Lambda as well.
    * <p>
    * Note: The S3EventNotification class does not work with messages routed
 through AWS EventBridge.
    */
   static void processS3Events(String bucketName, String queueUrl, String
 queueArn) {
       try {
           // Configure the bucket to send Object Created and Object Tagging
 notifications to an existing SQS queue.
           s3Client.putBucketNotificationConfiguration(b -> b
                   .notificationConfiguration(ncb -> ncb
                           .queueConfigurations(qcb -> qcb
                                   .events(Event.S3_OBJECT_CREATED,
 Event.S3_OBJECT_TAGGING)
                                   .queueArn(queueArn)))
                           .bucket(bucketName)
           ).join();

           triggerS3EventNotifications(bucketName);
```

```
            // Wait for event notifications to propagate.
            Thread.sleep(Duration.ofSeconds(5).toMillis());

            boolean didReceiveMessages = true;
            while (didReceiveMessages) {
                // Display the number of messages that are available in the
queue.
                sqsClient.getQueueAttributes(b -> b
                                .queueUrl(queueUrl)

.attributeNames(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)
                        ).thenAccept(attributeResponse ->
                                logger.info("Approximate number of messages in
the queue: {}",

attributeResponse.attributes().get(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)))
                        .join();

                // Receive the messages.
                ReceiveMessageResponse response = sqsClient.receiveMessage(b -> b
                        .queueUrl(queueUrl)
                ).get();
                logger.info("Count of received messages: {}",
response.messages().size());
                didReceiveMessages = !response.messages().isEmpty();

                // Create a collection to hold the received message for deletion
                // after we log the messages.
                HashSet<DeleteMessageBatchRequestEntry> messagesToDelete = new
HashSet<>();
                // Process each message.
                response.messages().forEach(message -> {
                    logger.info("Message id: {}", message.messageId());
                    // Deserialize JSON message body to a S3EventNotification
object
                    // to access messages in an object-oriented way.
                    S3EventNotification event =
S3EventNotification.fromJson(message.body());

                    // Log the S3 event notification record details.
                    if (event.getRecords() != null) {
                        event.getRecords().forEach(record -> {
                            String eventName = record.getEventName();
                            String key = record.getS3().getObject().getKey();
```

```
                            logger.info(record.toString());
                            logger.info("Event name is {} and key is {}",
eventName, key);
                        });
                    }
                    // Add logged messages to collection for batch deletion.
                    messagesToDelete.add(DeleteMessageBatchRequestEntry.builder()
                            .id(message.messageId())
                            .receiptHandle(message.receiptHandle())
                            .build());
                });
                // Delete messages.
                if (!messagesToDelete.isEmpty()) {

sqsClient.deleteMessageBatch(DeleteMessageBatchRequest.builder()
                            .queueUrl(queueUrl)
                            .entries(messagesToDelete)
                            .build()
                    ).join();
                }
            } // End of while block.
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
    }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - DeleteMessageBatch
  - GetQueueAttributes
  - PutBucketNotificationConfiguration
  - ReceiveMessage

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Mempublikasikan pesan Amazon SNS ke antrian Amazon SQS menggunakan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Buat topik (FIFO atau non-FIFO).
- Berlangganan beberapa antrian ke topik dengan opsi untuk menerapkan filter.
- Publikasikan pesan ke topik.
- Polling antrian untuk pesan yang diterima.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
 "sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
```

```
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
                    .AddTransient<SNSWrapper>()
                    .AddTransient<SQSWrapper>()
            )
            .Build();

        ServicesSetup(host);
        PrintDescription();

        await RunScenario();

    }

    /// <summary>
    /// Populate the services for use within the console application.
    /// </summary>
    /// <param name="host">The services host.</param>
    private static void ServicesSetup(IHost host)
    {
        SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
        SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
    }

    /// <summary>
    /// Run the scenario for working with topics and queues.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> RunScenario()
    {
        try
        {
            await SetupTopic();
```

```
            await SetupQueues();

            await PublishMessages();

            foreach (var queueUrl in _queueUrls)
            {
                var messages = await PollForMessages(queueUrl);
                if (messages.Any())
                {
                    await DeleteMessages(queueUrl, messages);
                }
            }
            await CleanupResources();

            Console.WriteLine("Messaging with topics and queues scenario is
complete.");
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
            await CleanupResources();
            Console.WriteLine(new string('-', 80));
            return false;
        }
    }

    /// <summary>
    /// Print a description for the tasks in the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static void PrintDescription()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Welcome to messaging with topics and queues.");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"In this scenario, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
                          $"\r\nYou can select from several options for
configuring the topic and the subscriptions for the 2 queues." +
```

```
                            $"\r\nYou can then post to the topic and see the
 results in the queues.\r\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
                            $"\r\nFIFO topics deliver messages in order and support
 deduplication and message filtering." +
                            $"\r\nYou can then post to the topic and see the
 results in the queues.\r\n");

       _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
 topics?");

       if (_useFifoTopic)
       {
           Console.WriteLine(new string('-', 80));
           _topicName = GetUserResponse("Enter a name for your SNS topic: ",
 "example-topic");
           Console.WriteLine(
               "Because you have selected a FIFO topic, '.fifo' must be appended
 to the topic name.\r\n");

           Console.WriteLine(new string('-', 80));
           Console.WriteLine($"Because you have chosen a FIFO topic,
 deduplication is supported." +
                               $"\r\nDeduplication IDs are either set in the
 message or automatically generated " +
                               $"\r\nfrom content using a hash function.\r\n" +
                               $"\r\nIf a message is successfully published to an
 SNS FIFO topic, any message " +
                               $"\r\npublished and determined to have the same
 deduplication ID, " +
                               $"\r\nwithin the five-minute deduplication
 interval, is accepted but not delivered.\r\n" +
```

```
                                $"\r\nFor more information about deduplication, " +
                                $"\r\nsee https://docs.aws.amazon.com/sns/latest/
dg/fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based
 deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName,
 _useFifoTopic, _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +
                          $"\r\nand Amazon Resource Name (ARN) {_topicArn}" +
                          $"\r\nhas been created.\r\n");

        Console.WriteLine(new string('-', 80));
        return _topicArn;
    }

    /// <summary>
    /// Set up the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task SetupQueues()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
 Service (Amazon SQS) queues to subscribe to the topic.");

        // Repeat this section for each queue.
        for (int i = 0; i < _queueCount; i++)
        {
            var queueName = GetUserResponse("Enter a name for an Amazon SQS
 queue: ", $"example-queue-{i}");
            if (_useFifoTopic)
            {
                // Only explain this once.
                if (i == 0)
                {
                    Console.WriteLine(
                        "Because you have selected a FIFO topic, '.fifo' must be
 appended to the queue name.");
                }
```

```
                var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

                _queueUrls[i] = queueUrl;

                Console.WriteLine($"Your new queue with the name {queueName}" +
                                  $"\r\nand queue URL {queueUrl}" +
                                  $"\r\nhas been created.\r\n");

                if (i == 0)
                {
                    Console.WriteLine(
                        $"The queue URL is used to retrieve the queue ARN,\r\n" +
                        $"which is used to create a subscription.");
                    Console.WriteLine(new string('-', 80));
                }

                var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

                if (i == 0)
                {
                    Console.WriteLine(
                        $"An AWS Identity and Access Management (IAM) policy must
be attached to an SQS queue, enabling it to receive\r\n" +
                        $"messages from an SNS topic");
                }

                await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

                await SetupFilters(i, queueArn, queueName);
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up filters with user options for a queue.
    /// </summary>
    /// <param name="queueCount">The number of this queue.</param>
    /// <param name="queueArn">The ARN of the queue.</param>
    /// <param name="queueName">The name of the queue.</param>
```

```
    /// <returns>Async Task.</returns>
    public static async Task SetupFilters(int queueCount, string queueArn, string
 queueName)
    {
        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            // Only explain this once.
            if (queueCount == 0)
            {
                Console.WriteLine(
                    "Subscriptions to a FIFO topic can have filters." +
                    "If you add a filter to this subscription, then only the
 filtered messages " +
                    "will be received in the queue.");

                Console.WriteLine(
                    "For information about message filtering, " +
                    "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

                Console.WriteLine(
                    "For this example, you can filter messages by a" +
                    "TONE attribute.");
            }

            var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
 subscription to the topic?");

            string? filterPolicy = null;
            if (useFilter)
            {
                filterPolicy = CreateFilterPolicy();
            }
            var subscriptionArn = await
 SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
                queueArn);
            _subscriptionArns[queueCount] = subscriptionArn;

            Console.WriteLine(
                $"The queue {queueName} has been subscribed to the topic
 {_topicName} " +
                $"with the subscription ARN {subscriptionArn}");
            Console.WriteLine(new string('-', 80));
```

```
        }
    }

    /// <summary>
    /// Use user input to create a filter policy for a subscription.
    /// </summary>
    /// <returns>The serialized filter policy.</returns>
    public static string CreateFilterPolicy()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            $"You can filter messages by one or more of the following" +
            $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" :
 "1");
            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }
```

```
    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                                  "\r\nAll messages within the same group will be
received in the order " +
                                  "they were published.");

                Console.WriteLine();
                var messageGroupId = GetUserResponse("Enter a message group ID
for this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                      "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this
message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
```

```
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"\t{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }

                var messageID = await SnsWrapper.PublishToTopicWithAttribute(
                    _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

                Console.WriteLine($"Message published with id {messageID}.");
            }

            keepSendingMessages = GetYesNoResponse("Send another message?",
false);
        }
    }

    /// <summary>
    /// Poll for the published messages to see the results of the user's choices.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task<List<Message>> PollForMessages(string queueUrl)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
                          "\r\nPress any key to continue.");
        if (UseConsole)
        {
            Console.ReadLine();
        }

        var moreMessages = true;
        var messages = new List<Message>();
```

```
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl,
10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the
queue at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                              $"\n\t{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message>
messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
                    if (deleteQueue)
                    {
                        await SqsWrapper.DeleteQueueByUrl(queueUrl);
                    }
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
```

```
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer =
 true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                           ynResponse.Equals("y",
                               StringComparison.InvariantCultureIgnoreCase);
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}
```

Buat kelas yang membungkus operasi 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;
    }
}
```

Buat kelas yang membungkus operasi 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;
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publikasikan
  - ReceiveMessage
  - SetQueueAttributes
  - Berlangganan
  - Berhenti berlangganan

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode 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 ostringstream;
            ostringstream << "Enter a name for " << (first ? "an" : "the next")
```

```
                        << " SQS queue. ";
            queueName = askQuestion(ostringstream.str());

            // 2.  Create an SQS queue.
            Aws::SQS::Model::CreateQueueRequest request;
            if (isFifoTopic) {

request.AddAttributes(Aws::SQS::Model::QueueAttributeName::FifoQueue,
                                    "true");
                queueName = queueName + FIFO_SUFFIX;

                if (first) // Only explain this once.
                {
                    std::cout
                            << "Because you are creating a FIFO SQS queue,
'.fifo' must "
                            << "be appended to the queue name." << std::endl;
                }
            }

            request.SetQueueName(queueName);
            queueNames.push_back(queueName);

            Aws::SQS::Model::CreateQueueOutcome outcome =
                    sqsClient.CreateQueue(request);

            if (outcome.IsSuccess()) {
                queueURL = outcome.GetResult().GetQueueUrl();
                std::cout << "Your new SQS queue with the name '" << queueName
                          << "' and the queue URL " << std::endl;
                std::cout << "'" << queueURL << "' has been created." <<
std::endl;
            }
            else {
                std::cerr << "Error with SQS::CreateQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);
```

```
                return false;
            }
        }
        queueURLS.push_back(queueURL);

        if (first) // Only explain this once.
        {
            std::cout
                    << "The queue URL is used to retrieve the queue ARN, which is
"
                    << "used to create a subscription." << std::endl;
        }

        Aws::String queueARN;
        {
            // 3.  Get the SQS queue ARN attribute.
            Aws::SQS::Model::GetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);

request.AddAttributeNames(Aws::SQS::Model::QueueAttributeName::QueueArn);

            Aws::SQS::Model::GetQueueAttributesOutcome outcome =
                    sqsClient.GetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                const Aws::Map<Aws::SQS::Model::QueueAttributeName, Aws::String>
&attributes =
                        outcome.GetResult().GetAttributes();
                const auto &iter = attributes.find(
                        Aws::SQS::Model::QueueAttributeName::QueueArn);
                if (iter != attributes.end()) {
                    queueARN = iter->second;
                    std::cout << "The queue ARN '" << queueARN
                              << "' has been retrieved."
                              << std::endl;
                }
                else {
                    std::cerr
                            << "Error ARN attribute not returned by
GetQueueAttribute."
                            << std::endl;

                    cleanUp(topicARN,
                            queueURLS,
```

```
                            subscriptionARNS,
                            snsClient,
                            sqsClient);

                    return false;
                }
            }
            else {
                std::cerr << "Error with SQS::GetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        if (first) {
            std::cout
                    << "An IAM policy must be attached to an SQS queue, enabling
it to receive "
                       "messages from an SNS topic." << std::endl;
        }

        {
            // 4.  Set the SQS queue policy attribute with a policy enabling the
receipt of SNS messages.
            Aws::SQS::Model::SetQueueAttributesRequest request;
            request.SetQueueUrl(queueURL);
            Aws::String policy = createPolicyForQueue(queueARN, topicARN);
            request.AddAttributes(Aws::SQS::Model::QueueAttributeName::Policy,
                                  policy);

            Aws::SQS::Model::SetQueueAttributesOutcome outcome =
                    sqsClient.SetQueueAttributes(request);

            if (outcome.IsSuccess()) {
                std::cout << "The attributes for the queue '" << queueName
                          << "' were successfully updated." << std::endl;
```

```
            }
            else {
                std::cerr << "Error with SQS::SetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        printAsterisksLine();

        {
            // 5.  Subscribe the SQS queue to the SNS topic.
            Aws::SNS::Model::SubscribeRequest request;
            request.SetTopicArn(topicARN);
            request.SetProtocol("sqs");
            request.SetEndpoint(queueARN);
            if (isFifoTopic) {
                if (first) {
                    std::cout << "Subscriptions to a FIFO topic can have
 filters."
                              << std::endl;
                    std::cout
                            << "If you add a filter to this subscription, then
 only the filtered messages "
                            << "will be received in the queue." << std::endl;
                    std::cout << "For information about message filtering, "
                              << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
                              << std::endl;
                    std::cout << "For this example, you can filter messages by a
 \""
                              << TONE_ATTRIBUTE << "\" attribute." << std::endl;
                }

                std::ostringstream ostringstream;
                ostringstream << "Filter messages for \"" << queueName
```

```
                            << "\"'s subscription to the topic \""
                            << topicName << "\"?  (y/n)";

                // Add filter if user answers yes.
                if (askYesNoQuestion(ostringstream.str())) {
                    Aws::String jsonPolicy = getFilterPolicyFromUser();
                    if (!jsonPolicy.empty()) {
                        filteringMessages = true;

                        std::cout << "This is the filter policy for this
subscription."
                                  << std::endl;
                        std::cout << jsonPolicy << std::endl;

                        request.AddAttributes("FilterPolicy", jsonPolicy);
                    }
                    else {
                        std::cout
                                << "Because you did not select any attributes, no
filter "
                                << "will be added to this subscription." <<
std::endl;
                    }
                }
            }  // if (isFifoTopic)
            Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

            if (outcome.IsSuccess()) {
                Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
                std::cout << "The queue '" << queueName
                          << "' has been subscribed to the topic '"
                          << "'" << topicName << "'" << std::endl;
                std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
                          << std::endl;
                subscriptionARNS.push_back(subscriptionARN);
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Subscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
```

```
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }

        first = false;
    }

    first = true;
    do {
        printAsterisksLine();

        // 6.  Publish a message to the SNS topic.
        Aws::SNS::Model::PublishRequest request;
        request.SetTopicArn(topicARN);
        Aws::String message = askQuestion("Enter a message text to publish.  ");
        request.SetMessage(message);
        if (isFifoTopic) {
            if (first) {
                std::cout
                        << "Because you are using a FIFO topic, you must set a
message group ID."
                        << std::endl;
                std::cout
                        << "All messages within the same group will be received
in the "
                        << "order they were published." << std::endl;
            }
            Aws::String messageGroupID = askQuestion(
                    "Enter a message group ID for this message. ");
            request.SetMessageGroupId(messageGroupID);
            if (!contentBasedDeduplication) {
                if (first) {
                    std::cout
                            << "Because you are not using content-based
deduplication, "
                            << "you must enter a deduplication ID." << std::endl;
                }
                Aws::String deduplicationID = askQuestion(
```

```
                    "Enter a deduplication ID for this message. ");
            request.SetMessageDeduplicationId(deduplicationID);
        }
    }

    if (filteringMessages && askYesNoQuestion(
            "Add an attribute to this message? (y/n) ")) {
        for (size_t i = 0; i < TONES.size(); ++i) {
            std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
        }
        int selection = askQuestionForIntRange(
                "Enter a number for an attribute. ",
                1, static_cast<int>(TONES.size()));
        Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
        messageAttributeValue.SetDataType("String");
        messageAttributeValue.SetStringValue(TONES[selection - 1]);
        request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
    }

    Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Your message was successfully published." << std::endl;
    }
    else {
        std::cerr << "Error with TopicsAndQueues::Publish. "
                  << outcome.GetError().GetMessage()
                  << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }

    first = false;
} while (askYesNoQuestion("Post another message? (y/n) "));

printAsterisksLine();

std::cout << "Now the SQS queue will be polled to retrieve the messages."
```

```
              << std::endl;
    askQuestion("Press any key to continue...", alwaysTrueTest);

    for (size_t i = 0; i < queueURLS.size(); ++i) {
        // 7.  Poll an SQS queue for its messages.
        std::vector<Aws::String> messages;
        std::vector<Aws::String> receiptHandles;
        while (true) {
            Aws::SQS::Model::ReceiveMessageRequest request;
            request.SetMaxNumberOfMessages(10);
            request.SetQueueUrl(queueURLS[i]);

            // Setting WaitTimeSeconds to non-zero enables long polling.
            // For information about long polling, see
            // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
            request.SetWaitTimeSeconds(1);
            Aws::SQS::Model::ReceiveMessageOutcome outcome =
                    sqsClient.ReceiveMessage(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::SQS::Model::Message> &newMessages =
 outcome.GetResult().GetMessages();
                if (newMessages.empty()) {
                    break;
                }
                else {
                    for (const Aws::SQS::Model::Message &message: newMessages) {
                        messages.push_back(message.GetBody());
                        receiptHandles.push_back(message.GetReceiptHandle());
                    }
                }
            }
            else {
                std::cerr << "Error with SQS::ReceiveMessage. "
                          << outcome.GetError().GetMessage()
                          << std::endl;

                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);
```

```
            return false;
        }
    }

    printAsterisksLine();

    if (messages.empty()) {
        std::cout << "No messages were ";
    }
    else if (messages.size() == 1) {
        std::cout << "One message was ";
    }
    else {
        std::cout << messages.size() << " messages were ";
    }
    std::cout << "received by the queue '" << queueNames[i]
              << "'." << std::endl;
    for (const Aws::String &message: messages) {
        std::cout << "  Message : '" << message << "'."
                  << std::endl;
    }

    // 8.  Delete a batch of messages from an SQS queue.
    if (!receiptHandles.empty()) {
        Aws::SQS::Model::DeleteMessageBatchRequest request;
        request.SetQueueUrl(queueURLS[i]);
        int id = 1; // Ids must be unique within a batch delete request.
        for (const Aws::String &receiptHandle: receiptHandles) {
            Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
            entry.SetId(std::to_string(id));
            ++id;
            entry.SetReceiptHandle(receiptHandle);
            request.AddEntries(entry);
        }

        Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
                sqsClient.DeleteMessageBatch(request);

        if (outcome.IsSuccess()) {
            std::cout << "The batch deletion of messages was successful."
                      << std::endl;
        }
        else {
            std::cerr << "Error with SQS::DeleteMessageBatch. "
```

```
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUp(topicARN,
                        queueURLS,
                        subscriptionARNS,
                        snsClient,
                        sqsClient);

                return false;
            }
        }
    }

    return cleanUp(topicARN,
                   queueURLS,
                   subscriptionARNS,
                   snsClient,
                   sqsClient,
                   true); // askUser
}


bool AwsDoc::TopicsAndQueues::cleanUp(const Aws::String &topicARN,
                                      const Aws::Vector<Aws::String> &queueURLS,
                                      const Aws::Vector<Aws::String>
 &subscriptionARNS,
                                      const Aws::SNS::SNSClient &snsClient,
                                      const Aws::SQS::SQSClient &sqsClient,
                                      bool askUser) {
    bool result = true;
    printAsterisksLine();
    if (!queueURLS.empty() && askUser &&
        askYesNoQuestion("Delete the SQS queues? (y/n) ")) {

        for (const auto &queueURL: queueURLS) {
            // 9.  Delete an SQS queue.
            Aws::SQS::Model::DeleteQueueRequest request;
            request.SetQueueUrl(queueURL);

            Aws::SQS::Model::DeleteQueueOutcome outcome =
                    sqsClient.DeleteQueue(request);

            if (outcome.IsSuccess()) {
                std::cout << "The queue with URL '" << queueURL
```

```
                        << "' was successfully deleted." << std::endl;
            }
            else {
                std::cerr << "Error with SQS::DeleteQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }

        for (const auto &subscriptionARN: subscriptionARNS) {
            // 10. Unsubscribe an SNS subscription.
            Aws::SNS::Model::UnsubscribeRequest request;
            request.SetSubscriptionArn(subscriptionARN);

            Aws::SNS::Model::UnsubscribeOutcome outcome =
                    snsClient.Unsubscribe(request);

            if (outcome.IsSuccess()) {
                std::cout << "Unsubscribe of subscription ARN '" <<
subscriptionARN
                          << "' was successful." << std::endl;
            }
            else {
                std::cerr << "Error with TopicsAndQueues::Unsubscribe. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    printAsterisksLine();
    if (!topicARN.empty() && askUser &&
        askYesNoQuestion("Delete the SNS topic? (y/n) ")) {

        // 11. Delete an SNS topic.
        Aws::SNS::Model::DeleteTopicRequest request;
        request.SetTopicArn(topicARN);

        Aws::SNS::Model::DeleteTopicOutcome outcome =
snsClient.DeleteTopic(request);

        if (outcome.IsSuccess()) {
```

```
            std::cout << "The topic with ARN '" << topicARN
                      << "' was successfully deleted." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::DeleteTopicRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}

//! Create an IAM policy that gives an SQS queue permission to receive messages
 from an SNS topic.
/*!
 \sa createPolicyForQueue()
 \param queueARN: The SQS queue Amazon Resource Name (ARN).
 \param topicARN: The SNS topic ARN.
 \return Aws::String: The policy as JSON.
 */
Aws::String AwsDoc::TopicsAndQueues::createPolicyForQueue(const Aws::String
 &queueARN,
                                                          const Aws::String
 &topicARN) {
    std::ostringstream policyStream;
    policyStream << R"({
        "Statement": [
        {
            "Effect": "Allow",
                    "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "sqs:SendMessage",
                    "Resource": ")" << queueARN << R"(",
                    "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": ")" << topicARN << R"("
                }
            }
        }
        ]
    })";
```

```
    return policyStream.str();
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk C++ .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publikasikan
  - ReceiveMessage
  - SetQueueAttributes
  - Berlangganan
  - Berhenti berlangganan

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
 "strings"
```

```
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions
 so that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string,
 bool, bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or
 standard.\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.
\n" +
   "Deduplication IDs are either set in the message or are automatically
 generated\n" +
```

```
   "from content using a hash function. If a message is successfully published to
\n" +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted
\n" +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to
\n"+
   "the topic name.", FIFO_SUFFIX)
 }

 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN)
 \n"+
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS
 queue. ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
  }
 }
```

```
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string,
 topicArn string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
 if err != nil {
  panic(err)
 }
 log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
  "messages to it.")
 log.Println(strings.Repeat("-", 88))

 var filterPolicy map[string][]string
 if isFifoTopic {
  if ordinal == "first" {
   log.Println("Subscriptions to a FIFO topic can have filters.\n" +
    "If you add a filter to this subscription, then only the filtered messages\n"
 +
    "will be received in the queue.\n" +
    "For information about message filtering, see\n" +
    "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
    "For this example, you can filter messages by a \"tone\" attribute.")
  }

  wantFiltering := runner.questioner.AskBool(
   fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
    "from the %v topic? (y/n) ", queueName, topicName), "y")
```

```
  if wantFiltering {
   log.Println("You can filter messages by one or more of the following \"tone\"
 attributes.")

   var toneSelections []string
   askAboutTones := true
   for askAboutTones {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelections = append(toneSelections, ToneChoices[toneIndex])
    askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
 the filter? (y/n) ", "y")
   }
   log.Printf("Your subscription will be filtered to only pass the following
 tones: %v\n", toneSelections)
   filterPolicy = map[string][]string{TONE_KEY: toneSelections}
  }
 }

 subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
 filterPolicy)
 if err != nil {
  panic(err)
 }
 log.Printf("The queue %v is now subscribed to the topic %v with the subscription
 ARN %v.\n",
  queueName, topicName, subscriptionArn)

 return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn
 string, isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
 var message string
 var groupId string
 var dedupId string
 var toneSelection string
 publishMore := true
 for publishMore {
  groupId = ""
  dedupId = ""
  toneSelection = ""
  message = runner.questioner.Ask("Enter a message to publish: ")
  if isFifoTopic {
```

```
    log.Println("Because you are using a FIFO topic, you must set a message group
 ID.\n" +
     "All messages within the same group will be received in the order they were
 published.")
    groupId = runner.questioner.Ask("Enter a message group ID: ")
    if !contentBasedDeduplication {
     log.Println("Because you are not using content-based deduplication,\n" +
      "you must enter a deduplication ID.")
     dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
    }
   }
   if usingFilters {
    if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
     toneIndex := runner.questioner.AskChoice(
      "Enter the number of the tone you want to filter by:\n", ToneChoices)
     toneSelection = ToneChoices[toneIndex]
    }
   }

   err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId,
 TONE_KEY, toneSelection)
   if err != nil {
    panic(err)
   }
   log.Println(("Your message was published."))

   publishMore = runner.questioner.AskBool("Do you want to publish another
 messsage? (y/n) ", "y")
  }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
    break
```

```
    }
    messages = append(messages, currentMsgs...)
   }
   if len(messages) == 0 {
    log.Printf("No messages were received by queue %v.\n", queueUrl)
   } else if len(messages) == 1 {
    log.Printf("One message was received by queue %v:\n", queueUrl)

   } else {
    log.Printf("%v messages were received by queue %v:\n", len(messages),
  queueUrl)
    }
   for msgIndex, message := range messages {
    messageBody := MessageBody{}
    err := json.Unmarshal([]byte(*message.Body), &messageBody)
    if err != nil {
     panic(err)
    }
    log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
   }

   if len(messages) > 0 {
    log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
    err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
    if err != nil {
     panic(err)
    }
   }
  }
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
  the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
```

```
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to
 the\n"+
  "topic. You can select from several options for configuring the topic and the
\n"+
  "subscriptions for the queues. You can then post to the topic and see the
 results\n"+
  "in the queues.\n", queueCount)

 log.Println(strings.Repeat("-", 88))

 runner := ScenarioRunner{
  questioner: questioner,
  snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
  sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
 }
 resources.snsActor = runner.snsActor
 resources.sqsActor = runner.sqsActor

 topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
 runner.CreateTopic(ctx)
 resources.topicArn = topicArn
 log.Println(strings.Repeat("-", 88))

 log.Printf("Now you will create %v SQS queues and subscribe them to the topic.
\n", queueCount)
 ordinals := []string{"first", "next"}
 usingFilters := false
```

```
 for _, ordinal := range ordinals {
  queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl,
 topicName, topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }

 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(ctx, resources.queueUrls)

 log.Println(strings.Repeat("-", 88))

 wantCleanup := questioner.AskBool("Do you want to remove all AWS resources
 created for this scenario? (y/n) ", "y")
 if wantCleanup {
  log.Println("Cleaning up resources...")
  resources.Cleanup(ctx)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Tentukan struct yang membungkus tindakan Amazon SNS yang digunakan dalam contoh ini.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
```

```
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
 an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent
 to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered
 according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string)
 error {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
 err)
 }
 return err
}
```

Tentukan struct yang membungkus tindakan Amazon SQS yang digunakan dalam contoh ini.

```
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
}
```

Pembersihan sumber daya

```
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)
  }
 }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Go .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publikasikan
  - ReceiveMessage
  - SetQueueAttributes
  - Berlangganan
  - Berhenti berlangganan

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode 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> queueAtts = response.attributesAsStrings();
       for (Map.Entry<String, String> queueAtt : queueAtts.entrySet())
           return queueAtt.getValue();

       return "";
   }

   public static String createQueue(SqsClient sqsClient, String queueName,
Boolean selectFIFO) {
```

```
        try {
            System.out.println("\nCreate Queue");
            if (selectFIFO) {
                Map<QueueAttributeName, String> attrs = new HashMap<>();
                attrs.put(QueueAttributeName.FIFO_QUEUE, "true");
                CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .attributes(attrs)
                    .build();

                sqsClient.createQueue(createQueueRequest);
                System.out.println("\nGet queue url");
                GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
                return getQueueUrlResponse.queueUrl();
            } else {
                CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();

                sqsClient.createQueue(createQueueRequest);
                System.out.println("\nGet queue url");
                GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
                return getQueueUrlResponse.queueUrl();
            }

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                .name(topicName)
                .build();
```

```
            result = snsClient.createTopic(request);
            return result.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }

    public static String createFIFO(SnsClient snsClient, String topicName, String
 duplication) {
        try {
            // Create a FIFO topic by using the SNS service client.
            Map<String, String> topicAttributes = new HashMap<>();
            if (duplication.compareTo("n") == 0) {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "false");
            } else {
                topicAttributes.put("FifoTopic", "true");
                topicAttributes.put("ContentBasedDeduplication", "true");
            }

            CreateTopicRequest topicRequest = CreateTopicRequest.builder()
                .name(topicName)
                .attributes(topicAttributes)
                .build();

            CreateTopicResponse response = snsClient.createTopic(topicRequest);
            return response.topicArn();

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - CreateQueue

- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publikasikan
- ReceiveMessage
- SetQueueAttributes
- Berlangganan
- Berhenti berlangganan

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Ini adalah titik masuk untuk skenario ini.

```
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();
```

```
};
```

Kode sebelumnya menyediakan dependensi yang diperlukan dan memulai skenario. Bagian selanjutnya berisi sebagian besar contoh.

```
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();
    }
  }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publikasikan
  - ReceiveMessage
  - SetQueueAttributes
  - Berlangganan
  - Berhenti berlangganan

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode 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 jsonObject = gson.fromJson(jsonString, JsonObject::class.java)
            val toneArray = jsonObject.getAsJsonArray("tone")
            for (value: String? in filterList) {
                toneArray.add(JsonPrimitive(value))
            }

            val updatedJsonString: String = gson.toJson(jsonObject)
            println(updatedJsonString)
            val attRequest = SetSubscriptionAttributesRequest {
                subscriptionArn = result.subscriptionArn
                attributeName = attributeNameVal
                attributeValue = updatedJsonString
            }

            snsClient.setSubscriptionAttributes(attRequest)
            return result.subscriptionArn
        }
    }
}

suspend fun setQueueAttr(queueUrlVal: String?, policy: String) {
    val attrMap: MutableMap<String, String> = HashMap()
    attrMap[QueueAttributeName.Policy.toString()] = policy

    val attributesRequest = SetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributes = attrMap
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.setQueueAttributes(attributesRequest)
        println("The policy has been successfully attached.")
    }
}

suspend fun getSQSQueueAttrs(queueUrlVal: String?): String {
    val atts: MutableList<QueueAttributeName> = ArrayList()
    atts.add(QueueAttributeName.QueueArn)
```

```
    val attributesRequest = GetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributeNames = atts
    }
    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.getQueueAttributes(attributesRequest)
        val mapAtts = response.attributes
        if (mapAtts != null) {
            mapAtts.forEach { entry ->
                println("${entry.key} : ${entry.value}")
                return entry.value
            }
        }
    }
    return ""
}

suspend fun createQueue(queueNameVal: String?, selectFIFO: Boolean): String? {
    println("\nCreate Queue")
    if (selectFIFO) {
        val attrs = mutableMapOf<String, String>()
        attrs[QueueAttributeName.FifoQueue.toString()] = "true"

        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
            attributes = attrs
        }

        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("\nGet queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    } else {
        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
        }
```

```
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("Get queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    }
}

suspend fun createSNSTopic(topicName: String?): String? {
    val request = CreateTopicRequest {
        name = topicName
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn
    }
}

suspend fun createFIFO(topicName: String?, duplication: String): String? {
    val topicAttributes: MutableMap<String, String> = HashMap()
    if (duplication.compareTo("n") == 0) {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "false"
    } else {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "true"
    }

    val topicRequest = CreateTopicRequest {
        name = topicName
        attributes = topicAttributes
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.createTopic(topicRequest)
        return response.topicArn
    }
```

```
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK For Kotlin.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publikasikan
  - ReceiveMessage
  - SetQueueAttributes
  - Berlangganan
  - Berhenti berlangganan

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
class TopicsAndQueuesScenario:
    """Manages the Topics and Queues feature scenario."""

    DASHES = "-" * 80

    def __init__(self, sns_wrapper: SnsWrapper, sqs_wrapper: SqsWrapper) -> None:
        """
        Initialize the Topics and Queues scenario.
```

```
        :param sns_wrapper: SnsWrapper instance for SNS operations.
        :param sqs_wrapper: SqsWrapper instance for SQS operations.
        """
        self.sns_wrapper = sns_wrapper
        self.sqs_wrapper = sqs_wrapper

        # Scenario state
        self.use_fifo_topic = False
        self.use_content_based_deduplication = False
        self.topic_name = None
        self.topic_arn = None
        self.queue_count = 2
        self.queue_urls = []
        self.subscription_arns = []
        self.tones = ["cheerful", "funny", "serious", "sincere"]

    def run_scenario(self) -> None:
        """Run the Topics and Queues feature scenario."""
        print(self.DASHES)
        print("Welcome to messaging with topics and queues.")
        print(self.DASHES)
        print(f"""
    In this scenario, you will create an SNS topic and subscribe
{self.queue_count} SQS queues to the topic.
    You can select from several options for configuring the topic and the
subscriptions for the queues.
    You can then post to the topic and see the results in the queues.
        """)

        try:
            # Setup Phase
            print(self.DASHES)
            self._setup_topic()
            print(self.DASHES)

            self._setup_queues()
            print(self.DASHES)

            # Demonstration Phase
            self._publish_messages()
            print(self.DASHES)

            # Examination Phase
            self._poll_queues_for_messages()
```

```
            print(self.DASHES)

            # Cleanup Phase
            self._cleanup_resources()
            print(self.DASHES)

        except Exception as e:
            logger.error(f"Scenario failed: {e}")
            print(f"There was a problem with the scenario: {e}")
            print("\nInitiating cleanup...")
            try:
                self._cleanup_resources()
            except Exception as cleanup_error:
                logger.error(f"Error during cleanup: {cleanup_error}")

        print("Messaging with topics and queues scenario is complete.")
        print(self.DASHES)

    def _setup_topic(self) -> None:
        """Set up the SNS topic to be used with the queues."""
        print("SNS topics can be configured as FIFO (First-In-First-Out).")
        print("FIFO topics deliver messages in order and support deduplication
 and message filtering.")
        print()

        self.use_fifo_topic = q.ask("Would you like to work with FIFO topics? (y/
n): ", q.is_yesno)

        if self.use_fifo_topic:
            print(self.DASHES)
            self.topic_name = q.ask("Enter a name for your SNS topic: ",
 q.non_empty)
            print("Because you have selected a FIFO topic, '.fifo' must be
 appended to the topic name.")
            print()

            print(self.DASHES)
            print("""
    Because you have chosen a FIFO topic, deduplication is supported.
    Deduplication IDs are either set in the message or automatically generated
    from content using a hash function.

    If a message is successfully published to an SNS FIFO topic, any message
    published and determined to have the same deduplication ID,
```

```
   within the five-minute deduplication interval, is accepted but not delivered.

   For more information about deduplication,
   see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.
           """)

           self.use_content_based_deduplication = q.ask(
               "Use content-based deduplication instead of entering a
deduplication ID? (y/n): ",
               q.is_yesno
           )
       else:
           self.topic_name = q.ask("Enter a name for your SNS topic: ",
q.non_empty)

       print(self.DASHES)

       # Create the topic
       self.topic_arn = self.sns_wrapper.create_topic(
           self.topic_name,
           self.use_fifo_topic,
           self.use_content_based_deduplication
       )

       print(f"Your new topic with the name {self.topic_name}")
       print(f"  and Amazon Resource Name (ARN) {self.topic_arn}")
       print(f"  has been created.")
       print()

   def _setup_queues(self) -> None:
       """Set up the SQS queues and subscribe them to the topic."""
       print(f"Now you will create {self.queue_count} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.")

       for i in range(self.queue_count):
           queue_name = q.ask(f"Enter a name for SQS queue #{i+1}: ",
q.non_empty)

           if self.use_fifo_topic and i == 0:
               print("Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.")

           # Create the queue
```

```
            queue_url = self.sqs_wrapper.create_queue(queue_name,
self.use_fifo_topic)
            self.queue_urls.append(queue_url)

            print(f"Your new queue with the name {queue_name}")
            print(f"  and queue URL {queue_url}")
            print(f"  has been created.")
            print()

            if i == 0:
                print("The queue URL is used to retrieve the queue ARN,")
                print("which is used to create a subscription.")
                print(self.DASHES)

            # Get queue ARN
            queue_arn = self.sqs_wrapper.get_queue_arn(queue_url)

            if i == 0:
                print("An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue,")
                print("enabling it to receive messages from an SNS topic.")

            # Set queue policy to allow SNS to send messages
            self.sqs_wrapper.set_queue_policy_for_topic(queue_arn,
self.topic_arn, queue_url)

            # Set up message filtering if using FIFO
            subscription_arn = self._setup_subscription_with_filter(i, queue_arn,
queue_name)
            self.subscription_arns.append(subscription_arn)

   def _setup_subscription_with_filter(self, queue_index: int, queue_arn: str,
queue_name: str) -> str:
       """Set up subscription with optional message filtering."""
       filter_policy = None

       if self.use_fifo_topic:
           print(self.DASHES)
           if queue_index == 0:
               print("Subscriptions to a FIFO topic can have filters.")
               print("If you add a filter to this subscription, then only the
filtered messages")
               print("will be received in the queue.")
               print()
```

```
                print("For information about message filtering,")
                print("see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html")
                print()
                print("For this example, you can filter messages by a TONE
 attribute.")

            use_filter = q.ask(f"Filter messages for {queue_name}'s subscription
 to the topic? (y/n): ", q.is_yesno)

            if use_filter:
                filter_policy = self._create_filter_policy()

        subscription_arn = self.sns_wrapper.subscribe_queue_to_topic(
            self.topic_arn, queue_arn, filter_policy
        )

        print(f"The queue {queue_name} has been subscribed to the topic
 {self.topic_name}")
        print(f"  with the subscription ARN {subscription_arn}")

        return subscription_arn

    def _create_filter_policy(self) -> str:
        """Create a message filter policy based on user selections."""
        print(self.DASHES)
        print("You can filter messages by one or more of the following TONE
 attributes.")

        filter_selections = []
        selection_number = 0

        while True:
            print("Enter a number to add a TONE filter, or enter 0 to stop adding
 filters.")
            for i, tone in enumerate(self.tones, 1):
                print(f"  {i}. {tone}")

            selection = q.ask("Your choice: ", q.is_int, q.in_range(0,
 len(self.tones)))

            if selection == 0:
                break
```

```
            elif selection > 0 and self.tones[selection - 1] not in
filter_selections:
                filter_selections.append(self.tones[selection - 1])
                print(f"Added '{self.tones[selection - 1]}' to filter list.")

        if filter_selections:
            filters = {"tone": filter_selections}
            return json.dumps(filters)
        return None

    def _publish_messages(self) -> None:
        """Publish messages to the topic with various options."""
        print("Now we can publish messages.")

        keep_sending = True
        while keep_sending:
            print()
            message = q.ask("Enter a message to publish: ", q.non_empty)

            message_group_id = None
            deduplication_id = None
            tone_attribute = None

            if self.use_fifo_topic:
                print("Because you are using a FIFO topic, you must set a message
group ID.")
                print("All messages within the same group will be received in the
order they were published.")
                print()
                message_group_id = q.ask("Enter a message group ID for this
message: ", q.non_empty)

                if not self.use_content_based_deduplication:
                    print("Because you are not using content-based
deduplication,")
                    print("you must enter a deduplication ID.")
                    deduplication_id = q.ask("Enter a deduplication ID for this
message: ", q.non_empty)

                # Ask about tone attribute
                add_attribute = q.ask("Add an attribute to this message? (y/n):
", q.is_yesno)
                if add_attribute:
                    print("Enter a number for an attribute:")
```

```
                    for i, tone in enumerate(self.tones, 1):
                        print(f"  {i}. {tone}")

                    selection = q.ask("Your choice: ", q.is_int, q.in_range(1,
 len(self.tones)))
                    if 1 <= selection <= len(self.tones):
                        tone_attribute = self.tones[selection - 1]

            # Publish the message
            message_id = self.sns_wrapper.publish_message(
                self.topic_arn,
                message,
                tone_attribute,
                deduplication_id,
                message_group_id
            )

            print(f"Message published with ID: {message_id}")

            keep_sending = q.ask("Send another message? (y/n): ", q.is_yesno)

    def _poll_queues_for_messages(self) -> None:
        """Poll all queues for messages and display results."""
        for i, queue_url in enumerate(self.queue_urls):
            print(f"Polling queue #{i+1} at {queue_url} for messages...")

            q.ask("Press Enter to continue...")

            messages = self._poll_queue_for_messages(queue_url)

            if messages:
                print(f"{len(messages)} message(s) were received by queue #{i
+1}")
                for j, message in enumerate(messages, 1):
                    print(f"  Message {j}:")
                    # Parse the SNS message body to get the actual message
                    try:
                        sns_message = json.loads(message['Body'])
                        actual_message = sns_message.get('Message',
 message['Body'])
                        print(f"    {actual_message}")
                    except (json.JSONDecodeError, KeyError):
                        print(f"    {message['Body']}")
```

```
                # Delete the messages
                self.sqs_wrapper.delete_messages(queue_url, messages)
                print(f"Messages deleted from queue #{i+1}")
            else:
                print(f"No messages received by queue #{i+1}")

            print(self.DASHES)

    def _poll_queue_for_messages(self, queue_url: str) -> List[Dict[str, Any]]:
        """Poll a single queue for messages."""
        all_messages = []
        max_polls = 3  # Limit polling to avoid infinite loops

        for poll_count in range(max_polls):
            messages = self.sqs_wrapper.receive_messages(queue_url, 10)

            if messages:
                all_messages.extend(messages)
                print(f"  Received {len(messages)} messages in poll {poll_count +
1}")
                # Small delay between polls
                time.sleep(1)
            else:
                print(f"  No messages in poll {poll_count + 1}")
                break

        return all_messages

    def _cleanup_resources(self) -> None:
        """Clean up all resources created during the scenario."""
        print("Cleaning up resources...")

        # Delete queues
        for i, queue_url in enumerate(self.queue_urls):
            if queue_url:
                delete_queue = q.ask(f"Delete queue #{i+1} with URL {queue_url}?
(y/n): ", q.is_yesno)
                if delete_queue:
                    try:
                        self.sqs_wrapper.delete_queue(queue_url)
                        print(f"Deleted queue #{i+1}")
                    except Exception as e:
                        print(f"Error deleting queue #{i+1}: {e}")
```

```
        # Unsubscribe from topic
        for i, subscription_arn in enumerate(self.subscription_arns):
            if subscription_arn:
                try:
                    self.sns_wrapper.unsubscribe(subscription_arn)
                    print(f"Unsubscribed subscription #{i+1}")
                except Exception as e:
                    print(f"Error unsubscribing #{i+1}: {e}")

        # Delete topic
        if self.topic_arn:
            delete_topic = q.ask(f"Delete topic {self.topic_name}? (y/n): ",
 q.is_yesno)
            if delete_topic:
                try:
                    self.sns_wrapper.delete_topic(self.topic_arn)
                    print(f"Deleted topic {self.topic_name}")
                except Exception as e:
                    print(f"Error deleting topic: {e}")

        print("Resource cleanup complete.")
```

Buat kelas yang membungkus operasi Amazon SNS dan Amazon SQS untuk digunakan dalam skenario.

```
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
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publikasikan
  - ReceiveMessage
  - SetQueueAttributes
  - Berlangganan

- Berhenti berlangganan

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode 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)
            }
        }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Swift.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publikasikan
  - ReceiveMessage
  - SetQueueAttributes
  - Berlangganan
  - Berhenti berlangganan

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Mengirim dan menerima kumpulan pesan dengan Amazon SQS menggunakan SDK AWS

Contoh-contoh kode berikut menunjukkan cara:

- Membuat antrean Amazon SQS.
- Kirim batch pesan ke antrian.
- Menerima kumpulan pesan dari antrian.
- Hapus kumpulan pesan dari antrian.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Seperti yang ditunjukkan dalam contoh berikut, Anda dapat menangani operasi pesan batch dengan Amazon SQS menggunakan dua pendekatan berbeda dengan: AWS SDK for Java 2.x

SendRecvBatch.java menggunakan operasi batch eksplisit. Anda secara manual membuat batch pesan dan menelepon `sendMessageBatch()` dan `deleteMessageBatch()` langsung. Anda juga menangani tanggapan batch, termasuk pesan yang gagal. Pendekatan ini memberi Anda kontrol penuh atas ukuran batch dan penanganan kesalahan. Namun, diperlukan lebih banyak kode untuk mengelola logika batching.

SimpleProducerConsumer.java menggunakan `SqsAsyncBatchManager` pustaka tingkat tinggi untuk batching permintaan otomatis. Anda membuat individu `sendMessage()` dan `deleteMessage()` panggilan dengan tanda tangan metode yang sama dengan klien standar. SDK secara otomatis menyangga panggilan ini dan mengirimkannya sebagai operasi batch. Pendekatan ini membutuhkan perubahan kode minimal sambil memberikan manfaat kinerja batching.

Gunakan batching eksplisit saat Anda membutuhkan kontrol halus atas komposisi batch dan penanganan kesalahan. Gunakan batching otomatis saat Anda ingin mengoptimalkan kinerja dengan perubahan kode minimal.

SendRecvBatch.java - Menggunakan operasi batch eksplisit dengan pesan.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.BatchResultErrorEntry;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchResultEntry;
```

```
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.MessageAttributeValue;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResultEntry;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

/**
 * This code demonstrates basic message operations in Amazon Simple Queue Service
 (Amazon SQS).
 */

public class SendRecvBatch {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SendRecvBatch.class);
    private static final SqsClient sqsClient = SqsClient.create();


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

```
   /**
    * Send a batch of messages in a single request to an SQS queue.
    * This request may return overall success even when some messages were not
sent.
    * The caller must inspect the Successful and Failed lists in the response
and
    * resend any failed messages.
    *
    * @param queueUrl  The URL of the queue to receive the messages.
    * @param messages  The messages to send to the queue. Each message contains
a body and attributes.
    * @return The response from SQS that contains the list of successful and
failed messages.
    */
   public static SendMessageBatchResponse sendMessages(
           String queueUrl, List<MessageEntry> messages) {

       try {
           List<SendMessageBatchRequestEntry> entries = new ArrayList<>();

           for (int i = 0; i < messages.size(); i++) {
               MessageEntry message = messages.get(i);
               entries.add(SendMessageBatchRequestEntry.builder()
                       .id(String.valueOf(i))
                       .messageBody(message.getBody())
                       .messageAttributes(message.getAttributes())
                       .build());
           }

           SendMessageBatchRequest sendBatchRequest =
SendMessageBatchRequest.builder()
                   .queueUrl(queueUrl)
                   .entries(entries)
                   .build();

           SendMessageBatchResponse response =
sqsClient.sendMessageBatch(sendBatchRequest);

           if (!response.successful().isEmpty()) {
               for (SendMessageBatchResultEntry resultEntry :
response.successful()) {
                   LOGGER.info("Message sent: {}: {}", resultEntry.messageId(),

messages.get(Integer.parseInt(resultEntry.id())).getBody());
```

```
                }
            }

            if (!response.failed().isEmpty()) {
                for (BatchResultErrorEntry errorEntry : response.failed()) {
                    LOGGER.warn("Failed to send: {}: {}", errorEntry.id(),

messages.get(Integer.parseInt(errorEntry.id())).getBody());
                }
            }

            return response;

        } catch (SqsException e) {
            LOGGER.error("Send messages failed to queue: {}", queueUrl, e);
            throw e;
        }
    }

    /**
     * Receive a batch of messages in a single request from an SQS queue.
     *
     * @param queueUrl   The URL of the queue from which to receive messages.
     * @param maxNumber  The maximum number of messages to receive (capped at 10
by SQS).
     *                   The actual number of messages received might be less.
     * @param waitTime   The maximum time to wait (in seconds) before returning.
When
     *                   this number is greater than zero, long polling is used.
This
     *                   can result in reduced costs and fewer false empty
responses.
     * @return The list of Message objects received. These each contain the body
     *         of the message and metadata and custom attributes.
     */
    public static List<Message> receiveMessages(String queueUrl, int maxNumber,
int waitTime) {
        try {
            ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(maxNumber)
                    .waitTimeSeconds(waitTime)
                    .messageAttributeNames("All")
```

```
                    .build();

            List<Message> messages =
sqsClient.receiveMessage(receiveRequest).messages();

            for (Message message : messages) {
                LOGGER.info("Received message: {}: {}", message.messageId(),
message.body());
            }

            return messages;

        } catch (SqsException e) {
            LOGGER.error("Couldn't receive messages from queue: {}", queueUrl,
e);
            throw e;
        }
    }

    /**
     * Delete a batch of messages from a queue in a single request.
     *
     * @param queueUrl  The URL of the queue from which to delete the messages.
     * @param messages  The list of messages to delete.
     * @return The response from SQS that contains the list of successful and
failed
     *         message deletions.
     */
    public static DeleteMessageBatchResponse deleteMessages(String queueUrl,
List<Message> messages) {
        try {
            List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();

            for (int i = 0; i < messages.size(); i++) {
                entries.add(DeleteMessageBatchRequestEntry.builder()
                        .id(String.valueOf(i))
                        .receiptHandle(messages.get(i).receiptHandle())
                        .build());
            }

            DeleteMessageBatchRequest deleteRequest =
DeleteMessageBatchRequest.builder()
                    .queueUrl(queueUrl)
                    .entries(entries)
```

```
                    .build();

            DeleteMessageBatchResponse response =
sqsClient.deleteMessageBatch(deleteRequest);

            if (!response.successful().isEmpty()) {
                for (DeleteMessageBatchResultEntry resultEntry :
response.successful()) {
                    LOGGER.info("Deleted {}",
messages.get(Integer.parseInt(resultEntry.id())).receiptHandle());
                }
            }

            if (!response.failed().isEmpty()) {
                for (BatchResultErrorEntry errorEntry : response.failed()) {
                    LOGGER.warn("Could not delete {}",
messages.get(Integer.parseInt(errorEntry.id())).receiptHandle());
                }
            }

            return response;

        } catch (SqsException e) {
            LOGGER.error("Couldn't delete messages from queue {}", queueUrl, e);
            throw e;
        }
    }

    /**
     * Helper class to represent a message with body and attributes.
     */
    public static class MessageEntry {
        private final String body;
        private final Map<String, MessageAttributeValue> attributes;

        public MessageEntry(String body, Map<String, MessageAttributeValue>
attributes) {
            this.body = body;
            this.attributes = attributes != null ? attributes : new HashMap<>();
        }

        public String getBody() {
            return body;
        }
```

```
        public Map<String, MessageAttributeValue> getAttributes() {
            return attributes;
        }
    }

    /**
     * Shows how to:
     * * Read the lines from a file and send the lines in
     *   batches of 10 as messages to a queue.
     * * Receive the messages in batches until the queue is empty.
     * * Reassemble the lines of the file and verify they match the original
file.
     */
    public static void usageDemo() {
        LOGGER.info("-".repeat(88));
        LOGGER.info("Welcome to the Amazon Simple Queue Service (Amazon SQS)
demo!");
        LOGGER.info("-".repeat(88));

        String queueUrl = null;
        try {
            // Create a queue for the demo.
            String queueName = "sqs-usage-demo-message-wrapper-" +
System.currentTimeMillis();
            CreateQueueRequest createRequest = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();
            queueUrl = sqsClient.createQueue(createRequest).queueUrl();
            LOGGER.info("Created queue: {}", queueUrl);

            try (InputStream inputStream =
SendRecvBatch.class.getResourceAsStream("/log4j2.xml");
                 BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream))) {

                List<String> lines = reader.lines().toList();

                // Send file lines in batches.
                int batchSize = 10;
                LOGGER.info("Sending file lines in batches of {} as messages.",
batchSize);

                for (int i = 0; i < lines.size(); i += batchSize) {
```

```
                    List<MessageEntry> messageBatch = new ArrayList<>();

                    for (int j = i; j < Math.min(i + batchSize, lines.size()); j+
+) {
                        String line = lines.get(j);
                        if (line == null || line.trim().isEmpty()) {
                            continue; // Skip empty lines.
                        }

                        Map<String, MessageAttributeValue> attributes = new
 HashMap<>();
                        attributes.put("line", MessageAttributeValue.builder()
                                .dataType("String")
                                .stringValue(String.valueOf(j))
                                .build());

                        messageBatch.add(new MessageEntry(lines.get(j),
 attributes));
                    }

                    sendMessages(queueUrl, messageBatch);
                    System.out.print(".");
                    System.out.flush();
                }

                LOGGER.info("\nDone. Sent {} messages.", lines.size());

                // Receive and process messages.
                LOGGER.info("Receiving, handling, and deleting messages in
batches of {}.", batchSize);
                String[] receivedLines = new String[lines.size()];
                boolean moreMessages = true;

                while (moreMessages) {
                    List<Message> receivedMessages = receiveMessages(queueUrl,
 batchSize, 5);

                    for (Message message : receivedMessages) {
                        int lineNumber =
Integer.parseInt(message.messageAttributes().get("line").stringValue());
                        receivedLines[lineNumber] = message.body();
                    }

                    if (!receivedMessages.isEmpty()) {
```

```
                    deleteMessages(queueUrl, receivedMessages);
                } else {
                    moreMessages = false;
                }
            }

            LOGGER.info("\nDone.");

            // Verify that all lines were received correctly.
            boolean allLinesMatch = true;
            for (int i = 0; i < lines.size(); i++) {
                String originalLine = lines.get(i);
                String receivedLine = receivedLines[i] == null ? "" :
receivedLines[i];

                if (!originalLine.equals(receivedLine)) {
                    allLinesMatch = false;
                    break;
                }
            }

            if (allLinesMatch) {
                LOGGER.info("Successfully reassembled all file lines!");
            } else {
                LOGGER.info("Uh oh, some lines were missed!");
            }
        }
    } catch (SqsException e) {
        LOGGER.error("SQS operation failed", e);
    } catch (RuntimeException | IOException e) {
        LOGGER.error("Unexpected runtime error during demo", e);
    } finally {
        // Clean up by deleting the queue if it was created.
        if (queueUrl != null) {
            try {
                DeleteQueueRequest deleteQueueRequest =
DeleteQueueRequest.builder()
                        .queueUrl(queueUrl)
                        .build();
                sqsClient.deleteQueue(deleteQueueRequest);
                LOGGER.info("Deleted queue: {}", queueUrl);
            } catch (SqsException e) {
                LOGGER.error("Failed to delete queue: {}", queueUrl, e);
            }
```

```
            }
        }

        LOGGER.info("Thanks for watching!");
        LOGGER.info("-".repeat(88));
    }
}
```

SimpleProducerConsumer.java - Menggunakan batching otomatis pesan.

```
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.
            }
        }
```

```
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat fungsi untuk membungkus fungsi pesan 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
```

Gunakan fungsi pembungkus untuk mengirim dan menerima pesan dalam 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)
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan AWS Message Processing Framework untuk.NET untuk mempublikasikan dan menerima pesan Amazon SQS

Contoh kode berikut menunjukkan cara membuat aplikasi yang menerbitkan dan menerima pesan Amazon SQS menggunakan AWS Message Processing Framework untuk.NET.

.NET

SDK untuk .NET

Menyediakan tutorial untuk AWS Message Processing Framework untuk.NET. Tutorial membuat aplikasi web yang memungkinkan pengguna untuk mempublikasikan pesan Amazon SQS dan aplikasi baris perintah yang menerima pesan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat tutorial lengkap di Panduan AWS SDK untuk .NET Pengembang dan contoh di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon SQS

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan Amazon SQS Java Messaging Library untuk bekerja dengan antarmuka Java Message Service (JMS) untuk Amazon SQS

Contoh kode berikut menunjukkan cara menggunakan Amazon SQS Java Messaging Library untuk bekerja dengan antarmuka JMS.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh berikut bekerja dengan antrian Amazon SQS standar dan termasuk:

- Mengirim pesan teks.
- Menerima pesan secara sinkron.
- Menerima pesan secara asinkron.

- Menerima pesan menggunakan mode CLIENT_ACCOGNEWNED.
- Menerima pesan menggunakan mode UNORDERED_ACCOGNEWNED.
- Menggunakan Spring untuk menyuntikkan dependensi.
- Kelas utilitas yang menyediakan metode umum yang digunakan oleh contoh lain.

Untuk informasi selengkapnya tentang penggunaan JMS dengan Amazon SQS, lihat Panduan Pengembang Amazon SQS.

Mengirim pesan teks.

```
   /**
    * 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);
        }
    }
```

Menerima pesan secara sinkron.

```
   /**
    * This method receives messages from a standard Amazon SQS queue using the
Amazon SQS Java
    * Messaging Library. It creates a connection to the queue using JMS (Java
Message Service),
    * waits for messages to arrive, and processes them one at a time. The method
handles all
    * necessary setup and cleanup of messaging resources.
    *
    * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
    */
   public static void doReceiveMessageSync() throws JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create a connection.
       try (SQSConnection connection = connectionFactory.createConnection() ) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

           // Create a session.
           Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
           MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

           connection.start();

           receiveMessages(consumer);
       }  // The connection closes automatically. This also closes the session.
       LOGGER.info("Connection closed");
   }
```

```
   /**
    * This method continuously checks for new messages from a standard Amazon
SQS queue using
    * the Amazon SQS Java Messaging Library. It waits up to 20 seconds for each
message, processes
    * it using JMS (Java Message Service), and confirms receipt. The method
stops checking for
    * messages after 20 seconds of no activity.
    *
    * @param consumer The JMS message consumer that receives messages from the
queue
    */
   private static void receiveMessages(MessageConsumer consumer) {
       try {
           while (true) {
               LOGGER.info("Waiting for messages...");
               // Wait 1 minute for a message
               Message message =
consumer.receive(Duration.ofSeconds(20).toMillis());
               if (message == null) {
                   LOGGER.info("Shutting down after 20 seconds of silence.");
                   break;
               }
               SqsJmsExampleUtils.handleMessage(message);
               message.acknowledge();
               LOGGER.info("Acknowledged message {}",
message.getJMSMessageID());
           }
       } catch (JMSException e) {
           LOGGER.error("Error receiving from SQS: {}", e.getMessage(), e);
       }
   }
```

Menerima pesan secara asinkron.

```
   /**
    * This method sets up automatic message handling for a standard Amazon SQS
queue using the
    * Amazon SQS Java Messaging Library. It creates a listener that processes
messages as soon
    * as they arrive using JMS (Java Message Service), runs for 5 seconds, then
cleans up all
```

```
     * messaging resources.
     *
     * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
     */
    public static void doReceiveMessageAsync() throws JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create a connection.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session.
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

            try {
                // Create a consumer for the queue.
                MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));
                // Provide an implementation of the MessageListener interface,
which has a single 'onMessage' method.
                // We use a lambda expression for the implementation.
                consumer.setMessageListener(message -> {
                    try {
                        SqsJmsExampleUtils.handleMessage(message);
                        message.acknowledge();
                    } catch (JMSException e) {
                        LOGGER.error("Error processing message: {}",
e.getMessage());
                    }
                });
                // Start receiving incoming messages.
                connection.start();
                LOGGER.info("Waiting for messages...");
            } catch (JMSException e) {
                throw new RuntimeException(e);
```

```
            }
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }  // The connection closes automatically. This also closes the session.
        LOGGER.info( "Connection closed" );
    }
```

Menerima pesan menggunakan mode CLIENT_ACCOGNEWNED.

```
    /**
     * This method demonstrates how message acknowledgment affects message
processing in a standard
     * Amazon SQS queue using the Amazon SQS Java Messaging Library. It sends
messages to the queue,
     * then shows how JMS (Java Message Service) client acknowledgment mode
handles both explicit
     * and implicit message confirmations, including how acknowledging one
message can automatically
     * acknowledge previous messages.
     *
     * @throws JMSException If there is a problem with the messaging operations
     */
    public static void doReceiveMessagesSyncClientAcknowledge() throws
JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create the connection in a try-with-resources statement so that it's
closed automatically.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);

            // Create a session with client acknowledge mode.
```

```
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

            // Create a producer and consumer.
            MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
            MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

            // Open the connection.
            connection.start();

            // Send two text messages.
            sendMessage(producer, session, "Message 1");
            sendMessage(producer, session, "Message 2");

            // Receive a message and don't acknowledge it.
            receiveMessage(consumer, false);

            // Receive another message and acknowledge it.
            receiveMessage(consumer, true);

            // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue,
            LOGGER.info("Waiting for visibility timeout...");
            try {
                Thread.sleep(TIME_OUT_MILLIS);
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, but none will be
available. This is because in
                CLIENT_ACKNOWLEDGE mode, when we acknowledged the second message,
all previous messages were
                automatically acknowledged as well. Therefore, although we never
directly acknowledged the first
                message, it was implicitly acknowledged when we confirmed the
second one. */
            receiveMessage(consumer, true);
        } // The connection closes automatically. This also closes the session.
```

```
        LOGGER.info("Connection closed.");

    }


    /**
     * Sends a text message using the specified JMS MessageProducer and Session.
     *
     * @param producer    The JMS MessageProducer used to send the message
     * @param session     The JMS Session used to create the text message
     * @param messageText The text content to be sent in the message
     * @throws JMSException If there is an error creating or sending the message
     */
    private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
        // Create a text message and send it.
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives and processes a message from a JMS queue using the specified
consumer.
     * The method waits for a message until the configured timeout period is
reached.
     * If a message is received, it is logged and optionally acknowledged based
on the
     * acknowledge parameter.
     *
     * @param consumer    The JMS MessageConsumer used to receive messages from
the queue
     * @param acknowledge Boolean flag indicating whether to acknowledge the
message.
     *                    If true, the message will be acknowledged after
processing
     * @throws JMSException If there is an error receiving, processing, or
acknowledging the message
     */
    private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSException {
        // Receive a message.
        Message message = consumer.receive(TIME_OUT_MILLIS);

        if (message == null) {
            LOGGER.info("Queue is empty!");
```

```
       } else {
           // Since this queue has only text messages, cast the message object
and print the text.
           LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
message).getText(), acknowledge);

           // Acknowledge the message if asked.
           if (acknowledge) message.acknowledge();
       }
   }
```

Menerima pesan menggunakan mode UNORDERED_ACCOGNEWNED.

```
   /**
    * Demonstrates message acknowledgment behavior in UNORDERED_ACKNOWLEDGE mode
with Amazon SQS JMS.
    * In this mode, each message must be explicitly acknowledged regardless of
receive order.
    * Unacknowledged messages return to the queue after the visibility timeout
expires,
    * unlike CLIENT_ACKNOWLEDGE mode where acknowledging one message
acknowledges all previous messages.
    *
    * @throws JMSException         If a JMS-related error occurs during message
operations
    */
   public static void doReceiveMessagesUnorderedAcknowledge() throws
JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create the connection in a try-with-resources statement so that it's
closed automatically.
       try( SQSConnection connection = connectionFactory.createConnection() ) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);
```

```
            // Create a session with unordered acknowledge mode.
            Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

            // Create the producer and consumer.
            MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
            MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

            // Open a connection.
            connection.start();

            // Send two text messages.
            sendMessage(producer, session, "Message 1");
            sendMessage(producer, session, "Message 2");

            // Receive a message and don't acknowledge it.
            receiveMessage(consumer, false);

            // Receive another message and acknowledge it.
            receiveMessage(consumer, true);

            // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue.
            LOGGER.info("Waiting for visibility timeout...");
            try {
                Thread.sleep(TIME_OUT_MILLIS);
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, and we'll get the
first message again. This occurs
                because in UNORDERED_ACKNOWLEDGE mode, each message requires its
own separate acknowledgment.
                Since we only acknowledged the second message, the first message
remains in the queue for
                redelivery. */
            receiveMessage(consumer, true);
```

```
            LOGGER.info("Connection closed.");
        } // The connection closes automatically. This also closes the session.
    }

    /**
     * Sends a text message to an Amazon SQS queue using JMS.
     *
     * @param producer    The JMS MessageProducer for the queue
     * @param session     The JMS Session for message creation
     * @param messageText The message content
     * @throws JMSException If message creation or sending fails
     */
    private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
        // Create a text message and send it.
        producer.send(session.createTextMessage(messageText));
    }
    /**
     * Synchronously receives a message from an Amazon SQS queue using the JMS
API
     * with an acknowledgment parameter.
     *
     * @param consumer    The JMS MessageConsumer for the queue
     * @param acknowledge If true, acknowledges the message after receipt
     * @throws JMSException If message reception or acknowledgment fails
     */
    private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSException {
        // Receive a message.
        Message message = consumer.receive(TIME_OUT_MILLIS);

        if (message == null) {
            LOGGER.info("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object
and print the text.
            LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
message).getText(), acknowledge);

            // Acknowledge the message if asked.
            if (acknowledge) message.acknowledge();
        }
    }
```

Menggunakan Spring untuk menyuntikkan dependensi.

```
package com.example.sqs.jms.spring;

import com.amazon.sqs.javamessaging.SQSConnection;
import com.example.sqs.jms.SqsJmsExampleUtils;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.NoSuchBeanDefinitionException;
import org.springframework.context.support.FileSystemXmlApplicationContext;

import java.io.File;
import java.net.URL;
import java.util.concurrent.TimeUnit;

/**
 * Demonstrates how to send and receive messages using the Amazon SQS Java
 Messaging Library
 * with Spring Framework integration. This example connects to a standard Amazon
 SQS message
 * queue using Spring's dependency injection to configure the connection and
 messaging components.
 * The application uses the JMS (Java Message Service) API to handle message
 operations.
 */
public class SpringExample {
    private static final Integer POLLING_SECONDS = 15;
    private static final String SPRING_XML_CONFIG_FILE =
 "SpringExampleConfiguration.xml.txt";
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SpringExample.class);

    /**
     * Demonstrates sending and receiving messages through a standard Amazon SQS
 message queue
     * using Spring Framework configuration. This method loads connection
 settings from an XML file,
     * establishes a messaging session using the Amazon SQS Java Messaging
 Library, and processes
```

```
    * messages using JMS (Java Message Service) operations. If the queue doesn't
exist, it will
    * be created automatically.
    *
    * @param args Command line arguments (not used)
    */
   public static void main(String[] args) {

       URL resource =
SpringExample.class.getClassLoader().getResource(SPRING_XML_CONFIG_FILE);
       File springFile = new File(resource.getFile());
       if (!springFile.exists() || !springFile.canRead()) {
           LOGGER.error("File " + SPRING_XML_CONFIG_FILE + " doesn't exist or
isn't readable.");
           System.exit(1);
       }

       try (FileSystemXmlApplicationContext context =
                    new FileSystemXmlApplicationContext("file://" +
springFile.getAbsolutePath())) {

           Connection connection;
           try {
               connection = context.getBean(Connection.class);
           } catch (NoSuchBeanDefinitionException e) {
               LOGGER.error("Can't find the JMS connection to use: " +
e.getMessage(), e);
               System.exit(2);
               return;
           }

           String queueName;
           try {
               queueName = context.getBean("queueName", String.class);
           } catch (NoSuchBeanDefinitionException e) {
               LOGGER.error("Can't find the name of the queue to use: " +
e.getMessage(), e);
               System.exit(3);
               return;
           }
           try {
               if (connection instanceof SQSConnection) {
                   SqsJmsExampleUtils.ensureQueueExists((SQSConnection)
connection, queueName, SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);
```

```
                }
                // Create the JMS session.
                Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

                SqsJmsExampleUtils.sendTextMessage(session, queueName);
                MessageConsumer consumer =
session.createConsumer(session.createQueue(queueName));

                receiveMessages(consumer);
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        }   // Spring context autocloses. Managed Spring beans that implement
AutoClosable, such as the
        // 'connection' bean, are also closed.
        LOGGER.info("Context closed");
    }

    /**
     * Continuously checks for and processes messages from a standard Amazon SQS
message queue
     * using the Amazon SQS Java Messaging Library underlying the JMS API. This
method waits for incoming messages,
     * processes them when they arrive, and acknowledges their receipt using JMS
(Java Message
     * Service) operations. The method will stop checking for messages after 15
seconds of
     * inactivity.
     *
     * @param consumer The JMS message consumer used to receive messages from the
queue
     */
    private static void receiveMessages(MessageConsumer consumer) {
        try {
            while (true) {
                LOGGER.info("Waiting for messages...");
                // Wait 15 seconds for a message.
                Message message =
consumer.receive(TimeUnit.SECONDS.toMillis(POLLING_SECONDS));
                if (message == null) {
                    LOGGER.info("Shutting down after {} seconds of silence.",
POLLING_SECONDS);
```

```
                    break;
                }
                SqsJmsExampleUtils.handleMessage(message);
                message.acknowledge();
                LOGGER.info("Message acknowledged.");
            }
        } catch (JMSException e) {
            LOGGER.error("Error receiving from SQS.", e);
        }
    }
}
```

Definisi kacang musim semi.

```
<?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>
```

Kelas utilitas yang menyediakan metode umum yang digunakan oleh contoh lain.

```
package com.example.sqs.jms;

import com.amazon.sqs.javamessaging.AmazonSQSMessagingClientWrapper;
import com.amazon.sqs.javamessaging.ProviderConfiguration;
import com.amazon.sqs.javamessaging.SQSConnection;
import com.amazon.sqs.javamessaging.SQSConnectionFactory;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.exception.SdkException;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
```

```
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;

import java.time.Duration;
import java.util.Base64;
import java.util.Map;

/**
 * This utility class provides helper methods for working with Amazon Simple
 Queue Service (Amazon SQS)
 * through the Java Message Service (JMS) interface. It contains common
 operations for managing message
 * queues and handling message delivery.
 */
public class SqsJmsExampleUtils {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SqsJmsExampleUtils.class);
    public static final Long QUEUE_VISIBILITY_TIMEOUT = 5L;

    /**
     * This method verifies that a message queue exists and creates it if
 necessary. The method checks for
     * an existing queue first to optimize performance.
     *
     * @param connection The active connection to the messaging service
     * @param queueName The name of the queue to verify or create
     * @param visibilityTimeout The duration in seconds that messages will be
 hidden after being received
     * @throws JMSException If there is an error accessing or creating the queue
     */
    public static void ensureQueueExists(SQSConnection connection, String
 queueName, Long visibilityTimeout) throws JMSException {
        AmazonSQSMessagingClientWrapper client =
 connection.getWrappedAmazonSQSClient();

      /* In most cases, you can do this with just a 'createQueue' call, but
 'getQueueUrl'
      (called by 'queueExists') is a faster operation for the common case where
 the queue
      already exists. Also, many users and roles have permission to call
 'getQueueUrl'
      but don't have permission to call 'createQueue'.
      */
       if( !client.queueExists(queueName) ) {
           CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
```

```
                    .queueName(queueName)
                    .attributes(Map.of(QueueAttributeName.VISIBILITY_TIMEOUT,
String.valueOf(visibilityTimeout)))
                    .build();
            client.createQueue( createQueueRequest );
        }
    }

    /**
     * This method sends a simple text message to a specified message queue. It
handles all necessary
     * setup for the message delivery process.
     *
     * @param session The active messaging session used to create and send the
message
     * @param queueName The name of the queue where the message will be sent
     */
    public static void sendTextMessage(Session session, String queueName) {
        // Rest of implementation...

        try {
            MessageProducer producer =
session.createProducer( session.createQueue( queueName) );
            Message message = session.createTextMessage("Hello world!");
            producer.send(message);
        } catch (JMSException e) {
            LOGGER.error( "Error receiving from SQS", e );
        }
    }

    /**
     * This method processes incoming messages and logs their content based on
the message type.
     * It supports text messages, binary data, and Java objects.
     *
     * @param message The message to be processed and logged
     * @throws JMSException If there is an error reading the message content
     */
    public static void handleMessage(Message message) throws JMSException {
        // Rest of implementation...
        LOGGER.info( "Got message {}", message.getJMSMessageID() );
        LOGGER.info( "Content: ");
        if(message instanceof TextMessage txtMessage) {
            LOGGER.info( "\t{}", txtMessage.getText() );
```

```
       } else if(message instanceof BytesMessage byteMessage){
           // Assume the length fits in an int - SQS only supports sizes up to
256k so that
           // should be true
           byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
           byteMessage.readBytes(bytes);
           LOGGER.info( "\t{}", Base64.getEncoder().encodeToString( bytes ) );
       } else if( message instanceof ObjectMessage) {
           ObjectMessage objMessage = (ObjectMessage) message;
           LOGGER.info( "\t{}", objMessage.getObject() );
       }
   }

   /**
    * This method sets up automatic message processing for a specified queue. It
creates a listener
    * that will receive and handle incoming messages without blocking the main
program.
    *
    * @param session The active messaging session
    * @param queueName The name of the queue to monitor
    * @param connection The active connection to the messaging service
    */
   public static void receiveMessagesAsync(Session session, String queueName,
Connection connection) {
       // Rest of implementation...
       try {
           // Create a consumer for the queue.
           MessageConsumer consumer =
session.createConsumer(session.createQueue(queueName));
           // Provide an implementation of the MessageListener interface, which
has a single 'onMessage' method.
           // We use a lambda expression for the implementation.
           consumer.setMessageListener(message -> {
               try {
                   SqsJmsExampleUtils.handleMessage(message);
                   message.acknowledge();
               } catch (JMSException e) {
                   LOGGER.error("Error processing message: {}", e.getMessage());
               }
           });
           // Start receiving incoming messages.
           connection.start();
       } catch (JMSException e) {
```

```
            throw new RuntimeException(e);
        }
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }


    /**
     * This method performs cleanup operations after message processing is
complete. It receives
     * any messages in the specified queue, removes the message queue and closes
all
     * active connections to prevent resource leaks.
     *
     * @param queueName The name of the queue to be removed
     * @param visibilityTimeout The duration in seconds that messages are hidden
after being received
     * @throws JMSException If there is an error during the cleanup process
     */
    public static void cleanUpExample(String queueName, Long visibilityTimeout)
throws JMSException {
        LOGGER.info("Performing cleanup.");

        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        try (SQSConnection connection = connectionFactory.createConnection() ) {
            ensureQueueExists(connection, queueName, visibilityTimeout);
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            receiveMessagesAsync(session, queueName, connection);

            SqsClient sqsClient =
connection.getWrappedAmazonSQSClient().getAmazonSQSClient();
            try {
                String queueUrl = sqsClient.getQueueUrl(b ->
b.queueName(queueName)).queueUrl();
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
```

```
                LOGGER.info("Queue deleted: {}", queueUrl);
            } catch (SdkException e) {
                LOGGER.error("Error during SQS operations: ", e);
            }
        }
        LOGGER.info("Clean up: Connection closed");
    }

    /**
     * This method creates a background task that sends multiple messages to a
specified queue
     * after waiting for a set time period. The task operates independently to
ensure efficient
     * message processing without interrupting other operations.
     *
     * @param queueName The name of the queue where messages will be sent
     * @param secondsToWait The number of seconds to wait before sending messages
     * @param numMessages The number of messages to send
     * @param visibilityTimeout The duration in seconds that messages remain
hidden after being received
     * @return A task that can be executed to send the messages
     */
    public static Runnable sendAMessageAsync(String queueName, Long
secondsToWait, Integer numMessages, Long visibilityTimeout) {
        return () -> {
            try {
                Thread.sleep(Duration.ofSeconds(secondsToWait).toMillis());
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException(e);
            }
            try {
                SQSConnectionFactory connectionFactory = new
SQSConnectionFactory(
                        new ProviderConfiguration(),
                        SqsClient.create()
                );
                try (SQSConnection connection =
connectionFactory.createConnection()) {
                    ensureQueueExists(connection, queueName, visibilityTimeout);
                    Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
                    for (int i = 1; i <= numMessages; i++) {
```

```
                        MessageProducer producer =
 session.createProducer(session.createQueue(queueName));
                        producer.send(session.createTextMessage("Hello World " +
 i + "!"));
                    }
                }
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        };
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - CreateQueue
  - DeleteQueue

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Bekerja dengan tag antrian dan Amazon SQS menggunakan SDK AWS

Contoh kode berikut menunjukkan cara melakukan operasi penandaan dengan Amazon SQS.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Contoh berikut membuat tag untuk antrian, daftar tag, dan menghapus 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 a tag a aueue.
     * @param queueUrl The URL of the queue to tag.
     */
    public static void addTags(String queueUrl) {
        // Build a map of the tags.
        final Map<String, String> tagsToAdd = Map.of(
                "Team", "Development",
                "Priority", "Beta",
                "Accounting ID", "456def");

        try {
            // Add tags to the queue using a Consumer<TagQueueRequest.Builder>
parameter.
            sqsClient.tagQueue(b -> b
                    .queueUrl(queueUrl)
                    .tags(tagsToAdd)
            );
        } catch (QueueDoesNotExistException e) {
            LOGGER.error("Queue does not exist: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }

    /** This method demonstrates how to view the tags for a queue.
     * @param queueUrl The URL of the queue whose tags you want to list.
     */
    public static void listTags(String queueUrl) {
        ListQueueTagsResponse response;
        try {
            // Call the listQueueTags method with a
Consumer<ListQueueTagsRequest.Builder> parameter that creates a
ListQueueTagsRequest.
            response = sqsClient.listQueueTags(b -> b
                    .queueUrl(queueUrl));
```

```
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }

        // Log the tags.
        response.tags()
                .forEach((k, v) ->
                        LOGGER.info("Key: {} -> Value: {}", k, v));
    }

    /**
     * This method demonstrates how to remove tags from a queue.
     * @param queueUrl The URL of the queue whose tags you want to remove.
     */
    public static void removeTags(String queueUrl) {
        try {
            // Call the untagQueue method with a
 Consumer<UntagQueueRequest.Builder> parameter.
            sqsClient.untagQueue(b -> b
                    .queueUrl(queueUrl)
                    .tagKeys("Accounting ID") // Remove a single tag.
            );
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - ListQueueTags
  - TagQueue
  - UntagQueue

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Contoh tanpa server untuk Amazon SQS

Contoh kode berikut menunjukkan cara menggunakan Amazon SQS dengan. AWS SDKs

Contoh

- Memanggil fungsi Lambda dari pemicu Amazon SQS
- Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS

Memanggil fungsi Lambda dari pemicu Amazon SQS

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima pesan dari antrian SQS. Fungsi mengambil pesan dari parameter peristiwa dan mencatat konten setiap pesan.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan.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.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
```

```
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan 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 untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan 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 untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan JavaScript Lambda menggunakan.

```
// 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;
  }
}
```

Mengkonsumsi acara SQS dengan TypeScript Lambda menggunakan.

```
// 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 untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan 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 untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan 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 untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan 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
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk fungsi Lambda yang menerima peristiwa dari antrian SQS. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan.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.DefaultLambdaJsonSeri
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
```

```
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan 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 untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan 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 untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan penggunaan JavaScript Lambda.

```
// 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}`);
}
```

Melaporkan kegagalan item batch SQS dengan penggunaan TypeScript Lambda.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
 from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];
```

```
    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan 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 untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan 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 untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan 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
}
```

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan Amazon SQS dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memecahkan masalah di Amazon SQS

Topik ini memberikan saran pemecahan masalah untuk kesalahan umum dan masalah yang mungkin Anda temui saat menggunakan konsol Amazon SQS, Amazon SQS API, atau alat lain dengan Amazon SQS. Jika Anda menemukan masalah yang tidak tercantum di sini, Anda dapat menggunakan tombol Umpan Balik di halaman ini untuk melaporkannya.

Untuk saran pemecahan masalah selengkapnya dan jawaban atas pertanyaan dukungan umum, kunjungi Pusat Pengetahuan AWS.

Topik

- Memecahkan masalah kesalahan akses ditolak di Amazon SQS
- Memecahkan masalah kesalahan Amazon SQS API
- Memecahkan masalah antrian surat mati Amazon SQS dan masalah redrive DLQ
- Memecahkan masalah pembatasan FIFO di Amazon SQS
- Memecahkan masalah pesan yang tidak dikembalikan untuk panggilan Amazon SQS API ReceiveMessage
- Memecahkan masalah kesalahan jaringan Amazon SQS
- Memecahkan masalah antrian Amazon Simple Queue Service menggunakan AWS X-Ray

Memecahkan masalah kesalahan akses ditolak di Amazon SQS

Topik berikut mencakup penyebab `AccessDenied` atau `AccessDeniedException` kesalahan paling umum pada panggilan API Amazon SQS. Untuk informasi selengkapnya tentang cara memecahkan masalah kesalahan ini, lihat Bagaimana cara memecahkan masalah kesalahan "" atau "AccessDenied" pada panggilan AccessDeniedException API Amazon SQS? dalam Panduan Pusat AWS Pengetahuan.

Contoh pesan kesalahan:

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to
        the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

- atau -

```
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.
```

Kebijakan antrian Amazon SQS dan kebijakan IAM

Untuk memverifikasi apakah pemohon memiliki izin yang tepat untuk melakukan operasi Amazon SQS, lakukan hal berikut:

- Identifikasi prinsip IAM yang membuat panggilan Amazon SQS API. Jika prinsipal IAM berasal dari akun yang sama, kebijakan antrian Amazon SQS atau AWS kebijakan Identity and Access Management (IAM) and Access Management (IAM) harus menyertakan izin untuk secara eksplisit mengizinkan akses untuk tindakan tersebut.
- Jika prinsipal adalah entitas IAM:
  - Anda dapat mengidentifikasi pengguna atau peran IAM Anda dengan memeriksa sudut kanan atas Konsol Manajemen AWS, atau dengan menggunakan perintah. `aws sts get-caller-identity`
  - Periksa kebijakan IAM yang terkait dengan peran atau pengguna IAM. Gunakan salah satu metode berikut:
    - Uji kebijakan IAM dengan IAM Policy Simulator.
    - Tinjau berbagai jenis kebijakan IAM yang berbeda.
  - Jika perlu, edit kebijakan pengguna IAM Anda.
  - Periksa kebijakan antrian dan edit jika diperlukan.
- Jika prinsipal adalah AWS layanan, maka kebijakan antrian Amazon SQS harus secara eksplisit mengizinkan akses.
- Jika prinsipal adalah prinsipal lintas akun, maka kebijakan antrian Amazon SQS dan kebijakan IAM harus secara eksplisit mengizinkan akses.
- Jika kebijakan menggunakan elemen kondisi, periksa apakah kondisi membatasi akses.

> **Important**
>
> Penolakan eksplisit di salah satu kebijakan mengesampingkan izin eksplisit. Berikut adalah beberapa contoh dasar kebijakan Amazon SQS.

AWS Key Management Service izin

Jika antrian Amazon SQS Anda mengaktifkan enkripsi sisi server (SSE) dengan pelanggan yang dikelola AWS KMS key, maka izin harus diberikan kepada produsen dan konsumen. Untuk mengonfirmasi apakah antrian dienkripsi, Anda dapat menggunakan **`KmsMasterKeyId`** atribut **`GetQueueAttributes`**API, atau dari konsol antrian di bawah Enkripsi.

- Izin yang diperlukan untuk produsen:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

- Izin yang diperlukan untuk konsumen:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- Izin yang diperlukan untuk akses lintas akun:

```
{
"Effect": "Allow",
"Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

Pilih salah satu opsi berikut untuk mengaktifkan enkripsi antrian Amazon SQS:

- SSE-Amazon SQS (Kunci enkripsi dibuat dan dikelola oleh layanan Amazon SQS.)
- AWS kunci default terkelola (alias/aws/sqs)
- Kunci yang dikelola pelanggan

Namun, jika Anda menggunakan kunci KMS AWS -managed, Anda tidak dapat mengubah kebijakan kunci default. Oleh karena itu, untuk menyediakan akses ke layanan lain dan lintas akun, gunakan kunci yang dikelola pelanggan. Melakukan hal ini memungkinkan Anda untuk mengedit kebijakan utama.

Kebijakan titik akhir VPC

Jika Anda mengakses Amazon SQS melalui titik akhir Amazon Virtual Private Cloud (Amazon VPC), kebijakan titik akhir Amazon SQS VPC harus mengizinkan akses. Anda dapat membuat kebijakan untuk titik akhir VPC Amazon untuk Amazon SQS, di mana Anda dapat menentukan hal berikut:

1. Prinsipal yang dapat melakukan tindakan.
2. Tindakan yang dapat dilakukan.
3. Sumber daya yang menjadi target tindakan.

Dalam contoh berikut, kebijakan titik akhir VPC menentukan bahwa pengguna IAM diizinkan mengirim pesan ke *MyUser* antrean Amazon SQS. *MyQueue* Tindakan lain, pengguna IAM, dan sumber daya Amazon SQS ditolak aksesnya melalui titik akhir VPC.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

Kebijakan kontrol layanan organisasi

Jika Anda Akun AWS termasuk dalam organisasi, AWS Organizations kebijakan dapat memblokir Anda dari mengakses antrian Amazon SQS Anda. Secara default, AWS Organizations kebijakan tidak memblokir permintaan apa pun ke Amazon SQS. Namun, pastikan AWS Organizations kebijakan Anda belum dikonfigurasi untuk memblokir akses ke antrian Amazon SQS. Untuk petunjuk tentang cara memeriksa AWS Organizations kebijakan Anda, lihat Mencantumkan semua kebijakan di Panduan AWS Organizations Pengguna.

Memecahkan masalah kesalahan Amazon SQS API

Topik berikut mencakup kesalahan paling umum yang dikembalikan saat melakukan panggilan Amazon SQS API, dan cara memecahkan masalah.

QueueDoesNotExist kesalahan

Kesalahan ini akan dikembalikan ketika layanan Amazon SQS tidak dapat menemukan antrian yang disebutkan untuk tindakan Amazon SQS.

Kemungkinan penyebab dan mitigasi:

- Wilayah salah: Tinjau konfigurasi klien Amazon SQS untuk mengonfirmasi bahwa Anda mengonfigurasi Wilayah yang benar pada klien. Bila Anda tidak mengonfigurasi Region pada klien, maka SDK atau AWS CLI memilih Region dari file konfigurasi atau variabel lingkungan. Jika SDK tidak menemukan Region dalam file konfigurasi, maka SDK menetapkan Region ke us-east-1 secara default.
- Antrian mungkin baru saja dihapus: Jika antrian dihapus sebelum panggilan API dibuat, maka panggilan API akan mengembalikan kesalahan ini. CloudTrailPeriksa DeleteQueueoperasi apa pun sebelum waktu kesalahan.
- Masalah izin: Jika pengguna atau peran yang meminta AWS Identity and Access Management (IAM) tidak memiliki izin yang diperlukan, maka Anda mungkin menerima kesalahan berikut:

  ```
  The specified queue does not exist or you do not have access to it.
  ```

  Periksa izin, dan lakukan panggilan API dengan izin yang benar.

Untuk detail selengkapnya tentang pemecahan masalah `QueueDoesNotExist` kesalahan, lihat Bagaimana cara memecahkan masalah QueueDoesNotExist kesalahan saat melakukan panggilan API ke antrean Amazon SQS saya? dalam Panduan Pusat AWS Pengetahuan.

InvalidAttributeValue kesalahan

Kesalahan ini akan dikembalikan setelah memperbarui kebijakan sumber daya antrian Amazon SQS, atau properti dengan kebijakan atau prinsipal yang salah.

Kemungkinan penyebab dan mitigasi:

- Kebijakan sumber daya tidak valid: Periksa apakah kebijakan sumber daya memiliki semua bidang yang diperlukan. Untuk informasi selengkapnya, lihat referensi elemen kebijakan IAM JSON dan Memvalidasi kebijakan IAM. Anda juga dapat menggunakan generator kebijakan IAM untuk membuat dan menguji kebijakan sumber daya Amazon SQS. Pastikan kebijakan tersebut dalam format JSON.
- Prinsipal tidak valid: Pastikan bahwa `Principal` elemen ada dalam kebijakan sumber daya dan nilainya valid. Jika `Principal` elemen kebijakan sumber daya Amazon SQS menyertakan entitas IAM, pastikan entitas tersebut ada sebelum Anda menggunakan kebijakan tersebut. Amazon SQS memvalidasi kebijakan sumber daya dan memeriksa entitas IAM. Jika entitas IAM tidak ada, Anda akan menerima kesalahan. Untuk mengonfirmasi entitas IAM, gunakan `GetRole`dan `GetUser` APIs.

Untuk informasi tambahan tentang cara memecahkan masalah `InvalidAttributeValue` kesalahan, lihat Bagaimana cara memecahkan masalah QueueDoesNotExist kesalahan saat melakukan panggilan API ke antrean Amazon SQS saya? dalam Panduan Pusat AWS Pengetahuan.

ReceiptHandle kesalahan

Setelah melakukan panggilan `DeleteMessage`API, kesalahan `ReceiptHandleIsInvalid` atau `InvalidParameterValue` mungkin dikembalikan jika pegangan tanda terima salah atau kedaluwarsa.

- ReceiptHandleIsInvalid error: Jika pegangan tanda terima salah, Anda akan menerima kesalahan yang mirip dengan contoh ini:

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
 The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- InvalidParameterValue error: Jika tanda terima kedaluwarsa, Anda akan menerima kesalahan yang mirip dengan contoh ini:

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:
 Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The
 receipt handle has expired.
```

Kemungkinan penyebab dan mitigasi:

Pegangan tanda terima dibuat untuk setiap pesan yang diterima, dan hanya berlaku untuk periode batas waktu visibilitas. Ketika periode batas waktu visibilitas berakhir, pesan menjadi terlihat pada antrian untuk konsumen. Ketika Anda menerima pesan lagi dari konsumen, Anda menerima pegangan tanda terima baru. Untuk mencegah kesalahan penanganan tanda terima yang salah atau kedaluwarsa, gunakan gagang tanda terima yang benar untuk menghapus pesan dalam periode batas waktu visibilitas antrian Amazon SQS.

Untuk informasi tambahan tentang cara memecahkan masalah `ReceiptHandle` kesalahan, lihat Bagaimana cara memecahkan masalah kesalahan "" dan "ReceiptHandleIsInvalid" saat saya menggunakan panggilan API Amazon SQS? InvalidParameterValue DeleteMessage dalam Panduan Pusat AWS Pengetahuan.

Memecahkan masalah antrian surat mati Amazon SQS dan masalah redrive DLQ

Topik berikut mencakup penyebab paling umum masalah Amazon SQS DLQ dan DLQ redrive, dan cara memecahkan masalah tersebut. Untuk informasi selengkapnya, lihat Bagaimana cara mengatasi masalah redrive Amazon SQS DLQ? dalam Panduan Pusat AWS Pengetahuan.

Masalah DLQ

Pelajari tentang masalah DLQ umum dan cara mengatasinya.

Melihat pesan menggunakan konsol dapat menyebabkan pesan dipindahkan ke antrean huruf mati

Amazon SQS menghitung melihat pesan di konsol terhadap kebijakan redrive antrean terkait. Oleh karena itu, jika Anda melihat pesan di konsol berapa kali ditentukan dalam kebijakan redrive antrian terkait, pesan akan dipindahkan ke antrean huruf mati antrian yang sesuai.

Untuk menyesuaikan perilaku ini, Anda dapat melakukan salah satu hal berikut:

- Meningkatkan pengaturan Penerimaan Maksimum untuk kebijakan redrive antrean terkait.
- Hindari melihat pesan antrian yang sesuai di konsol.

Antrian **`NumberOfMessagesSent`** dan **`NumberOfMessagesReceived`** untuk surat mati tidak cocok

Jika Anda mengirim pesan ke antrian huruf mati secara manual, pesan tersebut ditangkap oleh metrik. `NumberOfMessagesSent` Namun, jika pesan dikirim ke antrian huruf mati sebagai akibat dari upaya pemrosesan yang gagal, pesan tersebut tidak ditangkap oleh metrik ini. Oleh karena itu, dimungkinkan untuk nilai-nilai `NumberOfMessagesSent` dan `NumberOfMessagesReceived`untuk menjadi berbeda.

Membuat dan mengonfigurasi redrive antrian huruf mati

Penggerak ulang antrian huruf mati mengharuskan Anda menetapkan izin yang sesuai untuk Amazon SQS untuk menerima pesan dari antrian huruf mati, dan mengirim pesan ke antrean tujuan. Jika Anda tidak memiliki izin yang benar, tugas redrive antrian huruf mati bisa gagal. Anda dapat melihat status tugas penggerak ulang pesan untuk memperbaiki masalah, dan coba lagi.

Penanganan kegagalan pesan antrian standar dan FIFO

Antrian standar terus memproses pesan hingga berakhirnya periode retensi. Pemrosesan berkelanjutan ini meminimalkan kemungkinan antrian diblokir oleh pesan yang tidak dikonsumsi. Memiliki sejumlah besar pesan yang berulang kali gagal dihapus konsumen dapat meningkatkan biaya, dan menempatkan beban ekstra pada perangkat keras. Untuk menekan biaya, pindahkan pesan yang gagal ke antrian surat mati.

Antrian standar juga memungkinkan sejumlah besar pesan dalam penerbangan. Jika sebagian besar pesan Anda tidak dapat dikonsumsi, dan tidak dikirim ke antrian surat mati, kecepatan pemrosesan pesan Anda dapat melambat. Untuk menjaga efisiensi antrian Anda, pastikan aplikasi Anda menangani pemrosesan pesan dengan benar.

Antrian FIFO menyediakan pemrosesan tepat sekali dengan mengkonsumsi pesan secara berurutan dari grup pesan. Oleh karena itu, meskipun konsumen dapat terus mengambil pesan yang dipesan dari grup pesan lain, grup pesan pertama tetap tidak tersedia sampai pesan yang memblokir antrian diproses dengan sukses atau dipindahkan ke antrian huruf mati.

Selain itu, antrian FIFO memungkinkan jumlah pesan dalam penerbangan yang lebih rendah. Agar antrian FIFO Anda tidak diblokir oleh pesan, pastikan aplikasi Anda menangani pemrosesan pesan dengan benar.

Untuk informasi selengkapnya, lihat Kuota pesan Amazon SQS dan Praktik terbaik Amazon SQS.

Masalah DLQ-redrive

Pelajari tentang masalah umum DLQ-redrive dan cara mengatasinya.

AccessDenied masalah izin

`AccessDenied`Kesalahan terjadi ketika redrive DLQ gagal karena entitas AWS Identity and Access Management (IAM) tidak memiliki izin yang diperlukan.

Contoh pesan kesalahan:

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

Izin API berikut diperlukan untuk membuat permintaan redrive DLQ:

Untuk memulai redrive pesan:

- Izin antrian surat mati:
  - `sqs:StartMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt`— Ketika antrian huruf mati atau antrian sumber asli dienkripsi.
- Izin antrian tujuan:
  - `sqs:SendMessage`
  - `kms:GenerateDataKey`— Saat antrian tujuan dienkripsi.
  - `kms:Decrypt`— Saat antrian tujuan dienkripsi.

Untuk membatalkan redrive pesan yang sedang berlangsung:

- Izin antrian surat mati:

- `sqs:CancelMessageMoveTask`
- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`
- `kms:Decrypt`— Ketika antrian huruf mati atau antrian sumber asli dienkripsi.

Untuk menampilkan status pemindahan pesan:

- Izin antrian surat mati:
  - `sqs:ListMessageMoveTasks`
  - `sqs:GetQueueAttributes`

`NonExistentQueue` kesalahan

`NonExistentQueue`Kesalahan terjadi ketika antrean sumber Amazon SQS tidak ada, atau dihapus. Periksa dan redrive ke antrian Amazon SQS yang ada.

Contoh pesan kesalahan:

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource kesalahan

`CouldNotDetermineMessageSource`Kesalahan terjadi ketika Anda mencoba memulai redrive DLQ dengan skenario berikut:

- Pesan Amazon SQS dikirim langsung ke DLQ dengan API. `SendMessage`
- Pesan dari topik AWS Lambda atau fungsi Amazon Simple Notification Service (Amazon SNS) dengan DLQ yang dikonfigurasi.

Untuk mengatasi kesalahan ini, pilih Recrive ke tujuan kustom saat Anda memulai redrive. Kemudian, masukkan ARN antrian Amazon SQS untuk memindahkan semua pesan dari DLQ ke antrian tujuan.

Contoh pesan kesalahan:

```
Failed: CouldNotDetermineMessageSource
```

Memecahkan masalah pembatasan FIFO di Amazon SQS

Secara default, antrian FIFO mendukung 300 transaksi per detik, per tindakan API untuk, `SendMessage`, `ReceiveMessage`dan. `DeleteMessage` Permintaan lebih dari 300 TPS mendapatkan `ThrottlingException` kesalahan bahkan jika pesan dalam antrian tersedia. Untuk mengurangi ini, Anda dapat menggunakan metode berikut:

- Mengaktifkan throughput tinggi untuk antrian FIFO di Amazon SQS.
- Gunakan tindakan batch Amazon SQS API `SendMessageBatch``DeleteMessageBatch`, dan `ChangeMessageVisibilityBatch` untuk meningkatkan batas TPS hingga 3.000 pesan per detik per tindakan API, dan untuk mengurangi biaya. Untuk `ReceiveMessage` API, atur `MaxNumberofMessages` parameter untuk menerima hingga sepuluh pesan per transaksi. Untuk informasi selengkapnya, lihat Tindakan batch Amazon SQS.
- Untuk antrian FIFO dengan throughput tinggi, ikuti rekomendasi untuk mengoptimalkan pemanfaatan partisi. Kirim pesan dengan grup pesan yang sama IDs dalam batch. Hapus pesan, atau ubah nilai batas waktu visibilitas pesan dalam batch dengan pegangan tanda terima dari permintaan API yang sama`ReceiveMessage`.
- Tingkatkan jumlah `MessageGroupId`nilai unik. Hal ini memungkinkan distribusi merata di seluruh partisi antrian FIFO. Untuk informasi selengkapnya, lihat Menggunakan ID grup pesan Amazon SQS.

Untuk informasi selengkapnya, lihat Mengapa antrean Amazon SQS FIFO saya tidak menampilkan semua pesan atau pesan di grup pesan lain? dalam Panduan Pusat AWS Pengetahuan.

Memecahkan masalah pesan yang tidak dikembalikan untuk panggilan Amazon SQS API ReceiveMessage

Topik berikut mencakup penyebab paling umum mengapa pesan Amazon SQS mungkin tidak dikembalikan ke konsumen, dan cara memecahkan masalah mereka. Untuk informasi selengkapnya, lihat Mengapa saya tidak dapat menerima pesan dari antrian Amazon SQS saya? dalam Panduan Pusat AWS Pengetahuan.

Antrian kosong

Untuk menentukan apakah antrian kosong, gunakan polling panjang untuk memanggil API. `ReceiveMessage` Anda juga dapat

menggunakanApproximateNumberOfMessagesVisible,ApproximateNumberOfMessagesNotVisib dan ApproximateNumberOfMessagesDelayed CloudWatch metrik. Jika semua nilai metrik diatur ke 0 selama beberapa menit, antrian dianggap kosong.

Dalam batas penerbangan tercapai

Jika Anda menggunakan polling panjang dan jika antrian dalam batas penerbangan (120000 secara default) dilanggar, Amazon SQS tidak akan menampilkan pesan kesalahan yang melebihi batas kuota.

Penundaan pesan

Jika antrean Amazon SQS dikonfigurasi sebagai antrean penundaan, atau pesan dikirim dengan pengatur waktu pesan, maka pesan tidak akan terlihat hingga waktu tunda berakhir. Untuk memverifikasi apakah antrian dikonfigurasi sebagai antrean penundaan, gunakan **`DelaySeconds`** atribut **`GetQueueAttributes`**API, atau dari konsol antrian di bawah Penundaan pengiriman. Periksa `ApproximateNumberOfMessagesDelayed` CloudWatch metrik untuk memahami apakah ada pesan yang tertunda.

Pesan dalam penerbangan

Jika konsumen lain telah melakukan polling pesan, pesan akan dalam penerbangan atau tidak terlihat untuk periode batas waktu visibilitas. Jajak pendapat tambahan mungkin mengembalikan penerimaan kosong. Periksa ApproximateNumberOfMessagesVisible CloudWatchmetrik untuk memahami jumlah pesan yang tersedia untuk diterima. Dalam kasus antrian FIFO, jika pesan dengan ID grup pesan sedang dalam penerbangan, maka tidak ada lagi pesan yang akan dikembalikan kecuali Anda menghapus pesan, atau menjadi terlihat. Ini karena pengurutan pesan dipertahankan pada tingkat grup pesan dalam antrian FIFO.

Metode polling

Jika Anda menggunakan polling singkat, (WaitTimeSecondsadalah 0) Amazon SQS mengambil sampel subset dari servernya, dan mengembalikan pesan hanya dari server tersebut. Oleh karena itu, Anda mungkin tidak mendapatkan pesan bahkan jika mereka tersedia untuk diterima. Permintaan jajak pendapat berikutnya akan mengembalikan pesan.

Jika Anda menggunakan polling panjang, Amazon SQS polling semua server dan mengirimkan respons setelah mengumpulkan setidaknya satu pesan yang tersedia, dan hingga jumlah maksimum

yang ditentukan. Jika nilai untuk ReceiveMessage WaitTimeSecondsterlalu rendah, Anda mungkin tidak menerima semua pesan yang tersedia.

Memecahkan masalah kesalahan jaringan Amazon SQS

Topik berikut mencakup penyebab paling umum untuk masalah jaringan di Amazon SQS, dan cara memecahkan masalah mereka.

ETIMEOUT error

ETIMEOUTKesalahan terjadi ketika klien tidak dapat membuat koneksi TCP ke titik akhir Amazon SQS.

Pemecahan masalah:

- Periksa koneksi jaringan

  Uji koneksi jaringan Anda ke Amazon SQS dengan menjalankan perintah seperti. `telnet`

  Example: telnet sqs.us-east-1.amazonaws.com 443
- Periksa pengaturan jaringan
  - Pastikan aturan firewall lokal, rute, dan daftar kontrol akses (ACLs) mengizinkan lalu lintas di port yang Anda gunakan.
  - Aturan keluar (keluar) grup keamanan harus mengizinkan lalu lintas ke port 80 atau 443.
  - Aturan keluar (jalan keluar) ACL jaringan harus mengizinkan lalu lintas ke port TCP 80 atau 443.
  - Aturan inbound (ingress) ACL jaringan harus mengizinkan lalu lintas pada port TCP 1024-65535.
  - Instans Amazon Elastic Compute Cloud (Amazon EC2) yang terhubung ke internet publik harus memiliki konektivitas internet.
- Titik akhir Amazon Virtual Private Cloud (Amazon VPC)

  Jika Anda mengakses Amazon SQS melalui titik akhir VPC Amazon, grup keamanan titik akhir harus mengizinkan lalu lintas masuk ke grup keamanan klien di port 443. ACL jaringan yang terkait dengan subnet titik akhir VPC harus memiliki konfigurasi ini:
  - Aturan keluar (jalan keluar) ACL jaringan harus mengizinkan lalu lintas pada port TCP 1024-65535 (port sementara).
  - Aturan inbound (ingress) ACL jaringan harus mengizinkan lalu lintas di port 443.

Selain itu, kebijakan titik akhir Amazon SQS VPC AWS Identity and Access Management (IAM) harus mengizinkan akses. Contoh kebijakan titik akhir VPC berikut menetapkan bahwa pengguna *MyUser* IAM diizinkan mengirim pesan ke antrian Amazon SQS. *MyQueue* Tindakan lain, pengguna IAM, dan sumber daya Amazon SQS ditolak aksesnya melalui titik akhir 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

UnknownHostExceptionKesalahan terjadi ketika alamat IP host tidak dapat ditentukan.

Pemecahan masalah:

Gunakan nslookup utilitas untuk mengembalikan alamat IP yang terkait dengan nama host:

- Windows and Linux OS

  ```
  nslookup sqs.<region>.amazonaws.com
  ```

- AWS CLI atau SDK untuk titik akhir lama Python:

  ```
  nslookup <region>.queue.amazonaws.com
  ```

Jika Anda menerima output yang gagal, ikuti instruksi di Bagaimana cara kerja DNS dan bagaimana cara memecahkan masalah kegagalan DNS sebagian atau intermiten? dalam Panduan Pusat AWS Pengetahuan.

Jika Anda menerima output yang valid, maka kemungkinan akan menjadi masalah tingkat aplikasi. Untuk mengatasi masalah tingkat aplikasi, coba metode berikut:

- Mulai ulang aplikasi Anda.

- Konfirmasikan bahwa aplikasi Java Anda tidak memiliki cache DNS yang buruk. Jika memungkinkan, konfigurasikan aplikasi Anda untuk mematuhi DNS TTL. Untuk informasi selengkapnya, lihat Mengatur TTL JVM untuk pencarian nama DNS.

Untuk informasi tambahan tentang cara memecahkan masalah kesalahan jaringan, lihat Bagaimana cara mengatasi kesalahan koneksi “ETIMEOUT” dan “” Amazon SQS? UnknownHostException dalam Panduan Pusat AWS Pengetahuan.

Memecahkan masalah antrian Amazon Simple Queue Service menggunakan AWS X-Ray

AWS X-Ray mengumpulkan data tentang permintaan yang disajikan aplikasi Anda dan memungkinkan Anda melihat dan memfilter data untuk mengidentifikasi potensi masalah dan peluang untuk pengoptimalan. Untuk setiap permintaan yang dilacak ke aplikasi Anda, Anda dapat melihat informasi terperinci tentang permintaan, respons, dan panggilan yang dilakukan aplikasi Anda ke AWS sumber daya hilir, layanan mikro, database, dan web HTTP. APIs

Untuk mengirim header AWS X-Ray jejak melalui Amazon SQS, Anda dapat melakukan salah satu hal berikut:

- Gunakan header `X-Amzn-Trace-Id` tracing.
- Gunakan atribut sistem `AWSTraceHeader` pesan.

Untuk mengumpulkan data tentang kesalahan dan latensi, Anda harus instrumen `AmazonSQS`klien menggunakan AWS X-Ray SDK.

Anda dapat menggunakan AWS X-Ray konsol untuk melihat peta koneksi antara Amazon SQS dan layanan lain yang digunakan aplikasi Anda. Anda juga dapat menggunakan konsol tersebut untuk melihat metrik seperti tingkat latensi dan kegagalan rata-rata. Untuk informasi selengkapnya, lihat Amazon SQS dan AWS X-Ray di Panduan AWS X-Ray Pengembang.

Keamanan di Amazon SQS

Bagian ini menyediakan informasi tentang keamanan Amazon SQS, otentikasi dan kontrol akses, dan Bahasa Kebijakan Akses Amazon SQS.

Topik

- Perlindungan data di Amazon SQS
- Identitas dan manajemen akses di Amazon SQS
- Pencatatan dan pemantauan di Amazon SQS
- Validasi kepatuhan untuk Amazon SQS
- Ketahanan di Amazon SQS
- Keamanan infrastruktur di Amazon SQS
- Praktik terbaik keamanan Amazon SQS

Perlindungan data di Amazon SQS

Model tanggung jawab AWS bersama model berlaku untuk perlindungan data di Amazon Simple Queue Service. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Untuk informasi selengkapnya tentang privasi data, lihat FAQ Privasi Data AWS . Untuk informasi tentang perlindungan data di Eropa, lihat Pusat Peraturan Perlindungan Data Umum (GDPR).

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensyal dan mengatur pengguna individu dengan AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan AWS sumber daya. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.

- Siapkan API dan logging aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan CloudTrail jejak untuk menangkap AWS aktivitas, lihat Bekerja dengan CloudTrail jejak di AWS CloudTrail Panduan Pengguna.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-3 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-3.

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan Amazon SQS atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDK. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Bagian berikut memberikan informasi tentang perlindungan data di Amazon SQS.

Enkripsi data di Amazon SQS

Perlindungan data mengacu pada melindungi data saat dalam perjalanan (saat bepergian ke dan dari Amazon SQS) dan saat istirahat (saat disimpan di disk di pusat data Amazon SQS). Anda dapat melindungi data saat transit menggunakan Secure Socket Layer (SSL) atau enkripsi di sisi klien. Secara default, Amazon SQS menyimpan pesan dan file menggunakan enkripsi disk. Anda dapat melindungi data saat istirahat dengan meminta Amazon SQS untuk mengenkripsi pesan Anda sebelum menyimpannya ke sistem file terenkripsi di pusat datanya. Amazon SQS merekomendasikan penggunaan SSE untuk enkripsi data yang dioptimalkan.

Topik

- Enkripsi saat istirahat di Amazon SQS
- Manajemen Kunci Amazon SQS

Enkripsi saat istirahat di Amazon SQS

Server-side Enkripsi (SSE) memungkinkan Anda mengirimkan data sensitif dalam antrian terenkripsi. SSE melindungi isi pesan dalam antrian menggunakan kunci SQS-managed enkripsi (SSE-SQS) atau kunci yang dikelola di (). AWS Key Management Service SSE-KMS Untuk informasi tentang mengelola SSE menggunakan Konsol Manajemen AWS, lihat berikut ini:

- Mengkonfigurasi SSE-SQS antrian (konsol)
- Mengkonfigurasi SSE-KMS antrian (konsol)

Untuk informasi tentang mengelola SSE menggunakan AWS SDK untuk Java (dan`CreateQueue`,`SetQueueAttributes`, dan `GetQueueAttributes` tindakan), lihat contoh berikut:

- Menggunakan enkripsi sisi server dengan antrian Amazon SQS
- Mengkonfigurasi izin KMS untuk Layanan AWS

SSE mengenkripsi pesan segera setelah Amazon SQS menerimanya. Pesan disimpan dalam bentuk terenkripsi dan Amazon SQS mendekripsi pesan hanya ketika dikirim ke konsumen yang berwenang.

> **Important**
>
> Semua permintaan untuk antrian dengan SSE diaktifkan harus menggunakan HTTPS dan Signature Version 4.
> Antrian terenkripsi yang menggunakan kunci default (kunci KMS AWS terkelola untuk Amazon SQS) tidak dapat menjalankan fungsi Lambda secara berbeda. Akun AWS Beberapa fitur AWS layanan yang dapat mengirim pemberitahuan ke Amazon SQS menggunakan AWS Security Token Service `AssumeRole` tindakan kompatibel dengan SSE tetapi hanya berfungsi dengan antrian standar:
>
> - Kait Siklus Hidup Auto Scaling
> - AWS Lambda Dead-LetterAntrian
>
> Untuk informasi tentang kompatibilitas layanan lain dengan antrian terenkripsi, lihat Konfigurasikan izin KMS untuk AWS layanan dan dokumentasi layanan Anda.

AWS KMS menggabungkan perangkat keras dan perangkat lunak yang aman dan sangat tersedia untuk menyediakan sistem manajemen kunci yang diskalakan untuk cloud. Saat Anda menggunakan Amazon SQS AWS KMS, kunci data yang mengenkripsi data pesan Anda juga dienkripsi dan disimpan dengan data yang dilindunginya.

Berikut ini adalah manfaat menggunakan AWS KMS:

- Anda dapat membuat dan mengelola AWS KMS keyssendiri.
- Anda juga dapat menggunakan kunci KMS AWS terkelola untuk Amazon SQS, yang unik untuk setiap akun dan wilayah.
- Standar AWS KMS keamanan dapat membantu Anda memenuhi persyaratan kepatuhan terkait enkripsi.

Untuk informasi lebih lanjut, lihat Apa yang dimaksud AWS Key Management Service? dalam Panduan Developer AWS Key Management Service .

Lingkup enkripsi

SSE mengenkripsi isi pesan dalam antrian Amazon SQS.

SSE tidak mengenkripsi berikut ini:

- Metadata antrian (nama antrian dan atribut)
- Metadata pesan (ID pesan, stempel waktu, dan atribut)
- Per-queue metrik

Mengenkripsi pesan membuat isinya tidak tersedia untuk pengguna yang tidak sah atau anonim. Dengan SSE diaktifkan, anonim `SendMessage` dan `ReceiveMessage` permintaan ke antrian terenkripsi akan ditolak. Praktik terbaik keamanan Amazon SQS merekomendasikan agar tidak menggunakan permintaan anonim. Jika Anda ingin mengirim permintaan anonim ke antrian Amazon SQS, pastikan Anda menonaktifkan SSE. Ini tidak memengaruhi fungsi normal Amazon SQS:

- Pesan dienkripsi hanya jika dikirim setelah enkripsi antrian diaktifkan. Amazon SQS tidak mengenkripsi pesan backlog.
- Setiap pesan terenkripsi tetap dienkripsi bahkan jika enkripsi antreannya dinonaktifkan.

Memindahkan pesan ke antrian huruf mati tidak memengaruhi enkripsi:

- Saat Amazon SQS memindahkan pesan dari antrian sumber terenkripsi ke antrian huruf mati yang tidak terenkripsi, pesan tersebut tetap terenkripsi.
- Saat Amazon SQS memindahkan pesan dari antrian sumber yang tidak terenkripsi ke antrian huruf mati terenkripsi, pesan tetap tidak terenkripsi.

Istilah kunci

Istilah kunci berikut ini dapat membantu Anda lebih memahami fungsionalitas SSE. Untuk deskripsi mendetail, lihat Referensi API Layanan Antrian Sederhana Amazon.

Kunci data

Kunci (DEK) bertanggung jawab untuk mengenkripsi konten pesan Amazon SQS.

Untuk informasi selengkapnya, lihat Kunci Data di Panduan AWS Key Management Service Pengembang di Panduan AWS Encryption SDK Pengembang.

Periode penggunaan kembali kunci data

Lamanya waktu, dalam hitungan detik, Amazon SQS dapat menggunakan kembali kunci data untuk mengenkripsi atau mendekripsi pesan sebelum menelepon lagi. AWS KMS Bilangan bulat yang mewakili detik, antara 60 detik (1 menit) dan 86.400 detik (24 jam). Defaultnya adalah 300 (5 menit). Untuk informasi selengkapnya, lihat Memahami periode penggunaan kembali kunci data.

> **Note**
>
> Jika tidak dapat menjangkau AWS KMS, Amazon SQS terus menggunakan kunci data yang di-cache hingga koneksi dibuat kembali.

ID kunci KMS

Alias, alias ARN, ID kunci, atau ARN kunci dari kunci KMS AWS terkelola atau kunci KMS kustom —di akun Anda atau di akun lain. Sementara alias kunci KMS AWS terkelola untuk Amazon SQS `alias/aws/sqs` selalu, alias kunci KMS kustom dapat, misalnya,. `alias/MyAlias` Anda dapat menggunakan kunci KMS ini untuk melindungi pesan di antrian Amazon SQS.

> **Note**
>
> Ingatlah hal-hal berikut ini:

- Jika Anda tidak menentukan kunci KMS kustom, Amazon SQS menggunakan kunci KMS AWS terkelola untuk Amazon SQS.
- Pertama kali Anda menggunakan Konsol Manajemen AWS untuk menentukan kunci KMS AWS terkelola untuk Amazon SQS untuk antrean AWS KMS , membuat AWS kunci KMS terkelola untuk Amazon SQS.
- Atau, saat pertama kali Anda menggunakan `SendMessageBatch` tindakan `SendMessage` atau pada antrean dengan SSE diaktifkan, AWS KMS membuat kunci KMS AWS terkelola untuk Amazon SQS.

Anda dapat membuat kunci KMS, menentukan kebijakan yang mengontrol bagaimana kunci KMS dapat digunakan, dan mengaudit penggunaan kunci KMS menggunakan bagian Kunci terkelola Pelanggan di AWS KMS konsol atau tindakan. `CreateKey` AWS KMS Untuk informasi selengkapnya, lihat kunci KMS dan Membuat Kunci di Panduan AWS Key Management Service Pengembang. Untuk lebih banyak contoh pengidentifikasi kunci KMS, lihat KeyIddi Referensi AWS Key Management Service API. Untuk informasi tentang menemukan pengidentifikasi kunci KMS, lihat Menemukan ID Kunci dan ARN di Panduan Pengembang.AWS Key Management Service

> **Important**
>
> Ada biaya tambahan untuk penggunaan AWS KMS. Untuk informasi selengkapnya, lihat Memperkirakan AWS KMS ongkos dan Harga AWS Key Management Service.

Enkripsi Amplop

Keamanan data terenkripsi Anda sebagian bergantung pada perlindungan kunci data yang dapat mendekripsi itu. Amazon SQS menggunakan kunci KMS untuk mengenkripsi kunci data dan kemudian kunci data terenkripsi disimpan dengan pesan terenkripsi. Praktik menggunakan kunci KMS untuk mengenkripsi kunci data ini dikenal sebagai enkripsi amplop.

Untuk informasi selengkapnya, lihat Enkripsi envelope di Panduan Developer AWS Encryption SDK .

Manajemen Kunci Amazon SQS

Amazon SQS terintegrasi dengan AWS Key Management Service (KMS) untuk mengelola kunci KMS untuk enkripsi sisi server (SSE). Lihat Enkripsi saat istirahat di Amazon SQS untuk informasi SSE dan definisi manajemen kunci. Amazon SQS menggunakan kunci KMS untuk memvalidasi dan mengamankan kunci data yang mengenkripsi dan mendekripsi pesan. Bagian berikut memberikan informasi tentang bekerja dengan kunci KMS dan kunci data di layanan Amazon SQS.

Melakukan konfigurasi AWS KMS izin

Setiap kunci KMS harus memiliki kebijakan kunci. Perhatikan bahwa Anda tidak dapat mengubah kebijakan kunci kunci KMS AWS terkelola untuk Amazon SQS. Kebijakan untuk kunci KMS ini mencakup izin untuk semua prinsipal di akun (yang diizinkan untuk menggunakan Amazon SQS) untuk menggunakan antrian terenkripsi.

Amazon SQS membedakan antara penelepon berdasarkan AWS kredensialnya, apakah mereka menggunakan AWS akun yang berbeda, pengguna IAM, atau peran IAM. Selain itu, peran IAM yang sama dengan kebijakan pelingkupan yang berbeda akan diperlakukan sebagai pemohon yang berbeda. Namun, saat menggunakan sesi peran IAM, hanya memvariasikan `RoleSessionName` sementara mempertahankan peran IAM dan kebijakan pelingkupan yang sama tidak akan membuat pemohon yang berbeda. Oleh karena itu, ketika menentukan prinsip kebijakan AWS KMS utama, hindari mengandalkan `RoleSessionName` perbedaan saja, karena sesi ini akan diperlakukan sebagai pemohon yang sama.

Atau, Anda dapat menentukan izin yang diperlukan dalam kebijakan IAM yang ditetapkan ke prinsipal yang menghasilkan dan menggunakan pesan terenkripsi. Untuk informasi selengkapnya, lihat Menggunakan Kebijakan IAM dengan AWS KMS Panduan AWS Key Management Service Pengembang.

> **Note**
>
> Meskipun Anda dapat mengonfigurasi izin global untuk mengirim dan menerima dari Amazon SQS AWS KMS , memerlukan penamaan ARN lengkap kunci KMS secara eksplisit di wilayah tertentu di bagian kebijakan IAM. `Resource`

Konfigurasikan izin KMS untuk AWS layanan

Beberapa AWS layanan bertindak sebagai sumber acara yang dapat mengirim acara ke antrian Amazon SQS. Agar sumber peristiwa ini dapat bekerja dengan antrian terenkripsi, Anda harus

membuat kunci KMS yang dikelola pelanggan dan menambahkan izin dalam kebijakan kunci agar layanan dapat menggunakan metode API yang diperlukan. AWS KMS Lakukan langkah-langkah berikut untuk mengonfigurasi izin.

> **Warning**
>
> Saat mengubah kunci KMS untuk mengenkripsi pesan Amazon SQS Anda, ketahuilah bahwa pesan yang ada yang dienkripsi dengan kunci KMS lama akan tetap dienkripsi dengan kunci itu. Untuk mendekripsi pesan ini, Anda harus menyimpan kunci KMS lama dan memastikan bahwa kebijakan utamanya memberikan Amazon SQS izin untuk dan. `kms:Decrypt` `kms:GenerateDataKey` Setelah memperbarui ke kunci KMS baru untuk mengenkripsi pesan baru, pastikan semua pesan yang ada dienkripsi dengan kunci KMS lama diproses dan dihapus dari antrian sebelum menghapus atau menonaktifkan kunci KMS lama.

1. Buat kunci KMS yang dikelola pelanggan. Untuk informasi selengkapnya, lihat Membuat Kunci di Panduan Developer AWS Key Management Service .
2. Untuk mengizinkan sumber peristiwa AWS layanan menggunakan metode `kms:Decrypt` dan `kms:GenerateDataKey` API, tambahkan pernyataan berikut ke kebijakan kunci KMS.

   JSON

```
{
   "Version":"2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": "service.amazonaws.com"
         },
         "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
         ],
         "Resource": "*"
       }]
}
```

Ganti “layanan” pada contoh di atas dengan nama Layanan dari sumber acara. Sumber acara termasuk layanan berikut.

Sumber peristiwa	Nama layanan
CloudWatch Acara Amazon	`events.amazonaws.com`
Pemberitahuan acara Amazon S3	`s3.amazonaws.com`
Langganan topik Amazon SNS	`sns.amazonaws.com`

3. Konfigurasikan antrian SSE yang ada menggunakan ARN kunci KMS Anda.
4. Berikan ARN dari antrian terenkripsi ke sumber acara.

Konfigurasi AWS KMS izin untuk produsen

Ketika periode penggunaan kembali kunci data berakhir, panggilan berikutnya produsen ke `SendMessage` atau `SendMessageBatch` juga memicu panggilan ke dan. `kms:Decrypt` `kms:GenerateDataKey` Panggilan `kms:Decrypt` untuk memverifikasi integritas kunci data baru sebelum menggunakannya. Oleh karena itu, produsen harus memiliki `kms:Decrypt` dan `kms:GenerateDataKey` izin untuk kunci KMS.

Tambahkan pernyataan berikut ke kebijakan IAM produsen. Ingatlah untuk menggunakan nilai ARN yang benar untuk sumber daya kunci dan sumber daya antrian.

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"
        }
    ]
}
```

Konfigurasi AWS KMS izin untuk konsumen

Ketika periode penggunaan kembali kunci data berakhir, panggilan konsumen berikutnya `ReceiveMessage` juga memicu panggilan ke`kms:Decrypt`, untuk memverifikasi integritas kunci data baru sebelum menggunakannya. Oleh karena itu, konsumen harus memiliki `kms:Decrypt` izin untuk kunci KMS apa pun yang digunakan untuk mengenkripsi pesan dalam antrian yang ditentukan. Jika antrian bertindak sebagai antrian huruf mati, konsumen juga harus memiliki `kms:Decrypt` izin untuk kunci KMS apa pun yang digunakan untuk mengenkripsi pesan dalam antrian sumber. Tambahkan pernyataan berikut ke kebijakan IAM konsumen. Ingatlah untuk menggunakan nilai ARN yang benar untuk sumber daya kunci dan sumber daya antrian.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/111112222233333"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
```

```
    ]
}
```

Konfigurasi AWS KMS izin dengan perlindungan wakil yang membingungkan

Ketika prinsipal dalam pernyataan kebijakan kunci adalah prinsipal AWS layanan, Anda dapat menggunakan `aws:SourceArn`atau kunci kondisi `aws:SourceAccount`global untuk melindungi dari skenario wakil yang membingungkan. Untuk menggunakan kunci kondisi ini, tetapkan nilai ke Amazon Resource Name (ARN) dari sumber daya yang sedang dienkripsi. Jika Anda tidak tahu ARN sumber daya, gunakan `aws:SourceAccount` sebagai gantinya.

Dalam kebijakan kunci KMS ini, sumber daya tertentu dari layanan yang dimiliki oleh akun `111122223333` diizinkan untuk memanggil KMS untuk `Decrypt` dan `GenerateDataKey` tindakan, yang terjadi selama penggunaan SSE 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"
                    ]
                }
            }
        }
    ]
}
```

Saat menggunakan antrian Amazon SQS yang diaktifkan SSE, layanan berikut mendukung: `aws:SourceArn`

- Amazon SNS
- Amazon S3
- CloudWatch Acara
- AWS Lambda
- CodeBuild
- Connect Profil Pelanggan Pelanggan
- AWS Auto Scaling
- Amazon Chime

Memahami periode penggunaan kembali kunci data

Periode penggunaan kembali kunci data menentukan durasi maksimum Amazon SQS untuk menggunakan kembali kunci data yang sama. Ketika periode penggunaan kembali kunci data berakhir, Amazon SQS menghasilkan kunci data baru. Perhatikan panduan berikut tentang periode penggunaan kembali.

- Periode penggunaan kembali yang lebih pendek memberikan keamanan yang lebih baik tetapi menghasilkan lebih banyak panggilan ke AWS KMS, yang mungkin dikenakan biaya di luar Tingkat Gratis.
- Meskipun kunci data di-cache secara terpisah untuk enkripsi dan untuk dekripsi, periode penggunaan kembali berlaku untuk kedua salinan kunci data.
- Ketika periode penggunaan kembali kunci data berakhir, panggilan berikutnya ke `SendMessage` atau `SendMessageBatch` biasanya memicu panggilan ke AWS KMS `GenerateDataKey` metode untuk mendapatkan kunci data baru. Juga, panggilan berikutnya ke `SendMessage` dan masing-masing `ReceiveMessage` akan memicu panggilan AWS KMS `Decrypt` untuk memverifikasi integritas kunci data sebelum menggunakannya.
- Prinsipal (Akun AWS atau pengguna) tidak berbagi kunci data (pesan yang dikirim oleh prinsipal unik selalu mendapatkan kunci data unik). Oleh karena itu, volume panggilan ke AWS KMS adalah kelipatan dari jumlah prinsipal unik yang digunakan selama periode penggunaan kembali kunci data.

Memperkirakan AWS KMS ongkos

Untuk memprediksi biaya dan lebih memahami AWS tagihan Anda, Anda mungkin ingin tahu seberapa sering Amazon SQS menggunakan kunci KMS Anda.

> **Note**
>
> Meskipun rumus berikut dapat memberi Anda gambaran yang sangat baik tentang biaya yang diharapkan, biaya sebenarnya mungkin lebih tinggi karena sifat terdistribusi Amazon SQS.

Untuk menghitung jumlah permintaan API (R) per antrian, gunakan rumus berikut:

```
R = (B / D) * (2 * P + C)
```

B adalah periode penagihan (dalam detik).

Dadalah periode penggunaan kembali kunci data (dalam detik).

Padalah jumlah prinsipal produksi yang mengirim ke antrian Amazon SQS.

Cadalah jumlah prinsipal konsumsi yang menerima dari antrian Amazon SQS.

> **Important**
>
> Secara umum, prinsipal produksi dikenakan biaya dua kali lipat biaya konsumsi prinsipal. Untuk informasi selengkapnya, lihat Memahami periode penggunaan kembali kunci data. Jika produsen dan konsumen memiliki pengguna yang berbeda, biayanya meningkat.

Berikut ini adalah contoh perhitungan. Untuk informasi harga sebenranya, lihat Harga AWS Key Management Service.

Contoh 1: Menghitung jumlah AWS KMS API memanggil 2 prinsipal dan 1 antrian

Contoh ini mengasumsikan sebagai berikut:

- Periode penagihan adalah 1-31 Januari (2.678.400 detik).

- Periode penggunaan kembali kunci data diatur ke 5 menit (300 detik).
- Ada 1 antrian.
- Ada 1 pokok produksi dan 1 pokok konsumsi.

```
(2,678,400 / 300) * (2 * 1 + 1) = 26,784
```

Contoh 2: Menghitung jumlah AWS KMS API menyerukan beberapa produsen dan konsumen dan 2 antrian

Contoh ini mengasumsikan sebagai berikut:

- Periode penagihan adalah 1-28 Februari (2.419.200 detik).
- Periode penggunaan kembali kunci data diatur ke 24 jam (86.400 detik).
- Ada 2 antrian.
- Antrian pertama memiliki 3 prinsip produksi dan 1 pokok konsumsi.
- Antrian kedua memiliki 5 prinsip produksi dan 2 prinsip konsumsi.

```
(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532
```

AWS KMS kesalahan

Saat Anda bekerja dengan Amazon SQS dan AWS KMS, Anda mungkin mengalami kesalahan. Referensi berikut menjelaskan kesalahan dan kemungkinan solusi pemecahan masalah.

- AWS KMS Kesalahan umum
- AWS KMS Dekripsi kesalahan
- AWS KMS GenerateDataKey kesalahan

Privasi lalu lintas internetwork di Amazon SQS

Titik akhir Amazon Virtual Private Cloud (Amazon VPC) untuk Amazon SQS adalah entitas logis dalam VPC yang memungkinkan konektivitas hanya ke Amazon SQS. VPC merutekan permintaan ke Amazon SQS dan merutekan respons kembali ke VPC. Bagian berikut ini memberikan informasi tentang bekerja dengan VPC endpoint dan membuat kebijakan VPC endpoint.

Titik akhir Amazon Virtual Private Cloud untuk Amazon SQS

Jika Anda menggunakan Amazon VPC untuk meng-host AWS sumber daya Anda, Anda dapat membuat sambungan antara VPC dan Amazon SQS. Anda dapat menggunakan koneksi ini untuk mengirim pesan ke antrian Amazon SQS Anda tanpa melintasi internet publik.

Amazon VPC memungkinkan Anda meluncurkan AWS sumber daya di jaringan virtual khusus. Anda dapat menggunakan VPC untuk mengendalikan pengaturan jaringan, seperti rentang alamat IP, subnet, tabel rute, dan gateway jaringan. Untuk informasi lebih lanjut tentang Amazon VPC, lihat Panduan Pengguna Amazon VPC.

Untuk menghubungkan VPC Anda ke Amazon SQS, Anda harus terlebih dahulu menentukan titik akhir VPC antarmuka, yang memungkinkan Anda menghubungkan VPC ke layanan lain. AWS Titik akhir menyediakan konektivitas yang andal dan dapat diskalakan ke Amazon SQS tanpa memerlukan gateway internet, instance terjemahan alamat jaringan (NAT), atau koneksi VPN. Untuk informasi selengkapnya, lihat Tutorial: Mengirim pesan ke antrian Amazon SQS dari Amazon Virtual Private Cloud dan Contoh 5: Tolak akses jika bukan dari titik akhir VPC dalam panduan ini dan Titik Akhir VPC Antarmuka (AWS PrivateLink) di Panduan Pengguna Amazon VPC.

Important

- Anda dapat menggunakan Amazon Virtual Private Cloud hanya dengan titik akhir HTTPS Amazon SQS.
- Saat mengonfigurasi Amazon SQS untuk mengirim pesan dari Amazon VPC, Anda harus mengaktifkan DNS pribadi dan menentukan titik akhir dalam format `sqs.us-east-2.amazonaws.com` atau untuk titik akhir tumpukan ganda. `sqs.us-east-2.api.aws`
- Amazon SQS juga mendukung endpoint FIPS melalui PrivateLink penggunaan layanan endpoint. `com.amazonaws.region.sqs-fips` Anda dapat terhubung ke titik akhir FIPS dalam format. `sqs-fips.region.amazonaws.com`
- Saat menggunakan endpoint dual-stack di Amazon Virtual Private Cloud, permintaan akan dikirim menggunakan IPv4 dan IPv6.
- DNS pribadi tidak mendukung titik akhir lama seperti atau. `queue.amazonaws.com` `us-east-2.queue.amazonaws.com`

Membuat kebijakan titik akhir Amazon VPC untuk Amazon SQS

Anda dapat membuat kebijakan untuk titik akhir VPC Amazon untuk Amazon SQS yang Anda tentukan sebagai berikut:

- Prinsipal yang dapat melakukan tindakan.
- Tindakan yang dapat dilakukan.
- Sumber daya yang menjadi target tindakan.

Untuk informasi selengkapnya, lihat Mengontrol Akses ke Layanan dengan Titik Akhir VPC di Panduan Pengguna Amazon VPC

Contoh kebijakan titik akhir VPC berikut menetapkan bahwa pengguna `MyUser` diizinkan mengirim pesan ke antrean 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"
      }
   }]
}
```

Hal berikut ini ditolak:

- Tindakan API Amazon SQS lainnya, seperti `sqs:CreateQueue` dan. `sqs:DeleteQueue`
- Pengguna dan aturan lain yang mencoba menggunakan titik akhir VPC ini.
- `MyUser`mengirim pesan ke antrian Amazon SQS yang berbeda.

> **Note**
>
> Pengguna masih dapat menggunakan tindakan Amazon SQS API lainnya dari luar VPC. Untuk informasi selengkapnya, lihat Contoh 5: Tolak akses jika bukan dari titik akhir VPC.

Connect ke Amazon SQS menggunakan endpoint Dual-stack (IPv4 dan IPv6)

Dual-stack endpoint mendukung lalu lintas IPv4 dan IPv6. Saat Anda membuat permintaan ke titik akhir dual-stack, URL endpoint akan diselesaikan ke alamat IPv4 atau IPv6. Untuk informasi selengkapnya tentang dual-stack dan titik akhir FIPS, lihat panduan Referensi SDK.

Amazon SQS mendukung titik akhir dual-stack Regional, yang berarti Anda harus menentukan AWS Wilayah sebagai bagian dari nama titik akhir. Dual-stack nama endpoint menggunakan konvensi penamaan berikut:`sqs.Region.amazonaws.com`. Misalnya, nama titik akhir tumpukan ganda untuk Wilayah `eu-west-1` adalah `sqs.eu-west-1.amazonaws.com`.

Untuk daftar lengkap titik akhir Amazon SQS, lihat Referensi Umum.AWS

Identitas dan manajemen akses di Amazon SQS

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan di otorisasi (memiliki izin) untuk menggunakan sumber daya Amazon SQS. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda berdasarkan peran Anda:

- Pengguna layanan - minta izin dari administrator Anda jika Anda tidak dapat mengakses fitur (lihat Memecahkan masalah identitas dan akses Amazon Simple Queue Service)
- Administrator layanan - tentukan akses pengguna dan mengirimkan permintaan izin (lihat Cara kerja Amazon Simple Queue Service dengan IAM)
- Administrator IAM - tulis kebijakan untuk mengelola akses (lihat Praktik terbaik kebijakan)

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredentif identitas Anda. Anda harus diautentikasi sebagai Pengguna root akun AWS, pengguna IAM, atau dengan mengambil peran IAM.

Anda dapat masuk sebagai identitas federasi menggunakan kredenSIAL dari sumber identitas seperti AWS IAM Identity Center (Pusat Identitas IAM), otentikasi masuk tunggal, atau kredenSIAL. Google/ Facebook Untuk informasi selengkapnya tentang cara masuk, lihat Cara masuk ke Akun AWS Anda dalam Panduan Pengguna AWS Sign-In .

Untuk akses terprogram, AWS menyediakan SDK dan CLI untuk menandatangani permintaan kriptografis. Untuk informasi selengkapnya, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang disebut pengguna Akun AWS root yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Untuk tugas yang memerlukan kredensial pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS menggunakan kredenSIAL sementara.

Identi tas federasi adalah pengguna dari direktori perusahaan Anda, penyedia identitas web, atau Directory Service yang mengakses Layanan AWS menggunakan kredenSIAL dari sumber identitas. Identitas terfederasi mengambil peran yang memberikan kredensial sementara.

Untuk manajemen akses terpusat, kami menyarankan AWS IAM Identity Center. Untuk informasi selengkapnya, lihat Apa itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dengan izin khusus untuk satu orang atau aplikasi. Sebaiknya gunakan kredensial sementara alih-alih pengguna IAM dengan kredensial jangka panjang. Untuk informasi selengkapnya, lihat Memerlukan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS menggunakan kredenSIAL sementara di Panduan Pengguna IAM.

Grup IAM menentukan kumpulan pengguna IAM dan mempermudah pengelolaan izin untuk pengguna dalam jumlah besar. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dengan izin khusus yang menyediakan kredensial sementara. Anda dapat mengambil peran dengan beralih dari pengguna ke peran IAM (konsol) atau dengan memanggil operasi AWS CLI AWS API. Untuk informasi selengkapnya, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM berguna untuk akses pengguna terfederasi, izin pengguna IAM sementara, akses lintas akun, akses lintas layanan, dan aplikasi yang berjalan di Amazon EC2. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan menentukan izin saat dikaitkan dengan identitas atau sumber daya. AWS mengevaluasi kebijakan ini ketika kepala sekolah membuat permintaan. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Menggunakan kebijakan, administrator menentukan siapa yang memiliki akses ke apa dengan mendefinisikan principal mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Administrator IAM membuat kebijakan IAM dan menambahkannya ke peran, yang kemudian dapat diambil oleh pengguna. Kebijakan IAM mendefinisikan izin terlepas dari metode yang Anda gunakan untuk melakukan operasinya.

Identity-based kebijakan

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang Anda lampirkan ke identitas (pengguna, grup, atau peran). Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, terhadap sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan yang dikelola pelanggan dalam Panduan Pengguna IAM.

Identity-based kebijakan dapat berupa kebijakan sebaris (disematkan langsung ke dalam satu identitas) atau kebijakan ter kelola (kebijakan mandiri yang dilampirkan ke beberapa identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat Pilih antara kebijakan terkelola dan kebijakan inline dalam Panduan Pengguna IAM.

Resource-based kebijakan

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contohnya termasuk kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Anda harus menentukan principal dalam kebijakan berbasis sumber daya.

Resource-based kebijakan adalah kebijakan sebaris yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang dapat mengatur izin maksimum yang diberikan oleh jenis kebijakan yang lebih umum:

- Batasan izin – Menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM. Untuk informasi selengkapnya, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCP) – Menentukan izin maksimum untuk organisasi atau unit organisasi di AWS Organizations. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam Panduan Pengguna AWS Organizations .
- Kebijakan kontrol sumber daya (RCP) – Menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Kebijakan kontrol sumber daya (RCP) dalam Panduan Pengguna AWS Organizations .
- Kebijakan sesi – Kebijakan lanjutan yang diteruskan sebagai parameter saat membuat sesi sementara untuk peran atau pengguna terfederasi. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan ketika beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Ikhtisar pengelolaan akses di Amazon SQS

Setiap AWS sumber daya dimiliki oleh Akun AWS, dan izin untuk membuat atau mengakses sumber daya diatur oleh kebijakan izin. Administrator akun dapat melampirkan kebijakan izin ke identitas IAM (pengguna, grup, dan peran), dan beberapa layanan (seperti Amazon SQS) juga mendukung melampirkan kebijakan izin ke sumber daya.

> **Note**
>
> Administrator akun (atau pengguna administrator) adalah pengguna dengan hak administratif. Untuk informasi selengkapnya, lihat Praktik Terbaik IAM dalam Panduan Pengguna IAM.

Saat memberikan izin, Anda menentukan pengguna apa yang mendapatkan izin, sumber daya yang mereka dapatkan izin, dan tindakan spesifik yang ingin Anda izinkan pada sumber daya.

Sumber daya dan operasi Amazon Simple Queue Service

Di Amazon SQS, satu-satunya sumber daya adalah antri an. Dalam kebijakan, gunakan Nama Sumber Daya Amazon (ARN) untuk mengidentifikasi sumber daya yang diterapkan kebijakan tersebut. Sumber daya berikut memiliki ARN unik yang terkait dengannya:

Tipe sumber daya	Format ARN
Antrean	`arn:aws:sqs:` *`region`*`:`*`account_id`*`:`*`queue_name`*

Berikut ini adalah contoh format ARN untuk antrian:

- ARN untuk antrian yang dinam `my_queue` ai di wilayah AS Timur (Ohio), milik AWS Akun 123456789012:

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- ARN untuk antrian yang `my_queue` dinamai di masing-masing wilayah berbeda yang didukung Amazon SQS:

```
arn:aws:sqs:*:123456789012:my_queue
```

- ARN yang menggunakan * atau ? sebagai wildcard untuk nama antrian. Dalam contoh berikut, ARN mencocokkan semua antrian yang diawali dengan: `my_prefix_`

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

Anda bisa mendapatkan nilai ARN untuk antrian yang ada dengan memanggil `GetQueueAttributes` tindakan. Nilai `QueueArn` atribut adalah ARN dari antrian. Untuk informasi selengkapnya tentang ARN, lihat ARN IAM di Panduan Pengguna IAM.

Amazon SQS menyediakan serangkaian tindakan yang bekerja dengan sumber daya antrian. Untuk informasi selengkapnya, lihat Izin API Amazon SQS: Tindakan dan referensi sumber daya.

Memahami kepemilikan sumber daya

Memiliki Akun AWS sumber daya yang dibuat di akun, terlepas dari siapa yang membuat sumber daya. Secara khusus, pemilik sumber daya adalah entitas utama (yaitu, akun root, pengguna, atau peran IAM) yang mengotentikasi permintaan pembuatan sumber daya. Akun AWS Contoh berikut menggambarkan cara kerjanya:

- Jika Anda menggunakan kredentif akun root Akun AWS untuk membuat antrian Amazon SQS, Anda Akun AWS adalah pemilik sumber daya (di Amazon SQS, sumber dayanya adalah antrian Amazon SQS).
- Jika Anda membuat pengguna di Anda Akun AWS dan memberikan izin untuk membuat antrian kepada pengguna, pengguna dapat membuat antrian. Namun, Anda Akun AWS (milik pengguna) memiliki sumber daya antrian.
- Jika Anda membuat peran IAM Akun AWS dengan izin untuk membuat antrian Amazon SQS, siapa pun yang dapat mengambil peran tersebut dapat membuat antrian. Anda Akun AWS (yang menjadi milik peran) memiliki sumber daya antrian.

Mengelola akses ke sumber daya

Kebijakan izin menjelaskan izin yang diberikan ke akun. Bagian berikut menjelaskan opsi yang tersedia untuk membuat kebijakan izin.

> **Note**
>
> Bagian ini membahas penggunaan IAM dalam konteks Amazon SQS. Bagian ini tidak memberikan informasi yang mendetail tentang layanan IAM. Untuk dokumentasi lengkap IAM, lihat Apa yang Dimaksud dengan IAM? dalam Panduan Pengguna IAM. Untuk informasi tentang sintaksis dan penjelasan kebijakan IAM, lihat Referensi Kebijakan IAM AWS di Panduan Pengguna IAM.

Kebijakan yang terlampir pada identitas IAM disebut kebijakan (kebijakan IAM) berbasis identitas dan kebijakan yang dilampirkan pada sumber daya disebut kebijakan berbasis sumber daya.

Identity-based kebijakan

Ada dua cara untuk memberikan izin kepada pengguna Anda ke antrian Amazon SQS Anda: menggunakan sistem kebijakan Amazon SQS dan menggunakan sistem kebijakan IAM. Anda dapat menggunakan salah satu sistem, atau keduanya, untuk melampirkan kebijakan ke pengguna atau peran. Dalam kebanyakan kasus, Anda dapat mencapai hasil yang sama menggunakan salah satu sistem. Misalnya, Anda dapat melakukan hal berikut:

- Lampirkan kebijakan izin ke pengguna atau grup di akun Anda — Untuk memberikan izin pengguna untuk membuat antrian Amazon SQS, lampirkan kebijakan izin ke pengguna atau grup tempat pengguna tersebut berada.
- Melampirkan kebijakan izin ke pengguna lain Akun AWS — Anda dapat melampirkan kebijakan izin ke pengguna lain Akun AWS untuk memungkinkan mereka berinteraksi dengan antrian Amazon SQS. Namun, izin lintas akun tidak berlaku untuk tindakan berikut:

  Cross-account izin tidak berlaku untuk tindakan berikut:

  - `AddPermission`
  - `CancelMessageMoveTask`
  - `CreateQueue`
  - `DeleteQueue`
  - `ListMessageMoveTask`
  - `ListQueues`
  - `ListQueueTags`
  - `RemovePermission`

- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Untuk memberikan akses untuk tindakan ini, pengguna harus menjadi milik yang sama Akun AWS dengan yang memiliki antrian Amazon SQS.

- Melampirkan kebijakan izin ke peran (berikan izin lintas akun) — Untuk memberikan izin lintas akun ke antrian SQS, Anda harus menggabungkan IAM dan kebijakan berbasis sumber daya:
  1. Di Akun A (yang memiliki antrian):
     - Lampirkan kebijakan berbasis sumber daya ke antrian SQS. Kebijakan ini harus secara eksplisit memberikan izin yang diperlukan (misalnya SendMessage,, ReceiveMessage) kepada prinsipal di Akun B (seperti peran IAM).
  2. Di Akun A, buat peran IAM:
     - Kebijakan kepercayaan yang memungkinkan Akun B atau an Layanan AWS untuk mengambil peran tersebut.

       > **Note**
       >
       > Jika Anda ingin Layanan AWS (seperti Lambda atauEventBridge) untuk mengambil peran, tentukan prinsip layanan (misalnya,lambda.amazonaws.com) dalam kebijakan kepercayaan.

     - Kebijakan berbasis identitas yang memberikan izin peran yang diasumsikan untuk berinteraksi dengan antrian.
  3. Di Akun B, berikan izin untuk mengambil peran di Akun A.

Anda harus mengonfigurasi kebijakan akses antrian untuk mengizinkan prinsipal lintas akun. Kebijakan berbasis identitas IAM saja tidak cukup untuk akses lintas akun ke antrian SQS.

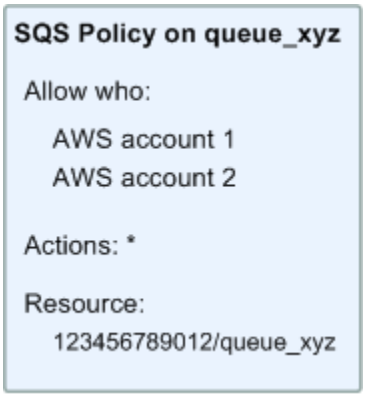
Untuk informasi selengkapnya tentang penggunaan IAM untuk mendelegasikan izin, lihat Manajemen Akses dalam Panduan Pengguna IAM.

Meskipun Amazon SQS bekerja dengan kebijakan IAM, ia memiliki infrastruktur kebijakannya sendiri. Anda dapat menggunakan kebijakan Amazon SQS dengan antrian untuk menentukan AWS Akun mana yang memiliki akses ke antrian. Anda dapat menentukan jenis akses dan kondisi (misalnya,

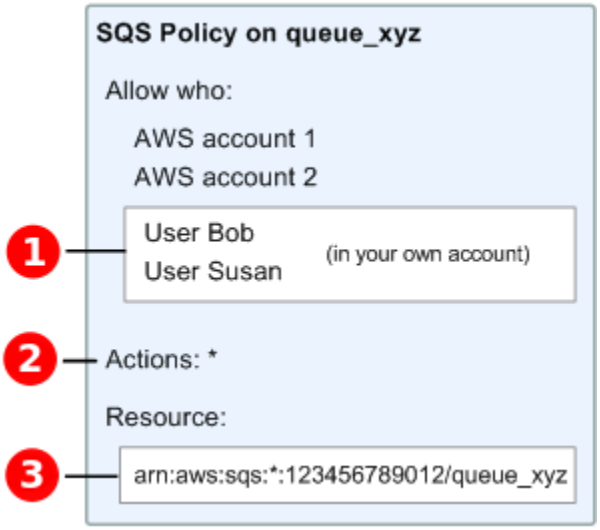
kondisi yang memberikan izin untuk digunakan`SendMessage`, `ReceiveMessage` jika permintaan dibuat sebelum 31 Desember 2010). Tindakan spesifik yang dapat Anda berikan izin adalah bagian dari daftar keseluruhan tindakan Amazon SQS. Saat Anda menulis kebijakan Amazon SQS dan menentukan * untuk “izinkan semua tindakan Amazon SQS,” itu berarti bahwa pengguna dapat melakukan semua tindakan di subset ini.

Diagram berikut menggambarkan konsep salah satu kebijakan dasar Amazon SQS yang mencakup subset tindakan. Kebijakan ini untuk`queue_xyz`, dan memberikan izin AWS Akun 1 dan AWS Akun 2 untuk menggunakan tindakan apa pun yang diizinkan dengan antrian yang ditentukan.

> **Note**
>
> Sumber daya dalam kebijakan ditentukan sebagai`123456789012/queue_xyz`, di `123456789012` mana ID AWS Akun dari akun yang memiliki antrian.



Dengan diperkenalkannya IAM dan konsep Pengguna dan Nama Sumber Daya Amazon (ARN), beberapa hal telah berubah tentang kebijakan SQS. Diagram dan tabel berikut menjelaskan perubahan.



1 Untuk informasi tentang memberikan izin kepada pengguna di akun yang berbeda, lihat Tutorial: Mendelegasikan Akses di AWS Seluruh Akun Menggunakan Peran IAM di Panduan Pengguna IAM.

2 Subset tindakan yang termasuk dalam * telah diperluas. Untuk daftar tindakan yang diizinkan, lihatIzin API Amazon SQS: Tindakan dan referensi sumber daya.

3 Anda dapat menentukan sumber daya menggunakan Nama Sumber Daya Amazon (ARN), cara standar untuk menentukan sumber daya dalam kebijakan IAM. Untuk informasi tentang format ARN untuk antrian Amazon SQS, lihat. Sumber daya dan operasi Amazon Simple Queue Service

Misalnya, menurut kebijakan Amazon SQS pada diagram sebelumnya, siapa pun yang memiliki kredenSIAL keamanan untuk AWS Akun 1 atau AWS Akun 2 dapat mengaksesqueue_xyz. Selain itu, Pengguna Bob dan Susan di Akun Anda sendiri AWS (dengan ID123456789012) dapat mengakses antrian.

Sebelum pengenalan IAM, Amazon SQS secara otomatis memberi pembuat antrian kontrol penuh atas antrian (yaitu, akses ke semua kemungkinan tindakan Amazon SQS pada antrian itu). Ini tidak lagi benar, kecuali pembuat menggunakan kreden AWS tif keamanan. Setiap pengguna yang memiliki izin untuk membuat antrian juga harus memiliki izin untuk menggunakan tindakan Amazon SQS lainnya untuk melakukan apa pun dengan antrian yang dibuat.

Berikut ini adalah contoh kebijakan yang memungkinkan pengguna untuk menggunakan semua tindakan Amazon SQS, tetapi hanya dengan antrian yang namanya diawali dengan string literal. `bob_queue_`

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
   }]
}
```

Untuk informasi selengkapnya, lihatMenggunakan kebijakan dengan Amazon SQS, dan Identitas (Pengguna, Grup, dan Peran) di Panduan Pengguna IAM.

Menentukan elemen kebijakan: Tindakan, efek, sumber daya, dan principal

Untuk setiap sumber daya Amazon Simple Queue Service, layanan menentukan serangkaian tindakan. Untuk memberikan izin untuk tindakan ini, Amazon SQS mendefinisikan serangkaian tindakan yang dapat Anda tentukan dalam kebijakan.

> **Note**
>
> Melakukan tindakan dapat memerlukan izin untuk lebih dari satu tindakan. Saat memberikan izin untuk tindakan tertentu, Anda juga mengidentifikasi sumber daya yang tindakannya diizinkan atau ditolak.

Berikut adalah elemen-elemen kebijakan yang paling dasar:

- Sumber daya – Dalam kebijakan, Anda menggunakan Amazon Resource Name (ARN) untuk mengidentifikasi sumber daya yang diatur kebijakan.
- Tindakan – Anda menggunakan kata kunci tindakan untuk mengidentifikasi tindakan sumber daya yang ingin Anda izinkan atau tolak. Misalnya, `sqs:CreateQueue` izin memungkinkan pengguna untuk melakukan `CreateQueue` tindakan Amazon Simple Queue Service.

- Efek – Anda menentukan efek ketika pengguna meminta tindakan tertentu—efek ini dapat berupa pemberian izin atau penolakan. Jika Anda tidak secara eksplisit memberikan akses ke sumber daya, akses secara implisit ditolak. Anda juga dapat secara eksplisit menolak akses ke sumber daya, yang dapat Anda lakukan untuk memastikan bahwa pengguna tidak dapat mengaksesnya, meskipun kebijakan yang berbeda memberikan akses.
- Prinsipal – Dalam kebijakan berbasis identitas (Kebijakan IAM), pengguna yang dilampiri kebijakan adalah prinsipal secara implisit. Untuk kebijakan berbasis sumber daya, Anda menentukan pengguna, akun, layanan, atau entitas lain yang diinginkan untuk menerima izin (berlaku hanya untuk kebijakan berbasis sumber daya).

Untuk mempelajari selengkapnya tentang sintaks dan deskripsi kebijakan Amazon SQS, lihat Referensi Kebijakan AWS IAM di Panduan Pengguna IAM.

Untuk tabel semua tindakan Amazon Simple Queue Service dan sumber daya yang diterapkan, lihatIzin API Amazon SQS: Tindakan dan referensi sumber daya.

Cara kerja Amazon Simple Queue Service dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke Amazon SQS, pelajari fitur IAM apa yang tersedia untuk digunakan dengan Amazon SQS.

Fitur IAM yang dapat Anda gunakan dengan Amazon Simple Queue Service

Fitur IAM	Dukungan Amazon SQS
Identity-based kebijakan	Ya
Resource-based kebijakan	Ya
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
kunci-kunci persyaratan kebijakan (spesifik layanan)	Ya
ACL	Tidak
ABAC (tanda dalam kebijakan)	Parsial

Fitur IAM	Dukungan Amazon SQS
Kredensial sementara	Ya
Sesi akses teruskan (FAS)	Ya
Peran layanan	Ya
Service-linked peran	Tidak

Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja Amazon SQS dan AWS layanan lainnya dengan sebagian besar fitur IAM, lihat AWS layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Kontrol akses

Daftar kontrol akses (ACL) mengendalikan principal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL serupa dengan kebijakan berbasis sumber daya, meskipun kebijakan tersebut tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung ACL. Untuk mempelajari ACL selengkapnya, lihat Gambaran umum daftar kontrol akses (ACL) dalam Panduan Developer Amazon Simple Storage Service.

> **Note**
>
> Penting untuk dipahami bahwa semua Akun AWS dapat mendelegasikan izin mereka kepada pengguna di bawah akun mereka. Cross-account akses memungkinkan Anda untuk berbagi akses ke AWS sumber daya Anda tanpa harus mengelola pengguna tambahan. Untuk informasi tentang menggunakan akses lintas akun, lihat Meng aktifkan Cross-Account Akses di Panduan Pengguna IAM.
> Lihat Batasan kebijakan kustom Amazon SQS untuk detail lebih lanjut tentang izin lintas konten dan kunci kondisi dalam kebijakan kustom Amazon SQS.

Identity-based kebijakan untuk Amazon SQS

Mendukung kebijakan berbasis identitas: Ya

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke identitas, seperti pengguna IAM, grup pengguna, atau peran. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkan atau ditolaknya tindakan tersebut. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Identity-based contoh kebijakan untuk Amazon SQS

Untuk melihat contoh kebijakan berbasis identitas Amazon SQS, lihat. Praktik terbaik kebijakan

Resource-based kebijakan dalam Amazon SQS

Mendukung kebijakan berbasis sumber daya: Ya

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh principal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan principal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai principal dalam kebijakan berbasis sumber daya. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Tindakan kebijakan untuk Amazon SQS

Mendukung tindakan kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan Amazon SQS, lihat Sumber Daya yang Ditetapkan oleh Layanan Antrian Sederhana Amazon di Referensi Otorisasi Layanan.

Tindakan kebijakan di Amazon SQS menggunakan awalan berikut sebelum tindakan:

```
sqs
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [
      "sqs:action1",
      "sqs:action2"
         ]
```

Untuk melihat contoh kebijakan berbasis identitas Amazon SQS, lihat. Praktik terbaik kebijakan

Sumber daya kebijakan untuk Amazon SQS

Mendukung sumber daya kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek yang menjadi target penerapan tindakan. Praktik terbaiknya, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar jenis sumber daya Amazon SQS dan ARN-nya, lihat Tindakan yang Ditetapkan oleh Layanan Antrian Sederhana Amazon di Referensi Otorisasi Layanan. Untuk mempelajari

tindakan mana Anda dapat menentukan ARN dari setiap sumber daya, lihat Sumber Daya yang Ditet apkan oleh Amazon Simple Queue Service.

Untuk melihat contoh kebijakan berbasis identitas Amazon SQS, lihat. Praktik terbaik kebijakan

Kunci kondisi kebijakan untuk Amazon SQS

Mendukung kunci kondisi kebijakan khusus layanan: Yes

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Elemen `Condition` menentukan ketika pernyataan dieksekusi berdasarkan kriteria yang ditetapkan. Anda dapat membuat ekspresi bersyarat yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Untuk melihat daftar kunci kondisi Amazon SQS, lihat Kunci Kondisi untuk Layanan Antrian Sederhana Amazon di Refer ensi O torisasi Layanan. Untuk mempelajari tindakan dan sumber daya mana yang dapat Anda gunakan kunci kondisi, lihat Sumber Daya yang Ditet apkan oleh Amazon Simple Queue Service.

Untuk melihat contoh kebijakan berbasis identitas Amazon SQS, lihat. Praktik terbaik kebijakan

ACL di Amazon SQS

Mendukung ACL: Tidak

Daftar kontrol akses (ACL) mengendalikan principal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL serupa dengan kebijakan berbasis sumber daya, meskipun kebijakan tersebut tidak menggunakan format dokumen kebijakan JSON.

ABAC dengan Amazon SQS

Mendukung ABAC (tag dalam kebijakan): Sebagian

Attribute-based kontrol akses (ABAC) adalah strategi otorisasi yang mendefinisikan izin berdasarkan atribut yang disebut tag. Anda dapat melampirkan tag ke entitas dan AWS sumber daya IAM, lalu

merancang kebijakan ABAC untuk mengizinkan operasi ketika tag kepala sekolah cocok dengan tag pada sumber daya.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di elemen kondisi dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat Tentukan izin dengan otorisasi ABAC dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) dalam Panduan Pengguna IAM.

Menggunakan kredenSIAL sementara dengan Amazon SQS

Mendukung kredensial sementara: Ya

KredenSIAL sementara menyediakan akses jangka pendek ke AWS sumber daya dan dibuat secara otomatis saat Anda menggunakan federasi atau beralih peran. AWS merekomendasikan agar Anda secara dinamis membuat kredenSIAL sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM dan Layanan AWS yang berfungsi dengan IAM dalam Panduan Pengguna IAM.

Teruskan sesi akses untuk Amazon SQS

Mendukung sesi akses terusan (FAS): Ya

Sesi akses maju (FAS) menggunakan izin dari kepala sekolah yang memanggil Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses terusan.

Peran layanan untuk Amazon SQS

Mendukung peran layanan: Ya

Peran layanan adalah peran IAM yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari

dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.

> **Warning**
>
> Mengubah izin untuk peran layanan dapat merusak fungsionalitas Amazon SQS. Edit peran layanan hanya jika Amazon SQS memberikan panduan untuk melakukannya.

Service-linked peran untuk Amazon SQS

Mendukung peran terkait layanan: Tidak

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat mengambil peran untuk melakukan tindakan atas nama Anda. Service-linked peran muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang pembuatan atau manajemen peran terkait layanan, lihat Layanan AWS yang berfungsi dengan IAM. Temukan layanan dalam tabel yang menyer Yes takan kolom Service-linked peran. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Amazon SQS memperbarui ke AWS kebijakan terkelola

Menambahkan izin ke para pengguna, grup, dan peran lebih mudah dilakukan dengan menggunakan kebijakan terkelola AWS dibandingkan dengan menulis kebijakan sendiri. Dibutuhkan waktu dan keahlian untuk membuat kebijakan terkelola pelanggan IAM yang hanya menyediakan izin sesuai kebutuhan tim Anda. Untuk mulai dengan cepat, Anda dapat menggunakan kebijakan-kebijakan terkelola AWS kami. Kebijakan-kebijakan ini mencakup kasus penggunaan umum dan tersedia di akun AWS Anda. Untuk informasi selengkapnya tentang AWS kebijakan terkelola, AWS lihat kebijakan ter kelola di Panduan Pengguna IAM.

AWS layanan memelihara dan memperbarui kebijakan AWS terkelola. Anda tidak dapat mengubah izin dalam kebijakan ter AWS kelola. Layanan terkadang menambahkan izin tambahan ke kebijakan ter AWS kelola untuk mendukung fitur baru. Jenis pembaruan ini akan memengaruhi semua identitas (pengguna, grup, dan peran) di mana kebijakan tersebut dilampirkan. Layanan kemungkinan

besar akan memperbarui kebijakan AWS terkelola saat fitur baru diluncurkan atau ketika operasi baru tersedia. Layanan tidak menghapus izin dari kebijakan ter AWS kelola, sehingga pembaruan kebijakan tidak akan merusak izin yang ada.

Selain itu, AWS mendukung kebijakan terkelola untuk fungsi pekerjaan yang mencakup beberapa layanan. Misalnya, kebijakan terkel ReadOnlyAccess AWS ola menyediakan akses baca-saja ke semua AWS layanan dan sumber daya. Saat layanan meluncurkan fitur baru, AWS menambahkan izin baca-saja untuk operasi dan sumber daya baru. Untuk melihat daftar dan deskripsi dari kebijakan fungsi tugas, lihat kebijakan yang dikelola AWS untuk fungsi tugas di Panduan Pengguna IAM.

AWS kebijakan yang dikelola: AmazonSQSFullAccess

Anda dapat melampirkan `AmazonSQSFullAccess` kebijakan ke identitas Amazon SQS Anda. Kebijakan ini memberikan izin yang memungkinkan akses penuh ke Amazon SQS.

Untuk melihat izin kebijakan ini, lihat AmazonSQSFullAccess di Referensi Kebijakan AWS Terkelola.

AWS kebijakan yang dikelola: AmazonSQSReadOnlyAccess

Anda dapat melampirkan `AmazonSQSReadOnlyAccess` kebijakan ke identitas Amazon SQS Anda. Kebijakan ini memberikan izin yang memungkinkan akses read-only ke Amazon SQS.

Untuk melihat izin kebijakan ini, lihat AmazonSQSReadOnlyAccess di Referensi Kebijakan AWS Terkelola.

AWS kebijakan yang dikelola: SQSUnlockQueuePolicy

Jika Anda salah mengonfigurasi kebijakan antrian untuk akun anggota untuk menolak semua pengguna mengakses antrian Amazon SQS Anda, Anda dapat menggunakan kebijakan terkel `SQSUnlockQueuePolicy` AWS ola untuk membuka kunci antrian.

Untuk informasi selengkapnya tentang cara menghapus kebijakan antrian yang salah konfigurasi yang menolak semua prinsipal mengakses antrian Amazon SQS, lihat Mel akukan tugas istimewa pada akun AWS Organizations anggota di Panduan Pengguna IAM.

Amazon SQS memperbarui ke AWS kebijakan terkelola

Lihat detail tentang pembaruan kebijakan ter AWS kelola untuk Amazon SQS sejak layanan ini mulai melacak perubahan ini. Untuk peringatan otomatis tentang perubahan pada halaman ini, berlangganan umpan RSS di halaman riwayat Dokumen Amazon SQS.

Ubah	Deskripsi	Date
SQSUnlockQueuePolicy	Amazon SQS menambahkan kebijakan yang AWS dikelola baru yang dipanggil `SQSUnlockQueuePolicy` untuk membuka kunci antrian dan menghapus kebijakan antrian yang salah konfigurasi yang menolak semua kepala sekolah mengakses antrian Amazon SQS.	November 15, 2024
AmazonSQSReadOnlyAccess	Amazon SQS menambahkan `ListQueueTags` tindakan, yang mengambil semua tag yang terkait dengan antrian Amazon SQS tertentu. Ini memungkinkan Anda untuk melihat pasangan kunci-nilai yang telah ditetapkan ke antrian untuk tujuan organisasi atau metadata. Tindakan ini dikaitkan dengan operasi API `ListQueueTags` .	Juni 20, 2024
AmazonSQSReadOnlyAccess	Amazon SQS menambahkan tindakan baru yang memungkinkan Anda mencantumkan tugas pergerakan pesan terbaru (hingga 10) di bawah antrian sumber tertentu. Tindakan ini	9 Juni 2023

Ubah	Deskripsi	Date
	dikaitkan dengan operasi API `ListMessageMoveTasks`.	

Memecahkan masalah identitas dan akses Amazon Simple Queue Service

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan Amazon SQS dan IAM.

Saya tidak berwenang untuk melakukan tindakan di Amazon SQS

Jika Anda menerima pesan kesalahan bahwa Anda tidak memiliki otorisasi untuk melakukan tindakan, kebijakan Anda harus diperbarui agar Anda dapat melakukan tindakan tersebut.

Contoh kesalahan berikut terjadi ketika pengguna `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *`my-example-widget`* fiktif, tetapi tidak memiliki izin `sqs:`*`GetWidget`* fiktif.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 sqs:GetWidget on resource: my-example-widget
```

Dalam hal ini, kebijakan Mateo harus diperbarui untuk memungkinkannya mengakses *`my-example-widget`* sumber daya menggunakan `sqs:`*`GetWidget`* tindakan.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan bahwa Anda tidak berwenang untuk melakukan `iam:PassRole` tindakan tersebut, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke Amazon SQS.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan itu alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di Amazon SQS. Namun, tindakan tersebut memerlukan

layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang-orang di luar saya Akun AWS untuk mengakses sumber daya Amazon SQS saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mengetahui apakah Amazon SQS mendukung fitur-fitur ini, lihatCara kerja Amazon Simple Queue Service dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM lainnya Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyedi Akun AWS akan akses milik pihak ketiga di Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Saya ingin membuka kunci antrian saya

Jika Anda Akun AWS milik organisasi, AWS Organizations kebijakan dapat memblokir Anda mengakses sumber daya Amazon SQS. Secara default, AWS Organizations kebijakan tidak memblokir permintaan apa pun ke Amazon SQS. Namun, pastikan AWS Organizations kebijakan Anda belum dikonfigurasi untuk memblokir akses ke antrian Amazon SQS. Untuk petunjuk tentang cara memeriksa AWS Organizations kebijakan Anda, lihat Menc antumkan semua kebijakan di Panduan AWS Organizations Pengguna.

Selain itu, jika Anda salah mengonfigurasi kebijakan antrian untuk akun anggota untuk menolak semua pengguna mengakses antrian Amazon SQS Anda, Anda dapat membuka kunci antrian dengan meluncurkan sesi istimewa untuk akun anggota di IAM. Setelah meluncurkan sesi istimewa, Anda dapat menghapus kebijakan antrian yang salah dikonfigurasi untuk mendapatkan kembali akses ke antrian. Untuk informasi selengkapnya, lihat Melakukan tugas istimewa pada akun AWS Organizations anggota di Panduan Pengguna IAM.

Menggunakan kebijakan dengan Amazon SQS

Topik ini memberikan contoh kebijakan berbasis identitas di mana administrator akun dapat melampirkan kebijakan izin ke identitas IAM (pengguna, grup, dan peran).

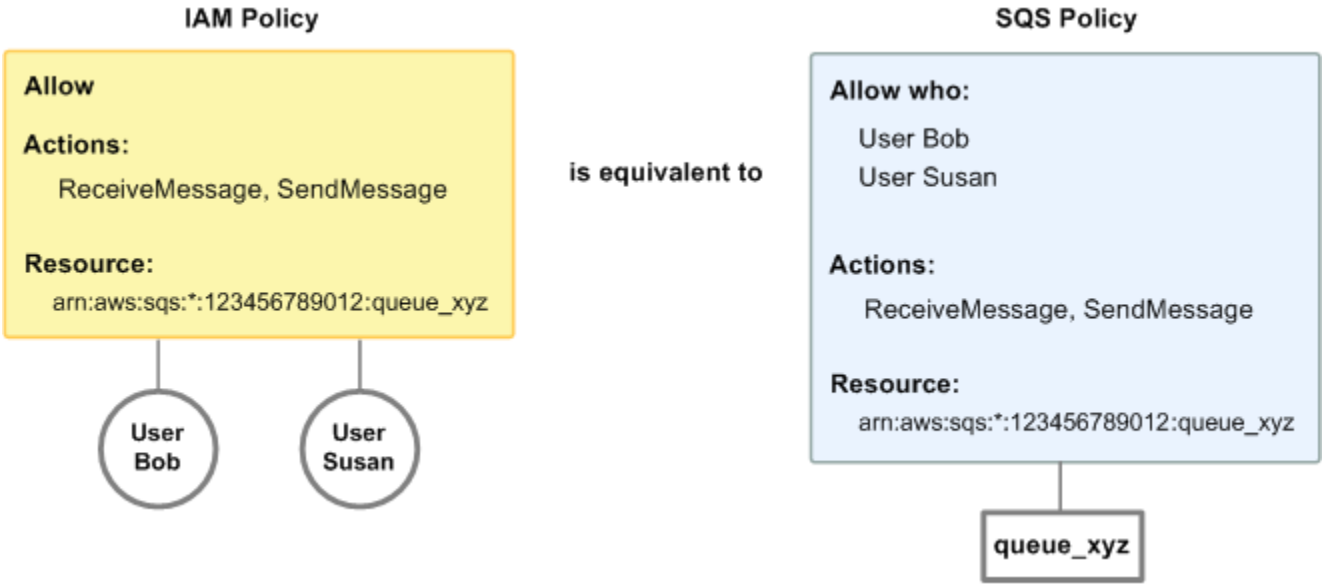
> **Important**
>
> Kami menyarankan Anda terlebih dahulu meninjau topik pengantar yang menjelaskan konsep dasar dan opsi yang tersedia bagi Anda untuk mengelola akses ke sumber daya Layanan Antrian Sederhana Amazon Anda. Untuk informasi selengkapnya, lihat Ikhtisar pengelolaan akses di Amazon SQS.
> Dengan pengecualian`ListQueues`, semua tindakan Amazon SQS mendukung izin tingkat sumber daya. Untuk informasi selengkapnya, lihat Izin API Amazon SQS: Tindakan dan referensi sumber daya.

Menggunakan kebijakan Amazon SQS dan IAM

Ada dua cara untuk memberikan izin kepada pengguna ke sumber daya Amazon SQS Anda: menggunakan sistem kebijakan Amazon SQS (kebijakan berbasis sumber daya) dan menggunakan sistem kebijakan IAM (kebijakan berbasis identitas). Anda dapat menggunakan satu atau kedua metode, dengan pengecualian `ListQueues` tindakan, yang merupakan izin regional yang hanya dapat ditetapkan dalam kebijakan IAM.

Misalnya, diagram berikut menunjukkan kebijakan IAM dan kebijakan Amazon SQS yang setara dengannya. Kebijakan IAM memberikan hak atas Amazon SQS `ReceiveMessage` dan `SendMessage` tindakan untuk antrian yang dipanggil `queue_xyz` di AWS Akun Anda, dan kebijakan dilampirkan ke pengguna bernama Bob dan Susan (Bob dan Susan memiliki izin yang tercantum dalam kebijakan). Kebijakan Amazon SQS ini juga memberi Bob dan Susan hak atas `SendMessage` tindakan `ReceiveMessage` dan untuk antrian yang sama.

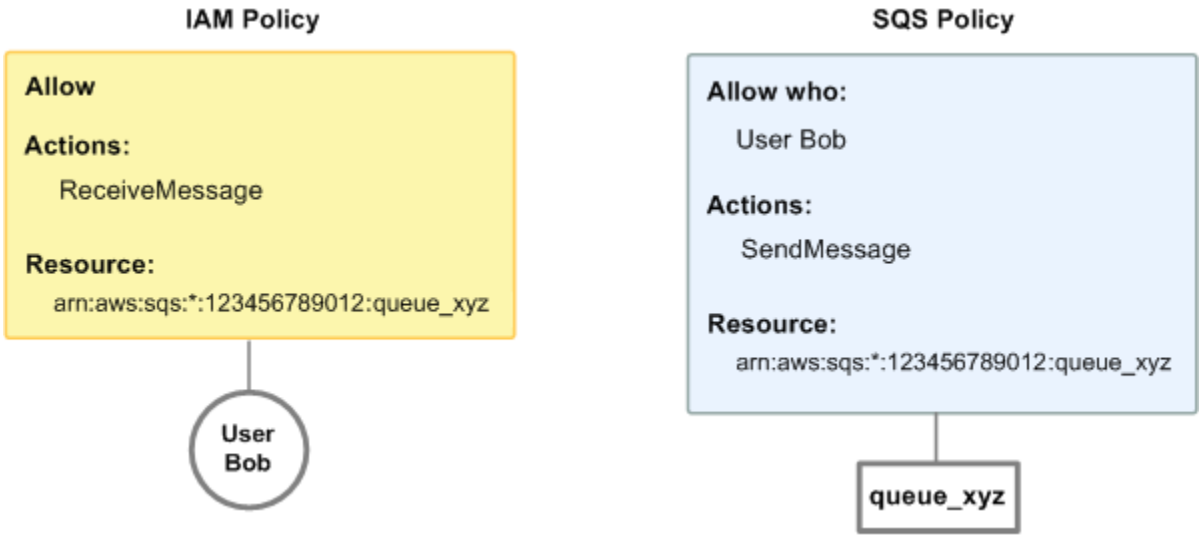
> **Note**
>
> Contoh berikut menunjukkan kebijakan sederhana tanpa kondisi. Anda dapat menentukan kondisi tertentu di salah satu kebijakan dan mendapatkan hasil yang sama.



Ada satu perbedaan utama antara kebijakan IAM dan Amazon SQS: sistem kebijakan Amazon SQS memungkinkan Anda memberikan izin ke AWS Akun lain, sedangkan IAM tidak.

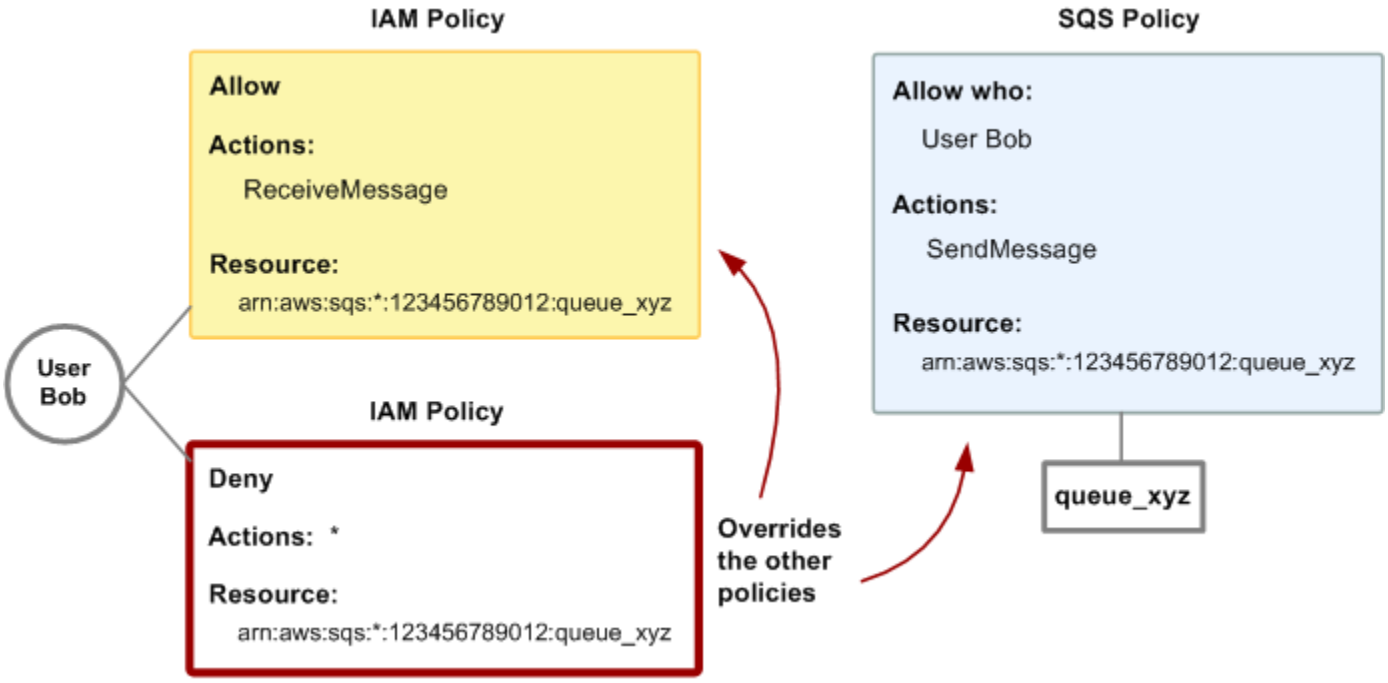
Terserah Anda bagaimana Anda menggunakan kedua sistem bersama-sama untuk mengelola izin Anda. Contoh berikut menunjukkan cara sistem dua kebijakan bekerja sama.

- Dalam contoh pertama, Bob memiliki kebijakan IAM dan kebijakan Amazon SQS yang berlaku untuk akunnya. Kebijakan IAM memberikan izin akunnya untuk `ReceiveMessage` tindakan aktif`queue_xyz`, sedangkan kebijakan Amazon SQS memberikan izin akunnya untuk `SendMessage` tindakan pada antrian yang sama. Diagram berikut menggambarkan konsep.



Jika Bob mengirim `ReceiveMessage` permintaan kequeue_xyz, kebijakan IAM mengizinkan tindakan tersebut. Jika Bob mengirim `SendMessage` permintaan kequeue_xyz, kebijakan Amazon SQS mengizinkan tindakan tersebut.

- Dalam contoh kedua, Bob menyalahgunakan aksesnyaqueue_xyz, sehingga menjadi perlu untuk menghapus seluruh aksesnya ke antrian. Hal termudah untuk dilakukan adalah menambahkan kebijakan yang menolaknya akses ke semua tindakan untuk antrian. Kebijakan ini mengesampingkan dua lainnya karena eksplisit `deny` selalu mengesampingkan an. `allow` Untuk informasi lebih lanjut tentang logika evaluasi kebijakan, lihatMenggunakan kebijakan khusus dengan Bahasa Kebijakan Akses Amazon SQS. Diagram berikut menggambarkan konsep.



Anda juga dapat menambahkan pernyataan tambahan ke kebijakan Amazon SQS yang menolak Bob jenis akses apa pun ke antrian. Ini memiliki efek yang sama seperti menambahkan kebijakan IAM yang menolak akses Bob ke antrian. Untuk contoh kebijakan yang mencakup tindakan dan sumber daya Amazon SQS, lihatContoh dasar kebijakan Amazon SQS. Untuk informasi selengkapnya tentang penulisan kebijakan Amazon SQS, lihatMenggunakan kebijakan khusus dengan Bahasa Kebijakan Akses Amazon SQS.

Izin diperlukan untuk menggunakan konsol Amazon SQS

Pengguna yang ingin bekerja dengan konsol Amazon SQS harus memiliki kumpulan izin minimum untuk bekerja dengan antrian Amazon SQS di pengguna. Akun AWS Misalnya, pengguna harus memiliki izin untuk memanggil `ListQueues` tindakan untuk dapat membuat daftar antrian, atau izin untuk memanggil `CreateQueue` tindakan untuk dapat membuat antrian. Selain izin Amazon SQS, untuk berlangganan antrian Amazon SQS ke topik Amazon SNS, konsol juga memerlukan izin untuk tindakan Amazon SNS.

Jika Anda membuat kebijakan IAM yang lebih ketat daripada izin minimum yang diperlukan, konsol mungkin tidak berfungsi sebagaimana dimaksudkan untuk pengguna dengan kebijakan IAM tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke tindakan AWS CLI atau Amazon SQS.

Identity-based contoh kebijakan untuk Amazon SQS

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi sumber daya Amazon SQS. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM dengan menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan IAM (konsol) di Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh Amazon SQS, termasuk format ARN untuk setiap jenis sumber daya, lihat Tindakan, Sumber Daya, dan Kunci Kondisi untuk Layanan Antrian Sederhana Amazon di Referensi Otorisasi Layanan.

> **Note**
>
> Saat mengonfigurasi kait siklus hidup untuk Amazon EC2 Auto Scaling, Anda tidak perlu menulis kebijakan untuk mengirim pesan ke antrian Amazon SQS. Untuk informasi

selengkapnya, lihat Amazon EC2 Auto Scaling Lifecycle Hooks di Panduan Pengguna Amazon EC2.

Praktik terbaik kebijakan

Identity-based kebijakan menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Amazon SQS di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Memulai kebijakan AWS terkelola dan beralih ke izin hak istimewa terendah — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan terkel AWS ola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat Kebijakan yang dikelola AWS atau Kebijakan yang dikelola AWS untuk fungsi tugas dalam Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat Kebijakan dan izin dalam IAM dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan kondisi untuk memberikan akses ke tindakan layanan jika digunakan melalui tindakan tertentu Layanan AWS, seperti CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Kondisi dalam Panduan Pengguna IAM.
- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk

informasi selengkapnya, lihat Validasi kebijakan dengan IAM Access Analyzer dalam Panduan Pengguna IAM.

- Memerlukan otentikasi multi-faktor (MFA) — Jika Anda memiliki skenario yang memerlukan pengguna IAM atau pengguna root Akun AWS, aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Amankan akses API dengan MFA dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM dalam Panduan Pengguna IAM.

Menggunakan konsol Amazon SQS

Untuk mengakses konsol Amazon Simple Queue Service, Anda harus memiliki kumpulan izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang sumber daya Amazon SQS di. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebagai gantinya, izinkan akses hanya ke tindakan yang sesuai dengan operasi API yang coba mereka lakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol Amazon SQS, lampirkan juga kebijakan yang `AmazonSQSReadOnlyAccess` AWS dikelola Amazon SQS ke entitas. Untuk informasi selengkapnya, lihat Menambah izin untuk pengguna dalam Panduan Pengguna IAM.

Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau secara terprogram menggunakan API AWS CLI or AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Izinkan pengguna membuat antrian

Dalam contoh berikut, kita membuat kebijakan untuk Bob yang memungkinkannya mengakses semua tindakan Amazon SQS, tetapi hanya dengan antrian yang namanya diawali dengan string literal. `alice_queue_`

Amazon SQS tidak secara otomatis memberikan izin kepada pembuat antrian untuk menggunakan antrian. Oleh karena itu, kami harus secara eksplisit memberikan izin Bob untuk menggunakan semua tindakan Amazon SQS selain `CreateQueue` tindakan dalam kebijakan IAM.

JSON

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

```
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
   }]
}
```

Izinkan pengembang menulis pesan ke antrian bersama

Dalam contoh berikut, kita membuat grup untuk pengembang dan melampirkan kebijakan yang memungkinkan grup menggunakan `SendMessage` tindakan Amazon SQS, tetapi hanya dengan antrian yang termasuk dalam yang ditentukan Akun AWS dan diberi `MyCompanyQueue` nama.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
   }]
}
```

Anda dapat menggunakan * alih `SendMessage` -alih untuk memberikan tindakan berikut kepada kepala sekolah pada antrian bersama:`ChangeMessageVisibility`,`DeleteMessage`,`GetQueueAttributes`,`GetQueueUrl`,`Receiv` dan`SendMessage`.

> **Note**
>
> Meskipun * mencakup akses yang disediakan oleh jenis izin lain, Amazon SQS mempertimbangkan izin secara terpisah. Misalnya, dimungkinkan untuk memberikan keduanya * dan `SendMessage` izin kepada pengguna, meskipun a menyer * takan akses yang disediakan oleh`SendMessage`.
> Konsep ini juga berlaku ketika Anda menghapus izin. Jika kepala sekolah hanya memiliki * izin, meminta untuk menghapus `SendMessage` izin tidak meninggalkan kepala sekolah dengan izin segalanya kecuali. Sebaliknya, permintaan tidak berpengaruh, karena kepala sekolah tidak memiliki `SendMessage` izin eksplisit. Untuk meninggalkan kepala sekolah

hanya dengan `ReceiveMessage` izin, pertama-tama tambahkan `ReceiveMessage` izin dan kemudian hapus `*` izin.

Izinkan manajer untuk mendapatkan ukuran umum antrian

Dalam contoh berikut, kita membuat grup untuk manajer dan melampirkan kebijakan yang memungkinkan grup menggunakan `GetQueueAttributes` tindakan Amazon SQS dengan semua antrian milik akun yang ditentukan AWS .

JSON

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

Izinkan mitra mengirim pesan ke antrian tertentu

Anda dapat menyelesaikan tugas ini menggunakan kebijakan Amazon SQS atau kebijakan IAM. Jika pasangan Anda memilik Akun AWS inya, mungkin lebih mudah menggunakan kebijakan Amazon SQS. Namun, setiap pengguna di perusahaan mitra yang memiliki kredenSIAL AWS keamanan dapat mengirim pesan ke antrian. Jika Anda ingin membatasi akses ke pengguna atau aplikasi tertentu, Anda harus memperlakukan mitra seperti pengguna di perusahaan Anda sendiri dan menggunakan kebijakan IAM alih-alih kebijakan Amazon SQS.

Contoh ini melakukan tindakan berikut:

1. Buat grup yang di WidgetCo panggil untuk mewakili perusahaan mitra.
2. Buat pengguna untuk pengguna atau aplikasi tertentu di perusahaan mitra yang membutuhkan akses.
3. Tambahkan pengguna ke grup .
4. Lampirkan kebijakan yang memberikan akses grup hanya ke `SendMessage` tindakan hanya untuk antrian bernama`WidgetPartnerQueue`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "sqs:SendMessage",
         "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
   }]
}
```

Contoh dasar kebijakan Amazon SQS

Bagian ini menunjukkan contoh kebijakan untuk kasus penggunaan Amazon SQS umum.

Anda dapat menggunakan konsol untuk memverifikasi efek setiap kebijakan saat Anda melampirkan kebijakan kepada pengguna. Awalnya, pengguna tidak memiliki izin dan tidak akan dapat melakukan apa pun di konsol. Saat Anda melampirkan kebijakan ke pengguna, Anda dapat memverifikasi bahwa pengguna dapat melakukan berbagai tindakan di konsol.

> **Note**
>
> Kami menyarankan Anda menggunakan dua jendela browser: satu untuk memberikan izin dan yang lainnya untuk masuk Konsol Manajemen AWS menggunakan kredenSIAL pengguna untuk memverifikasi izin saat Anda memberikannya kepada pengguna.

Contoh 1: Berikan satu izin kepada satu Akun AWS

Kebijakan contoh berikut memberikan Akun AWS nomor `111122223333` `SendMessage` izin untuk antrian yang dinam `444455556666/queue1` ai di wilayah AS Timur (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"
   }]
}
```

Contoh 2: Berikan dua izin untuk satu Akun AWS

Kebijakan contoh berikut memberikan Akun AWS nomor `111122223333` `SendMessage` dan `ReceiveMessage` izin untuk antrian yang diberi nama`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"
   }]
}
```

Contoh 3: Berikan semua izin ke dua Akun AWS

Kebijakan contoh berikut memberikan dua izin Akun AWS nomor (111122223333dan444455556666) berbeda untuk menggunakan semua tindakan yang Amazon SQS mengizinkan akses bersama untuk antrian yang dinam 123456789012/queue1 ai di wilayah AS Timur (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"
   }]
}
```

Contoh 4: Memberikan izin lintas akun ke peran dan nama pengguna

Contoh kebijakan berikut memberikan izin 111122223333 lintas akun username1 di bawah Akun AWS nomor untuk menggunakan semua tindakan yang Amazon SQS mengizinkan akses bersama untuk antrian yang dinamai 123456789012/queue1 di wilayah AS Timur (Ohio). role1

Cross-account izin tidak berlaku untuk tindakan berikut:

- 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"
   }]
}
```

Contoh 5: Berikan izin kepada semua pengguna

Kebijakan contoh berikut memberikan semua pengguna (pengguna anonim) `ReceiveMessage` izin untuk antrian bernama`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"
   }]
}
```

Contoh 6: Memberikan izin terbatas waktu kepada semua pengguna

Kebijakan contoh berikut memberikan `ReceiveMessage` izin kepada semua pengguna (pengguna anonim) untuk antrian bernama`111122223333/queue1`, tetapi hanya antara pukul 12:00 siang dan 15:00 pada tanggal 31 Januari 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"
         }
      }
   }]
}
```

Contoh 7: Berikan semua izin kepada semua pengguna dalam rentang CIDR

Kebijakan contoh berikut memberi semua pengguna (pengguna anonim) izin untuk menggunakan semua kemungkinan tindakan Amazon SQS yang dapat dibagikan untuk antrian bernama`111122223333/queue1`, tetapi hanya jika permintaan berasal dari rentang `192.0.2.0/24` CIDR.

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"
         }
      }
   }]
}
```

Contoh 8: Izin daftar izin dan daftar blokir untuk pengguna dalam rentang CIDR yang berbeda

Contoh kebijakan berikut memiliki dua pernyataan:

- Pernyataan pertama memberikan semua pengguna (pengguna anonim) dalam rentang `192.0.2.0/24` CIDR (kecuali untuk`192.0.2.188`) izin untuk menggunakan `SendMessage` tindakan untuk antrian bernama `111122223333` /queue1.
- Pernyataan kedua memblokir semua pengguna (pengguna anonim) dalam rentang `12.148.72.0/23` CIDR dari menggunakan antrian.

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"
         }
      }
   }]
}
```

Menggunakan kebijakan khusus dengan Bahasa Kebijakan Akses Amazon SQS

Untuk memberikan izin dasar (seperti `SendMessage` atau `ReceiveMessage`) hanya berdasarkan Akun AWS ID, Anda tidak perlu menulis kebijakan khusus. Sebagai gantinya, gunakan `AddPermission` tindakan Amazon SQS.

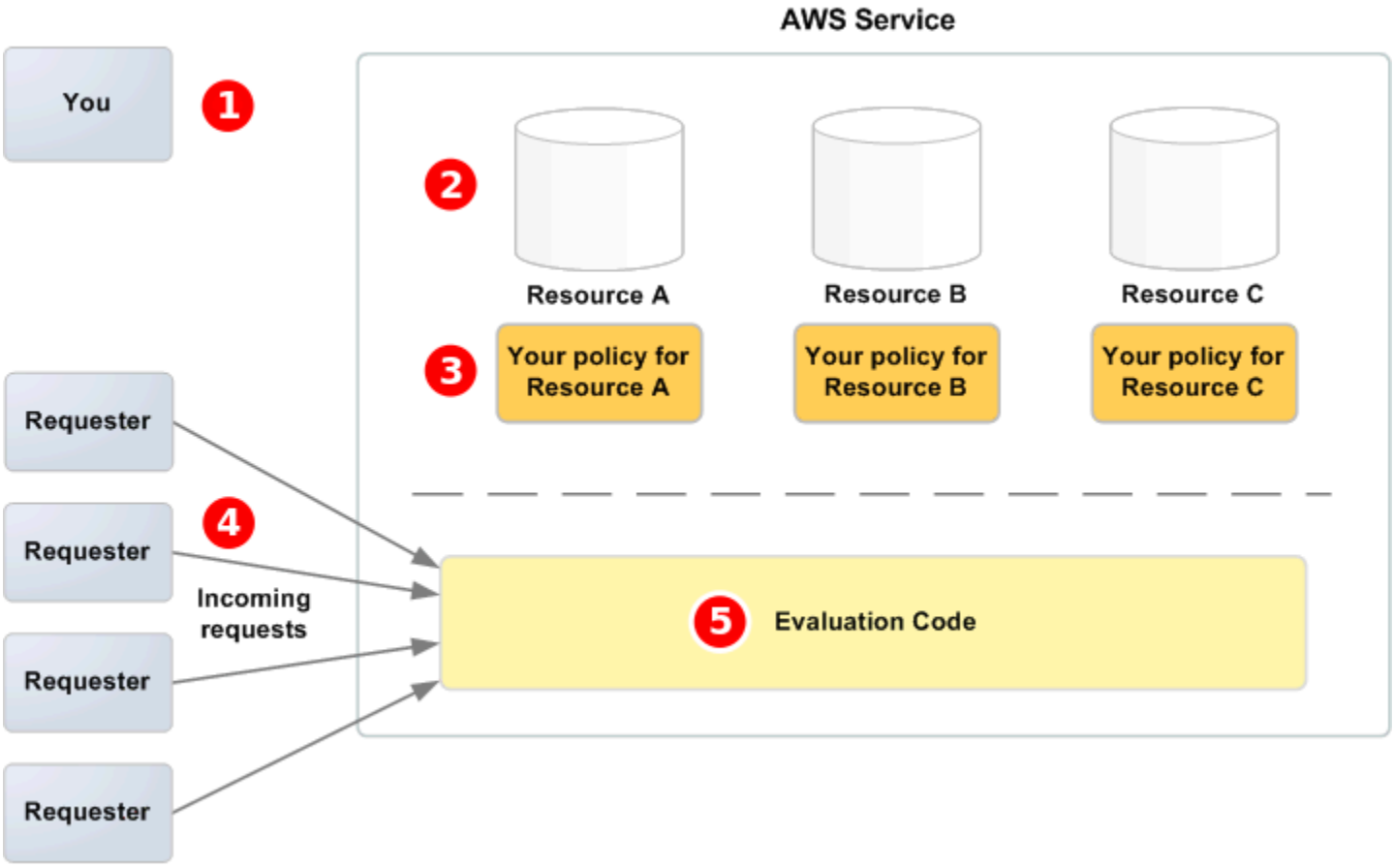
Untuk mengizinkan atau menolak akses berdasarkan kondisi tertentu, seperti waktu permintaan atau alamat IP pemohon, Anda harus membuat kebijakan Amazon SQS khusus dan mengunggahnya menggunakan tindakan tersebut SetQueueAttributes.

Topik

- Arsitektur kontrol akses Amazon SQS
- Alur kerja proses kontrol akses Amazon SQS
- Konsep kunci bahasa Kebijakan Akses Amazon SQS
- Kebijakan Akses Amazon SQS Logika evaluasi bahasa
- Hubungan antara penolakan eksplisit dan default dalam Bahasa Kebijakan Akses Amazon SQS
- Batasan kebijakan kustom Amazon SQS
- Contoh Bahasa Kebijakan Akses Amazon SQS Kustom

Arsitektur kontrol akses Amazon SQS

Diagram berikut menjelaskan kontrol akses untuk sumber daya Amazon SQS Anda.



1 Anda, pemilik sumber daya.

2 Sumbe daya Anda yang terkandung dalam AWS layanan (misalnya, antrian Amazon SQS).

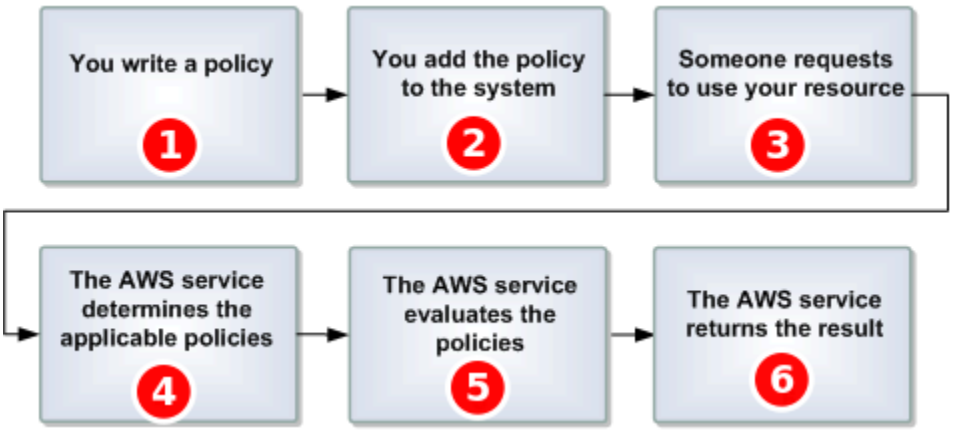
3 Kebijak Anda. Ini adalah praktik yang baik untuk memiliki satu kebijakan per sumber daya. AWS Layanan ini menyediakan API yang Anda gunakan untuk mengunggah dan mengelola kebijakan Anda.

4 Pemoh dan permintaan masuk mereka ke AWS layanan.

5 Kode evaluasi bahasa kebijakan akses. Ini adalah kumpulan kode dalam AWS layanan yang mengevaluasi permintaan masuk terhadap kebijakan yang berlaku dan menentukan apakah pemohon diizinkan mengakses sumber daya.

Alur kerja proses kontrol akses Amazon SQS

Diagram berikut menjelaskan alur kerja umum kontrol akses dengan bahasa kebijakan akses Amazon SQS.



1 Anda menulis kebijakan Amazon SQS untuk antrian Anda.

2 Anda mengunggah kebijakan Anda ke AWS. AWS Layanan ini menyediakan API yang Anda gunakan

untuk mengunggah kebijakan Anda. Misalnya, Anda menggunakan `SetQueueAttributes` tindakan Amazon SQS untuk mengunggah kebijakan untuk antrian Amazon SQS tertentu.

3 Seseor

mengirim permintaan untuk menggunakan antrian Amazon SQS Anda.

4 Amazo

SQS memeriksa semua kebijakan Amazon SQS yang tersedia dan menentukan kebijakan mana yang berlaku.

5 Amazo

SQS mengevaluasi kebijakan dan menentukan apakah pemohon diizinkan untuk menggunakan antrian Anda.

6 Berdas

hasil evaluasi kebijakan, Amazon SQS mengembalikan `Access denied` kesalahan kepada pemohon atau terus memproses permintaan.

Konsep kunci bahasa Kebijakan Akses Amazon SQS

Untuk menulis kebijakan Anda sendiri, Anda harus terbiasa dengan JSON dan sejumlah konsep kunci.

Izinkan

Hasil dari a Pernyataan yang telah Efek ditetapkan `keallow`.

Tindakan

Aktivitas yang Kepala Sekolah memiliki izin untuk dilakukan, biasanya permintaan AWS.

Default-deny

Hasil dari a Pernyataan yang tidak memiliki Explicit-deny pengaturan Izinkan atau.

Kondisi

Setiap batasan atau detail tentang aIzin . Kondisi khas terkait dengan tanggal dan waktu dan alamat IP.

Efek

Hasil yang Anda inginkan Kebijakan untuk dikembalikan pada waktu evaluasi. Pernyataan Anda menentukan `allow` nilai `deny` atau saat menulis pernyataan kebijakan. Ada tiga kemungkinan hasil pada waktu evaluasi kebijakan:Default-deny,Izinkan , danExplicit-deny.

Explicit-deny

Hasil dari a Pernyataan yang telah Efek ditetapkan ke`deny`.

Evaluasi

Proses yang digunakan Amazon SQS untuk menentukan apakah permintaan masuk harus ditolak atau diizinkan berdasarkan aKebijakan .

Penerbit

Pengguna yang menulis a Kebijakan untuk memberikan izin ke sumber daya. Penerbit, menurut definisi selalu pemilik sumber daya. AWS tidak mengizinkan pengguna Amazon SQS membuat kebijakan untuk sumber daya yang tidak mereka miliki.

Kunci

Karakteristik spesifik yang menjadi dasar pembatasan akses.

Izin

Konsep mengizinkan atau melarang akses ke sumber daya menggunakan a Kondisi dan aKunci .

Kebijakan

Dokumen yang bertindak sebagai wadah untuk satu atau lebih pernyataan .



Amazon SQS menggunakan kebijakan untuk menentukan apakah akan memberikan akses kepada pengguna untuk sumber daya.

Kepala Sekolah

Pengguna yang menerima Izin di Kebijakan .

Sumber Daya

Objek yang Kepala Sekolah meminta akses ke.

Pernyataan

Deskripsi formal dari izin tunggal, ditulis dalam bahasa kebijakan akses sebagai bagian dari Kebijakan dokumen yang lebih luas.

Pemohon

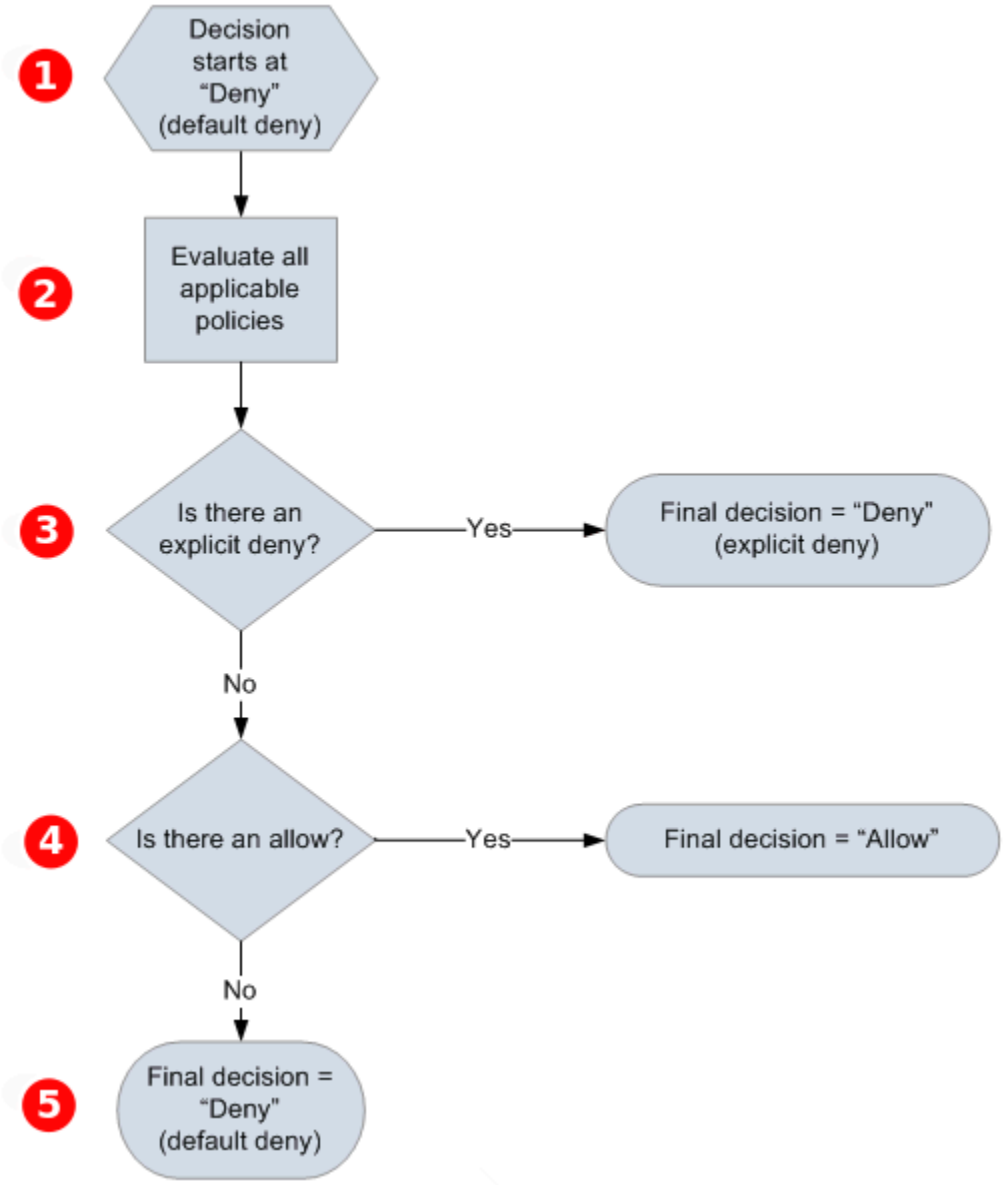
Pengguna yang mengirim permintaan untuk akses ke a Sumber Daya.

Kebijakan Akses Amazon SQS Logika evaluasi bahasa

Pada saat evaluasi, Amazon SQS menentukan apakah permintaan dari orang lain selain pemilik sumber daya harus diizinkan atau ditolak. Logika evaluasi mengikuti beberapa aturan dasar:

- Secara default, semua permintaan untuk menggunakan sumber daya Anda berasal dari siapa pun kecuali Anda ditolak.
- A menges Izinkan ampingkan apa pun Default-deny.
- A menges Explicit-deny ampingkan izin apa pun.
- Urutan di mana kebijakan dievaluasi tidak penting.

Diagram berikut menjelaskan secara rinci bagaimana Amazon SQS mengevaluasi keputusan tentang izin akses.



1 Keputu dimulai dengan default-Deny.

2 Kode penegakan mengevaluasi semua kebijakan yang berlaku untuk permintaan (berdasarkan sumber daya, prinsip, tindakan, dan kondisi). Urutan kode penegakan mengevaluasi kebijakan tidak penting.

3 Kode penegakan mencari instruksi pen olakan eksplisit yang dapat diterapkan pada permintaan. Jika menemukan satu pun, kode penegakan mengembalikan keputusan pen ol akan dan proses selesai.

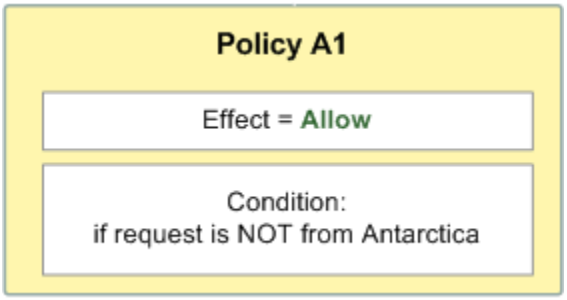
4 Jika tidak ada instruksi explicit-tolak yang ditemukan, kode penegakan mencari instruksi izinkan apa pun yang dapat diterapkan pada permintaan. Jika menemukan satu pun, kode penegakan mengembalikan keputusan izinkan dan proses selesai (layanan terus memproses permintaan).

5 Jika tidak ada instruksi izinkan yang ditemukan, maka keputusan akhir adalah menolak (karena tidak ada explicit- atau allow, ini dianggap sebagai defaul t-menolak).

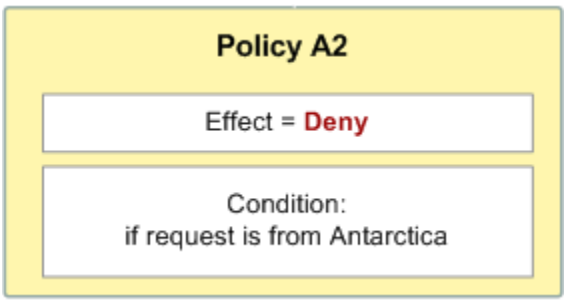
Hubungan antara penolakan eksplisit dan default dalam Bahasa Kebijakan Akses Amazon SQS

Jika kebijakan Amazon SQS tidak langsung berlaku untuk permintaan, permintaan tersebut menghasilkan a Default-deny. Misalnya, jika pengguna meminta izin untuk menggunakan Amazon SQS tetapi satu-satunya kebijakan yang berlaku untuk pengguna dapat menggunakan DynamoDB, permintaan tersebut menghasilkan penolakan default.

Jika kondisi dalam pernyataan tidak terpenuhi, permintaan akan menghasilkan default- tolak. Jika semua kondisi dalam pernyataan terpenuhi, permintaan menghasilkan an Izinkan atau Explicit-deny berdasarkan nilai Efek elemen kebijakan. Kebijakan tidak menentukan apa yang harus dilakukan jika suatu kondisi tidak terpenuhi, jadi hasil default dalam kasus ini adalah default-tolak. Misalnya, Anda ingin mencegah permintaan yang datang dari Antartika. Anda menulis Kebijakan A1 yang mengizinkan permintaan hanya jika tidak berasal dari Antartika. Diagram berikut menggambarkan kebijakan Amazon SQS.

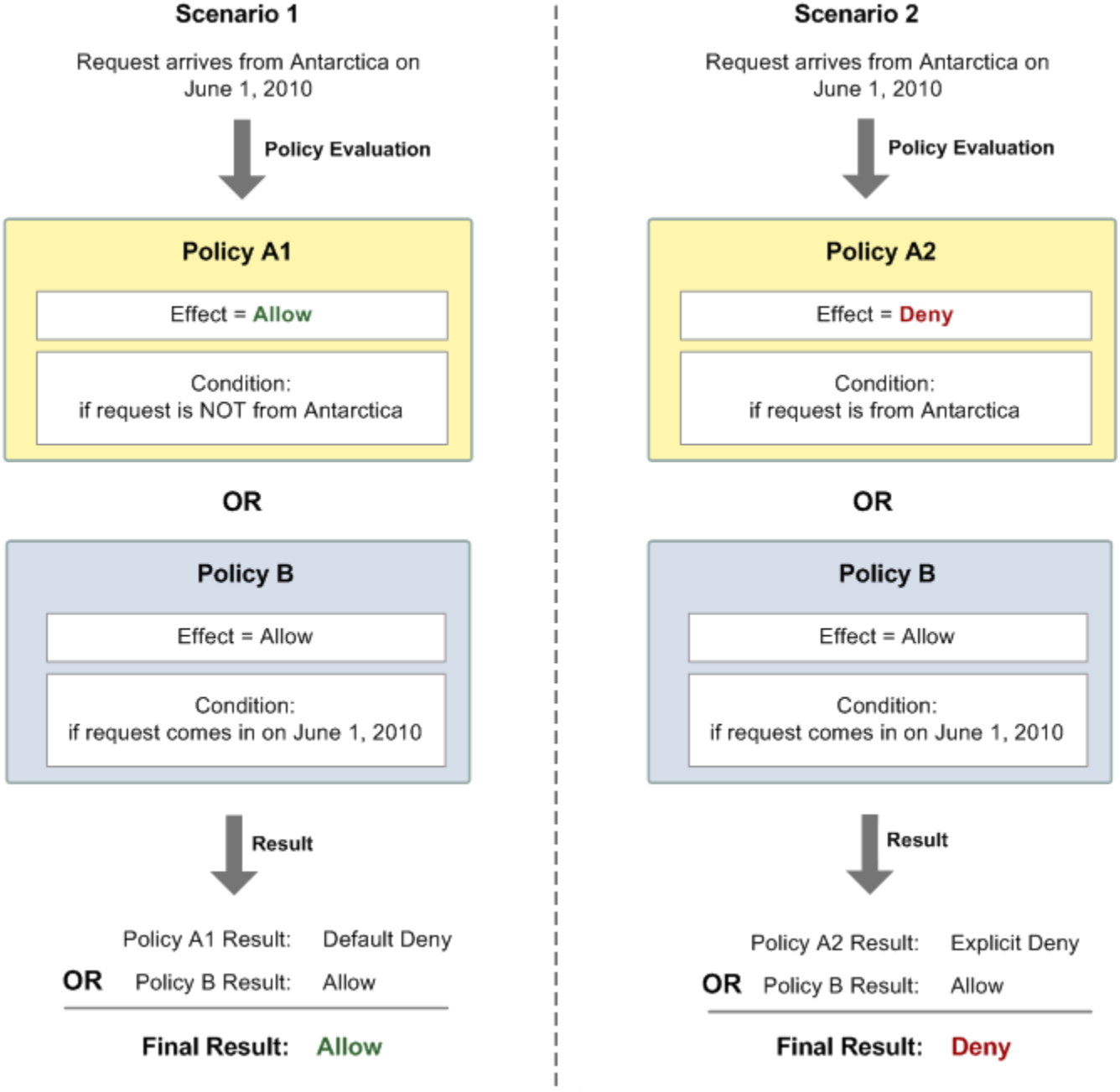


Jika pengguna mengirim permintaan dari U.S., kondisi terpenuhi (permintaan bukan dari Antartika), dan permintaan menghasilkan izin. Namun, jika pengguna mengirim permintaan dari Antartika, kondisinya tidak terpenuhi dan permintaan default menjadi penolakan default. Anda dapat mengubah hasilnya menjadi explicit- dengan menulis Kebijakan A2 yang secara eksplisit menolak permintaan jika itu berasal dari Antartika. Diagram berikut menggambarkan kebijakan.



Jika pengguna mengirim permintaan dari Antartika, kondisi terpenuhi dan permintaan tersebut menghasilkan penolakan ekspl isit.

Perbedaan antara default-deny dan explicit- penting karena izin dapat mengganti yang pertama tetapi tidak yang terakhir. Misalnya, Kebijakan B mengizinkan permintaan jika mereka tiba pada 1 Juni 2010. Diagram berikut membandingkan menggabungkan kebijakan ini dengan Kebijakan A1 dan Kebijakan A2.



Dalam Skenario 1, Kebijakan A1 menghasilkan penolakan default dan Kebijakan B menghasilkan izin karena kebijakan mengizinkan permintaan yang masuk pada 1 Juni 2010. Iz in dari Kebijakan B mengesampingkan pen olakan default dari Kebijakan A1, dan permintaan diizinkan.

Dalam Skenario 2, Kebijakan B2 menghasilkan pen olakan ekspl isit dan Kebijakan B menghasilkan izin. Pen olakan eksplisit dari Kebijakan A2 mengesampingkan izin dari Kebijakan B, dan permintaan ditolak.

Batasan kebijakan kustom Amazon SQS

Cross-account akses

Cross-account izin tidak berlaku untuk tindakan berikut:

- `AddPermission`
- `CancelMessageMoveTask`
- `CreateQueue`
- `DeleteQueue`
- `ListMessageMoveTask`
- `ListQueues`
- `ListQueueTags`
- `RemovePermission`
- `SetQueueAttributes`
- `StartMessageMoveTask`
- `TagQueue`
- `UntagQueue`

Kunci syarat

Saat ini, Amazon SQS hanya mendukung subset terbatas dari kunci kondisi yang tersedia di IAM. Untuk informasi selengkapnya, lihat Izin API Amazon SQS: Tindakan dan referensi sumber daya.

Contoh Bahasa Kebijakan Akses Amazon SQS Kustom

Berikut ini adalah contoh kebijakan akses Amazon SQS yang khas.

Contoh 1: Berikan izin ke satu akun

Contoh kebijakan Amazon SQS berikut memberikan izin Akun AWS 111122223333 untuk mengirim dan menerima dari milik 444455556666. `queue2` Akun AWS

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"
   }]
}
```

Contoh 2: Berikan izin ke satu atau lebih akun

Contoh kebijakan Amazon SQS berikut memberikan satu atau lebih Akun AWS akses ke antrian yang dimiliki oleh akun Anda untuk jangka waktu tertentu. Penting untuk menulis kebijakan ini dan mengunggahnya ke Amazon SQS menggunakan tindakan tersebut karena `SetQueueAttributes` `AddPermission` tindakan tersebut tidak mengizinkan menentukan batasan waktu saat memberikan akses ke antrian.

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"
         }
      }
   }]
}
```

Contoh 3: Memberikan izin untuk permintaan dari instans Amazon EC2

Contoh kebijakan Amazon SQS berikut memberikan akses ke permintaan yang berasal dari instans Amazon EC2. Contoh ini dibangun di atas contoh Contoh 2: Berikan izin ke satu atau lebih akun "": membatasi akses sebelum 30 Juni 2009 pukul 12 siang (UTC), itu membatasi akses ke rentang IP. `203.0.113.0/24` Penting untuk menulis kebijakan ini dan mengunggahnya ke Amazon SQS menggunakan tindakan tersebut karena `SetQueueAttributes` tindakan tersebut `AddPermission` tidak mengizinkan menentukan pembatasan alamat IP saat memberikan akses ke antrian.

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"
         }
      }
   }]
}
```

Contoh 4: Tolak akses ke akun tertentu

Contoh berikut kebijakan Amazon SQS menolak Akun AWS akses tertentu ke antrian Anda. Contoh ini dibangun di atas contoh Contoh 1: Berikan izin ke satu akun "": ia menolak akses ke yang ditentukan Akun AWS. Penting untuk menulis kebijakan ini dan mengunggahnya ke Amazon SQS menggunakan tindakan tersebut karena `SetQueueAttributes` tindakan tersebut tidak mengizinkan tolak akses ke antrian (`AddPermission`tindakan ini hanya memungkinkan pemberian akses ke antrian).

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"
   }]
}
```

Contoh 5: Tolak akses jika bukan dari titik akhir VPC

Contoh berikut kebijakan Amazon SQS membatasi akses kequeue1: 111122223333 dapat melakukan `ReceiveMessage` tindakan `SendMessage` dan hanya dari ID `vpce-1a2b3c4d` titik akhir VPC (ditentukan menggunakan kondisi). `aws:sourceVpce` Untuk informasi selengkapnya, lihat Titik akhir Amazon Virtual Private Cloud untuk Amazon SQS.

> **Note**
>
> - `aws:sourceVpce`Kondisi ini tidak memerlukan ARN untuk sumber daya titik akhir VPC, hanya ID titik akhir VPC.
> - Anda dapat memodifikasi contoh berikut untuk membatasi semua tindakan ke titik akhir VPC tertentu dengan menolak semua tindakan Amazon SQS (`sqs:*`) dalam pernyataan kedua. Namun, pernyataan kebijakan tersebut akan menetapkan bahwa semua tindakan (termasuk tindakan administratif yang diperlukan untuk mengubah izin antrian) harus dilakukan melalui titik akhir VPC tertentu yang ditentukan dalam kebijakan, yang berpotensi mencegah pengguna memodifikasi izin antrian di masa mendatang.

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"
            }
         }
      }
   ]
}
```

Menggunakan kredentif keamanan sementara dengan Amazon SQS

Selain membuat pengguna dengan kredenSIAL keamanan mereka sendiri, IAM juga memungkinkan Anda untuk memberikan kredenSIAL keamanan sementara kepada pengguna mana pun, memungkinkan pengguna untuk mengakses AWS layanan dan sumber daya Anda. Anda dapat mengelola pengguna yang memiliki Akun AWS. Anda juga dapat mengelola pengguna untuk sistem Anda yang tidak memiliki Akun AWS (pengguna federasi). Selain itu, aplikasi yang Anda buat untuk mengakses AWS sumber daya Anda juga dapat dianggap sebagai “pengguna.”

Anda dapat menggunakan kredentif keamanan sementara ini untuk membuat permintaan ke Amazon SQS. Pustaka API menghitung nilai tanda tangan yang diperlukan menggunakan kredensial tersebut untuk mengautentikasi permintaan Anda. Jika Anda mengirim permintaan menggunakan kredenSIAL kedaluwarsa, Amazon SQS menolak permintaan tersebut.

> **Note**
>
> Anda tidak dapat menetapkan kebijakan berdasarkan kredensi sementara.

Prasyarat

1. Gunakan IAM untuk membuat kredenSIAL keamanan sementara:
   - Token keamanan

- Access key ID
- Kunci Akses Rahasia

2. Siapkan string Anda untuk ditandatangani dengan ID Kunci Akses sementara dan token keamanan.
3. Gunakan Kunci Akses Rahasia sementara alih-alih Kunci Akses Rahasia Anda sendiri untuk menandatangani permintaan API Kueri Anda.

> **Note**
>
> Saat Anda mengirimkan permintaan API Kueri yang ditandatangani, gunakan ID Kunci Akses sementara alih-alih ID Kunci Akses Anda sendiri dan untuk menyertakan token keamanan. Untuk informasi selengkapnya tentang dukungan IAM untuk kredenSIAL keamanan sementara, lihat Mem berikan Akses Sementara ke Sumber AWS Daya Anda di Panduan Pengguna IAM.

Untuk memanggil tindakan Amazon SQS Query API menggunakan kredenSIAL keamanan sementara

1. Minta token keamanan sementara menggunakan AWS Identity and Access Management. Untuk informasi selengkapnya, lihat Membuat KredenSIAL Keamanan Sementara untuk Mengaktifkan Akses bagi Pengguna IAM di Panduan Pengguna IAM.

   IAM mengembalikan token keamanan, ID Kunci Akses, dan Kunci Akses Rahasia.
2. Siapkan kueri Anda menggunakan ID Kunci Akses sementara alih-alih ID Kunci Akses Anda sendiri dan sertakan token keamanan. Tanda tangani permintaan Anda menggunakan Kunci Akses Rahasia sementara, bukan milik Anda sendiri.
3. Kirim string kueri yang ditandatangani dengan ID Kunci Akses sementara dan token keamanan.

   Contoh berikut menunjukkan cara menggunakan kredentif keamanan sementara untuk mengotentikasi permintaan Amazon SQS. Struktur *AUTHPARAMS* tergantung pada tanda tangan permintaan API. Untuk informasi selengkapnya, lihat Menandatangani Permintaan AWS API di Referensi Umum Amazon Web Services.

```
https://sqs.us-east-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
```

```
&QueueName=MyQueue
&Attribute.1.Name=VisibilityTimeout
&Attribute.1.Value=40
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

Contoh berikut menggunakan kredentif keamanan sementara untuk mengirim dua pesan menggunakan `SendMessageBatch` tindakan.

```
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
```

Manajemen akses untuk antrian Amazon SQS terenkripsi dengan kebijakan hak istimewa paling sedikit

Anda dapat menggunakan Amazon SQS untuk bertukar data sensitif antar aplikasi dengan menggunakan enkripsi sisi server (SSE) yang terintegrasi dengan (KMS).AWS Key Management Service Dengan integrasi Amazon SQS dan AWS KMS, Anda dapat mengelola kunci yang melindungi Amazon SQS secara terpusat, serta kunci yang melindungi sumber daya Anda yang lain. AWS

Beberapa AWS layanan dapat bertindak sebagai sumber peristiwa yang mengirim acara ke Amazon SQS. Untuk mengaktifkan sumber peristiwa untuk mengakses antrean Amazon SQS terenkripsi, Anda perlu mengonfigurasi antrian dengan kunci yang dikelola pelanggan. AWS KMS Kemudian, gunakan kebijakan kunci untuk mengizinkan layanan menggunakan metode AWS KMS API yang diperlukan. Layanan ini juga memerlukan izin untuk mengautentikasi akses untuk mengaktifkan antrian untuk mengirim acara. Anda dapat mencapai ini dengan menggunakan kebijakan Amazon

SQS, yang merupakan kebijakan berbasis sumber daya yang dapat Anda gunakan untuk mengontrol akses ke antrean Amazon SQS dan datanya.

Bagian berikut memberikan informasi tentang cara mengontrol akses ke antrean Amazon SQS terenkripsi melalui kebijakan Amazon SQS dan kebijakan utama. AWS KMS Kebijakan dalam panduan ini akan membantu Anda mencapai hak istimewa paling sedikit.

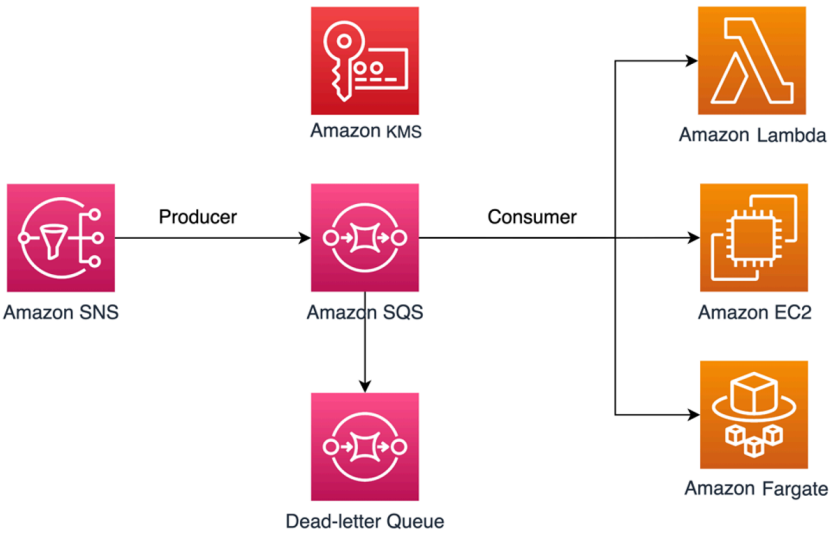
Panduan ini juga menjelaskan bagaimana kebijakan berbasis sumber daya mengatasi masalah wakil yang membingungkan dengan menggunakan kunci konteks kondisi IAM `aws:SourceArn`,, `aws:SourceAccount`dan global. `aws:PrincipalOrgID`

Topik

- Ikhtisar
- Kebijakan kunci hak istimewa paling sedikit untuk Amazon SQS
- Pernyataan kebijakan Amazon SQS untuk antrian huruf mati
- Mencegah masalah wakil lintas layanan yang membingungkan
- Gunakan IAM Access Analyzer untuk meninjau akses lintas akun

Ikhtisar

Dalam topik ini, kami akan memandu Anda melalui kasus penggunaan umum untuk menggambarkan bagaimana Anda dapat membangun kebijakan kunci dan kebijakan antrian Amazon SQS. Kasus penggunaan ini ditunjukkan pada gambar berikut.



Dalam contoh ini, produsen pesan adalah topik Amazon Simple Notification Service (SNS), yang dikonfigurasi untuk meng-fanout pesan ke antrean Amazon SQS terenkripsi Anda. Konsumen pesan

adalah layanan komputasi, seperti AWS Lambdafungsi, instans Amazon Elastic Compute Cloud (EC2), atau container. AWS Fargate Antrian Amazon SQS Anda kemudian dikonfigurasi untuk mengirim pesan gagal ke Dead-Letter Queue (DLQ). Ini berguna untuk men-debug aplikasi atau sistem pesan Anda karena DLQs memungkinkan Anda mengisolasi pesan yang tidak dikonsumsi untuk menentukan mengapa pemrosesannya tidak berhasil. Dalam solusi yang ditentukan dalam topik ini, layanan komputasi seperti fungsi Lambda digunakan untuk memproses pesan yang disimpan dalam antrean Amazon SQS. Jika konsumen pesan berada di cloud pribadi virtual (VPC), pernyataan `DenyReceivingIfNotThroughVPCE`kebijakan yang disertakan dalam panduan ini memungkinkan Anda membatasi penerimaan pesan ke VPC tertentu.

> **Note**
>
> Panduan ini hanya berisi izin IAM yang diperlukan dalam bentuk pernyataan kebijakan. Untuk membuat kebijakan, Anda perlu menambahkan pernyataan ke kebijakan Amazon SQS atau kebijakan utama AWS KMS Anda. Panduan ini tidak memberikan petunjuk tentang cara membuat antrean Amazon SQS atau kuncinya. AWS KMS Untuk petunjuk tentang cara membuat sumber daya ini, lihat Membuat antrian Amazon SQS dan Membuat kunci. Kebijakan Amazon SQS yang ditentukan dalam panduan ini tidak mendukung penggerak ulang pesan secara langsung ke antrean Amazon SQS yang sama atau berbeda.

Kebijakan kunci hak istimewa paling sedikit untuk Amazon SQS

Di bagian ini, kami menjelaskan izin hak istimewa terkecil yang diperlukan untuk kunci yang dikelola pelanggan yang Anda gunakan AWS KMS untuk mengenkripsi antrean Amazon SQS Anda. Dengan izin ini, Anda dapat membatasi akses hanya ke entitas yang dituju sambil menerapkan hak istimewa paling sedikit. Kebijakan utama harus terdiri dari pernyataan kebijakan berikut, yang kami jelaskan secara rinci di bawah ini:

- Berikan izin administrator ke kunci AWS KMS
- Berikan akses hanya-baca ke metadata kunci
- Berikan izin Amazon SNS KMS ke Amazon SNS untuk mempublikasikan pesan ke antrian
- Memungkinkan konsumen untuk mendekripsi pesan dari antrian

Berikan izin administrator ke kunci AWS KMS

Untuk membuat AWS KMS kunci, Anda perlu memberikan izin AWS KMS administrator ke peran IAM yang Anda gunakan untuk menyebarkan kunci. AWS KMS Izin administrator ini didefinisikan dalam pernyataan `AllowKeyAdminPermissions` kebijakan berikut. Saat menambahkan pernyataan ini ke kebijakan AWS KMS kunci, pastikan untuk mengganti *<admin-role ARN>* dengan Amazon Resource Name (ARN) peran IAM yang digunakan untuk menyebarkan AWS KMS kunci, mengelola kunci, atau keduanya AWS KMS . Ini bisa berupa peran IAM dari pipeline penerapan Anda, atau peran administrator untuk organisasi Anda di AWS Organizations Anda.

```
{
  "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**
>
> Dalam kebijakan AWS KMS kunci, nilai `Resource` elemen harus`*`, yang berarti " AWS KMS kunci ini". Tanda bintang (`*`) mengidentifikasi AWS KMS kunci yang dilampirkan kebijakan kunci.

Berikan akses hanya-baca ke metadata kunci

Untuk memberikan akses hanya-baca peran IAM lainnya ke metadata kunci Anda, tambahkan `AllowReadAccessToKeyMetaData` pernyataan tersebut ke kebijakan kunci Anda. Misalnya, pernyataan berikut memungkinkan Anda mencantumkan semua AWS KMS kunci di akun Anda untuk tujuan audit. Pernyataan ini memberikan akses read-only kepada pengguna AWS root ke metadata kunci. Oleh karena itu, setiap prinsipal IAM di akun dapat memiliki akses ke metadata kunci ketika kebijakan berbasis identitas mereka memiliki izin yang tercantum dalam pernyataan berikut:,, dan. `kms:Describe*` `kms:Get*` `kms:List*` Pastikan untuk mengganti *<account-ID>* dengan informasi Anda sendiri.

```
{
  "Sid": "AllowReadAcesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}
```

Berikan izin Amazon SNS KMS ke Amazon SNS untuk mempublikasikan pesan ke antrian

Untuk mengizinkan topik Amazon SNS Anda memublikasikan pesan ke antrean Amazon SQS terenkripsi, tambahkan `AllowSNSToSendToSQS` pernyataan kebijakan ke kebijakan utama Anda. Pernyataan ini memberikan izin Amazon SNS untuk menggunakan AWS KMS kunci untuk

mempublikasikan ke antrian Amazon SQS Anda. Pastikan untuk mengganti *<account-ID>* dengan informasi Anda sendiri.

> **Note**
>
> `Condition`Dalam pernyataan membatasi akses hanya ke layanan Amazon SNS di akun yang sama AWS .

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
}
```

Memungkinkan konsumen untuk mendekripsi pesan dari antrian

`AllowConsumersToReceiveFromTheQueue`Pernyataan berikut memberi konsumen pesan Amazon SQS izin yang diperlukan untuk mendekripsi pesan yang diterima dari antrian Amazon SQS terenkripsi. Saat Anda melampirkan pernyataan kebijakan, ganti *<consumer's runtime role ARN>* dengan ARN peran runtime IAM dari konsumen pesan.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
```

```
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

Kebijakan Amazon SQS dengan hak istimewa paling sedikit

Bagian ini memandu Anda melalui kebijakan antrian Amazon SQS dengan hak istimewa paling sedikit untuk kasus penggunaan yang dicakup oleh panduan ini (misalnya, Amazon SNS ke Amazon SQS). Kebijakan yang ditetapkan dirancang untuk mencegah akses yang tidak diinginkan dengan menggunakan campuran keduanya `Deny` dan `Allow` pernyataan. `Allow`Pernyataan memberikan akses ke entitas atau entitas yang dimaksud. `Deny`Pernyataan tersebut mencegah entitas lain yang tidak diinginkan mengakses antrean Amazon SQS, sementara mengecualikan entitas yang dimaksud dalam kondisi kebijakan.

Kebijakan Amazon SQS mencakup pernyataan berikut, yang kami jelaskan secara rinci di bawah ini:

- Batasi izin manajemen Amazon SQS
- Batasi tindakan antrian Amazon SQS dari organisasi yang ditentukan
- Berikan izin Amazon SQS kepada konsumen
- Menegakkan enkripsi dalam perjalanan
- Batasi transmisi pesan ke topik Amazon SNS tertentu
- (Opsional) Batasi penerimaan pesan ke titik akhir VPC tertentu

Batasi izin manajemen Amazon SQS

Pernyataan `RestrictAdminQueueActions` kebijakan berikut membatasi izin pengelolaan Amazon SQS hanya untuk peran atau peran IAM yang Anda gunakan untuk menerapkan antrean, mengelola antrian, atau keduanya. Pastikan untuk mengganti *<placeholder values>* dengan informasi Anda sendiri. Tentukan ARN peran IAM yang digunakan untuk menerapkan antrean Amazon SQS, serta peran administrator apa pun yang harus memiliki izin ARNs pengelolaan Amazon SQS.

```
{
```

```
  "Sid": "RestrictAdminQueueActions",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes"
  ],
  "Resource": "<SQS Queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": [
        "arn:aws:iam::<account-id>:role/<deployment-role-name>",
        "<admin-role ARN>"
      ]
    }
  }
}
```

Batasi tindakan antrian Amazon SQS dari organisasi yang ditentukan

Untuk membantu melindungi sumber daya Amazon SQS Anda dari akses eksternal (akses oleh entitas di luar AWS organisasi Anda), gunakan pernyataan berikut. Pernyataan ini membatasi akses antrian Amazon SQS ke organisasi yang Anda tentukan di. `Condition` Pastikan untuk mengganti *<SQS queue ARN>* dengan ARN peran IAM yang digunakan untuk menerapkan antrean Amazon SQS; dan, dengan ID organisasi Anda. *<org-id>*

```
{
  "Sid": "DenyQueueActionsOutsideOrg",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes",
    "sqs:ReceiveMessage"
```

```
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:PrincipalOrgID": [
        "<org-id>"
      ]
    }
  }
}
```

Berikan izin Amazon SQS kepada konsumen

Untuk menerima pesan dari antrian Amazon SQS, Anda perlu memberikan izin yang diperlukan kepada konsumen pesan. Pernyataan kebijakan berikut memberi konsumen, yang Anda tentukan, izin yang diperlukan untuk menggunakan pesan dari antrean Amazon SQS. Saat menambahkan pernyataan ke kebijakan Amazon SQS Anda, pastikan untuk mengganti *<consumer's IAM runtime role ARN>* dengan ARN peran runtime IAM yang digunakan oleh konsumen; dan, *<SQS queue ARN>* dengan ARN peran IAM yang digunakan untuk menerapkan antrean 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>"
}
```

Untuk mencegah entitas lain menerima pesan dari antrean Amazon SQS, tambahkan `DenyOtherConsumersFromReceiving` pernyataan ke kebijakan antrian Amazon SQS. Pernyataan ini membatasi konsumsi pesan kepada konsumen yang Anda tetapkan—tidak memungkinkan konsumen lain untuk memiliki akses, bahkan ketika izin identitas mereka akan

memberi mereka akses. Pastikan untuk mengganti *<SQS queue ARN>* dan *<consumer's runtime role ARN>* dengan informasi Anda sendiri.

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

Menegakkan enkripsi dalam perjalanan

Pernyataan `DenyUnsecureTransport` kebijakan berikut memberlakukan konsumen dan produsen untuk menggunakan saluran aman (koneksi TLS) untuk mengirim dan menerima pesan dari antrian Amazon SQS. Pastikan untuk mengganti *<SQS queue ARN>* dengan ARN dari peran IAM yang digunakan untuk menyebarkan antrian Amazon SQS.

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
```

```
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}
```

Batasi transmisi pesan ke topik Amazon SNS tertentu

Pernyataan `AllowSNSToSendToTheQueue` kebijakan berikut memungkinkan topik Amazon SNS yang ditentukan untuk mengirim pesan ke antrean Amazon SQS. Pastikan untuk mengganti *<SQS queue ARN>* dengan ARN peran IAM yang digunakan untuk menyebarkan antrian Amazon SQS; dan, dengan *<SNS topic ARN>* topik Amazon SNS ARN.

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

Pernyataan `DenyAllProducersExceptSNSFromSending` kebijakan berikut mencegah produsen lain mengirim pesan ke antrian. Ganti *<SQS queue ARN>* dan *<SNS topic ARN>* dengan informasi Anda sendiri.

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
```

```
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(Opsional) Batasi penerimaan pesan ke titik akhir VPC tertentu

Untuk membatasi penerimaan pesan hanya ke titik akhir VPC tertentu, tambahkan pernyataan kebijakan berikut ke kebijakan antrian Amazon SQS Anda. Pernyataan ini mencegah konsumen pesan menerima pesan dari antrian kecuali pesan tersebut berasal dari titik akhir VPC yang diinginkan. Ganti *<SQS queue ARN>* dengan ARN dari peran IAM yang digunakan untuk menyebarkan antrian Amazon SQS; dan dengan *<vpce_id>* ID titik akhir VPC.

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

Pernyataan kebijakan Amazon SQS untuk antrian huruf mati

Tambahkan pernyataan kebijakan berikut, yang diidentifikasi oleh ID pernyataan mereka, ke kebijakan akses DLQ Anda:

- `RestrictAdminQueueActions`
- `DenyQueueActionsOutsideOrg`
- `AllowConsumersToReceiveFromTheQueue`

- `DenyOtherConsumersFromReceiving`
- `DenyUnsecureTransport`

Selain menambahkan pernyataan kebijakan sebelumnya ke kebijakan akses DLQ Anda, Anda juga harus menambahkan pernyataan untuk membatasi transmisi pesan ke antrian Amazon SQS, seperti yang dijelaskan di bagian berikut.

Batasi transmisi pesan ke antrian Amazon SQS

Untuk membatasi akses hanya antrian Amazon SQS dari akun yang sama, tambahkan pernyataan kebijakan `DenyAnyProducersExceptSQS` berikut ke kebijakan antrian DLQ. Pernyataan ini tidak membatasi transmisi pesan ke antrian tertentu karena Anda perlu menerapkan DLQ sebelum membuat antrian utama, sehingga Anda tidak akan tahu Amazon SQS ARN saat membuat DLQ. Jika Anda perlu membatasi akses ke hanya satu antrian Amazon SQS, ubah `aws:SourceArn` di dengan ARN dari antrian sumber Amazon SQS Anda saat Anda mengetahuinya. `Condition`

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

Important

Kebijakan antrean Amazon SQS yang ditentukan dalam panduan ini tidak membatasi `sqs:PurgeQueue` tindakan ke peran atau peran IAM tertentu. `sqs:PurgeQueue`Tindakan ini memungkinkan Anda untuk menghapus semua pesan dalam antrean Amazon SQS. Anda juga dapat menggunakan tindakan ini untuk membuat perubahan pada format pesan tanpa mengganti antrean Amazon SQS. Saat men-debug aplikasi, Anda dapat menghapus antrean Amazon SQS untuk menghapus pesan yang berpotensi salah. Saat menguji aplikasi,

Anda dapat mengarahkan volume pesan tinggi melalui antrian Amazon SQS dan kemudian membersihkan antrian untuk memulai yang baru sebelum memasuki produksi. Alasan untuk tidak membatasi tindakan ini ke peran tertentu adalah bahwa peran ini mungkin tidak diketahui saat menerapkan antrian Amazon SQS. Anda perlu menambahkan izin ini ke kebijakan berbasis identitas peran agar dapat membersihkan antrian.

Mencegah masalah wakil lintas layanan yang membingungkan

Masalah deputi yang membingungkan adalah masalah keamanan di mana entitas yang tidak memiliki izin untuk melakukan tindakan dapat memaksa entitas yang lebih istimewa untuk melakukan tindakan. Untuk mencegah hal ini, AWS sediakan alat yang membantu Anda melindungi akun Anda jika Anda memberi pihak ketiga (dikenal sebagai lintas akun) atau AWS layanan lain (dikenal sebagai lintas layanan) akses ke sumber daya di akun Anda. Pernyataan kebijakan di bagian ini dapat membantu Anda mencegah masalah wakil lintas layanan yang membingungkan.

Peniruan identitas lintas layanan dapat terjadi ketika satu layanan (layanan yang dipanggil) memanggil layanan lain (layanan yang dipanggil). Layanan panggilan dapat dimanipulasi untuk menggunakan izinnya untuk bertindak atas sumber daya pelanggan lain dengan cara yang seharusnya tidak memiliki izin untuk mengakses. Untuk membantu melindungi dari masalah ini, kebijakan berbasis sumber daya yang ditentukan dalam posting ini menggunakan kunci konteks kondisi IAM `aws:SourceArn`aws:SourceAccount, dan `aws:PrincipalOrgID`global. Ini membatasi izin yang dimiliki layanan untuk sumber daya tertentu, akun tertentu, atau organisasi tertentu di AWS Organizations.

Gunakan IAM Access Analyzer untuk meninjau akses lintas akun

Anda dapat menggunakan AWS IAM Access Analyzer untuk meninjau kebijakan antrean Amazon SQS AWS KMS dan kebijakan kunci serta memberi tahu Anda saat antrian Amazon SQS atau kunci AWS KMS memberikan akses ke entitas eksternal. IAM Access Analyzer membantu mengidentifikasi sumber daya di organisasi dan akun Anda yang dibagikan dengan entitas di luar zona kepercayaan. Zona kepercayaan ini dapat berupa AWS akun atau AWS organisasi dalam Organizations yang Anda tentukan saat mengaktifkan IAM Access Analyzer.

IAM Access Analyzer mengidentifikasi sumber daya yang dibagikan dengan prinsipal eksternal dengan menggunakan penalaran berbasis logika untuk menganalisis kebijakan berbasis sumber daya di lingkungan Anda. AWS Untuk setiap contoh sumber daya yang dibagikan di luar zona kepercayaan Anda, Access Analyzer menghasilkan temuan. Temuan mencakup informasi tentang akses dan prinsip eksternal yang diberikan kepadanya. Tinjau temuan untuk menentukan apakah

akses dimaksudkan dan aman, atau apakah akses tidak diinginkan dan risiko keamanan. Untuk akses yang tidak diinginkan, tinjau kebijakan yang terpengaruh dan perbaiki. Lihat posting blog ini untuk informasi lebih lanjut tentang bagaimana AWS IAM Access Analyzer mengidentifikasi akses yang tidak diinginkan ke sumber daya Anda. AWS

Untuk informasi selengkapnya tentang AWS IAM Access Analyzer, lihat dokumentasi AWS IAM Access Analyzer.

Izin API Amazon SQS: Tindakan dan referensi sumber daya

Saat Anda mengatur Kontrol akses dan menulis kebijakan izin yang dapat Anda lampirkan ke identitas IAM, Anda dapat menggunakan tabel berikut sebagai referensi. setiap tindakan Amazon Simple Queue Service, tindakan terkait yang dapat Anda berikan izin untuk melakukan tindakan, dan AWS sumber daya yang dapat Anda berikan izin.

Tentukan tindakan di `Action` bidang kebijakan, dan nilai sumber daya di `Resource` bidang kebijakan. Untuk menentukan tindakan, gunakan `sqs:` awalan diikuti dengan nama tindakan (misalnya,`sqs:CreateQueue`).

Saat ini, Amazon SQS mendukung kunci konteks kondisi global yang tersedia di I AM.

Amazon Simple Queue Service API dan izin yang diperlukan untuk tindakan

AddPermission

Tindakan: `sqs:AddPermission`

Sumber daya: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

ChangeMessageVisibility

Tindakan: `sqs:ChangeMessageVisibility`

Sumber daya: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

ChangeMessageVisibilityBatch

Tindakan: `sqs:ChangeMessageVisibilityBatch`

Sumber daya: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

CreateQueue

Tindakan: `sqs:CreateQueue`

Sumber daya: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

DeleteMessage

Tindakan: `sqs:DeleteMessage`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessageBatch

Tindakan: `sqs:DeleteMessageBatch`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

DeleteQueue

Tindakan: `sqs:DeleteQueue`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

GetQueueAttributes

Tindakan: `sqs:GetQueueAttributes`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

GetQueueUrl

Tindakan: `sqs:GetQueueUrl`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

ListDeadLetterSourceQueues

Tindakan: `sqs:ListDeadLetterSourceQueues`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

ListQueues

Tindakan: `sqs:ListQueues`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

ListQueueTags

Tindakan: `sqs:ListQueueTags`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

PurgeQueue

Tindakan: `sqs:PurgeQueue`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

ReceiveMessage

Tindakan: `sqs:ReceiveMessage`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

RemovePermission

Tindakan: `sqs:RemovePermission`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

SendMessage dan SendMessageBatch

Tindakan: `sqs:SendMessage`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

SetQueueAttributes

Tindakan: `sqs:SetQueueAttributes`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

TagQueue

Tindakan: `sqs:TagQueue`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

UntagQueue

Tindakan: `sqs:UntagQueue`

Sumber daya: `arn:aws:sqs:region:account_id:queue_name`

Pencatatan dan pemantauan di Amazon SQS

Amazon Simple Queue Service terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau. Layanan AWS CloudTrail menangkap semua panggilan API untuk Amazon SQS sebagai peristiwa. Panggilan yang diambil termasuk panggilan dari konsol Amazon SQS dan panggilan kode ke operasi Amazon SQS API. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan

yang dibuat untuk Amazon SQS, alamat IP dari mana permintaan dibuat, kapan dibuat, dan detail tambahan.

Setiap entri peristiwa atau log berisi informasi tentang entitas yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut hal ini:

- Baik permintaan tersebut dibuat dengan kredensial pengguna root atau pengguna.
- Apakah permintaan dibuat atas nama pengguna IAM Identity Center.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan tersebut dibuat oleh Layanan AWS lain.

CloudTrail aktif di Anda Akun AWS ketika Anda membuat akun dan Anda secara otomatis memiliki akses ke riwayat CloudTrail Acara. Riwayat CloudTrail Acara menyediakan catatan yang dapat dilihat, dapat dicari, dapat diunduh, dan tidak dapat diubah dari 90 hari terakhir dari peristiwa manajemen yang direkam dalam file. Wilayah AWS Untuk informasi selengkapnya, lihat Bekerja dengan riwayat CloudTrail Acara di Panduan AWS CloudTrail Pengguna. Tidak ada CloudTrail biaya untuk melihat riwayat Acara.

Untuk catatan acara yang sedang berlangsung dalam 90 hari Akun AWS terakhir Anda, buat jejak atau penyimpanan data acara CloudTrailDanau.

CloudWatch Alarm Amazon

Pantau satu metrik selama periode waktu yang Anda tentukan, dan lakukan satu atau beberapa tindakan berdasarkan nilai metrik relatif terhadap ambang batas yang ditentukan selama beberapa periode. Misalnya, Anda dapat mengonfigurasi CloudWatch alarm untuk mengirim pemberitahuan ke topik Amazon SNS atau memicu tindakan untuk mengirim pesan ke antrean Amazon SQS. CloudWatch alarm tidak melakukan tindakan hanya karena mereka berada dalam keadaan tertentu; negara harus berubah dan tetap dalam keadaan itu untuk sejumlah periode tertentu.

Untuk informasi selengkapnya, lihat Membuat CloudWatch alarm untuk metrik Amazon SQS dan Membuat alarm untuk antrian huruf mati menggunakan Amazon CloudWatch.

CloudWatch Log Amazon

Pantau, simpan, dan akses file log yang terkait dengan Amazon SQS dengan mengonfigurasi aplikasi atau fungsi Lambda Anda yang memproses pesan untuk mengirim log ke Log.

CloudWatch Anda dapat menggunakan log ini untuk menganalisis pemrosesan pesan, masalah debug, dan memantau kinerja alur kerja Amazon SQS Anda.

Untuk informasi selengkapnya, lihat Mencatat panggilan API Layanan Antrian Sederhana Amazon menggunakan AWS CloudTrail.

CloudWatch Acara Amazon

Gunakan Amazon CloudWatch Events untuk mendeteksi perubahan atau peristiwa tertentu di AWS lingkungan Anda dan mengarahkannya ke antrian Amazon SQS. Ini memungkinkan Anda untuk menangkap data peristiwa, memicu alur kerja, atau menyimpan peristiwa untuk diproses nanti.

Untuk informasi lebih lanjut, lihat Mengotomatiskan pemberitahuan dari AWS layanan ke Amazon SQS menggunakan Amazon EventBridge di panduan ini dan EventBridge merupakan evolusi CloudWatch Acara Amazon di Panduan EventBridge Pengguna Amazon.

AWS CloudTrail Log

CloudTrail menangkap catatan rinci tindakan yang dilakukan di Amazon SQS oleh pengguna, peran, atau. Layanan AWS Log ini memungkinkan Anda melacak panggilan API `SendMessage`, seperti `ReceiveMessage`,, atau `DeleteQueue`, dan memberikan detail penting seperti siapa yang membuat permintaan, kapan itu terjadi, dan alamat IP asal.

Untuk informasi selengkapnya, lihat Mencatat panggilan API Layanan Antrian Sederhana Amazon menggunakan AWS CloudTrail.

AWS Trusted Advisor

Trusted Advisor menggunakan praktik terbaik yang dikembangkan dari melayani AWS pelanggan untuk membantu mengoptimalkan penggunaan Amazon SQS Anda. Ini meninjau antrian Amazon SQS Anda dan memberikan rekomendasi yang dapat ditindaklanjuti untuk meningkatkan keamanan, meningkatkan keandalan pemrosesan pesan, dan mengurangi biaya. Misalnya, mungkin menyarankan mengaktifkan antrian surat mati atau untuk meningkatkan kebijakan akses antrian Anda untuk memastikan operasi yang aman.

Untuk informasi selengkapnya, lihat AWS Trusted Advisor di Dukungan Panduan Pengguna.

CloudTrail jalan setapak

Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Semua jalur yang dibuat menggunakan Konsol Manajemen AWS Multi-region. Anda dapat membuat jalur Single-region atau Multi-region dengan menggunakan. AWS CLI Membuat jejak Multi-wilayah

disarankan karena Anda menangkap aktivitas Wilayah AWS di semua akun Anda. Jika Anda membuat jejak wilayah Tunggal, Anda hanya dapat melihat peristiwa yang dicatat di jejak. Wilayah AWS Untuk informasi selengkapnya tentang jejak, lihat Membuat jejak untuk Anda Akun AWS dan Membuat jejak untuk organisasi di Panduan AWS CloudTrail Pengguna.

Anda dapat mengirimkan satu salinan acara manajemen yang sedang berlangsung ke bucket Amazon S3 Anda tanpa biaya CloudTrail dengan membuat jejak, namun, ada biaya penyimpanan Amazon S3. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga. Untuk informasi tentang harga Amazon S3, lihat Harga Amazon S3.

CloudTrail Penyimpanan data acara danau

CloudTrail Lake memungkinkan Anda menjalankan kueri berbasis SQL pada acara Anda. CloudTrail Lake mengubah peristiwa yang ada dalam format JSON berbasis baris ke format Apache ORC. ORC adalah format penyimpanan kolumnar yang dioptimalkan untuk pengambilan data dengan cepat. Peristiwa digabungkan ke dalam penyimpanan data peristiwa, yang merupakan kumpulan peristiwa yang tidak dapat diubah berdasarkan kriteria yang Anda pilih dengan menerapkan pemilih acara tingkat lanjut. Penyeleksi yang Anda terapkan ke penyimpanan data acara mengontrol peristiwa mana yang bertahan dan tersedia untuk Anda kueri. Untuk informasi lebih lanjut tentang CloudTrail Danau, lihat Bekerja dengan AWS CloudTrail Danau di Panduan AWS CloudTrail Pengguna.

CloudTrail Penyimpanan data acara danau dan kueri menimbulkan biaya. Saat Anda membuat penyimpanan data acara, Anda memilih opsi harga yang ingin Anda gunakan untuk penyimpanan data acara. Opsi penetapan harga menentukan biaya untuk menelan dan menyimpan peristiwa, dan periode retensi default dan maksimum untuk penyimpanan data acara. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga.

Mencatat panggilan API Layanan Antrian Sederhana Amazon menggunakan AWS CloudTrail

CloudTrail memungkinkan Anda untuk log dan memantau operasi Amazon SQS menggunakan dua jenis peristiwa: peristiwa data dan peristiwa manajemen. Ini memudahkan untuk melacak dan mengaudit aktivitas Amazon SQS di akun Anda.

Peristiwa data Amazon SQS di CloudTrail

Peristiwa data memberikan informasi tentang operasi sumber daya yang dilakukan pada atau di sumber daya (misalnya, mengirim pesan ke objek Amazon SQS). Ini juga dikenal sebagai operasi

bidang data. Peristiwa data sering kali merupakan aktivitas bervolume tinggi. Secara default, CloudTrail tidak mencatat peristiwa data. Riwayat CloudTrail peristiwa tidak merekam peristiwa data.

Biaya tambahan berlaku untuk peristiwa data. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga.

Anda dapat mencatat peristiwa data untuk jenis sumber daya Amazon SQS menggunakan CloudTrail konsol AWS CLI, atau operasi CloudTrail API. Untuk informasi selengkapnya tentang cara mencatat peristiwa data, lihat Mencatat peristiwa data dengan Konsol Manajemen AWS dan Logging peristiwa data dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

Untuk mencatat peristiwa data Amazon SQS CloudTrail, Anda harus menggunakan pemilih peristiwa lanjutan untuk mengonfigurasi sumber daya atau tindakan Amazon SQS tertentu yang ingin Anda log. Sertakan jenis sumber daya `AWS::SQS::Queue`untuk menangkap tindakan terkait antrian. Anda dapat menyempurnakan preferensi logging Anda lebih jauh dengan menggunakan filter seperti `eventName` (seperti `SendMessage`peristiwa). Untuk informasi selengkapnya, lihat AdvancedEventSelector di dalam Referensi API CloudTrail .

Jenis peristiwa data (konsol)	nilai resources.type	Data APIs masuk ke CloudTrail
Antrian Amazon SQS	`AWS::SQS::Queue`	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch • GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

Gunakan pemilih acara lanjutan untuk memfilter bidang dan hanya mencatat peristiwa penting. Untuk informasi selengkapnya tentang bidang ini, lihat AdvancedFieldSelectordi Referensi AWS CloudTrail API.

Acara manajemen Amazon SQS di CloudTrail

Acara manajemen memberikan informasi tentang operasi manajemen yang dilakukan pada sumber daya di Anda Akun AWS. Ini juga dikenal sebagai operasi bidang kontrol. Secara default, CloudTrail mencatat peristiwa manajemen.

Amazon SQS mencatat operasi bidang kontrol berikut CloudTrail sebagai peristiwa manajemen.

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTasks
- PurgeQueue
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Contoh acara Amazon SQS

Peristiwa mewakili permintaan tunggal dari sumber manapun dan mencakup informasi tentang operasi API yang diminta, tanggal dan waktu operasi, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, sehingga peristiwa tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan CloudTrail peristiwa yang menunjukkan `SendMessage` operasi.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
```

```
    "principalId": "EXAMPLE_PRINCIPAL_ID",
    "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/SessionName",
    "accountId": "123456789012",
    "accessKeyId": "ACCESS_KEY_ID",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed",
        "accountId": "123456789012",
        "userName": "RoleToBeAssumed"
      },
      "attributes": {
        "creationDate": "2023-11-07T22:13:06Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2023-11-07T23:59:11Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessage",
  "awsRegion": "ap-southeast-4",
  "sourceIPAddress": "10.0.118.80",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/
linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/
pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue",
    "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageDeduplicationId": "MsgDedupIdSdk1ae1958f2-bbe8-4442-83e7-4916e3b035aa",
    "messageGroupId": "MsgGroupIdSdk16"
  },
  "responseElements": {
    "mD5OfMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
    "mD5OfMessageSystemAttributes": "f88f0587f951b7f5551f18ae699c3a9d",
    "messageId": "93bb6e2d-1090-416c-81b0-31eb1faa8cd8",
    "sequenceNumber": "18881790870905840128"
  },
  "requestID": "c4584600-fe8a-5aa3-a5ba-1bc42f055fae",
  "eventID": "98c735d8-70e0-4644-9432-b6ced4d791b1",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
```

```
      "type": "AWS::SQS::Queue",
      "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sqs.ap-southeast-4.amazonaws.com"
  }
```

> **Note**
>
> `ListQueues`Operasi adalah kasus unik karena tidak bertindak pada sumber daya tertentu. Akibatnya, bidang ARN tidak menyertakan nama antrian dan menggunakan wildcard (*) sebagai gantinya.

Untuk informasi tentang konten CloudTrail rekaman, lihat konten CloudTrail rekaman di Panduan AWS CloudTrail Pengguna.

Memantau antrian Amazon SQS menggunakan CloudWatch

Amazon SQS dan Amazon CloudWatch terintegrasi sehingga Anda dapat menggunakannya CloudWatch untuk melihat dan menganalisis metrik untuk antrian Amazon SQS Anda. Anda dapat melihat dan menganalisis metrik antrian Anda dari konsol Amazon SQS, CloudWatch konsol, menggunakan AWS CLI, atau menggunakan API. CloudWatch Anda juga dapat mengatur CloudWatch alarm untuk metrik Amazon SQS.

CloudWatch metrik untuk antrian Amazon SQS Anda secara otomatis dikumpulkan dan didorong ke dalam CloudWatch interval satu menit. Metrik ini dikumpulkan pada semua antrian yang memenuhi CloudWatch pedoman untuk aktif. CloudWatch menganggap antrian aktif hingga enam jam jika berisi pesan, atau jika ada tindakan yang mengaksesnya.

Ketika antrian Amazon SQS tidak aktif selama lebih dari enam jam, layanan Amazon SQS dianggap tidur dan berhenti mengirimkan metrik ke CloudWatch layanan. Data yang hilang, atau data yang

mewakili nol, tidak dapat divisualisasikan dalam CloudWatch metrik untuk Amazon SQS untuk periode waktu antrian Amazon SQS Anda tidak aktif.

> **Note**
>
> - Antrian Amazon SQS dapat diaktifkan ketika pengguna memanggil API terhadap antrian tidak diotorisasi, dan permintaan gagal.
> - Konsol Amazon SQS melakukan panggilan `GetQueueAttributes` API saat halaman antrian dibuka. Per `GetQueueAttributes` mintaan API mengaktifkan antrian.
> - Penundaan hingga 15 menit terjadi dalam CloudWatch metrik ketika antrian diaktifkan dari keadaan tidak aktif.
> - Tidak ada biaya untuk metrik Amazon SQS yang dilaporkan di CloudWatch. Mereka disediakan sebagai bagian dari layanan Amazon SQS.
> - CloudWatch metrik didukung untuk antrian standar dan FIFO.

Mengakses CloudWatch metrik untuk Amazon SQS

Amazon SQS dan Amazon CloudWatch terintegrasi sehingga Anda dapat menggunakannya CloudWatch untuk melihat dan menganalisis metrik untuk antrian Amazon SQS Anda. Anda dapat melihat dan menganalisis metrik antrian Anda dari konsol Amazon SQS, CloudWatch konsol, menggunakan AWS CLI, atau menggunakan API. CloudWatch Anda juga dapat mengatur CloudWatch alarm untuk metrik Amazon SQS.

Menggunakan konsol Amazon SQS

Gunakan konsol Amazon SQS untuk mengakses dan menganalisis metrik hingga 10 antrian Amazon SQS.

1. Masuk ke konsol Amazon SQS.
2. Dalam daftar antrian, pilih (centang) kotak untuk antrian yang ingin Anda akses metriknya. Anda dapat menampilkan metrik hingga 10 antrian.
3. Pilih tab Pemantauan.

   Berbagai grafik ditampilkan di bagian metrik SQS.
4. Untuk memahami apa yang diwakili oleh grafik tertentu, arahkan kur

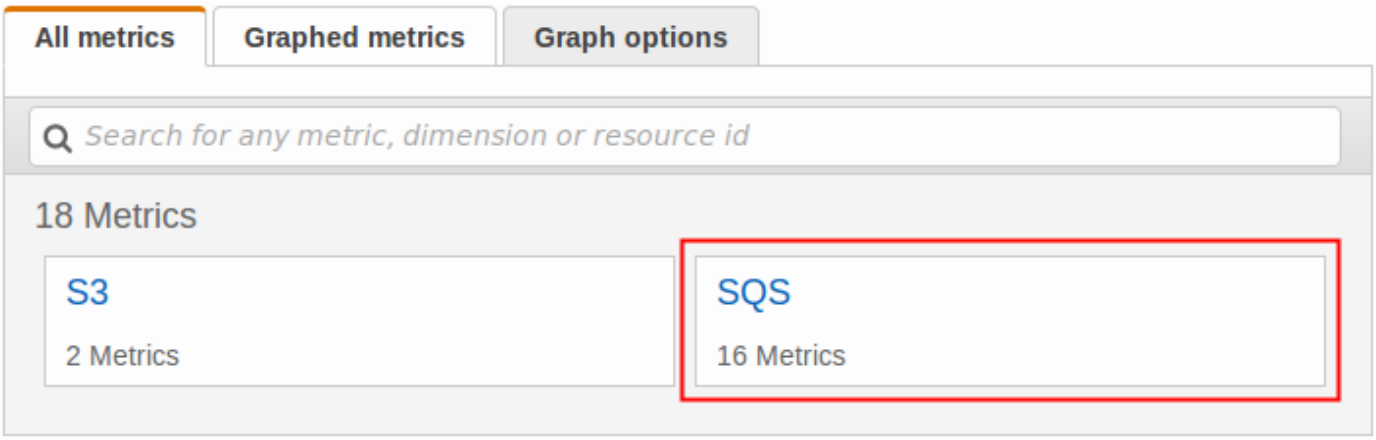
sor di sebelah grafik yang diinginkan, atau lihatCloudWatch Metrik yang tersedia untuk Amazon SQS.

5. Untuk mengubah rentang waktu untuk semua grafik secara bersamaan, untuk Rentang Waktu, pilih rentang waktu yang diinginkan (misalnya, Jam Terakhir).
6. Untuk melihat statistik tambahan untuk grafik individual, pilih grafik.
7. Di kotak dialog Rincian Pem CloudWatch antauan, pilih Statistik, (misalnya, Jumlah). Untuk daftar statistik yang didukung, lihatCloudWatch Metrik yang tersedia untuk Amazon SQS.
8. Untuk mengubah rentang waktu dan interval waktu yang ditampilkan grafik individual (misalnya, untuk menampilkan rentang waktu 24 jam terakhir, bukan 5 menit terakhir, atau untuk menampilkan periode waktu setiap jam, bukan setiap 5 menit), dengan kotak dialog grafik masih ditampilkan, untuk Rentang Waktu, pilih rentang waktu yang diinginkan (misalnya, 24 Jam Ter akhir). Untuk Peri ode, pilih periode waktu yang diinginkan dalam rentang waktu yang ditentukan (misalnya, 1 Jam). Setelah selesai melihat grafik, pilih Tutup.
9. (Opsional) Untuk bekerja dengan CloudWatch fitur tambahan, pada tab Pem antauan, pilih Lihat semua CloudWatch metrik, lalu ikuti petunjuk dalam Menggunakan CloudWatch konsol Amazon prosedur.

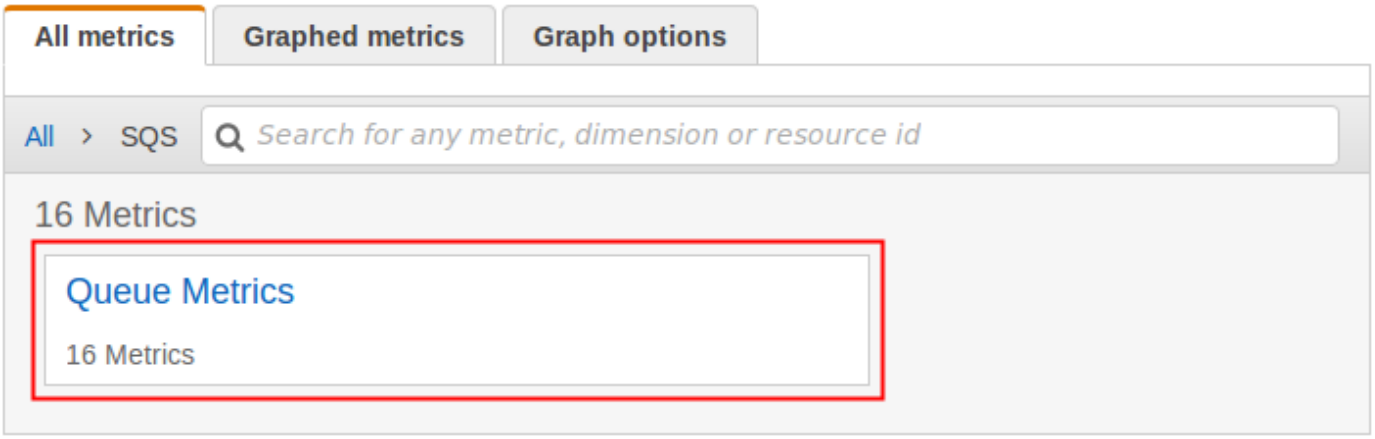
Menggunakan CloudWatch konsol Amazon

Gunakan CloudWatch konsol untuk mengakses dan menganalisis metrik Amazon SQS.

1. Masuk ke konsol CloudWatch tersebut.
2. Di panel navigasi, pilih Metrics (Metrik).
3. Pilih namespace metrik SQS.

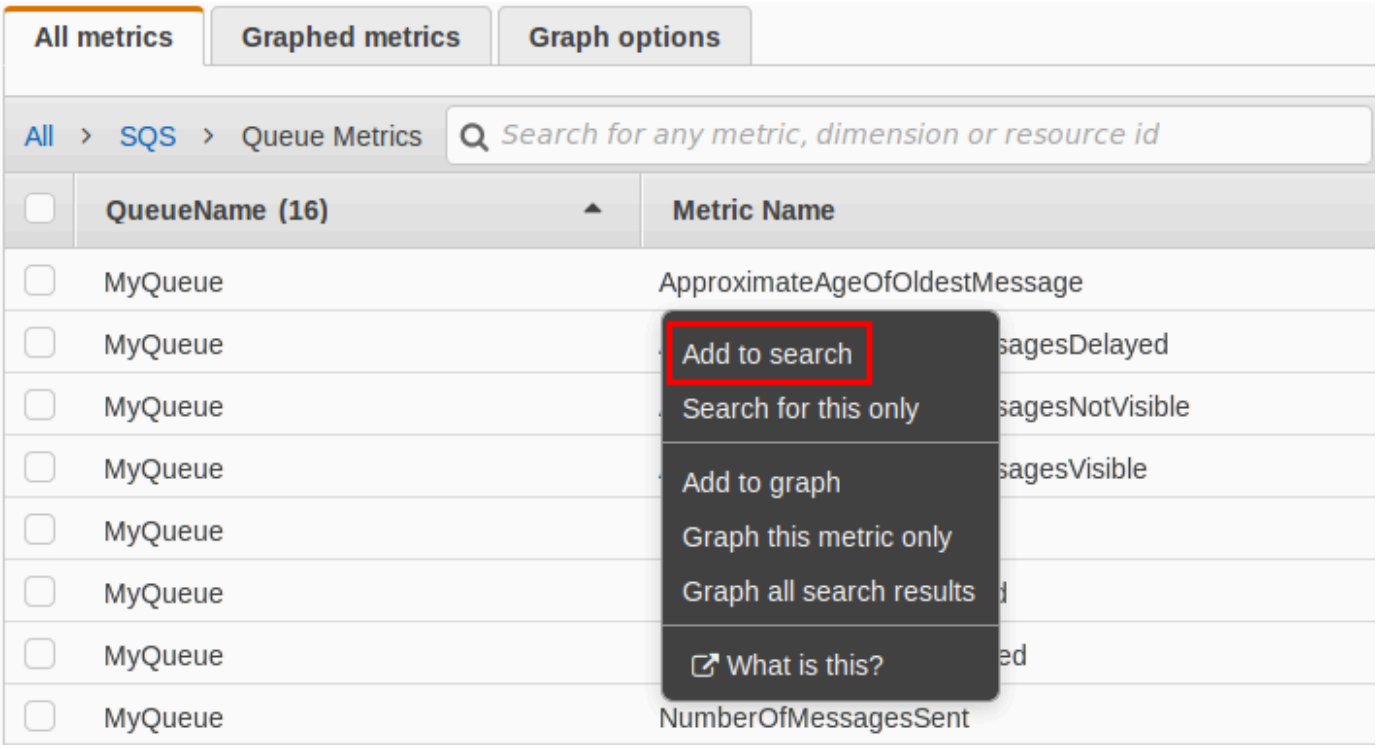


4. Pilih dimensi metrik Antrian Metrik.



5. Anda sekarang dapat memeriksa metrik Amazon SQS Anda:

   - Untuk mengurutkan metrik, gunakan judul kolom.
   - Untuk membuat grafik sebuah metrik, pilih kotak centang di samping metrik.
   - Untuk menyaring berdasarkan metrik, pilih nama metrik, kemudian pilih Tambahkan ke pencarian.



Untuk informasi selengkapnya dan opsi tambahan, lihat Metrik Grafik dan Menggunakan CloudWatch Dasbor Amazon di Panduan CloudWatch Pengguna Amazon.

Menggunakan AWS Command Line Interface

Untuk mengakses metrik Amazon SQS menggunakan AWS CLI, jalankan `get-metric-statistics` perintah.

Untuk informasi selengkapnya, lihat Dapatkan Statistik untuk Metrik di Panduan CloudWatch Pengguna Amazon.

Menggunakan CloudWatch API

Untuk mengakses metrik Amazon SQS menggunakan CloudWatch API, gunakan `GetMetricStatistics` tindakan.

Untuk informasi selengkapnya, lihat Dapatkan Statistik untuk Metrik di Panduan CloudWatch Pengguna Amazon.

Membuat CloudWatch alarm untuk metrik Amazon SQS

CloudWatch memungkinkan Anda memicu alarm ketika metrik mencapai ambang batas tertentu. Misalnya, Anda dapat membuat alarm untuk `NumberOfMessagesSent` metrik. Misalnya, jika lebih dari 100 pesan dikirim ke an `MyQueue` trian dalam 1 jam, pemberitahuan email dikirim. Untuk informasi selengkapnya, lihat Membuat CloudWatch Alarm Amazon di Panduan CloudWatch Pengguna Amazon.

1. Masuk ke Konsol Manajemen AWS dan buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pilih Alarm, lalu pilih Buat Alarm.
3. Di bagian Pilih Metrik pada kotak dialog Buat Alarm, pilih Browse Metrics, SQS.
4. Untuk SQS > Metrik Antrian, pilih QueueName dan Nama Metrik untuk menyetel alarm, lalu pilih Ber ikutnya. Untuk daftar metrik yang tersedia, lihatCloudWatch Metrik yang tersedia untuk Amazon SQS.

   Dalam contoh berikut, pemilihan adalah untuk alarm untuk `NumberOfMessagesSent` metrik untuk `MyQueue` antrian. Alarm dipicu ketika jumlah pesan yang dikirim melebihi 100.
5. Di bagian Define Alarm pada kotak dialog Create Alarm, lakukan hal berikut:

   a. Di bawah Ambang Alarm, ketik Nama dan Deskripsi untuk alarm.

   b. Set el adalah > 100.

c. Set el untuk 1 dari 1 titik data.

d. Di bawah pratinjau Alarm, atur Peri ode ke 1 Jam.

e. Atur Statistik ke Standar, Jumlah.

f. Di bawah Tind akan, set el Kapan pun alarm ini ke Status adalah ALARM.

Jika Anda CloudWatch ingin mengirim pemberitahuan saat alarm dipicu, pilih topik Amazon SNS yang ada atau pilih Daftar baru dan masukkan alamat email yang dipisahkan dengan koma.

> **Note**
>
> Jika Anda membuat topik Amazon SNS baru, alamat email harus diverifikasi sebelum menerima pemberitahuan apa pun. Jika status alarm berubah sebelum alamat email diverifikasi, notifikasi tidak dikirimkan.

6. Pilih Buat Alarm.

Alarm dibuat.

CloudWatch Metrik yang tersedia untuk Amazon SQS

Amazon SQS mengirimkan metrik berikut ke CloudWatch.

> **Note**
>
> Untuk beberapa metrik, hasilnya adalah perkiraan karena arsitektur terdistribusi Amazon SQS. Dalam kebanyakan kasus, jumlah harus mendekati jumlah sebenarnya dari pesan dalam antrian.

Metrik Amazon SQS

Amazon SQS secara otomatis menerbitkan metrik operasional ke Amazon CloudWatch di bawah `AWS/SQS` namespace. Metrik ini membantu Anda memantau kesehatan dan kinerja antrian. Karena sifat SQS yang terdistribusi, banyak nilai perkiraan, tetapi cukup akurat untuk sebagian besar keputusan operasional.

> **Note**
>
> - Semua metrik memancarkan nilai non-negatif hanya saat antrian aktif.
> - Beberapa metrik (sepertiSentMessageSize) tidak dipancarkan sampai setidaknya satu pesan dikirim.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
ApproximateAgeOfOldestMessage	Usia pesan tertua yang belum diproses dalam antrian.	Detik	Dilaporkan jika antrian berisi setidaknya satu pesan aktif.	• Untuk antrian standar, jika pesan diterima tiga kali atau lebih dan tidak dihapus, SQS memindahkannya ke belakang antrian. Metrik kemudian mencerminkan usia pesan berikutnya yang belum melebihi ambang penerimaan. Penataan ulang ini terjadi bahkan ketika kebijakan redrive diterapkan. • Poison-pill pesan (yang diterima berulang kali tetapi tidak pernah dihapus) dikecualikan dari metrik ini sampai berhasil diproses. • Ketika pesan dipindahkan ke DLQ setelah melebihimaxReceiveCount , usia diatur ulang. Dalam hal ini, metrik DLQ mencerminkan waktu pesan dipindahkan—bukan saat pesan itu awalnya dikirim.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
				• Antrian FIFO tidak menyusun ulang pesan untuk menjaga ketertiba n. Pesan yang gagal memblokir grup pesannya sampai dihapus atau kedaluwarsa. Jika DLQ dikonfigurasi, pesan dikirim ke sana setelah ambang penerimaan terpenuhi.
`Approxima teNumber0 fGroupsWi thInfligh tMessages`	Hanya untuk FIFO. Jumlah grup pesan dengan satu atau lebih pesan dalam penerbangan.	Hitungan	Dilaporkan jika antrian FIFO aktif.	• Pesan dianggap dalam penerbangan setelah diterima dari antrian oleh konsumen tetapi belum dihapus atau kedaluwarsa. • Metrik ini membantu Anda memecahkan masalah dan mengoptimalkan throughpu t antrian FIFO. Nilai tinggi biasanya menunjukkan konkurensi yang kuat. • Jika antrian memiliki backlog besar dan nilai ini tetap rendah, pertimban gkan untuk menskalakan konsumen atau meningkat kan jumlah grup pesan aktif. • Untuk throughput dan batas dalam penerbangan, lihatKuota Amazon SQS.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`ApproximateNumberOfMessagesDelayed`	Jumlah pesan dalam antrian yang tertunda dan tidak segera tersedia untuk diambil.	Hitungan	Dilaporkan jika ada pesan tertunda dalam antrian.	• Berlaku untuk antrian yang dikonfigurasi dengan penundaan default dan ke pesan individual yang dikirim dengan `DelaySeconds` parameter. • Pesan yang tertunda tetap tersembunyi dari konsumen sampai periode penundaan mereka berakhir, yang dapat memengaruhi backlog antrian atau throughput yang dirasakan.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`ApproximateNumberOfMessagesNotVisible`	Jumlah pesan dalam penerbangan yang telah diterima tetapi belum dihapus atau kedaluwarsa.	Hitungan	Dilaporkan jika ada pesan dalam penerbangan.	• Pesan memasuki status dalam penerbangan setelah dikirim ke konsumen melalui `ReceiveMessage` API. • Pesan-pesan ini disembunyikan sementara dari konsumen lain selama jendela batas waktu visibilitas. • Gunakan metrik ini untuk melacak keterlambatan pemrosesan pesan atau konsumen yang macet. • Pada kesempatan yang jarang terjadi, salah satu server yang digunakan Amazon SQS untuk menyimpan pesan Anda mungkin tidak tersedia saat Amazon SQS melaporkan metrik antrian. Ketika ini terjadi, metrik ini mungkin melaporkan nilai bukan nol, termasuk saat antrian kosong atau tidak memiliki `ReceiveMessage` panggilan. Biasanya diselesaikan dalam hitungan menit, jadi jika Anda menyetel alarm untuk metrik ini, evaluasi beberapa titik

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
				data berturut-turut daripada nilai tunggal bukan nol.
`ApproximateNumberOfMessagesVisible`	Jumlah pesan yang saat ini tersedia untuk pengambilan dan pemrosesan.	Hitungan	Dilaporkan jika antrian aktif.	• Mencerminkan backlog pemrosesan saat ini dalam antrian. • Tidak ada batasan ketat pada berapa banyak pesan yang dapat terakumulasi, tetapi pesan tersebut tunduk pada periode retensi yang dikonfigurasi antrian. • Nilai tinggi secara konsisten dapat mengindikasikan konsumen yang kurang disediakan atau logika pemrosesan yang macet.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`NumberOfEmptyReceives` [1]	Jumlah panggilan ReceiveMessage API yang tidak mengembalikan pesan.	Hitungan	Dilaporkan selama operasi penerimaan.	• Metrik ini dapat membantu mengidentifikasi inefisiensi dalam perilaku polling atau contoh konsumen yang kurang dimanfaatkan. • Nilai tinggi dapat terjadi ketika antrian kosong, konsumen menggunakan polling pendek, atau pesan diproses lebih cepat daripada yang dihasilkan. • Ini bukan indikator yang tepat dari status antrian. Ini mencerminkan perilaku sisi layanan dan mungkin termasuk percobaan ulang.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`NumberOfDeduplicatedSentMessages`	Hanya untuk FIFO. Jumlah pesan terkirim yang dideduplikasi dan tidak ditambahkan ke antrian.	Hitungan	Dilaporkan jika `MessageDeduplicationId` nilai atau konten duplikat terdeteksi.	• SQS mendeduplikasi pesan berdasarkan hashing berbasis `MessageDeduplicationId` konten (jika diaktifkan). • Nilai tinggi dapat menunjukkan bahwa produsen berulang kali mengirim pesan yang sama dalam jendela deduplikasi 5 menit. • Gunakan metrik ini untuk memecahkan masalah logika produsen yang berlebihan atau mengonfirmasi bahwa deduplikasi berfungsi sebagaimana dimaksud.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci

`NumberOfMessagesDeleted` [1]	Jumlah pesan yang berhasil dihapus dari antrian.	Hitungan	Dilaporkan untuk setiap permintaan penghapusan dengan pegangan tanda terima yang valid.	• Metrik ini menghitung semua operasi penghapusan yang berhasil — bahkan jika pesan yang sama dihapus lebih dari satu kali.
				• Alasan umum untuk nilai yang lebih tinggi dari yang diharapkan meliputi:
				  • Beberapa penghapusan pesan yang sama menggunakan pegangan tanda terima yang berbeda, setelah batas waktu visibilitas berakhir dan pesan diterima lagi.
				  • Penghapusan duplikat menggunakan pegangan tanda terima yang sama, yang masih mengembalikan status sukses dan menambah metrik.
				• Gunakan metrik ini untuk melacak keberhasilan pemrosesan pesan, tetapi jangan memperlakukannya sebagai jumlah pasti pesan unik yang dihapus.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`NumberOfMessagesReceived` [1]	Jumlah pesan yang dikembalikan oleh ReceiveMessage API.	Hitungan	Dilaporkan selama operasi penerimaan.	• Ini termasuk semua pesan yang dikembalikan ke konsumen, termasuk pesan yang kemudian dikembalikan ke antrian karena kedaluwarsa batas waktu visibilitas. • Satu pesan dapat diterima beberapa kali jika tidak dihapus, yang dapat menyebabkan metrik ini melebihi jumlah pesan yang dikirim. • Gunakan ini untuk melacak aktivitas konsumen, tetapi jangan memperlakukannya sebagai hitungan pesan unik yang diproses.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`NumberOfMessagesSent` [1]	Jumlah pesan yang berhasil ditambahkan ke antrian.	Hitungan	Dilaporkan untuk setiap pengiriman manual yang berhasil.	• Panggilan manual ke `SendMessage` atau `SendMessageBatch` dihitung, termasuk panggilan yang menargetkan DLQ secara langsung. • Pesan yang secara otomatis dipindahkan ke DLQ setelah melebihi tidak `maxReceiveCount` disertakan dalam metrik ini. • Akibatnya, `NumberOfMessagesSent` mungkin lebih rendah dari `NumberOfMessagesReceived` —terutama jika kebijakan redrive memindahkan banyak pesan ke DLQ di belakang layar.
`SentMessageSize` [1]	Ukuran pesan berhasil dikirim ke antrian.	Byte	Tidak dipancarkan sampai setidaknya satu pesan dikirim.	• Metrik ini tidak akan muncul di CloudWatch konsol sampai antrian menerima pesan pertamanya. • Gunakan metrik ini untuk melacak ukuran setiap pesan dalam byte. Ini berguna untuk menganalisis tren muatan atau memperkirakan biaya throughput. • Ukuran pesan maksimum untuk SQS adalah 1 MiB.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`ApproximateNumberOfNoisyGroups`	Jumlah grup pesan yang dianggap berisik dalam antrian yang adil. Grup pesan berisik mewakili penyewa tetangga yang berisik dari antrian multi-penyewa.	Hitungan	Nilai non-negatif dilaporkan jika antrian aktif.	• Membantu mengidentifikasi potensi masalah tetangga yang bising di lingkungan multi-penyewa dengan melacak grup pesan yang menggunakan sumber daya yang tidak proporsional. • Gunakan metrik ini untuk menyetel alarm yang dipicu ketika jumlah grup berisik melebihi ambang batas yang dapat diterima, yang menunjukkan potensi masalah keadilan antrian.
`ApproximateNumberOfMessagesVisibleInQuietGroups`	Jumlah pesan yang terlihat tidak termasuk pesan dari grup pesan berisik.	Hitungan	Nilai non-negatif dilaporkan jika antrian aktif.	• Memberikan visibilitas ke backlog antrian untuk grup pesan tingkat standar, tidak termasuk pesan dari tetangga yang bising. • Membantu mengidentifikasi backlog pemrosesan yang sebenarnya untuk grup pesan biasa dengan menyaring dampak tetangga yang bising.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`ApproximateNumberOfMessagesNotVisibleInQuietGroups`	Jumlah pesan dalam penerbangan tidak termasuk pesan dari grup pesan berisik.	Hitungan	Nilai non-negatif dilaporkan jika antrian aktif.	• Melacak pesan dalam penerbangan (sedang diproses tetapi belum dihapus) dari grup pesan yang berperilaku baik. • Gunakan metrik ini untuk memantau throughput pemrosesan grup pesan normal dan mendeteksi hambatan pemrosesan yang tidak disebabkan oleh tetangga yang bising.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`ApproximateNumberOfMessagesDelayedInQuietGroups`	Jumlah pesan yang tidak termasuk pesan dari grup pesan berisik yang tertunda dan tidak tersedia untuk dibaca segera. Pesan tertunda terjadi ketika antrian dikonfigurasi sebagai antrean penundaan atau ketika pesan telah dikirim dengan parameter delay.	Hitungan	Nilai non-negatif dilaporkan jika antrian aktif.	• Membantu memantau backlog pesan tertunda dari grup pesan dengan pola throughput normal atau yang diharapkan (sebagai lawan dari grup volume tinggi atau bising) • Berguna untuk memahami persyaratan pemrosesan di masa depan dan perencanaan kapasitas untuk beban kerja tipikal.

Metrik	Deskripsi	Unit	Perilaku pelaporan	Catatan kunci
`ApproximateAgeOfOldestMessageInQuietGroups`	Usia pesan tertua yang tidak dihapus dalam antrian tidak termasuk pesan dari grup pesan berisik.	Detik	Nilai non-negatif dilaporkan jika antrian aktif.	• Digunakan untuk memantau kepatuhan SLA dan mendeteksi hambatan pemrosesan dalam grup pesan dengan pola throughput normal atau yang diharapkan (sebagai lawan dari grup pesan volume tinggi atau berisik yang mungkin mengubah metrik). • Gunakan metrik ini untuk menyetel alarm untuk batas waktu pemrosesan pesan yang mengabaikan pesan tua buatan dari tetangga yang berisik.

[1] Metrik ini mencerminkan aktivitas tingkat sistem dan mungkin termasuk percobaan ulang, duplikat, atau pesan tertunda. Jangan gunakan jumlah mentah untuk memperkirakan status antrian real-time tanpa memperhitungkan perilaku siklus hidup pesan.

Dead-letter antrian (DLQ) dan metrik CloudWatch

Saat bekerja dengan DLQ, penting untuk memahami bagaimana metrik Amazon SQS berperilaku:

- **`NumberOfMessagesSent`**— Metrik ini berperilaku berbeda untuk DLQ:
  - Pengiriman Manual — Pesan yang dikirim secara manual ke DLQ ditangkap oleh metrik ini.
  - Automatic Redrive — Pesan secara otomatis dipindahkan ke DLQ karena kegagalan pemrosesan tidak dit angkap oleh metrik ini. Akibatnya, `NumberOfMessagesReceived` metrik `NumberOfMessagesSent` dan dapat menunjukkan perbedaan untuk DLQ.

- Metrik yang Direkomendasikan untuk DLQ — Untuk memantau keadaan DLQ, gunakan metrik. `ApproximateNumberOfMessagesVisible` Metrik ini menunjukkan jumlah pesan yang saat ini tersedia untuk diproses di DLQ.

Antrian dan CloudWatch metrik yang adil

Saat Anda menggunakan antrian yang adil, Amazon SQS memancarkan metrik tambahan berikut:

- `ApproximateNumberOfNoisyGroups`
- `ApproximateNumberOfMessagesVisibleInQuietGroups`
- `ApproximateNumberOfMessagesNotVisibleInQuietGroups`
- `ApproximateNumberOfMessagesDelayedInQuietGroups`
- `ApproximateAgeOfOldestMessageInQuietGroups`

> **Note**
>
> Setiap `QuietGroup` metrik adalah bagian dari met `Approximate` rik tingkat antrian standar yang setara, tetapi tidak termasuk pesan dari grup tetangga yang bising.

Kelompok yang berisik

Grup pesan berisik mewakili penyewa tetangga yang berisik dari antrian multi-penyewa.

Kelompok yang tenang

Grup pesan tidak termasuk grup berisik.

Mengamati perilaku antrian adil SQS

Untuk memantau efek antrian adil Amazon SQS, Anda dapat membandingkan `Approximate..InQuietGroups` metrik dengan metrik tingkat antrian standar. Selama lonjakan lalu lintas untuk penyewa tertentu, metrik tingkat antrian umum dapat mengungkapkan peningkatan backlog atau usia pesan yang lebih tua. Namun, melihat grup yang tenang secara terpisah, Anda dapat mengidentifikasi bahwa sebagian besar grup pesan atau penyewa yang tidak berisik tidak terpengaruh, dan memberikan perkiraan jumlah total grup pesan yang terkena dampak.

Meskipun metrik baru ini memberikan gambaran yang baik tentang perilaku antrian adil Amazon SQS, akan bermanfaat untuk memahami penyewa spesifik mana yang menyebabkan beban. W CloudWatch awasan kontributor Amazon memungkinkan Anda melihat metrik tentang kontributor Top-N, jumlah total kontributor unik, dan penggunaannya. Ini sangat membantu dalam skenario di mana Anda berurusan dengan ribuan penyewa yang sebaliknya akan mengarah pada data kardinalitas tinggi (dan biaya) saat memancarkan metrik tradisional.

Untuk contoh konfigurasi pemantauan untuk antrian yang adil, lihat sampel di GitHub.

Dimensi untuk metrik Amazon SQS

Metrik Amazon SQS CloudWatch menggunakan satu dimensi: **`QueueName`**. Semua data metrik dikelompokkan dan difilter berdasarkan nama antrian.

Kiat pemantauan

Pantau SQS secara efektif menggunakan metrik dan CloudWatch alarm utama untuk mendeteksi backlog antrian, mengoptimalkan kinerja, dan tetap dalam batas layanan.

- Atur CloudWatch alarm berdasarkan `ApproximateNumberOfMessagesVisible` untuk menangkap pertumbuhan backlog.
- Pantau `NumberOfEmptyReceives` untuk menyesuaikan frekuensi polling dan mengurangi biaya API.
- Gunakan `ApproximateNumberOfGroupsWithInflightMessages` dalam antrian FIFO untuk mendiagnosis batas throughput.
- Tinjau kuota SQS untuk memahami ambang metrik dan batas layanan.

Validasi kepatuhan untuk Amazon SQS

Pengalaman AWS pendaftaran baru kami tidak dirancang untuk beban kerja yang diatur. Jika Anda menggunakan pengalaman AWS pendaftaran baru kami, tetapi ingin menggunakannya AWS untuk beban kerja yang diatur, Anda dapat mendaftar untuk AWS (lanjutan) atau mengaktifkan fitur lanjutan untuk lingkungan Anda AWS .

Untuk mengetahui apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat Layanan AWS di Lingkup berdasarkan Program Kepatuhan dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat Program Kep AWS atuhan Program Jaminan .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh Laporan di AWS Artifact .

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. Untuk informasi selengkapnya tentang tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS, lihat Dokumentasi AWS Keamanan.

Ketahanan di Amazon SQS

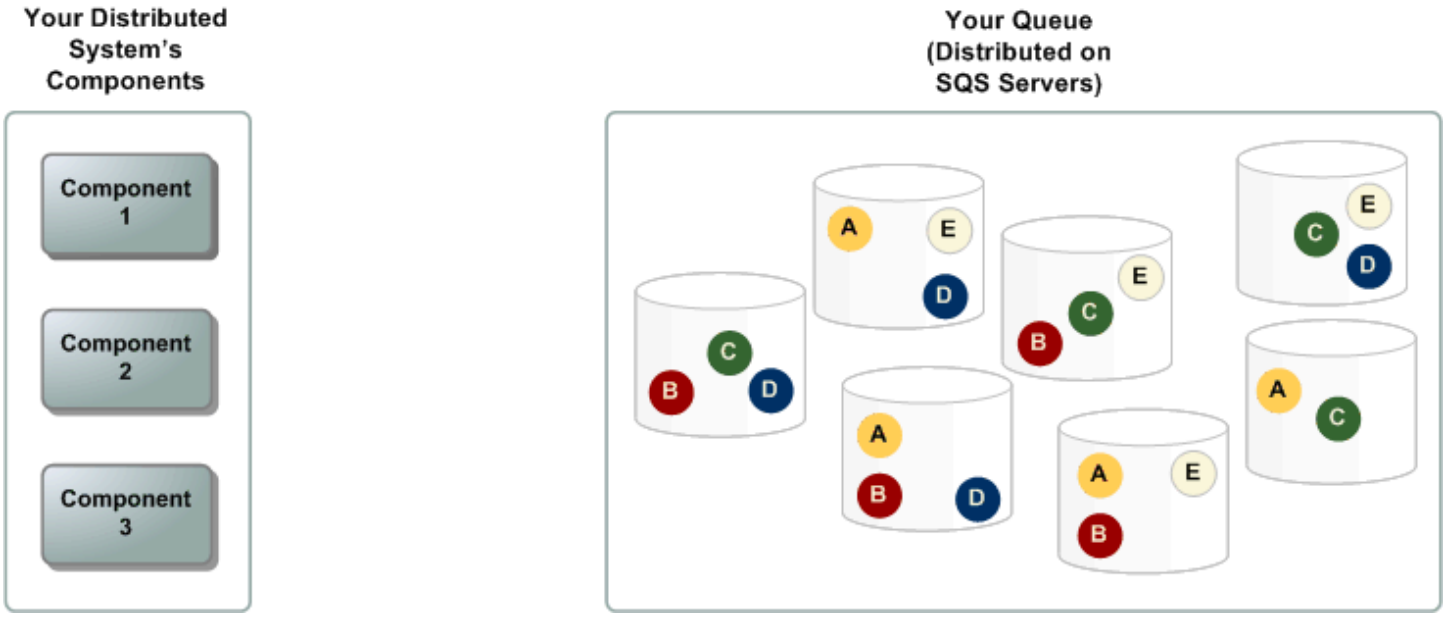
Infrastruktur AWS global dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional. Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS Global.

Selain infrastruktur AWS global, Amazon SQS menawarkan antrian terdistribusi.

Antrian terdistribusi

Ada tiga bagian utama dalam sistem pesan terdistribusi: komponen sistem terdistribusi Anda, antrian Anda (didistribusikan di server Amazon SQS), dan pesan dalam antrian.

Dalam skenario berikut, sistem Anda memiliki beberapa produsen (komponen yang mengirim pesan ke antrian) dan konsumen (komponen yang menerima pesan dari antrian). Antrian (yang menyimpan pesan A hingga E) menyimpan pesan secara berlebihan di beberapa server Amazon SQS.



Keamanan infrastruktur di Amazon SQS

Sebagai layanan terkelola, Amazon SQS dilindungi oleh prosedur keamanan jaringan AWS global yang dijelaskan dalam whitepaper Amazon Web Services: Overview of Security Processes.

Anda menggunakan tindakan API yang AWS dipublikasikan untuk mengakses Amazon SQS melalui jaringan. Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2 atau versi yang lebih baru. Selain itu, klien harus mendukung suite sandi dengan Perfect Forward Secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Diffie-Hellman Ephemeral (ECDHE).

Anda harus menandatangani permintaan menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan prinsipal IAM. Atau, Anda dapat menggunakanAWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk permintaan penandatanganan.

Anda dapat memanggil tindakan API ini dari lokasi jaringan mana pun, tetapi Amazon SQS mendukung kebijakan akses berbasis sumber daya, yang dapat mencakup pembatasan berdasarkan alamat IP sumber. Anda juga dapat menggunakan kebijakan Amazon SQS untuk mengontrol akses dari titik akhir VPC Amazon tertentu atau spesifik. VPCs Ini secara efektif mengisolasi akses jaringan ke antrian Amazon SQS tertentu dari hanya VPC tertentu dalam jaringan. AWS Lihat informasi yang lebih lengkap di Contoh 5: Tolak akses jika bukan dari titik akhir VPC.

Praktik terbaik keamanan Amazon SQS

AWS menyediakan banyak fitur keamanan untuk Amazon SQS, yang harus Anda tinjau dalam konteks kebijakan keamanan Anda sendiri. Berikut ini adalah praktik terbaik keamanan preventif untuk Amazon SQS.

> **Note**
>
> Panduan implementasi khusus yang diberikan adalah untuk kasus penggunaan umum dan implementasi. Kami menyarankan agar Anda melihat praktik terbaik ini dalam konteks kasus penggunaan, arsitektur, dan model ancaman spesifik Anda.

Pastikan antrian tidak dapat diakses publik

Kecuali Anda secara eksplisit meminta siapa pun di internet untuk dapat membaca atau menulis ke antrian Amazon SQS Anda, Anda harus memastikan bahwa antrian Anda tidak dapat diakses publik (dapat diakses oleh semua orang di dunia atau oleh pengguna yang diautentikasi). AWS

- Hindari pembuatan kebijakan dengan `Principal` diatur ke `""`.
- Hindari penggunaan wildcard (`*`). Sebagai gantinya, beri nama pengguna tertentu.

Menerapkan akses hak istimewa yang paling rendah

Saat Anda memberikan izin, Anda memutuskan siapa yang menerimanya, untuk antrian izin, dan tindakan API tertentu yang ingin Anda izinkan untuk antrian ini. Menerapkan hak istimewa paling sedikit penting untuk mengurangi risiko keamanan dan mengurangi efek kesalahan atau niat jahat.

Ikuti saran keamanan standar pemberian hak istimewa paling rendah. Artinya, hanya berikan izin yang diperlukan untuk melakukan tugas tertentu. Anda dapat menerapkan ini menggunakan kombinasi kebijakan keamanan.

Amazon SQS menggunakan model produsen-konsumen, yang membutuhkan tiga jenis akses akun pengguna:

- Administrator — Akses untuk membuat, memodifikasi, dan menghapus antrian. Administrator juga mengontrol kebijakan antrian.

- Produser — Akses untuk mengirim pesan ke antrian.
- Konsumen — Akses untuk menerima dan menghapus pesan dari antrian.

Untuk informasi selengkapnya, lihat bagian berikut:

- Identitas dan manajemen akses di Amazon SQS
- Izin API Amazon SQS: Tindakan dan referensi sumber daya
- Menggunakan kebijakan khusus dengan Bahasa Kebijakan Akses Amazon SQS

Gunakan peran IAM untuk aplikasi dan AWS layanan yang memerlukan akses Amazon SQS

Untuk aplikasi atau AWS layanan seperti Amazon EC2 untuk mengakses antrian Amazon SQS, mereka harus menggunakan kredensyal yang AWS valid dalam permintaan API mereka. AWS Karena kredensyal ini tidak diputar secara otomatis, Anda tidak boleh menyimpan AWS kredensyal secara langsung di aplikasi atau instans EC2.

Anda harus menggunakan peran IAM untuk mengelola kredensyal sementara untuk aplikasi atau layanan yang perlu mengakses Amazon SQS. Saat Anda menggunakan peran, Anda tidak perlu mendistribusikan kredensyal jangka panjang (seperti nama pengguna, kata sandi, dan kunci akses) ke instans atau AWS layanan EC2 seperti. AWS Lambda Sebagai gantinya, peran menyediakan izin sementara yang dapat digunakan aplikasi saat mereka melakukan panggilan ke AWS sumber daya lain.

Untuk informasi selengkapnya, lihat IAM Role dan Skenario Umum untuk Peran: Pengguna, Aplikasi, dan Layanan di Panduan Pengguna IAM.

Menerapkan enkripsi sisi server

Untuk mitigasi masalah kebocoran data, gunakan enkripsi saat istirahat untuk mengenkripsi pesan menggunakan kunci yang disimpan di lokasi berbeda dari lokasi yang menyimpan pesan Anda. Enkripsi sisi server (SSE) menyediakan enkripsi data saat istirahat. Amazon SQS mengenkripsi data Anda pada tingkat pesan saat menyimpannya, dan mendekripsi pesan untuk Anda saat Anda mengaksesnya. SSE menggunakan kunci yang dikelola di AWS Key Management Service. Selama Anda mengautentikasi permintaan Anda dan memiliki izin akses, tidak ada perbedaan antara mengakses antrian terenkripsi dan tidak terenkripsi.

Untuk informasi selengkapnya, lihat Enkripsi saat istirahat di Amazon SQS dan Manajemen Kunci Amazon SQS.

Menegakkan enkripsi data saat transit

Tanpa HTTPS (TLS), penyerang berbasis jaringan dapat menguping lalu lintas jaringan atau memanipulasinya, menggunakan serangan seperti. man-in-the-middle Izinkan hanya koneksi terenkripsi melalui HTTPS (TLS) menggunakan `aws:SecureTransport`kondisi dalam kebijakan antrian untuk memaksa permintaan menggunakan SSL.

Pertimbangkan untuk menggunakan titik akhir VPC untuk mengakses Amazon SQS

Jika Anda memiliki antrian yang harus dapat berinteraksi dengan Anda tetapi yang sama sekali tidak boleh diekspos ke internet, gunakan titik akhir VPC untuk mengantri akses hanya ke host dalam VPC tertentu. Anda dapat menggunakan kebijakan antrian untuk mengontrol akses ke antrian dari titik akhir VPC Amazon tertentu atau dari titik tertentu. VPCs

Titik akhir Amazon SQS VPC menyediakan dua cara untuk mengontrol akses ke pesan Anda:

- Anda dapat mengontrol permintaan, pengguna, atau grup yang diizinkan melalui titik akhir VPC tertentu.
- Anda dapat mengontrol titik akhir VPC VPCs atau mana yang memiliki akses ke antrian Anda menggunakan kebijakan antrian.

Lihat informasi yang lebih lengkap di Titik akhir Amazon Virtual Private Cloud untuk Amazon SQS dan Membuat kebijakan titik akhir Amazon VPC untuk Amazon SQS.

Sumber daya Amazon SQS terkait

Berikut ini adalah tabel yang mencantumkan daftar sumber daya terkait yang akan berguna saat Anda bekerja dengan layanan ini.

Sumber Daya	Deskripsi
Referensi API Layanan Antrian Sederhana Amazon	Deskripsi tindakan, parameter, dan tipe data dan daftar kesalahan yang dikembalikan layanan.
Amazon SQS di Referensi Perintah AWS CLI	Deskripsi AWS CLI perintah yang dapat Anda gunakan untuk bekerja dengan antrian.
Wilayah dan Titik Akhir	Informasi tentang wilayah dan titik akhir Amazon SQS
Halaman Produk	Halaman web utama untuk informasi tentang Amazon SQS.
Forum Diskusi	Forum berbasis komunitas bagi pengembang untuk mendiskusikan pertanyaan teknis yang terkait dengan Amazon SQS.
AWS Informasi Support Premium	Halaman web utama untuk informasi tentang Dukungan AWS Premium, saluran dukungan respons cepat untuk membantu Anda membangun dan menjalankan aplikasi pada layanan AWS infrastruktur. one-on-one

Riwayat dokumentasi

Tabel berikut menjelaskan perubahan penting pada Panduan Pengembang Layanan Antrian Sederhana Amazon sejak Januari 2019. Untuk pemberitahuan tentang pembaruan dokumentasi ini, berlangganan umpan RSS.

Fitur layanan terkadang diluncurkan secara bertahap ke AWS Wilayah tempat layanan tersedia. Kami memperbarui dokumentasi ini hanya untuk rilis pertama. Kami tidak memberikan informasi tentang ketersediaan Wilayah atau mengumumkan peluncuran Wilayah berikutnya. Untuk informasi tentang ketersediaan fitur layanan wilayah dan untuk berlangganan pemberitahuan tentang pembaruan, lihat Apa yang Baru dengan AWS? .

Perubahan	Deskripsi	Tanggal
Dukungan Fair Queues	Dukungan Fair Queues ditambahkan untuk antrian standar.	Juli 21, 2025
Menambahkan dukungan untuk titik akhir dual-stack (IPv4 dan IPv6)	Amazon SQS sekarang mendukung titik akhir dual-stack (IPv4 dan IPv6), memungkinkan antrian diakses melalui kedua protokol IP.	17 April 2025
CloudTrail integrasi untuk semua Amazon SQS APIs	CloudTrail integrasi ditambahkan untuk semua Amazon APIs SQS.	Januari 10, 2025
SQSUnlockQueuePolicy	Amazon SQS menambahkan kebijakan AWS terkelola baru yang dipanggil `SQSUnlockQueuePolicy` untuk membuka kunci antrian dan menghapus kebijakan antrian yang salah konfigurasi yang menolak semua prinsipal	November 15, 2024

	mengakses antrian Amazon SQS.	
AWS kms:Decrypt	Amazon SQS tidak lagi memerlukan `kms:Decrypt` izin untuk API. `SendMessage` Pelanggan sekarang hanya memerlukan `kms:GenerateDataKey` izin pada kunci KMS yang digunakan untuk mengenkripsi antrian, tetapi masih membutuhkan `kms:Decrypt` izin untuk menelepon. `ReceiveMessage`	Juli 24, 2024
Pembaruan metrik FIFO	Support untuk `NumberOfDeduplicatedSentMessages` dan `ApproximateNumberOfGroupsWithInflightMessages` ditambahkan ke metrik Amazon SQS FIFO.	Juli 3, 2024
`ListQueueTags` tindakan yang didukung dalam kebijakan SQSReadOnlyAccess terkelola Amazon	Kebijakan SQSReadOnlyAccess terkelola Amazon mendukung `ListQueueTags` untuk mengambil semua tag yang terkait dengan antrean Amazon SQS yang ditentukan.	2 Mei 2024
AWS Protokol JSON	Buat permintaan API menggunakan protokol AWS JSON.	Juli 27, 2023

Bagian baru yang menjelaskan kebijakan AWS terkelola untuk Amazon SQS dan pembaruan kebijakan ini	Amazon SQS menambahkan tindakan baru yang memungkinkan Anda mencantumkan tugas pergerakan pesan terbaru (hingga 10) di bawah antrian sumber tertentu. Tindakan ini dikaitkan dengan operasi API `ListMessageMoveTasks` .	Juni 7, 2023
Penggerak ulang antrian huruf mati menggunakan APIs	Konfigurasikan redrive antrian huruf mati menggunakan Amazon SQS. APIs	Juni 7, 2023
ABAC untuk Amazon SQS	Kontrol akses berbasis atribut (ABAC) menggunakan tag antrian untuk izin akses yang fleksibel dan dapat diskalakan.	10 November 2022
Batas throughput tinggi FIFO meningkat	Peningkatan kuota default untuk mode throughput tinggi FIFO di Wilayah komersial, ditambah optimasi dokumen throughput tinggi FIFO.	20 Oktober 2022
Enkripsi sisi server default (SSE) tersedia	Enkripsi sisi server (SSE) menggunakan enkripsi milik SQS (SSE-SQS) secara default.	26 September 2022

Dukungan perlindungan wakil kebingungan Amazon SQS tersedia	Perlindungan wakil yang bingung memungkinkan Anda menentukan header baru dalam permintaan mereka, yang diperiksa terhadap kondisi dalam kebijakan KMS saat menggunakan SSE terkelola Amazon SQS.	Desember 29, 2021
SSE terkelola tersedia	Amazon SQS managed SSE (SSE-SQS) adalah enkripsi sisi server terkelola yang menggunakan kunci enkripsi milik Amazon SQS untuk melindungi data sensitif yang dikirim melalui antrian pesan.	23 November 2021
Penggerak ulang antrian huruf mati tersedia	Amazon SQS mendukung redrive antrian huruf mati untuk antrian standar.	November 10, 2021
Throughput tinggi untuk pesan dalam antrian FIFO tersedia	Throughput tinggi untuk antrian FIFO Amazon SQS memberikan jumlah transaksi per detik (TPS) yang lebih tinggi untuk pesan dalam antrian FIFO. Untuk informasi tentang kuota throughput, lihat Kuota yang terkait dengan pesan.	27 Mei 2021

Throughput tinggi untuk pesan dalam antrian FIFO tersedia dalam rilis pratinjau	Throughput tinggi untuk antrian Amazon SQS FIFO dalam rilis pratinjau dan dapat berubah sewaktu-waktu. Fitur ini memberikan jumlah transaksi per detik (TPS) yang lebih tinggi untuk pesan dalam antrian FIFO. Untuk informasi tentang kuota throughput, lihat Kuota yang terkait dengan pesan.	17 Desember 2020
Desain konsol Amazon SQS baru	Untuk menyederhanakan alur kerja pengembangan dan produksi, konsol Amazon SQS memiliki pengalaman pengguna baru.	8 Juli 2020
Amazon SQS mendukung pagination untuk ListQueues dan listDeadLetter SourceQueues	Anda dapat menentuka n jumlah maksimum hasil yang akan dikembalikan dari ListQueues atau permintaan. listDeadLetterSourceQueues	22 Juni 2020
Amazon SQS mendukung metrik CloudWatch Amazon 1 menit di AWS semua Wilayah, kecuali AWS GovCloud Wilayah (AS)	CloudWatch Metrik satu menit untuk Amazon SQS tersedia di semua Wilayah, kecuali AWS GovCloud (US) Wilayah.	9 Januari 2020
Amazon SQS mendukung metrik 1 menit CloudWatch	CloudWatch Metrik satu menit untuk Amazon SQS saat ini hanya tersedia di Wilayah berikut: AS Timur (Ohio), Eropa (Irlandia), Eropa (Stockholm), dan Asia Pasifik (Tokyo).	25 November 2019

AWS Lambda pemicu untuk antrian Amazon SQS FIFO tersedia	Anda dapat mengonfigurasi pesan yang masuk dalam antrian FIFO sebagai pemicu fungsi Lambda.	25 November 2019
Enkripsi sisi server (SSE) untuk Amazon SQS tersedia di Wilayah Tiongkok	SSE untuk Amazon SQS tersedia di Wilayah Tiongkok.	13 November 2019
Antrian FIFO tersedia di Wilayah Timur Tengah (Bahrain)	Antrian FIFO tersedia di Wilayah Timur Tengah (Bahrain).	10 Oktober 2019
Titik akhir Amazon Virtual Private Cloud (Amazon VPC) untuk Amazon SQS tersedia di Wilayah AWS GovCloud (AS-Timur) dan (AS-Barat) AWS GovCloud	Anda dapat mengirim pesan ke antrian Amazon SQS dari Amazon VPC di AWS GovCloud Wilayah (AS-Timur) dan (AS-Barat). AWS GovCloud	Selasa, 05 September 2019
Amazon SQS memungkinkan pemecahan masalah antrian menggunakan atribut sistem pesan AWS X-Ray	Anda dapat memecahkan masalah pesan yang melewati antrian Amazon SQS menggunakan X-Ray. Rilis ini menambahkan parameter `MessageSystemAttribute` permintaan (yang memungkinkan Anda mengirim header jejak X-Ray melalui Amazon SQS) ke `SendMessage` operasi `SendMessageBatch` dan API, `AWSTraceHeader` atribut ke `ReceiveMessage` operasi API, dan tipe data`MessageSystemAttributeValue` .	28 Agustus 2019

Anda dapat menandai antrian Amazon SQS saat pembuatan	Anda dapat menggunakan satu panggilan Amazon SQS API, fungsi AWS SDK, atau perintah AWS Command Line Interface (AWS CLI) untuk secara bersamaan membuat antrian dan menentukan tag-nya. Selain itu, Amazon SQS mendukung kunci `aws:TagKeys` dan `aws:RequestTag` AWS Identity and Access Management (IAM).	22 Agustus 2019
Klien antrian sementara untuk Amazon SQS sekarang tersedia	Antrian sementara membantu Anda menghemat waktu pengembangan dan biaya penerapan saat menggunakan pola pesan umum seperti respon permintaan. Anda dapat menggunakan Klien Antrian Sementara untuk membuat antrian sementara yang dikelola aplikasi dengan throughput tinggi, hemat biaya, dan dikelola.	25 Juli 2019
SSE untuk Amazon SQS tersedia di Wilayah (AWS GovCloud AS-Timur)	Enkripsi sisi server (SSE) untuk Amazon SQS tersedia di Wilayah (AS-Timur). AWS GovCloud	20 Juni 2019
Antrian FIFO tersedia di wilayah Asia Pasifik (Hong Kong), Tiongkok (Beijing) , (AS-Timur), dan AWS GovCloud (AS-Barat) AWS GovCloud	Antrian FIFO tersedia di Asia Pasifik (Hong Kong), Tiongkok (Beijing), (AS-Timur), dan AWS GovCloud AWS GovCloud (AS-Barat).	15 Mei 2019

Kebijakan titik akhir Amazon VPC tersedia untuk Amazon SQS	Anda dapat membuat kebijakan titik akhir Amazon VPC untuk Amazon SQS.	4 April 2019
Antrian FIFO tersedia di Wilayah Eropa (Stockholm) dan Tiongkok (Ningxia)	Antrian FIFO tersedia di Wilayah Eropa (Stockholm) dan Tiongkok (Ningxia).	14 Maret 2019
Antrian FIFO tersedia di semua Wilayah di mana Amazon SQS tersedia	Antrian FIFO tersedia di AS Timur (Ohio), AS Timur (Virginia N.), AS Barat (California N.), AS Barat (Oregon), Asia Pasifik (Mumbai), Asia Pasifik (Seoul), Asia Pasifik (Singapura), Asia Pasifik (Sydney), Asia Pasifik (Tokyo), Kanada (Tengah), Eropa (Frankfurt), Eropa (Irlandia), Eropa (Irlandia), Eropa (Wilayah London), Eropa (Paris), dan Amerika Selatan (São Paulo).	7 Februari 2019

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.