Guía para desarrolladores

Amazon Simple Queue Service

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

Amazon Simple Queue Service: Guía para desarrolladores

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

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

¿Qué es Amazon SQS? ... 1
Beneficios de utilizar Amazon SQS ... 1
Arquitectura básica ... 1
Colas distribuidas ... 2
Ciclo de vida del mensaje ... 2
Diferencias entre Amazon SQS, Amazon MQ y Amazon SNS ... 4
Introducción ... 6
Configuración ... 6
Paso 1: Cree un usuario Cuenta de AWS de IAM ... 6
Paso 2: Conceder acceso mediante programación ... 8
Paso 3: prepararse para usar el código de ejemplo ... 10
Pasos a seguir a continuación ... 11
Descripción de la consola de Amazon SQS ... 11
Tipo de colas ... 12
Implementación de sistemas de solicitud-respuesta ... 16
Creación de una cola estándar ... 16
Creación de una cola ... 16
Enviar un mensaje mediante una cola estándar ... 18
Creación de una cola FIFO ... 19
Creación de una cola ... 19
Enviar un mensaje mediante una cola FIFO ... 22
Tareas comunes ... 23
Administración de una cola ... 24
Edición de una cola ... 24
Recepción y eliminación de mensajes ... 25
Confirmación de que una cola está vacía ... 26
Eliminación de una cola ... 27
Depuración de una cola ... 28
Colas estándar ... 30
Entrega de Amazon SQS at-least-once ... 31
Identificadores de colas y mensajes ... 31
Identificadores de colas estándar ... 31
Colas FIFO ... 33
Términos clave de cola FIFO ... 34

Lógica de entrega FIFO ... 36
Envío de mensajes ... 36
Recepción de mensajes ... 37
Varios reintentos ... 37
Notas adicionales sobre el comportamiento de la FIFO ... 38
Ejemplos para una mejor comprensión ... 39
Procesamiento único ... 39
Cambio de una cola estándar a una cola FIFO ... 40
Comportamiento de la cola FIFO y de la simultaneidad de Lambda ... 41
Agrupación de mensajes de cola FIFO ... 42
Simultaneidad de Lambda con colas FIFO ... 42
Ejemplo de casos de uso ... 43
Alto rendimiento para las colas FIFO ... 43
Casos de uso ... 44
Particiones y distribución de datos ... 44
Habilitación del alto rendimiento para las colas FIFO ... 47
Identificadores de colas y mensajes ... 48
Identificadores de colas FIFO ... 31
Identificadores adicionales para las colas FIFO ... 50
Cuotas ... 51
Cuotas de colas FIFO ... 51
Cuotas de Amazon SQS ... 51
Cuotas de cola estándar ... 53
Cuotas de mensajes ... 55
Cuotas de políticas ... 61
Características y funciones básicas ... 62
Colas de mensajes fallidos ... 62
Uso de políticas para colas de mensajes fallidos ... 63
Descripción de los períodos de retención de mensajes en la cola de mensajes fallidos ... 63
Configuración de una cola de mensajes fallidos ... 64
Configuración de un redireccionamiento de cola de mensajes fallidos ... 64
CloudTrail requisitos de actualización y permiso ... 75
Creación de alarmas para colas de cartas sin salida con Amazon CloudWatch ... 79
Metadatos de mensajes para Amazon SQS ... 79
Atributos de mensajes ... 80
Atributos del sistema de mensajes ... 84

Recursos necesarios para procesar mensajes ... 84
Paginación de colas de listas ... 85
Etiquetas de asignación de costos ... 86
Sondeos cortos y largos ... 87
Consumo de mensajes mediante sondeo corto ... 87
Consumo de mensajes mediante sondeo largo ... 88
Diferencias entre el sondeo corto y el sondeo largo ... 89
Tiempo de espera de visibilidad ... 89
Casos de uso del tiempo de espera de visibilidad ... 90
Establecer y ajustar el tiempo de espera de visibilidad ... 90
Mensajes y cuotas en vuelo ... 91
Descripción del tiempo de espera de visibilidad en colas estándar y FIFO ... 91
Manejar los fallos ... 92
Cambio y finalización del tiempo de espera de visibilidad ... 92
Prácticas recomendadas ... 93
Colas justas ... 93
¿Qué diferencia hay con las colas FIFO ... 95
Utilizar colas justas ... 96
Métricas de colas CloudWatch justas ... 96
Colas con retraso ... 97
Colas temporales ... 98
Colas virtuales ... 99
Patrón de mensajes de respuesta a solicitudes (colas virtuales) ... 100
Situación de ejemplo: procesamiento de una solicitud de inicio de sesión ... 101
Limpieza de colas ... 103
Temporizadores de mensajes ... 104
Acceder a EventBridge las tuberías ... 104
Administración de mensajes grandes ... 106
Uso de la biblioteca de clientes ampliada para Java ... 106
Uso de la biblioteca de clientes ampliada para Python ... 113
Configuración de Amazon SQS ... 117
ABAC para Amazon SQS ... 117
¿Qué es ABAC? ... 117
¿Por qué debo utilizar ABAC en Amazon SQS? ... 118
Etiquetado para el control de acceso ... 119
Creación de usuarios de IAM y colas de Amazon SQS ... 119

Prueba del control de acceso basado en atributos ... 123
Configuración de los parámetros de la cola ... 124
Configuración de una política de acceso ... 126
Configuración de SSE-SQS para una cola ... 127
Configuración de SSE-KMS para una cola ... 128
Configuración de etiquetas para una cola ... 130
Suscripción de una cola a un tema ... 130
Suscripciones multicuenta ... 131
Suscripciones entre regiones ... 132
Configurar un desencadenante de Lambda ... 133
Requisitos previos ... 133
Automatizar las notificaciones mediante EventBridge ... 135
Atributos de mensajes ... 135
Prácticas recomendadas ... 137
Gestión de errores y mensajes problemáticos ... 137
Gestión de errores de solicitudes en Amazon SQS ... 137
Captura de mensajes problemáticos en Amazon SQS ... 138
Configuración de la retención de la cola de mensajes fallidos en Amazon SQS ... 138
Desduplicación y agrupación de mensajes ... 139
Cómo evitar el procesamiento incoherente de los mensajes en Amazon SQS ... 139
Uso del ID de desduplicación de mensajes ... 139
Uso del ID de grupo de mensajes ... 142
Uso del ID de intento de solicitud de recepción ... 144
Procesamiento y temporización de mensajes ... 144
Procesamiento de los mensajes a tiempo en Amazon SQS ... 145
Configuración del sondeo largo en Amazon SQS ... 146
Uso del modo de sondeo apropiado en Amazon SQS ... 147
Ejemplos de SDK de Java ... 148
Uso del cifrado del servidor ... 148
Agregar SSE a una cola existente ... 148
Desactivación de SSE para una cola ... 150
Creación de una cola con SSE ... 150
Recuperación de atributos de SSE ... 151
Configuraciones de etiquetas ... 151
Enumeración de etiquetas ... 151
Adición o actualización de etiquetas ... 152

Eliminación de etiquetas ... 152
Envío de atributos de mensaje ... 153
Definición de atributos ... 153
Envío de un mensaje con atributos ... 155
Usando APIs ... 156
Realizar solicitudes de API de consulta mediante el protocolo AWS JSON ... 157
Construcción de un punto de enlace ... 157
Realizar una solicitud POST ... 158
Interpretación de las respuestas de la API JSON de Amazon SQS ... 159
Protocolo Amazon SQS JSON AWS FAQs ... 160
Realizar solicitudes a la API de consulta mediante el protocolo de AWS consulta ... 164
Construcción de un punto de enlace ... 164
Realizar una solicitud GET ... 165
Realizar una solicitud POST ... 158
Interpretación de las respuestas de la API XML de Amazon SQS ... 166
Autenticación de solicitudes ... 168
Proceso básico de autenticación con HMAC-SHA ... 168
Parte 1: la solicitud del usuario ... 170
Parte 2: La respuesta de AWS ... 171
Acciones de procesamiento por lotes ... 172
Procesamiento por lotes de las acciones de los mensajes ... 173
Habilitación del almacenamiento en búfer del cliente y del agrupamiento en lotes de solicitudes con Amazon SQS ... 174
Aumento del rendimiento mediante el escalado horizontal y agrupación en lotes de acciones con Amazon SQS ... 187
Trabajando con AWS SDKs ... 199
Uso de JMS ... 201
Requisitos previos ... 201
Uso de la Biblioteca de mensajes Java ... 202
Creación de una conexión JMS ... 203
Creación de una cola de Amazon SQS ... 204
Envío de mensajes de forma sincrónica ... 205
Recepción de mensajes de forma sincrónica ... 206
Recepción de mensajes de forma asincrónica ... 208
Uso del modo de reconocimiento del cliente ... 209
Uso del modo de reconocimiento sin orden ... 210

Uso del cliente de JMS con otros clientes de Amazon SQS ... 211
Ejemplos funcionales en Java del uso de JMS con colas estándar ... 212
ExampleConfiguration.java ... 212
TextMessageSender.java ... 215
SyncMessageReceiver.java ... 217
AsyncMessageReceiver.java ... 219
SyncMessageReceiverClientAcknowledge.java ... 221
SyncMessageReceiverUnorderedAcknowledge.java ... 224
SpringExampleConfiguration.xml ... 228
SpringExample.java ... 229
ExampleCommon.java ... 232
Implementaciones de JMS 1.1 admitidas ... 233
Interfaces comunes admitidas ... 233
Tipos de mensajes admitidos ... 234
Modos de confirmación de mensajes admitidos ... 234
Propiedades reservadas y encabezados definidos por JMS ... 234
Tutoriales ... 236
Creación de una cola de Amazon SQS mediante AWS CloudFormation ... 236
Envío de mensajes desde una VPC ... 238
Paso 1: Crear un EC2 key pair de Amazon ... 239
Paso 2: Crea recursos AWS ... 239
Paso 3: Confirma que tu EC2 instancia no es de acceso público ... 240
Paso 4: Crear un punto de conexión de VPC para Amazon SQS ... 242
Paso 5: enviar un mensaje a la cola de Amazon SQS ... 243
Ejemplos de código ... 245
Conceptos básicos ... 259
Introducción a Amazon SQS ... 260
Acciones ... 272
Escenarios ... 437
Creación de una aplicación de mensajería ... 438
Creación de una aplicación de mensajería ... 439
Creación de una aplicación de exploración de Amazon Textract ... 440
Creación y publicación en un tema FIFO ... 442
Detección de personas y objetos en un video ... 454
Gestione mensajes de gran tamaño con S3 ... 455
Procese notificaciones de eventos de S3 ... 459

Publicación de mensajes en colas ... 463
Enviar y recibir lotes de mensajes ... 578
Utilice el marco de procesamiento de AWS mensajes para.NET con Amazon SQS ... 610
Utilice la biblioteca de mensajería Java de Amazon SQS para trabajar con la interfaz JMS . 610
Trabaje con etiquetas de cola ... 633
Ejemplos de tecnología sin servidor ... 636
Invocar una función de Lambda desde un desencadenador de Amazon SQS ... 636
Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS. ... 645
Solución de problemas ... 656
Error de acceso denegado ... 656
Política de colas y política de IAM de Amazon SQS ... 657
AWS Key Management Service (AWS KMS) permisos ... 658
Política de punto de conexión de VPC ... 659
Política de control de servicios de la organización ... 660
errores de API ... 660
QueueDoesNotExist error ... 660
InvalidAttributeValue error ... 661
ReceiptHandle error ... 661
Problemas de redireccionamiento con DLQ y DLQ ... 662
Problemas de DLQ ... 662
Problemas de redireccionamiento de DLQ ... 664
Problemas de limitación de FIFO ... 666
No se devuelven los mensajes de una llamada a la API ReceiveMessage ... 667
Cola vacía ... 667
Se ha superado el límite de tránsito ... 667
Retraso de mensajes ... 667
El mensaje está en tránsito ... 667
Método de sondeo ... 668
Errores de red ... 668
ETIMEOUT error ... 668
UnknownHostException error ... 669
Solución de problemas de colas mediante X-Ray ... 670
Seguridad ... 672
Protección de los datos ... 672
Cifrado de datos ... 673

Privacidad del tráfico entre redes ... 686
Uso de puntos de conexión de doble pila para la conectividad ... 688
Identity and Access Management ... 689
Público ... 689
Autenticación con identidades ... 689
Administración de acceso mediante políticas ... 693
Descripción general ... 696
Cómo funciona Amazon Simple Queue Service con IAM ... 703
AWS políticas gestionadas ... 711
Solución de problemas ... 713
Uso de políticas de ... 716
Registro y supervisión ... 765
Registro de llamadas a la API ... 768
Supervisión de colas ... 772
Validación de conformidad ... 795
Resiliencia ... 797
Colas distribuidas ... 797
Seguridad de la infraestructura ... 798
Prácticas recomendadas ... 798
Comprobación de que las colas no sean accesibles de forma pública ... 799
Implementación del acceso a los privilegios mínimos ... 799
Utilice funciones de IAM para aplicaciones y AWS servicios que requieren acceso a Amazon SQS ... 800
Implementación del cifrado en el servidor ... 800
Aplicación del cifrado de los datos en tránsito ... 800
Consideración del uso de puntos de conexión de VPC para obtener acceso a Amazon SQS ... 801
Recursos relacionados ... 802
Historial de documentación ... 803
... dcccxii

What is Amazon Simple Queue Service?

Con Amazon Simple Queue Service (Amazon SQS), se ofrece una cola alojada segura, duradera y disponible que le permite integrar y desacoplar sistemas y componentes de software distribuidos. Amazon SQS ofrece constructos comunes, como colas de mensajes fallidos y etiquetas de asignación de costos. Proporciona una API de servicios web genérica a la que puede acceder mediante cualquier lenguaje de programación compatible con el AWS SDK.

Beneficios de utilizar Amazon SQS

- Seguridad: usted controla quién puede enviar mensajes a la cola de Amazon SQS o recibirlos de ella. Puede elegir entre transmitir datos confidenciales mediante la protección del contenido de los mensajes en las colas por medio del cifrado del servidor (SSE) administrado por Amazon SQS de forma predeterminada o mediante el uso de claves SSE personalizadas administradas en AWS Key Management Service (AWS KMS).
- Durabilidad: para mantener seguros sus mensajes, Amazon SQS los almacena en varios servidores. Las colas estándar admiten la entrega de at-least-once mensajes y las colas FIFO admiten el procesamiento de mensajes exactamente una vez y el modo de alto rendimiento.
- Disponibilidad: Amazon SQS utiliza una infraestructura redundante para proporcionar acceso con un alto grado de simultaneidad a los mensajes y alta disponibilidad para producir y consumir mensajes.
- Escalabilidad: Amazon SQS puede procesar cada solicitud en búfer independientemente, con un escalado transparente para controlar cualquier aumento o pico de carga sin instrucciones de aprovisionamiento.
- Fiabilidad: Amazon SQS bloquea sus mensajes durante el procesamiento, de forma que varios productores puedan enviar mensajes y varios consumidores puedan recibirlos al mismo tiempo.
- Personalización: sus colas no tienen por qué ser exactamente iguales; por ejemplo, puede establecer un retraso predeterminado en una cola. Puede almacenar el contenido de los mensajes de más de 1 MiB mediante Amazon Simple Storage Service (Amazon S3) o Amazon DynamoDB, manteniendo Amazon SQS pulsando el puntero sobre el objeto de Amazon S3, o puede dividir un mensaje grande en mensajes más pequeños.

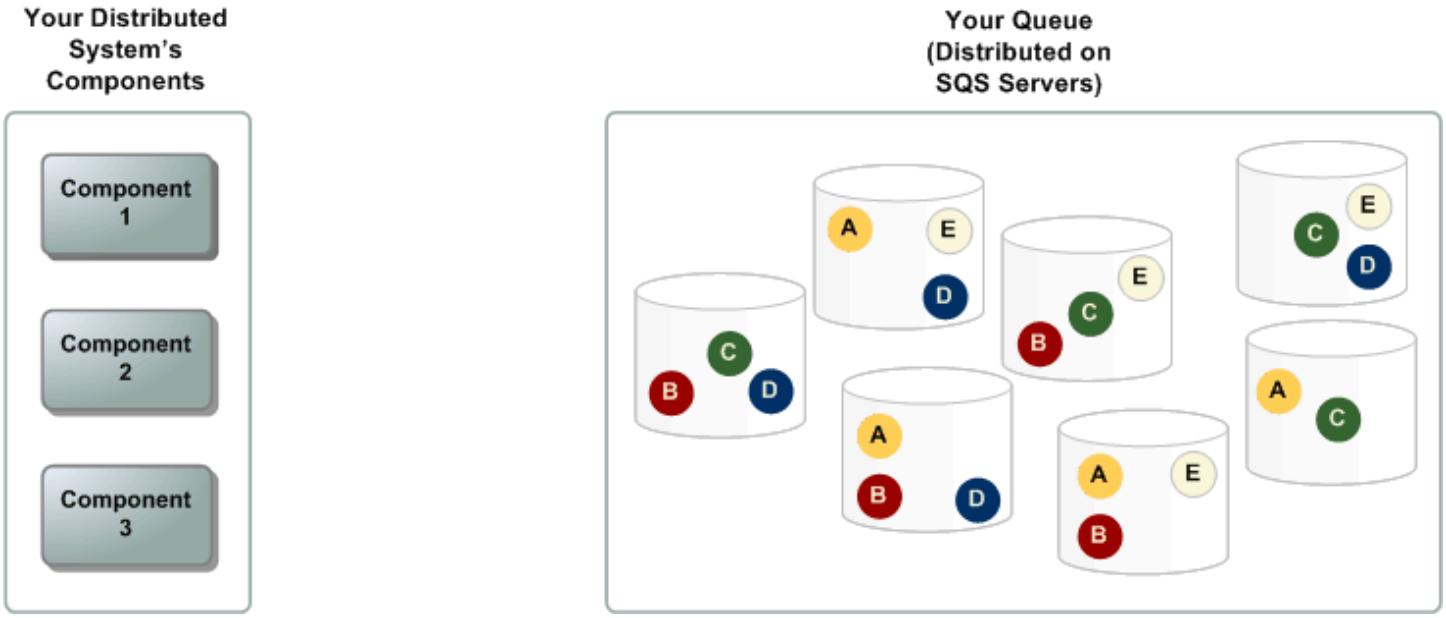
Arquitectura de Amazon SQS básica

En esta sección se describen los componentes de un sistema de mensajería distribuida y se explica el ciclo de vida de un mensaje de Amazon SQS.

Colas distribuidas

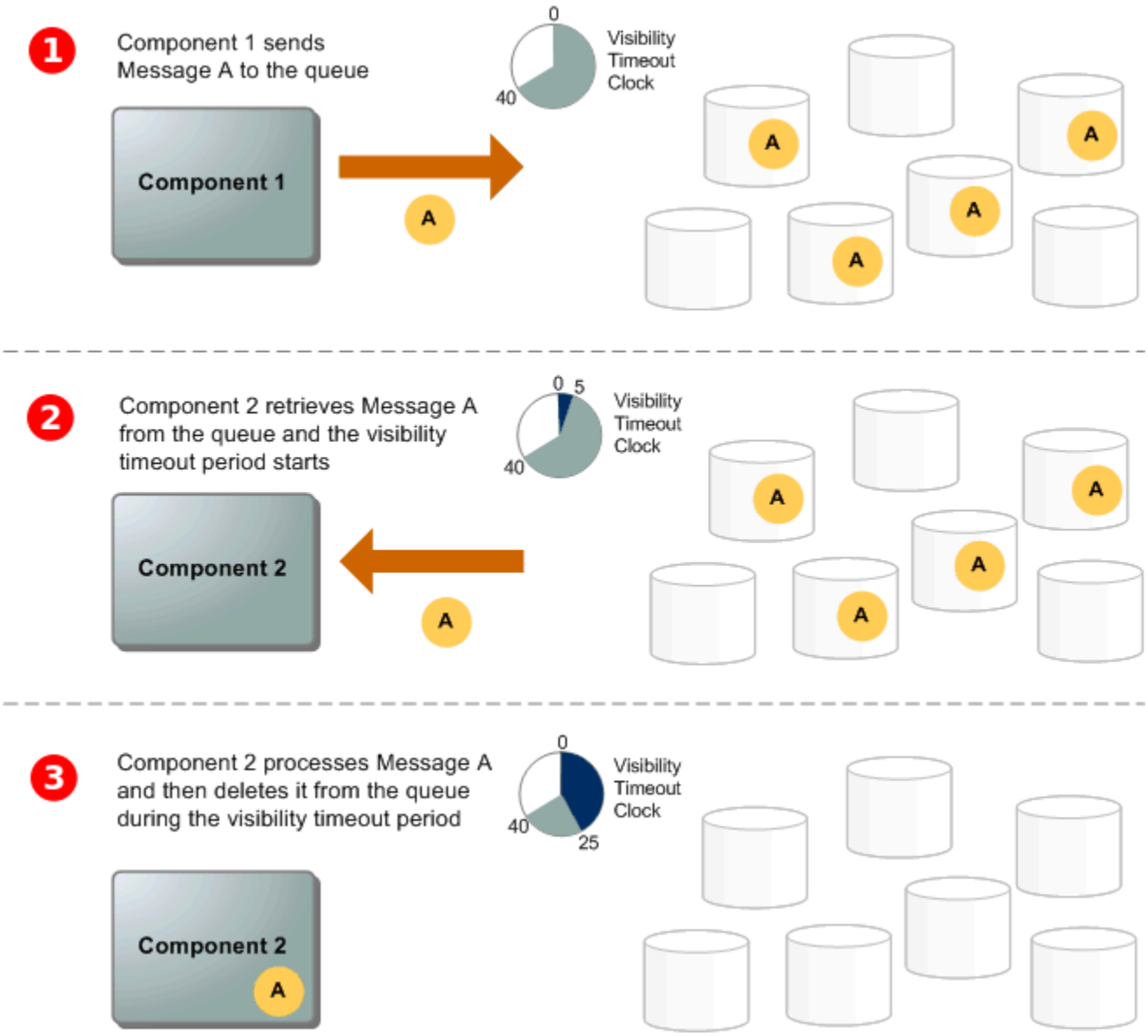
Un sistema de mensajería distribuida consta de tres partes principales: los componentes del sistema distribuido, la cola (distribuida en los servidores Amazon SQS) y los mensajes de la cola.

En el siguiente escenario, el sistema tiene varios productores (componentes que envían mensajes a la cola) y consumidores (componentes que reciben mensajes de la cola). La cola (que contiene los mensajes A a E) almacena de forma redundante los mensajes en varios servidores de Amazon SQS.



Ciclo de vida del mensaje

En el escenario siguiente se describe el ciclo de vida de un mensaje de Amazon SQS en una cola, desde la creación hasta la eliminación.



1 Un productor (componente 1) envía el mensaje A a una cola y el mensaje se distribuye por los servidores de Amazon SQS de forma redundante.

2 Cuando un consumidor (componente 2) está preparado para procesar los mensajes, consume los mensajes de la cola y devuelve el mensaje A. Mientras se procesa, el mensaje A permanece en la cola y no se devuelve a las solicitudes de recepción posteriores durante el tiempo de espera de visibilidad.

3 El consumidor (componente 2) elimina el mensaje A de la cola para evitar que se reciba y procese de nuevo cuando caduque el tiempo de espera de visibilidad.

> **Note**
>
> Amazon SQS elimina automáticamente los mensajes que han estado en una cola durante más tiempo que el periodo máximo de retención de mensajes. El periodo de retención de mensajes predeterminado es de 4 días. Sin embargo, puede establecer el periodo de retención de un mensaje en un valor comprendido entre 60 y 1.209.600 segundos (14 días) mediante la acción `SetQueueAttributes` .

Diferencias entre Amazon SQS, Amazon MQ y Amazon SNS

Amazon SQS, Amazon SNS y Amazon MQ ofrecen servicios de mensajería gestionados easy-to-use y altamente escalables, cada uno diseñado para funciones específicas dentro de los sistemas distribuidos. A continuación, encontrará información general sobre las diferencias entre estos servicios:

Amazon SQS desacopla y escala los sistemas y componentes de software distribuidos como un servicio de colas. Por lo general, procesa los mensajes a través de un único suscriptor, lo que resulta ideal para flujos de trabajo en los que la prevención de orden y pérdidas es fundamental. Para una distribución más amplia, la integración de Amazon SQS con Amazon SNS permite un patrón de mensajería de distribución ramificada que envía mensajes de manera efectiva a varios suscriptores a la vez.

Amazon SNS permite a los publicadores enviar mensajes a varios suscriptores a través de temas, que sirven como canales de comunicación. Los suscriptores reciben mensajes publicados mediante un tipo de punto de conexión compatible, como Amazon Data Firehose, Amazon SQS, Lambda, HTTP, correo electrónico, notificaciones push para dispositivos móviles y mensajes de texto a través del móvil (SMS). Este servicio es ideal para situaciones que requieren notificaciones inmediatas, como la participación de los usuarios en tiempo real o los sistemas de alarma. Para evitar la pérdida de mensajes cuando los suscriptores están desconectados, la integración de Amazon SNS con los mensajes en cola de Amazon SQS garantiza una entrega uniforme.

Amazon MQ es adecuado para las empresas que desean migrar desde agentes de mensajes tradicionales, ya que admite protocolos de mensajería estándar como AMQP y MQTT, junto con

Apache ActiveMQ y RabbitMQ. Ofrece compatibilidad con los sistemas heredados que requieren una mensajería estable y fiable sin necesidad de una reconfiguración significativa.

En el siguiente gráfico se ofrece información general sobre los tipos de recursos de cada servicio:

Tipo de recurso	Amazon SNS	Amazon SQS	Amazon MQ
Síncrono	No	No	Sí
Asíncrono	Sí	Sí	Sí
Colas	No	Sí	Sí
Mensajería de publicador y suscriptor	Sí	No	Sí
Agentes de mensajes	No	No	Sí

Se recomiendan Amazon SQS y Amazon SNS para aplicaciones nuevas que puedan beneficiarse de una escalabilidad prácticamente ilimitada y sencilla. APIs Por lo general, con sus precios, ofrecen soluciones más rentables para aplicaciones de gran volumen. pay-as-you-go Recomendamos Amazon MQ para migrar aplicaciones de los intermediarios de mensajes existentes que dependen de la compatibilidad con, como JMS, o protocolos APIs como el Protocolo avanzado de colas de mensajes (AMQP), el MQTT y el Protocolo simple de mensajes orientados al texto (OpenWireSTOMP).

Introducción a Amazon SQS

Este tema le guía sobre el uso de la consola Amazon SQS para crear y gestionar colas estándar y colas FIFO. Aprenderá a navegar por la consola, a ver los atributos de las colas y a distinguir entre los tipos de colas. Las tareas clave incluyen enviar, recibir y configurar mensajes, ajustar parámetros como el tiempo de espera de visibilidad y la retención de mensajes, y administrar el acceso a las colas mediante políticas.

Temas

- Configuración de Amazon SQS
- Descripción de la consola de Amazon SQS
- Tipos de colas de Amazon SQS
- Creación de una cola estándar de Amazon SQS y envío de un mensaje
- Creación de una cola FIFO de Amazon SQS y envío de un mensaje
- Tareas habituales para empezar a usar Amazon SQS

Configuración de Amazon SQS

Antes de poder utilizar Amazon SQS por primera vez, debe completar los siguientes pasos:

Paso 1: Cree un usuario Cuenta de AWS de IAM

Para acceder a cualquier AWS servicio, primero tienes que crear una Cuenta de AWScuenta de Amazon.com que pueda usar AWS productos. Puedes utilizarla Cuenta de AWS para ver tus informes de actividad y uso y para gestionar la autenticación y el acceso.

Para evitar utilizar el usuario Cuenta de AWS raíz para las acciones de Amazon SQS, se recomienda crear un usuario de IAM para cada persona que necesite acceso administrativo a Amazon SQS.

Inscríbase en una Cuenta de AWS

Si no tiene uno Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirte a una Cuenta de AWS

1. Abrir https://portal.aws.amazon.com/billing/registro.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

Cuando te registras en un Cuenta de AWS, Usuario raíz de la cuenta de AWSse crea un. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar tareas que requieren acceso de usuario raíz.

AWS te envía un correo electrónico de confirmación una vez finalizado el proceso de registro. En cualquier momento, puede ver la actividad de su cuenta actual y administrarla accediendo a https://aws.amazon.com/y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de crear un usuario administrativo Cuenta de AWS, asegúrelo Usuario raíz de la cuenta de AWS AWS IAM Identity Center, habilite y cree un usuario administrativo para no usar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión AWS Management Consolecomo propietario de la cuenta seleccionando el usuario root e introduciendo su dirección de Cuenta de AWS correo electrónico. En la siguiente página, escriba su contraseña.

   Para obtener ayuda para iniciar sesión con el usuario raíz, consulte Iniciar sesión como usuario raíz en la Guía del usuario de AWS Sign-In .
2. Active la autenticación multifactor (MFA) para el usuario raíz.

   Para obtener instrucciones, consulte Habilitar un dispositivo MFA virtual para el usuario Cuenta de AWS raíz (consola) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

   Consulte las instrucciones en Activar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center .
2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre su uso Directorio de IAM Identity Center como fuente de identidad, consulte Configurar el acceso de los usuarios con la configuración predeterminada Directorio de IAM Identity Center en la Guía del AWS IAM Identity Center usuario.

Inicio de sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, use la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

  Para obtener ayuda para iniciar sesión con un usuario del Centro de identidades de IAM, consulte Iniciar sesión en el portal de AWS acceso en la Guía del AWS Sign-In usuario.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

   Para conocer las instrucciones, consulte Create a permission set en la Guía del usuario de AWS IAM Identity Center .

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

   Para conocer las instrucciones, consulte Add groups en la Guía del usuario de AWS IAM Identity Center .

Paso 2: Conceder acceso mediante programación

Para utilizar las acciones de Amazon SQS (por ejemplo, mediante Java o a través de AWS Command Line Interface), necesita un identificador de clave de acceso y una clave de acceso secreta.

> **Note**
>
> El identificador de la clave de acceso y la clave de acceso secreta son específicos de AWS Identity and Access Management. No las confundas con las credenciales de otros AWS servicios, como los pares de EC2 claves de Amazon.

Los usuarios necesitan acceso programático si quieren interactuar con personas AWS ajenas a. AWS Management Console La forma de conceder el acceso programático depende del tipo de usuario que acceda. AWS

Para conceder acceso programático a los usuarios, elija una de las siguientes opciones.

¿Qué usuario necesita acceso programático?	Para	Mediante
Identidad del personal (Usuarios administrados en el IAM Identity Center)	Usa credenciales temporales para firmar las solicitudes programáticas dirigidas al AWS CLI AWS SDKs, o. AWS APIs	Siga las instrucciones de la interfaz que desea utilizar: • Para ello AWS CLI, consulte Configuración del AWS CLI uso AWS IAM Identity Center en la Guía del AWS Command Line Interface usuario. • Para AWS SDKs ver las herramientas y AWS APIs, consulte la autenticación del Centro de Identidad de IAM en la Guía de referencia de herramientas AWS SDKs y herramientas.
IAM	Utilice credenciales temporales para firmar las solicitudes programáticas dirigidas al AWS CLI AWS SDKs, o. AWS APIs	Siga las instrucciones de Uso de credenciales temporales con AWS recursos de la Guía del usuario de IAM.
IAM	(No recomendado) Utilice credenciales de larga duración para firmar las solicitudes programáticas dirigidas al AWS CLI AWS SDKs, o. AWS APIs	Siga las instrucciones de la interfaz que desea utilizar: • Para ello AWS CLI, consulte Autenticación con credenciales de usuario de IAM en la

¿Qué usuario necesita acceso programático?	Para	Mediante
		Guía del AWS Command Line Interface usuario. • Para obtener AWS SDKs información sobre las herramientas, consulte Autenticarse con credencia les de larga duración en la Guía de referencia de herramientas AWS SDKs y herramientas. • Para ello AWS APIs, consulte Administrar las claves de acceso para los usuarios de IAM en la Guía del usuario de IAM.

Paso 3: prepararse para usar el código de ejemplo

Esta guía incluye ejemplos que utilizan el AWS SDK for Java. Para ejecutar el código de ejemplo, siga las instrucciones de configuración de Introducción a AWS SDK para Java 2.0.

Puede desarrollar AWS aplicaciones en otros lenguajes de programación, como GoJavaScript, Python y Ruby. Para obtener más información, consulte Herramientas sobre las que construir AWS.

> **Note**
>
> Puede explorar Amazon SQS sin necesidad de escribir código con herramientas como AWS Command Line Interface (AWS CLI) o Windows. PowerShell Puede encontrar AWS CLI ejemplos en la sección Amazon SQS de la Referencia de AWS CLI comandos. Puede encontrar PowerShell ejemplos de Windows en la sección Amazon Simple Queue Service de la referencia de Herramientas de AWS para PowerShell cmdlets.

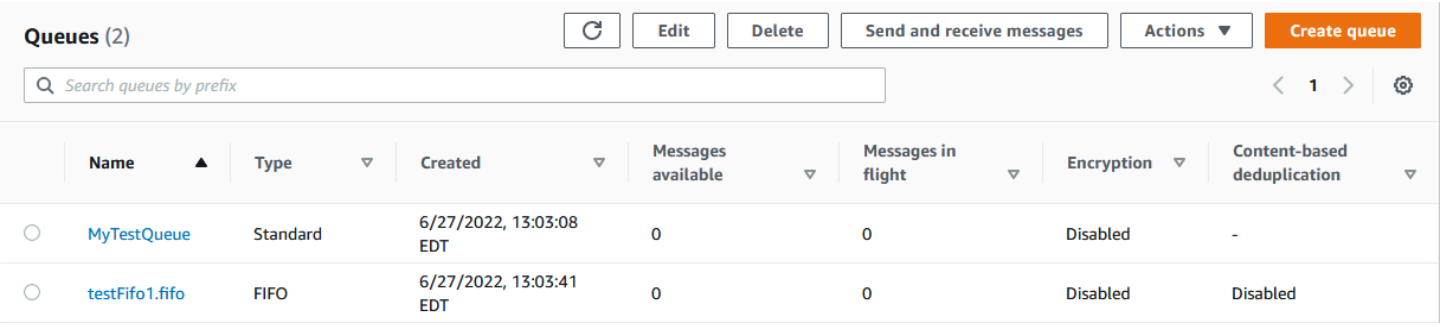
Pasos a seguir a continuación

Ya está preparado para empezar a administrar las colas y los mensajes de Amazon SQS mediante la AWS Management Console.

Descripción de la consola de Amazon SQS

Cuando abra la consola de Amazon SQS, seleccione Colas en el panel de navegación. La página Colas proporciona información sobre todas las colas de la región activa.

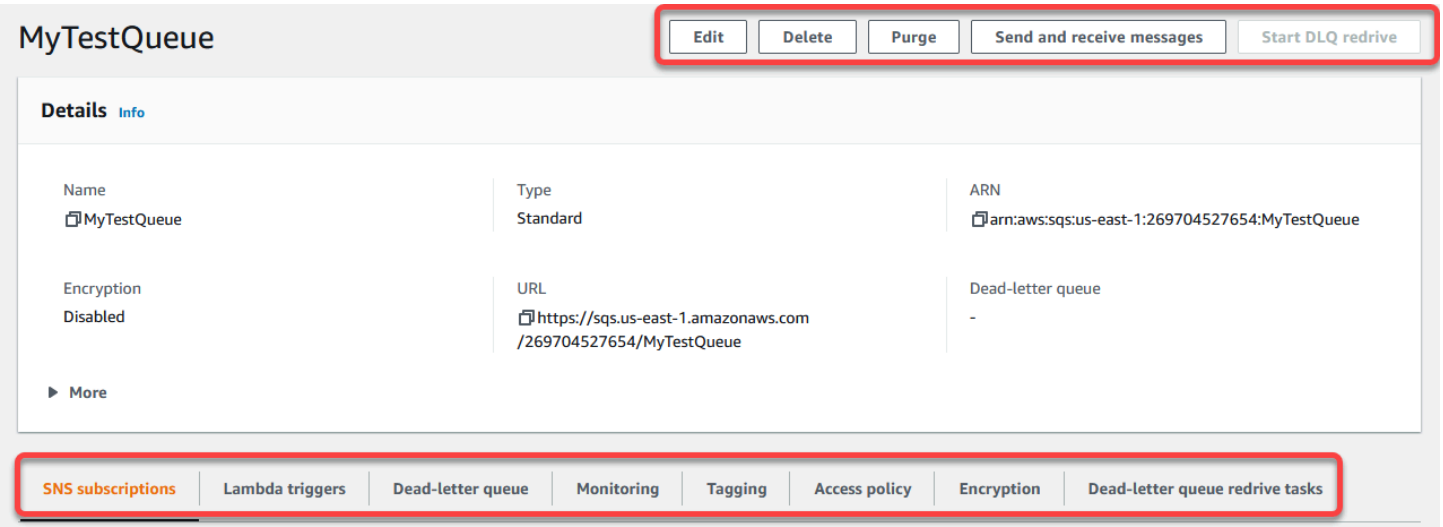
Cada entrada de cola proporciona información esencial sobre la cola, incluidos su tipo y sus atributos clave. Las colas estándar, optimizadas para obtener el máximo rendimiento y ordenar los mensajes de la mejor manera, se diferencian de las colas First-In-First-Out (FIFO), que priorizan el orden de los mensajes y la unicidad de los mensajes para las aplicaciones que requieren una secuenciación estricta de los mensajes.



Elementos y acciones interactivos

En la página Colas, tiene varias opciones para administrar sus colas:

1. Acciones rápidas: junto al nombre de cada cola, hay un menú desplegable que ofrece un acceso rápido a las acciones más habituales, como enviar mensajes, ver o eliminar mensajes, configurar los desencadenadores y eliminar la propia cola.
2. Vista detallada y configuración: al hacer clic en el nombre de una cola, se abre su página de detalles, donde puede profundizar en los ajustes y configuraciones de la cola. Aquí puede ajustar parámetros como el período de retención de los mensajes, el tiempo de espera de visibilidad y el tamaño máximo de los mensajes para adaptar la cola a los requisitos de su aplicación.



Selección de regiones y etiquetas de recursos

Asegúrese de estar en lo correcto para acceder Región de AWS a sus colas y administrarlas de forma eficaz. Además, considere la posibilidad de utilizar etiquetas de recursos para organizar y categorizar las colas, lo que permitirá una mejor administración de los recursos, la asignación de costos y el control del acceso en su AWS entorno compartido.

Al aprovechar las características y funcionalidades que ofrece la consola de Amazon SQS, puede administrar de forma eficiente su infraestructura de mensajería, optimizar el rendimiento de las colas y garantizar una entrega fiable de los mensajes para sus aplicaciones.

Tipos de colas de Amazon SQS

Amazon SQS admite dos tipos de colas: colas estándar y colas FIFO. Utilice la siguiente tabla para determinar qué cola se adapta mejor a sus necesidades.

Colas estándar	Colas FIFO
Rendimiento ilimitado: las colas estándar admiten un número muy alto, casi ilimitado , de llamadas a la API por segundo y por acción (`SendMessage` , `ReceiveMessage` o `DeleteMessage`). Este alto rendimien to las hace ideales para casos de uso que requieren procesar grandes volúmenes de	Alto rendimiento: si utiliza el procesamiento por lotes, las colas FIFO procesan hasta 3000 mensajes por segundo por método de API (`SendMessageBatch` , `ReceiveMe ssage` o `DeleteMessageBatch`). Este rendimiento se basa en 300 llamadas a la API por segundo, y cada llamada a la API

Colas estándar	Colas FIFO
mensajes con rapidez, como el flujo de datos en tiempo real o las aplicaciones a gran escala. Si bien las colas estándar se escalan automáticamente en función de la demanda, es esencial supervisar los patrones de uso para garantizar un rendimiento óptimo, especialmente en las regiones con mayores cargas de trabajo. At-least-once entrega: at-least-once entrega garantizada, lo que significa que cada mensaje se entrega al menos una vez, pero en algunos casos, un mensaje puede entregarse más de una vez debido a reintentos o retrasos en la red. Debe diseñar su aplicación para administrar posibles mensajes duplicados mediante operaciones idempotentes, que garanticen que procesar el mismo mensaje varias veces no afecte al estado del sistema. Orden de mejor esfuerzo: permite orden de mejor esfuerzo, lo que significa que, si bien Amazon SQS intenta entregar los mensajes en el orden en que se enviaron, no lo garantiza. En algunos casos, los mensajes pueden llegar desordenados, especialmente en condiciones de alto rendimiento o por errores de recuperación. En el caso de las aplicaciones en las que el orden del procesamiento de los mensajes es crucial, debería administrar la lógica de reordenamiento dentro de la aplicación o utilizar colas FIFO para garantizar un orden estricto. Durabilidad y redundancia: las colas estándar garantizan una alta durabilidad al almacenar varias copias de cada mensaje en varias AWS	administra un lote de 10 mensajes. Al habilitar el modo de alto rendimiento, puede escalar hasta 30 000 transacciones por segundo (TPS) con un orden flexible dentro de los grupos de mensajes. Sin el procesamiento por lotes, las colas FIFO admiten hasta 300 llamadas a la API por segundo, por método API (`SendMessage`, `ReceiveMessage` o `DeleteMessage`). Si necesita incrementar el rendimiento, puede solicitar un aumento de cuota a través del Centro de soporte de AWS. Para activar el modo de alto rendimiento, consulte Habilitación del alto rendimiento para las colas FIFO en Amazon SQS. Procesamiento único: las colas FIFO entregan cada mensaje una vez y lo mantienen disponible hasta que usted lo procese y elimine. Al utilizar características como `MessageDeduplicationId` o la desduplicación basada en el contenido, se evitan los mensajes duplicados, incluso cuando se vuelven a intentar debido a problemas de red o a tiempos de espera. First-in-first-out entrega: las colas FIFO garantizan que reciba los mensajes en el orden en que se envían dentro de cada grupo de mensajes. Al distribuir los mensajes entre varios grupos, puede procesarlos en paralelo y, al mismo tiempo, mantener el orden dentro de cada grupo.

Colas estándar	Colas FIFO
zonas de disponibilidad. Esto garantiza que los mensajes no se pierdan, incluso en caso de que se produzcan fallos en la infraestructura. Tiempo de espera de visibilidad: Amazon SQS le permite configurar un tiempo de espera de visibilidad para controlar cuánto tiempo permanece oculto un mensaje después de recibirlo, lo que garantiza que otros consumido res no procesen el mensaje hasta que se haya procesado por completo o caduque el tiempo de espera.	

Colas estándar	Colas FIFO
Utilice colas estándar para enviar datos entre aplicaciones cuando el rendimiento sea crucial, por ejemplo: • Desacoplar solicitudes de usuario en tiempo real de un trabajo en segundo plano intensivo. Permita que los usuarios carguen contenido multimedia rápidamente mientras usted procesa tareas como el cambio de tamaño o la codificación en segundo plano, lo que garantiza tiempos de respuesta rápidos sin sobrecargar el sistema. • Asignar tareas a varios nodos de trabajo. Distribuya un elevado número de solicitudes de validación de tarjetas de crédito en varios nodos de trabajo y administre mensajes duplicados con operaciones idempotentes para evitar errores de procesamiento. • Agrupar por lotes mensajes para procesarlos en el futuro. Coloque en cola varias entradas para añadirlas por lotes a una base de datos. Como el orden de los mensajes no está garantizado, diseñe su sistema para que se encargue del out-of-order procesamiento si es necesario.	Use colas FIFO para enviar datos entre aplicaciones cuando el orden de los eventos sea importante, por ejemplo: • Asegurarse de que los comandos introduci dos por el usuario se ejecutan en el orden correcto. Este es un caso de uso clave para las colas FIFO, en las que el orden de los comandos es crucial. Por ejemplo, si un usuario realiza una secuencia de acciones en una aplicación, las colas FIFO garantiza n que las acciones se procesen en el mismo orden en que se ingresaron. • Mostrar el precio correcto del producto enviando las modificaciones de precios en el orden adecuado. Las colas FIFO garantiza n que se reciban varias actualizaciones del precio de un producto y se procesen secuencialmente. Sin FIFO, una reducción de precio podría procesarse después de un aumento de precio, lo que provocaría que se mostraran datos incorrectos. • Evitar que un estudiante se matricule en un curso sin antes de registrarse para obtener una cuenta. Al utilizar las colas FIFO, se asegura de que el proceso de registro se lleve a cabo en la secuencia correcta. El sistema procesa primero el registro de la cuenta y, después, la inscripción al curso, lo que evita que la solicitud de inscripción se ejecute prematuramente.

Implementación de sistemas de solicitud-respuesta

Al implementar un sistema de solicitud-respuesta o de llamada a procedimiento remoto (RPC), tenga en cuenta las siguientes prácticas recomendadas:

- Cree colas de respuesta al inicio: en lugar de crear colas de respuesta por mensaje, créelas al inicio, por productor. Utilice un atributo de mensaje de ID de correlación para asignar las respuestas a las solicitudes de manera eficiente.
- Evite compartir las colas de respuestas entre los productores: asegúrese de que cada productor tenga su propia cola de respuestas. Si se comparten las colas de respuesta, un productor puede recibir mensajes de respuesta destinados a otro productor.

Para obtener más información acerca de cómo implementar el patrón de solicitud de respuesta utilizando el Cliente de colas temporales, consulte Patrón de mensajes de respuesta a solicitudes (colas virtuales).

Creación de una cola estándar de Amazon SQS y envío de un mensaje

Puede crear una cola estándar y enviar mensajes mediante la consola Amazon SQS. Este tema también hace hincapié en las mejores prácticas, como evitar la información confidencial en los nombres de las colas y utilizar el cifrado administrado del lado del servidor.

Creación de una cola estándar con la consola Amazon SQS

> **Important**
>
> El 17 de agosto de 2022, se aplicó de forma predeterminada el cifrado del servidor (SSE) a todas las colas de Amazon SQS.
> No agregue información de identificación personal (PII) ni ninguna otra información confidencial o sensible en los nombres de las colas. Muchos Amazon Web Services pueden acceder a los nombres de las colas, incluidos los CloudWatch registros y la facturación. Los nombres de las colas no están diseñados para contener información privada o confidencial.

Creación de una cola estándar de Amazon SQS

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. Elige Crear cola.
3. En Tipo, el tipo de cola estándar se establece de forma predeterminada.

> **Note**
>
> No se puede cambiar el tipo de cola después de crearla.

4. Escriba un Nombre para la cola.
5. (Opcional) La consola establece valores predeterminados para los parámetros de configuración de la cola. En Configuración, puede establecer nuevos valores para los siguientes parámetros:

   a. En Tiempo de espera de visibilidad, introduzca la duración y las unidades. El intervalo es de 0 segundos a 12 horas. El valor predeterminado es de 30 segundos.

   b. En Periodo de retención del mensaje, introduzca la duración y las unidades. El intervalo es de 1 minuto a 14 días. El valor predeterminado es 4 días.

   c. En Retraso de entrega, introduzca la duración y las unidades. El intervalo es de 0 segundos a 15 minutos. El valor predeterminado es 0 segundos.

   d. En Tamaño máximo del mensaje, introduzca un valor. El rango va de 1 KiB a 1024 KiB. El valor predeterminado es 1024 KiB.

   e. En Tiempo de espera de recepción del mensaje, introduzca un valor. El intervalo es de 0 a 20 segundos. El valor predeterminado es 0 segundos, lo que establece un sondeo corto. Cualquier valor distinto de cero establece un sondeo largo.

6. (Opcional) Defina una política de acceso. La política de acceso define las cuentas, los usuarios y los roles que pueden acceder a la cola. La política de acceso también define las acciones (como `SendMessage`, `ReceiveMessage` o`DeleteMessage`) a las que pueden acceder los usuarios. La política predeterminada permite que solo el propietario de la cola envíe y reciba mensajes.

   Para definir la política de acceso, realice una de las siguientes acciones:

   - Elija Básico para configurar quién puede enviar mensajes a la cola y quién puede recibir mensajes de la cola. La consola crea la política basándose en sus elecciones y muestra la política de acceso resultante en el panel JSON de solo lectura.

- Elija Avanzado para modificar directamente la política de acceso de JSON. De este modo, puede especificar un conjunto personalizado de acciones que puede realizar cada entidad principal (cuenta, usuario o rol).

7. En Política de permiso de redireccionamiento, elija Habilitado. Seleccione una de las siguientes opciones: Permitir todo, Por cola o Denegar todo. Al elegir Por cola, especifique una lista de hasta diez colas de origen por nombre de recurso de Amazon (ARN).
8. De forma predeterminada, Amazon SQS proporciona un cifrado del servidor administrado. Para elegir un tipo de clave de cifrado o para desactivar el cifrado del servidor administrado por Amazon SQS, expanda Cifrado. Para obtener más información sobre los tipos de claves de cifrado, consulte Configuración del cifrado del servidor para una cola mediante claves de cifrado administradas por SQS y Configuración del cifrado del servidor para una cola mediante la consola de Amazon SQS.

> **Note**
>
> Con SSE activado, se rechazarán las solicitudes `SendMessage` y `ReceiveMessage` anónimas a la cola cifrada. Las prácticas recomendadas de seguridad de Amazon SQS desaconsejan utilizar solicitudes anónimas. Si desea enviar solicitudes anónimas a una cola de Amazon SQS, asegúrese de desactivar SSE.

9. (Opcional) Para configurar una cola de mensajes fallidos a fin de recibir mensajes que no se pueden entregar, expanda Cola de mensajes fallidos.
10. (Opcional) Para agregar etiquetas a la cola, expanda Etiquetas.
11. Elige Crear cola. Amazon SQS crea la cola y muestra la página Detalles de la cola.

Amazon SQS propaga la información sobre la nueva cola por todo el sistema. Dado que Amazon SQS es un sistema distribuido, es posible que experimente un ligero retraso antes de que la consola muestre la cola en la página Colas.

Enviar un mensaje mediante una cola estándar

Una vez creada la cola, puede enviarle un mensaje.

1. En el panel de navegación izquierdo, elija Colas. En la lista de colas, seleccione la cola que ha creado.
2. En Acciones, elija Enviar y recibir mensajes.

La consola muestra la página Enviar y recibir mensajes.

3. En Cuerpo del mensaje, especifique el texto del mensaje.
4. Para una cola estándar, puede introducir un valor para Retraso de entrega y elegir las unidades. Por ejemplo, introduzca `60` y elija segundos. Para obtener más información, consulte Temporizadores de mensajes de Amazon SQS.
5. Elija Enviar mensaje.

   Cuando se envía el mensaje, la consola muestra un mensaje de confirmación. Elija Ver detalles para mostrar información sobre el mensaje enviado.

Creación de una cola FIFO de Amazon SQS y envío de un mensaje

Puede crear una cola FIFO de Amazon SQS y enviar mensajes mediante la consola. En este tema se explica cómo configurar los parámetros de las colas, incluidos el tiempo de espera de visibilidad, la retención de mensajes y la deduplicación, sin dejar de seguir las mejores prácticas de seguridad, como evitar la información confidencial en los nombres de las colas y habilitar el cifrado del lado del servidor. También se trata de la definición de las políticas de acceso, la configuración de las colas de correo no válido y el envío de mensajes con atributos específicos de la FIFO, como el identificador de grupo de mensajes y el identificador de deduplicación.

Creación de una cola FIFO mediante la consola Amazon SQS

Puede usar la consola de Amazon SQS para crear colas FIFO. La consola proporciona valores predeterminados para todas las configuraciones excepto para el nombre de la cola.

> **Important**
>
> El 17 de agosto de 2022, se aplicó de forma predeterminada el cifrado del servidor (SSE) a todas las colas de Amazon SQS.
> No agregue información de identificación personal (PII) ni ninguna otra información confidencial o sensible en los nombres de las colas. Muchos Amazon Web Services pueden acceder a los nombres de las colas, incluidos los CloudWatch registros y la facturación. Los nombres de las colas no están diseñados para contener información privada o confidencial.

Creación de una cola FIFO de Amazon SQS

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. Elige Crear cola.
3. En Tipo, el tipo de cola estándar se establece de forma predeterminada. Para crear una cola FIFO, elija FIFO.

   > **Note**
   >
   > No se puede cambiar el tipo de cola después de crearla.

4. Escriba un Nombre para la cola.

   La cola FIFO debe finalizar con el sufijo `.fifo`. El sufijo cuenta para la cuota de nombre de cola de 80 caracteres. Para determinar si una cola es FIFO, puede comprobar si el nombre de la cola termina con el sufijo.

5. (Opcional) La consola establece valores predeterminados para los parámetros de configuración de la cola. En Configuración, puede establecer nuevos valores para los siguientes parámetros:

   a. En Tiempo de espera de visibilidad, introduzca la duración y las unidades. El intervalo es de 0 segundos a 12 horas. El valor predeterminado es de 30 segundos.

   b. En Periodo de retención del mensaje, introduzca la duración y las unidades. El intervalo es de 1 minuto a 14 días. El valor predeterminado es 4 días.

   c. En Retraso de entrega, introduzca la duración y las unidades. El intervalo es de 0 segundos a 15 minutos. El valor predeterminado es 0 segundos.

   d. En Tamaño máximo del mensaje, introduzca un valor. El rango va de 1 KiB a 1024 KiB. El valor predeterminado es 1024 KiB.

   e. En Tiempo de espera de recepción del mensaje, introduzca un valor. El intervalo es de 0 a 20 segundos. El valor predeterminado es 0 segundos, lo que establece un sondeo corto. Cualquier valor distinto de cero establece un sondeo largo.

   f. Para una cola FIFO, elija Desduplicación basada en el contenido para activar la desduplicación basada en el contenido. La configuración predeterminada está desactivada.

   g. (Opcional) Para que una cola FIFO permita un mayor rendimiento en el envío y recepción de mensajes en la cola, elija Habilitar FIFO de alto rendimiento.

Al elegir esta opción, las opciones relacionadas (Ámbito de desduplicación y Límite de rendimiento FIFO) cambian a la configuración necesaria para habilitar un alto rendimiento para las colas FIFO. Si cambia alguna de las configuraciones necesarias para utilizar FIFO de alto rendimiento, el rendimiento normal estará en vigor para la cola y la desduplicación se producirá según lo especificado. Para obtener más información, consulte Alto rendimiento para las colas FIFO en Amazon SQS y Cuotas de mensajes de Amazon SQS.

6. (Opcional) Defina una política de acceso. La política de acceso define las cuentas, los usuarios y los roles que pueden acceder a la cola. La política de acceso también define las acciones (como `SendMessage`, `ReceiveMessage` o`DeleteMessage`) a las que pueden acceder los usuarios. La política predeterminada permite que solo el propietario de la cola envíe y reciba mensajes.

   Para definir la política de acceso, realice una de las siguientes acciones:

   - Elija Básico para configurar quién puede enviar mensajes a la cola y quién puede recibir mensajes de la cola. La consola crea la política basándose en sus elecciones y muestra la política de acceso resultante en el panel JSON de solo lectura.
   - Elija Avanzado para modificar directamente la política de acceso de JSON. De este modo, puede especificar un conjunto personalizado de acciones que puede realizar cada entidad principal (cuenta, usuario o rol).

7. En Política de permiso de redireccionamiento, elija Habilitado. Seleccione una de las siguientes opciones: Permitir todo, Por cola o Denegar todo. Al elegir Por cola, especifique una lista de hasta diez colas de origen por nombre de recurso de Amazon (ARN).

8. De forma predeterminada, Amazon SQS proporciona un cifrado del servidor administrado. Para elegir un tipo de clave de cifrado o para desactivar el cifrado del servidor administrado por Amazon SQS, expanda Cifrado. Para obtener más información sobre los tipos de claves de cifrado, consulte Configuración del cifrado del servidor para una cola mediante claves de cifrado administradas por SQS y Configuración del cifrado del servidor para una cola mediante la consola de Amazon SQS.

   > **Note**
   >
   > Con SSE activado, se rechazarán las solicitudes `SendMessage` y `ReceiveMessage` anónimas a la cola cifrada. Las prácticas recomendadas de seguridad de Amazon SQS desaconsejan utilizar solicitudes anónimas. Si desea enviar solicitudes anónimas a una cola de Amazon SQS, asegúrese de desactivar SSE.

9. (Opcional) Para configurar una cola de mensajes fallidos a fin de recibir mensajes que no se pueden entregar, expanda Cola de mensajes fallidos.
10. (Opcional) Para agregar etiquetas a la cola, expanda Etiquetas.
11. Elige Crear cola. Amazon SQS crea la cola y muestra la página Detalles de la cola.

Amazon SQS propaga la información sobre la nueva cola por todo el sistema. Dado que Amazon SQS es un sistema distribuido, es posible que experimente un ligero retraso antes de que la consola muestre la cola en la página Colas.

Tras crear una cola, puede enviarle mensajes, así como recibirlos y eliminarlos. También puede editar cualquiera de las opciones de configuración de la cola, excepto el tipo de cola.

Enviar un mensaje mediante una cola FIFO

Después de crear la cola, puede enviarle un mensaje.

1. En el panel de navegación izquierdo, elija Colas. En la lista de colas, seleccione la cola que ha creado.
2. En Acciones, elija Enviar y recibir mensajes.

   La consola muestra la página Enviar y recibir mensajes.
3. En Cuerpo del mensaje, especifique el texto del mensaje.
4. Para una cola First-In-First-Out (FIFO), introduzca un ID de grupo de mensajes. Para obtener más información, consulte Lógica de entrega de colas FIFO en Amazon SQS.
5. (Opcional) Para una cola FIFO, puede introducir un ID de desduplicación de mensajes. Si ha activado la desduplicación basada en el contenido para la cola, el ID de desduplicación de mensajes no es necesario. Para obtener más información, consulte Lógica de entrega de colas FIFO en Amazon SQS.
6. Las colas FIFO no admiten temporizadores en los mensajes individuales. Para obtener más información, consulte Temporizadores de mensajes de Amazon SQS.
7. Elija Enviar mensaje.

   Cuando se envía el mensaje, la consola muestra un mensaje de confirmación. Elija Ver detalles para mostrar información sobre el mensaje enviado.

Tareas habituales para empezar a usar Amazon SQS

Una vez que hayas creado una cola y hayas aprendido a enviar, recibir y eliminar mensajes, te recomendamos que pruebes lo siguiente:

- Active una función Lambda para procesar los mensajes entrantes automáticamente, lo que permite flujos de trabajo basados en eventos sin necesidad de realizar sondeos continuos.
- Configure las colas, incluidas las SSE y otras funciones.
- Enviar un mensaje con atributos.
- Enviar un mensaje desde una VPC.
- Descubra la funcionalidad y la arquitectura de Amazon SQS.
- Descubra las directrices y advertencias que le ayudarán a sacar el máximo provecho de Amazon SQS.
- Explore los ejemplos de Amazon SQS para un AWS SDK, como la Guía para AWS SDK for Java 2.x desarrolladores.
- Obtenga información sobre los comandos de Amazon SQS. AWS CLI
- Más información sobre las acciones de la API de Amazon SQS.
- Aprenda a interactuar con Amazon SQS mediante programación. Consulte Cómo trabajar con el Centro de APIs Desarrollo y explore el mismo AWS :
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows y .NET

- Aprenda a monitorear los costos y los recursos.
- Aprenda a proteger sus datos.
- Obtenga más información sobre el flujo de trabajo de Amazon SQS.

Administración de una cola de Amazon SQS

Aprenda a gestionar las colas de Amazon SQS mediante la consola, lo que incluye editar la configuración de las colas, recibir y eliminar mensajes, confirmar que las colas están vacías y eliminar o purgar las colas. Conozca las mejores prácticas para gestionar los mensajes de forma eficiente, como utilizar sondeos prolongados, gestionar los tiempos de espera de visibilidad y verificar las métricas mediante la supervisión de los paneles de control o el. AWS CLI Siga las medidas prácticas para mantener las colas y gestionar los mensajes de forma eficaz y, al mismo tiempo, minimizar las interrupciones.

Edición de una cola de Amazon SQS mediante la consola

Puede utilizar la consola Amazon SQS para editar los parámetros de configuración de colas (excepto el tipo de cola) y modificar o eliminar funciones según sea necesario.

Edición de una cola de Amazon SQS (consola)

1. En la consola de Amazon SQS, abra la página Colas.
2. Seleccione un tema y, a continuación, elija Editar.
3. (Opcional) En Configuración, actualice los parámetros de configuración de la cola.
4. (Opcional) Para actualizar la política de acceso, en Política de acceso, modifique la política de JSON.
5. (Opcional) Para actualizar una política de permiso de redireccionamiento de una cola de mensajes fallidos, expanda Política de permiso de redireccionamiento.
6. (Opcional) Para actualizar o eliminar el cifrado, expanda Cifrado.
7. (Opcional) Para agregar, actualizar o eliminar una cola de mensajes fallidos (que le permite recibir mensajes que no se pueden entregar), expanda Cola de mensajes fallidos.
8. (Opcional) Para agregar, actualizar o eliminar las etiquetas de la cola, expanda Etiquetas.
9. Seleccione Save.

    - La consola muestra la página Detalles de la cola.

Recepción y eliminación de mensajes en Amazon SQS

Tras enviar los mensajes a una cola de Amazon SQS, puede recuperarlos y eliminarlos para procesar el flujo de trabajo de la aplicación. Este proceso garantiza una gestión segura y fiable de los mensajes. En este tema se explica cómo recuperar y eliminar mensajes mediante la consola Amazon SQS y se explican los ajustes clave para optimizar esta operación. Los siguientes son conceptos clave para recibir y eliminar mensajes:

1. Recepción de mensajes
   - Cuando recupera mensajes de una cola de Amazon SQS, no puede segmentar mensajes específicos. En su lugar, especifique el número máximo de mensajes que se van a recuperar en una sola solicitud (hasta 10).
   - Debido a la naturaleza distribuida de Amazon SQS, la recuperación de una cola con pocos mensajes puede arrojar una respuesta vacía. Para mitigar esta situación:
     - Utilice una votación prolongada, que espere hasta que haya un mensaje disponible o hasta que se agote el tiempo de espera de la encuesta. Este enfoque reduce los costos de votación innecesarios y mejora la eficiencia.
     - Vuelva a emitir la solicitud si es necesario.
2. Visibilidad y eliminación de los mensajes
   - Los mensajes no se eliminan automáticamente tras su recuperación. Esta función garantiza que pueda volver a procesar los mensajes en caso de que se produzcan fallos en las aplicaciones o interrupciones de la red.
   - Tras el procesamiento, debe enviar de forma explícita una solicitud de eliminación para eliminar el mensaje de forma permanente. Esta acción confirma que la gestión se ha realizado correctamente.
   - Los mensajes recuperados con la consola Amazon SQS permanecen visibles para volver a recuperarlos. Ajuste la configuración del tiempo de espera de visibilidad para los entornos automatizados a fin de ocultar temporalmente los mensajes a otros consumidores mientras se procesan.
3. Tiempo de espera de visibilidad
   - Esta configuración determina cuánto tiempo permanece oculto un mensaje tras su recuperación. Establezca un tiempo de espera adecuado para garantizar que los mensajes se procesen solo una vez y para evitar la duplicación durante el procesamiento distribuido.

Recepción y eliminación de un mensaje mediante la consola

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. En la página Colas, elija la cola de la que desea recibir los mensajes y, a continuación, seleccione Enviar y recibir mensajes.
4. En la página Enviar y recibir mensajes, selecciona Buscar mensajes.

   Amazon SQS muestra una barra de progreso que indica la duración del sondeo. Los mensajes recuperados aparecerán en la sección Mensajes, donde se muestra:

   - Message ID
   - Fecha de envío
   - Tamaño
   - Recuento de recepción
5. Para eliminar los mensajes, elige los que quieras eliminar y selecciona Eliminar.

   Confirme la eliminación en el cuadro de diálogo Eliminar mensajes seleccionando Eliminar.

Para obtener más información sobre las operaciones avanzadas, incluida la recuperación y eliminación de mensajes basadas en API, consulte la Guía de referencia de las API de Amazon SQS.

Confirmación de que una cola de Amazon SQS está vacía

En la mayoría de los casos, puede utilizar el sondeo largo para determinar si una cola está vacía. En contados casos, es posible que reciba respuestas vacías incluso cuando una cola aún contenga mensajes, sobre si especifica un valor bajo para Tiempo de espera de recepción del mensaje al crear la cola. En esta sección se describe cómo confirmar que una cola está vacía.

Confirmación de que una cola está vacía (consola)

1. Detenga el envío de mensajes de todos los productores.
2. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
3. En el panel de navegación, elija Colas.
4. En la página Colas, elija una cola.
5. Elija la pestaña Monitorización.

6. En la parte superior derecha de los paneles de monitorización, elija la flecha hacia abajo situada junto al símbolo Actualizar. En el menú desplegable, elija Actualización automática. Deje el Intervalo de actualización en 1 minuto.
7. Observe los siguientes paneles:
   - Número aproximado de mensajes retrasados
   - Número aproximado de mensajes no visibles
   - Número aproximado de mensajes visibles

   Si todos muestran valores `0` durante varios minutos, la cola está vacía.

Para confirmar que una cola está vacía (AWS CLI, AWS API)

1. Detenga el envío de mensajes de todos los productores.
2. Ejecute uno de los siguientes comandos de forma repetida:
   - AWS CLI: `get-queue-attributes`
   - AWS API: `GetQueueAttributes`
3. Observe las métricas de los siguientes atributos:
   - `ApproximateNumberOfMessagesDelayed`
   - `ApproximateNumberOfMessagesNotVisible`
   - `ApproximateNumberOfMessagesVisible`

   Si todos son `0` durante varios minutos, la cola está vacía.

Si confías en CloudWatch las métricas de Amazon, asegúrate de ver varios puntos de datos cero consecutivos antes de considerar que la cola está vacía. Para obtener más información sobre CloudWatch las métricas, consulta CloudWatch Métricas disponibles para Amazon SQS.

Eliminación de una cola de Amazon SQS

Si ya no utiliza una cola de Amazon SQS y no piensa utilizarla en un futuro próximo, elimínela.

> **Tip**
>
> Si desea verificar que una cola esté vacía antes de eliminarla, consulte Confirmación de que una cola de Amazon SQS está vacía.

Puede eliminar una cola aunque no esté vacía. Para eliminar los mensajes de una cola, pero no la propia cola, púrguela.

Eliminación de una cola (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. En la página Colas, elija la cola que desea eliminar.
4. Elija Eliminar.
5. En el cuadro de diálogo Eliminar cola, introduzca **`delete`** para confirmar la eliminación.
6. Elija Eliminar.

Para eliminar una cola (AWS CLI y una API)

Elige el método adecuado para eliminar la cola en función de tus necesidades:

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

Purgado de mensajes de una cola mediante la consola de Amazon SQS

Para mantener una cola de Amazon SQS pero eliminar todos sus mensajes, puede purgarla. De este modo, se eliminarán todos los mensajes, incluidos los que estén actualmente invisibles (en movimiento). El proceso de purga puede tardar hasta 60 segundos, así que espere los 60 segundos completos, independientemente del tamaño de la cola.

> **Important**
>
> Al purgar una cola, no puede recuperar ningún mensaje eliminado.

Purga de una cola (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. En la página Colas, elija la cola que desea purgar.
4. En Acciones, elija Purgar.
5. En el cuadro de diálogo Purgar cola, introduzca **`purge`** y elija Purgar para confirmar el purgado.
   - Todos los mensajes se purgan de la cola. La consola mostrará un banner de confirmación.

Colas estándar de Amazon SQS

Amazon SQS proporciona colas estándar como tipo de cola predeterminado y admite un número casi ilimitado de llamadas a la API por segundo para acciones como `SendMessage`, `ReceiveMessage` y `DeleteMessage`. Las colas estándar garantizan la entrega de los at-least-once mensajes, pero debido a la arquitectura altamente distribuida, es posible que se entregue más de una copia de un mensaje y, en ocasiones, los mensajes lleguen desordenados. A pesar de ello, las colas estándar hacen todo lo posible por mantener el orden en el que se envían los mensajes.

Cuando envía un mensaje utilizando`SendMessage`, Amazon SQS almacena el mensaje de forma redundante en varias zonas de disponibilidad (AZs) antes de confirmarlo. Esta redundancia garantiza que ningún error de equipo, red o zona de disponibilidad pueda provocar que no se pueda acceder a los mensajes.

Puede crear y configurar colas mediante la consola de Amazon SQS. Para obtener instrucciones detalladas, consulta Creación de una cola estándar con la consola Amazon SQS. Para ver ejemplos específicos de Java, consulte Ejemplos de SDK de Java de Amazon SQS.

Casos de uso para colas estándar

Las colas de mensajes estándar son adecuadas para varios escenarios, siempre que su aplicación pueda administrar los mensajes que pueden llegar más de una vez o desordenados. Entre los ejemplos se incluyen:

- Desacoplar las solicitudes de los usuarios en directo del trabajo en segundo plano intenso: los usuarios pueden cargar contenido multimedia mientras el sistema cambia su tamaño o lo codifica en segundo plano.
- Asignar tareas a varios nodos de trabajo: por ejemplo, administrar un elevado volumen de solicitudes de validación de tarjetas de crédito.
- Agrupar mensajes para procesarlos más adelante: programar varias entradas para agregarlas a una base de datos más tarde.

Para obtener información sobre las cuotas relacionadas con las colas estándar, consulte Cuotas de colas estándar de Amazon SQS.

Para conocer las prácticas recomendadas del trabajo con colas estándar, consulte Prácticas recomendadas para Amazon SQS.

Entrega de Amazon SQS at-least-once

Amazon SQS almacena copias de los mensajes en varios servidores para mejorar la redundancia y lograr una alta disponibilidad. En raras ocasiones, uno de los servidores que almacena una copia de un mensaje podría no estar disponible cuando usted reciba o elimine un mensaje.

En este caso, la copia no se elimina en el servidor no disponible y podría recibir una nueva copia al recibir mensajes. Diseñe sus aplicaciones de modo que sean idempotentes (no deben verse afectadas negativamente si se procesa el mismo mensaje más de una vez).

Identificadores de colas y mensajes de Amazon SQS

En este tema se describen los identificadores de las colas estándar y FIFO. Estos identificadores pueden ayudarle a encontrar y manipular colas y mensajes específicos.

identificadores de colas estándar de Amazon SQS

Para obtener más información sobre los siguientes identificadores, consulte la Referencia de la API de Amazon Simple Queue Service.

Nombre de cola y URL

Cuando se crea una nueva cola, debe especificar un nombre de cola único para su cuenta y región de AWS . Amazon SQS asigna a cada cola que crea un identificador llamado una URL de cola que incluye el nombre de la cola y otros componentes de Amazon SQS. Siempre que desee realizar una acción en una cola, tiene que proporcionar su URL de cola.

A continuación se muestra la URL de una cola denominada `MyQueue` que pertenece a un usuario que tiene el número de cuenta de AWS `123456789012`.

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

Puede recuperar la dirección URL de una cola mediante programación enumerando las colas y analizando la cadena que sigue al número de cuenta. Para obtener más información, consulte `ListQueues`.

Message ID

Cada mensaje recibe un ID de mensaje asignado por el sistema que Amazon SQS le devuelve en la respuesta `SendMessage`. Este identificador es útil para identificar los mensajes. La longitud máxima de un ID de mensaje es 100 caracteres.

Identificador de recepción

Cada vez que recibe un mensaje de una cola, recibe un identificador de recepción para ese mensaje. Este controlador está asociado a la acción de recepción del mensaje, no al propio mensaje. Para eliminar el mensaje o cambiar la visibilidad de los mensajes, debe proporcionar el identificador de recepción (no el ID de mensaje). Por tanto, siempre debe recibir un mensaje para poder eliminarlo (no puede poner un mensaje en la cola y, a continuación, recuperarlo). La longitud máxima de un identificador de recepción es 1024 caracteres.

> **Important**
>
> Si recibe un mensaje más de una vez, cada vez que lo reciba, obtendrá un identificador de recepción diferente. Cuando solicite eliminar el mensaje, debe proporcionar el identificador de recepción recibido más recientemente (de lo contrario, el mensaje podría no eliminarse).

A continuación se muestra un ejemplo de un identificador de recibo dividido en tres líneas.


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

Colas FIFO de Amazon SQS

Las colas FIFO (primero en entrar, primero en salir) tienen todas las capacidades de las colas estándar, pero están diseñadas para mejorar la mensajería entre aplicaciones cuando el orden de las operaciones y los eventos es crítico, o cuando no se pueden tolerar los duplicados.

Las características más importantes de las colas FIFO son entrega FIFO (primero en entrar, primero en salir) y procesamiento único:

- El orden en que se envían y reciben los mensajes se conserva estrictamente, y un mensaje se envía una sola vez y permanece no disponible hasta que un consumidor lo procesa y elimina.
- No se introducen duplicados en la cola.

Además, las colas FIFO admiten grupos de mensajes que permiten varios grupos de mensajes ordenados en una única cola. No hay cuota para el número de grupos de mensajes en una cola FIFO.

Algunos ejemplos de situaciones en las que puede utilizar colas FIFO son las siguientes:

1. Sistema de administración de pedidos de comercio electrónico en el que el orden es fundamental
2. Integración con sistemas de terceros en los que es necesario procesar eventos en orden
3. Procesamiento de las entradas introducidas por el usuario en el orden introducido
4. Comunicaciones y redes: envío y recepción de datos e información en el mismo orden
5. Sistemas de computación: comprobación de que los comandos introducidos por el usuario se ejecutan en el orden correcto
6. Institutos de enseñanza: evitar que un estudiante se matricule en un curso antes de registrarse para obtener una cuenta
7. Sistema de venta de entradas en línea: las entradas se distribuyen por orden de llegada

> **Note**
>
> Las colas FIFO también proporcionan un procesamiento único, pero tienen un número limitado de transacciones por segundo (TPS). Puede utilizar el modo de alto rendimiento de Amazon SQS con la cola FIFO para aumentar el límite de transacciones. Para obtener más información sobre el uso del modo de alto rendimiento, consulte Alto rendimiento para

las colas FIFO en Amazon SQS. Para obtener información sobre las cuotas de rendimiento, consulte the section called “Cuotas de mensajes”.

Las colas FIFO de Amazon SQS están disponibles en todas las regiones en las que está disponible Amazon SQS.

Para obtener más información sobre cómo utilizar las colas FIFO con ordenaciones complejas, consulte Resolución de problemas de ordenaciones complejas con las colas FIFO de Amazon SQS.

Para obtener información sobre cómo crear y configurar colas mediante la consola de Amazon SQS, consulte Creación de una cola estándar con la consola Amazon SQS. Para ver ejemplos de Java, consulte Ejemplos de SDK de Java de Amazon SQS.

Para conocer las prácticas recomendadas del trabajo con colas FIFO, consulte Prácticas recomendadas para Amazon SQS.

Términos clave de cola FIFO de Amazon SQS

Los siguientes términos clave pueden ayudarlo a comprender mejor la funcionalidad de las colas FIFO. Para obtener más información, consulte la Referencia de la API de Amazon Simple Queue Service.

Clientes

El cliente asincrónico con búfer de Amazon SQS no admite actualmente las colas FIFO.

ID de desduplicación de mensajes

Un token que se utiliza en las colas FIFO de Amazon SQS para identificar de forma exclusiva los mensajes y evitar la duplicación. Si se envían varios mensajes con el mismo ID de desduplicación en un intervalo de desduplicación de 5 minutos, se tratan como duplicados y solo se entrega una copia. Si no especifica un ID de desduplicación y está habilitada la desduplicación basada en contenido, Amazon SQS genera un ID de desduplicación mediante un hash del cuerpo del mensaje. Este mecanismo garantiza la entrega una sola vez al eliminar los mensajes duplicados dentro del período de tiempo especificado.

> **Note**
>
> Amazon SQS sigue rastreando el ID de deduplicación incluso después de recibir y eliminar el mensaje.

ID de grupo de mensajes

En las colas FIFO (primero en entrar, primero en salir), `MessageGroupId` es un atributo que organiza los mensajes en grupos distintos. Los mensajes del mismo grupo de mensajes siempre se procesan de uno en uno, siguiendo un orden estricto, lo que garantiza que no se procesen dos mensajes del mismo grupo simultáneamente. En las colas estándar, el uso `MessageGroupId` permite hacer colas justas. Si se requiere un orden estricto, utilice una cola FIFO.

ID de intento de solicitud de recepción

El ID de intento de solicitud de recepción es un token único que se utiliza para deduplicar `ReceiveMessage`llamadas en Amazon SQS.

Número de secuencia

El número grande y no consecutivo que Amazon SQS asigna a cada mensaje.

Servicios

Si su aplicación utiliza varios AWS servicios o una combinación de AWS servicios externos, es importante entender qué funcionalidad del servicio no admite las colas FIFO.

Es posible que algunos servicios externos AWS o algunos que envían notificaciones a Amazon SQS no sean compatibles con las colas FIFO, a pesar de que le permiten establecer una cola FIFO como destino.

Las siguientes características de los AWS servicios no son compatibles actualmente con las colas FIFO:

- Notificaciones de eventos de Amazon S3
- Enlaces de ciclo de vida de escalado automático
- AWS IoT Acciones de reglas
- AWS Lambda Colas de mensajes fallidos

Para obtener información acerca de la compatibilidad de otros servicios con colas FIFO, consulte la documentación de los servicios.

Lógica de entrega de colas FIFO en Amazon SQS

Los siguientes conceptos aclaran cómo gestionan las colas FIFO de Amazon SQS el envío y la recepción de mensajes, especialmente cuando se trata del orden y el grupo de mensajes. IDs

Envío de mensajes

Las colas FIFO de Amazon SQS conservan el orden de los mensajes mediante una IDs deduplicación y un grupo de mensajes únicos. IDs En este tema se destaca la importancia de los grupos de mensajes IDs para mantener un orden estricto dentro de los grupos y se destacan las mejores prácticas para garantizar una entrega de mensajes ordenada y fiable entre varios productores.

1. Preservación de pedidos
   - Cuando se envían varios mensajes sucesivamente a una cola FIFO con una deduplicación de mensajes única, Amazon IDs SQS los almacena y confirma su transmisión. A continuación, estos mensajes se reciben y procesan en el orden exacto en que se transmitieron.
2. ID de grupo de mensajes
   - En las colas FIFO, los mensajes se ordenan en función de su ID de grupo de mensajes. Si varios productores o hilos envían mensajes con el mismo ID de grupo de mensajes, Amazon SQS se asegura de que se almacenen y procesen en el orden en que llegan.
   - Práctica recomendada: Para garantizar un orden estricto de los mensajes entre varios productores, asigne un identificador de grupo de mensajes único a todos los mensajes de cada productor.
3. Pedido por grupo
   - La lógica de colas FIFO se aplica en función del ID de cada grupo de mensajes:
     - Cada ID de grupo de mensajes representa un grupo de mensajes distinto y ordenado.
     - Dentro de un ID de grupo de mensajes, todos los mensajes se envían y reciben siguiendo un orden estricto.
     - Los mensajes con un grupo de mensajes diferente IDs pueden llegar o procesarse de forma desordenada unos con respecto a otros.
   - Requisito: debes asociar un identificador de grupo de mensajes a cada mensaje. Si se envía un mensaje sin un ID de grupo, se produce un error en la acción.
   - Escenario de un solo grupo: si necesita que todos los mensajes se procesen siguiendo un orden estricto, utilice el mismo ID de grupo de mensajes para todos los mensajes.

Recepción de mensajes

Las colas FIFO de Amazon SQS gestionan la recuperación de mensajes, incluido el procesamiento por lotes, las garantías de los pedidos FIFO y las limitaciones a la hora de solicitar un grupo de mensajes específico. IDs En este tema se explica cómo Amazon SQS recupera los mensajes dentro y entre grupos de mensajes, IDs manteniendo normas estrictas de orden y visibilidad.

1. Recuperación de lotes
   - Al recibir mensajes de una cola FIFO con varios grupos de mensajes IDs, Amazon SQS:
     - Intenta devolver tantos mensajes como sea posible con el mismo ID de grupo de mensajes en una sola llamada.
     - Permite a otros consumidores procesar mensajes de diferentes grupos de mensajes de forma IDs simultánea.
   - Aclaración importante
     - Puede recibir varios mensajes del mismo ID de grupo de mensajes en un lote (hasta 10 mensajes en una sola llamada con el `MaxNumberOfMessages` parámetro).
     - Sin embargo, no podrás recibir más mensajes del mismo ID de grupo de mensajes en solicitudes posteriores hasta que:
       - Se eliminen los mensajes recibidos actualmente, o
       - Vuelven a ser visibles (por ejemplo, una vez transcurrido el tiempo de espera de visibilidad).
2. Garantía de pedido FIFO
   - Los mensajes recuperados en un lote conservan su orden FIFO dentro del grupo.
   - Si hay menos de 10 mensajes disponibles para el mismo ID de grupo de mensajes, Amazon SQS puede incluir mensajes de otro grupo de mensajes IDs en el mismo lote, pero cada grupo conserva el orden FIFO.
3. Limitaciones para el consumidor
   - No puedes solicitar explícitamente recibir mensajes de un ID de grupo de mensajes específico.

Varios reintentos

Los productores y los consumidores pueden volver a intentar de forma segura las acciones fallidas en las colas FIFO de Amazon SQS sin interrumpir el orden de los mensajes ni introducir duplicados. En este tema se destaca cómo los tiempos de espera de la deduplicación IDs y la visibilidad garantizan la integridad de los mensajes durante los reintentos.

1. El productor vuelve a intentarlo
   - Si se produce un error en una `SendMessage`acción, el productor puede volver a intentar enviar el mensaje varias veces con el mismo ID de deduplicación de mensajes.
   - Siempre que el productor reciba al menos un acuse de recibo antes de que venza el intervalo de deduplicación, lo volverá a intentar:
     - No introduzca mensajes duplicados.
     - No altere el orden de los mensajes.
2. El consumidor vuelve a intentarlo
   - Si una `ReceiveMessage`acción falla, el consumidor puede volver a intentarlo tantas veces como sea necesario con el mismo ID de intento de solicitud de recepción.
   - Siempre que el consumidor reciba al menos un acuse de recibo antes de que caduque el tiempo de espera de visibilidad, lo volverá a intentar:
     - No altere el orden de los mensajes.

Notas adicionales sobre el comportamiento de la FIFO

Obtenga información sobre cómo gestionar los tiempos de espera de visibilidad, habilitar el procesamiento paralelo con varios grupos IDs de mensajes y garantizar un procesamiento secuencial estricto en escenarios de un solo grupo.

1. Cómo gestionar los tiempos de espera de visibilidad
   - Cuando se recupera un mensaje pero no se elimina, permanece invisible hasta que caduque el tiempo de espera de visibilidad.
   - No se devuelve ningún mensaje adicional del mismo ID de grupo de mensajes hasta que se elimine el primer mensaje o vuelva a estar visible.
2. Procesamiento simultáneo y paralelo
   - Las colas FIFO permiten el procesamiento paralelo de mensajes en diferentes grupos de mensajes. IDs
   - Para maximizar la simultaneidad, diseñe su sistema con varios grupos de mensajes IDs para lograr flujos de trabajo independientes.
3. Escenarios de un solo grupo
   - Para un procesamiento secuencial estricto de todos los mensajes de una cola FIFO, utilice un único ID de grupo de mensajes para todos los mensajes de la cola.

Ejemplos para una mejor comprensión

Los siguientes son escenarios prácticos que ilustran el comportamiento de las colas FIFO en Amazon SQS.

1. Escenario 1: ID de grupo único
   - Un productor envía cinco mensajes con el mismo ID de grupo de mensajes (Grupo A.
   - El consumidor recibe estos mensajes en orden FIFO. Hasta que el consumidor elimine estos mensajes o se agote el tiempo de espera de visibilidad, no se recibirá ningún mensaje adicional del Grupo A.
2. Escenario 2: Grupo múltiple IDs
   - Un productor envía cinco mensajes al grupo A y 5 al grupo B.
   - El consumidor 1 procesa los mensajes del grupo A, mientras que el consumidor 2 procesa los mensajes del grupo B. Esto permite el procesamiento en paralelo con un orden estricto dentro de cada grupo.
3. Escenario 3: Recuperación de lotes
   - Un productor envía siete mensajes al Grupo A y tres al Grupo B.
   - Un solo consumidor recupera hasta 10 mensajes. Si la cola lo permite, puede devolver:
     - Siete mensajes del grupo A y tres del grupo B (o menos si hay menos mensajes disponibles de un solo grupo).

Procesamiento una sola vez en Amazon SQS

A diferencia de las colas estándar, las colas FIFO no generan mensajes duplicados. Las colas FIFO le ayudan a evitar el envío de duplicados a una cola. Si reintenta la acción `SendMessage` dentro del intervalo de desduplicación de cinco minutos, Amazon SQS no genera ningún duplicado en la cola.

Para configurar la desduplicación, debe realizar una de las siguientes acciones:

- Habilitar la desduplicación basada en el contenido. Esto indica a Amazon SQS que debe utilizar un hash SHA-256 para generar el ID de desduplicación de mensajes mediante el cuerpo del mensaje, pero no los atributos del mensaje. Para obtener más información, consulte la documentación de las acciones `CreateQueue`, `GetQueueAttributes` y `SetQueueAttributes` en la Referencia de la API de Amazon Simple Queue Service.

- Proporcionar de manera explícita el ID de desduplicación de mensajes (o ver el número secuencial) del mensaje. Para obtener más información, consulte la documentación de las acciones `SendMessage`, `SendMessageBatch` y `ReceiveMessage` en la Referencia de la API de Amazon Simple Queue Service.

Cambio de una cola estándar a una cola FIFO en Amazon SQS

Si su aplicación utiliza colas estándar y desea aprovechar las características de ordenación o procesamiento único de las colas FIFO, debe configurar correctamente tanto la cola como la aplicación.

Consideraciones clave

- Creación de una cola FIFO: No puede convertir una cola estándar existente en una cola FIFO. Debe crear una nueva cola FIFO para su aplicación o eliminar la cola estándar existente y volver a crearla como una cola FIFO.
- Parámetro de retraso: Delay Parameter: Las colas FIFO no admiten retrasos por mensaje, solo por cola. Si su aplicación establece el parámetro `DelaySeconds` en todos los mensajes, debe modificarla para establecer `DelaySeconds` en toda la cola.
- ID de grupo de mensajes: proporcione un ID de grupo de mensajes para cada mensaje enviado. Este identificador permite el procesamiento paralelo de los mensajes manteniendo su orden respectivo. Para escalar mejor con colas FIFO, utilice una dimensión empresarial más detallada para el ID de mensaje. Cuantos más grupos de mensajes IDs distribuya los mensajes, mayor será el número de mensajes disponibles para su consumo.
- Modo de alto rendimiento: utilice el modo de alto rendimiento recomendado para FIFO a fin de lograr un mayor rendimiento. Para obtener más información sobre cuotas de mensajería, consulte Cuotas de mensajes de Amazon SQS.

Lista de verificación para trasladares a colas FIFO

Antes de enviar mensajes a una cola FIFO, confirme lo siguiente:

1. Configuración de los ajustes de retardo
   - Modifique su aplicación para eliminar los retrasos por mensaje.
   - Defina el `DelaySeconds` parámetro en toda la cola.
2. Configure el grupo de mensajes IDs

- Organice los mensajes en grupos de mensajes mediante la especificación de un ID de grupo de mensajes basándose en una dimensión empresarial.
- Utilice dimensiones empresariales más granulares para mejorar la escalabilidad.

3. Gestión de desduplicación de mensajes
   - Si tu aplicación no puede enviar mensajes con cuerpos de mensaje idénticos, proporciona un identificador de deduplicación de mensajes único para cada mensaje.
   - Si su aplicación envía mensajes con cuerpos únicos, habilite la desduplicación basada en el contenido.
4. Configuración del consumidor
   - Por lo general, el consumidor no necesita cambiar el código.
   - Si se tarda mucho tiempo en procesar los mensajes y el tiempo de espera de visibilidad es alto, considere la posibilidad de añadir un ID de intento de solicitud de recepción para cada acción `ReceiveMessage`. Esto le permite reintentar la recepción en caso de que se produzcan errores de red e impide que las colas se pausen debido a intentos fallidos de recepción.

Si sigue estos pasos, puede asegurarse de que su aplicación funcione correctamente con las colas FIFO, aprovechando al máximo sus características de orden y procesamiento de una sola vez. Para obtener más información, consulte la Referencia de la API de Amazon Simple Storage Service.

Comportamiento de la cola FIFO de Amazon SQS y de la simultaneidad de Lambda

Al utilizar una cola FIFO (primero en entrar, primero en salir) con Lambda, puede garantizar un procesamiento ordenado de los mensajes dentro de cada grupo de mensajes. La función de Lambda no ejecutará varias instancias para el mismo grupo de mensajes simultáneamente, por lo que mantendrá el orden. Sin embargo, puede escalarse para administrar varios grupos de mensajes en paralelo, lo que garantiza un procesamiento eficiente de la carga de trabajo de la cola. Los siguientes puntos describen el comportamiento de las funciones Lambda al procesar mensajes de una cola FIFO de Amazon SQS con respecto al grupo de mensajes: IDs

- Instancia única por grupo de mensajes: en cualquier momento, solo una instancia de Lambda procesará los mensajes de un ID de grupo de mensajes específico. Esto garantiza que los mensajes del mismo grupo se procesen en orden, manteniendo la integridad de la secuencia FIFO.

- Procesamiento simultáneo de diferentes grupos: Lambda puede procesar simultáneamente mensajes de diferentes IDs grupos de mensajes mediante varias instancias. Esto significa que, mientras una instancia de la función Lambda gestiona los mensajes de un ID de grupo de mensajes, otras instancias pueden gestionar simultáneamente los mensajes de otro grupo de mensajes IDs, lo que aprovecha las capacidades de simultaneidad de Lambda para procesar varios grupos en paralelo.

Agrupación de mensajes de cola FIFO

Las colas FIFO garantizan que los mensajes se procesen en el orden exacto en que se envían. Utilizan un ID de grupo de mensajes para agrupar los mensajes que deben procesarse secuencialmente.

Los mensajes del mismo grupo de mensajes se procesan en orden y solo se procesa un mensaje de cada grupo a la vez para mantener este orden.

Simultaneidad de Lambda con colas FIFO

Después de crear la cola, puede enviarle un mensaje.

Al configurar una función de Lambda para procesar los mensajes de una cola FIFO de Amazon SQS, Lambda respeta las garantías de orden que ofrece la cola FIFO. Los siguientes puntos describen el comportamiento de las funciones de Lambda en términos de simultaneidad y escalamiento al procesar mensajes de una cola FIFO de Amazon SQS cuando se utiliza un grupo de mensajes. IDs

- Simultaneidad dentro de los grupos de mensajes: solo una instancia de Lambda procesa los mensajes de un ID de grupo de mensajes concreto a la vez. Esto garantiza que los mensajes de un grupo se administren de forma secuencial.
- Escalado y varios grupos de mensajes: si bien Lambda puede escalar para procesar mensajes simultáneamente, este escalado se produce en diferentes grupos de mensajes. Si tiene varios grupos de mensajes, Lambda puede procesar varios grupos en paralelo, y cada grupo lo administra una instancia de Lambda independiente.

Para obtener más información, consulte Escalado y simultaneidad en Lambda en la Guía del operador de AWS Lambda .

Ejemplo de casos de uso

Supongamos que la cola FIFO recibe mensajes con el mismo ID de grupo de mensajes y que la función de Lambda tiene un límite de simultaneidad alto (hasta 1000).

Si se está procesando un mensaje del ID de grupo “A” y llega otro mensaje del ID de grupo “A”, el segundo mensaje no activará una nueva instancia de Lambda hasta que el primer mensaje se procese por completo.

Sin embargo, si llegan mensajes de los grupos «A IDs » y «B», ambos mensajes se pueden procesar simultáneamente en distintas instancias de Lambda.

Alto rendimiento para las colas FIFO en Amazon SQS

Las colas FIFO de alto rendimiento de Amazon SQS administran de forma eficiente el alto rendimiento de los mensajes y, al mismo tiempo, mantienen un orden estricto de los mensajes, lo que garantiza la fiabilidad y la escalabilidad de las aplicaciones que procesan numerosos mensajes. Esta solución es ideal para situaciones que exigen tanto un alto rendimiento como una entrega de mensajes ordenada.

Las colas FIFO de alto rendimiento de Amazon SQS no son necesarias en situaciones en las que el orden estricto de los mensajes no es crucial y en las que el volumen de mensajes entrantes es relativamente bajo o esporádico. Por ejemplo, si tiene una aplicación a pequeña escala que procesa mensajes poco frecuentes o no secuenciales, es posible que la complejidad y el costo adicionales asociados a las colas FIFO de alto rendimiento no estén justificados. Además, si su aplicación no requiere las capacidades de rendimiento mejoradas que ofrecen las colas FIFO de alto rendimiento, optar por una cola Amazon SQS estándar podría resultar más rentable y sencillo de administrar.

Para mejorar la capacidad de solicitudes en las colas FIFO de alto rendimiento, se recomienda aumentar el número de grupos de mensajes. Para obtener más información sobre las cuotas de mensajes de alto rendimiento, consulte Service Quotas de Amazon SQS en la Referencia general de Amazon Web Services.

Para obtener información sobre las cuotas por cola y las estrategias de distribución de datos, consulte Cuotas de mensajes de Amazon SQS y Particiones y distribución de datos para obtener alto rendimiento en las colas FIFO de SQS.

Casos de uso de alto rendimiento para colas FIFO de Amazon SQS

Los siguientes casos de uso destacan las diversas aplicaciones de las colas FIFO de alto rendimiento y muestran su eficacia en todos los sectores y escenarios:

1. Procesamiento de datos en tiempo real: las aplicaciones que se ocupan de flujos de datos en tiempo real, como el procesamiento de eventos o la ingesta de datos telemétricos, pueden beneficiarse de las colas FIFO de alto rendimiento para administrar la afluencia continua de mensajes y, al mismo tiempo, conservar su orden para un análisis preciso.
2. Procesamiento de pedidos de comercio electrónico: en las plataformas de comercio electrónico en las que es fundamental mantener el orden de las transacciones de los clientes, las colas FIFO de alto rendimiento garantizan que los pedidos se procesen de forma secuencial y sin retrasos, incluso durante las temporadas altas de compras.
3. Servicios financieros: las instituciones financieras que administran datos comerciales o transaccionales de alta frecuencia utilizan colas FIFO de alto rendimiento para procesar los datos de mercado y las transacciones con una latencia mínima y, al mismo tiempo, cumplen con los estrictos requisitos reglamentarios para la gestión de mensajes.
4. Transmisión multimedia: las plataformas de transmisión y los servicios de distribución multimedia utilizan colas FIFO de alto rendimiento para administrar la entrega de archivos multimedia y contenido en streaming, lo que garantiza una experiencia de reproducción fluida para los usuarios y, al mismo tiempo, mantiene el orden correcto de entrega del contenido.

Particiones y distribución de datos para obtener alto rendimiento en las colas FIFO de SQS

Amazon SQS almacena los datos de la cola FIFO en particiones. Una partición es una asignación de almacenamiento para una cola que se replica automáticamente en varias zonas de disponibilidad de una región. AWS Usted no administra las particiones. En su lugar, Amazon SQS se encarga de la administración de particiones.

Para las colas FIFO, Amazon SQS modifica el número de particiones de una cola en las siguientes situaciones:

- Si la tasa de solicitudes actual se aproxima o supera lo que pueden admitir las particiones existentes, se asignan particiones adicionales hasta que la cola alcanza la cuota regional. Para obtener información sobre las cuotas, consulte Cuotas de mensajes de Amazon SQS.

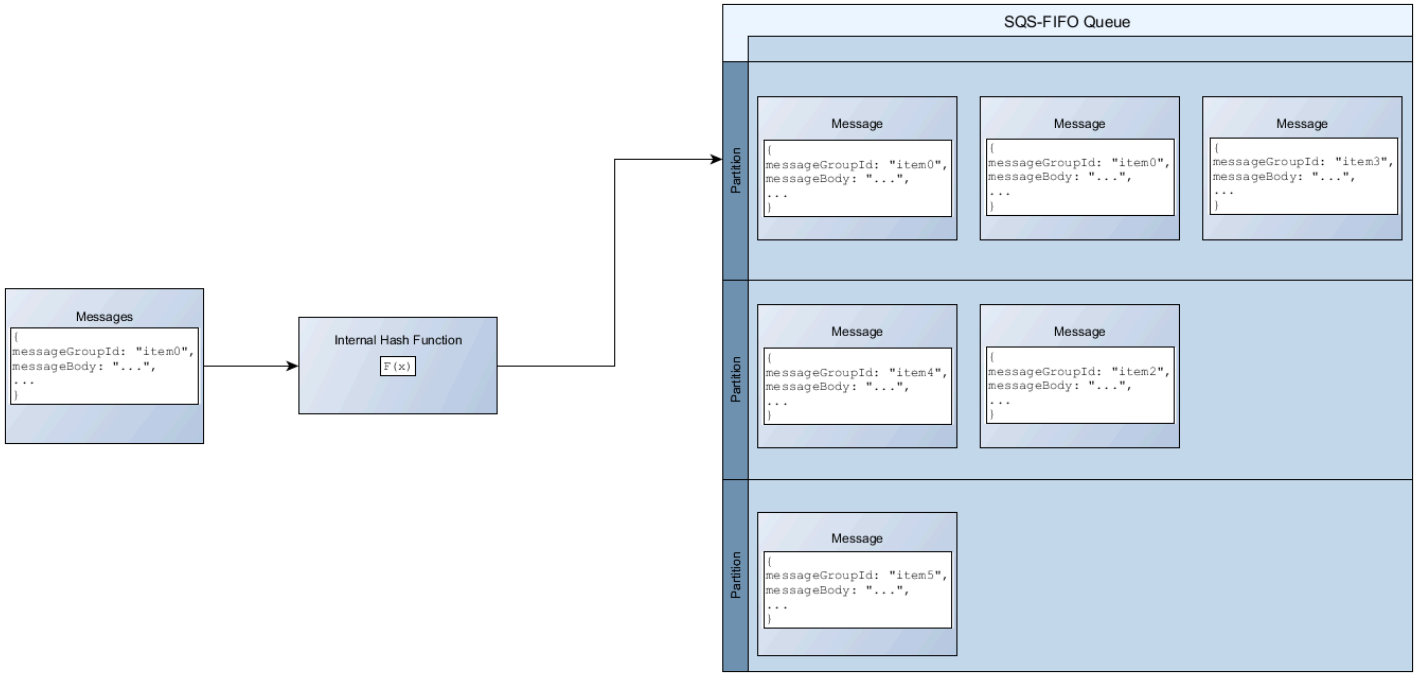
- Si las particiones actuales se utilizan poco, es posible que se reduzca el número de particiones.

La administración de las particiones tiene lugar automáticamente en segundo plano y es transparente para las aplicaciones. Su cola y sus mensajes están disponibles en todo momento.

Distribuir los datos por grupo de mensajes IDs

Para agregar un mensaje a una cola FIFO, Amazon SQS utiliza el valor del ID de grupo de mensajes de cada mensaje como entrada a una función hash interna. El valor del resultado de la función hash determina la partición que almacena el mensaje.

En el siguiente diagrama se muestra una tabla denominada que abarca varias particiones. El ID de grupo de mensajes de la cola se basa en el número de elemento. Amazon SQS utiliza la función hash para determinar dónde se almacenará un nuevo elemento; en este caso, se basa en el valor hash de la cadena `item0`. Tenga en cuenta que los elementos se almacenan en el mismo orden en el que se agregan a la cola. La ubicación de cada elemento viene determinada por el valor hash de su ID de grupo de mensajes.



> **Note**
>
> Amazon SQS está optimizado para una distribución uniforme de los elementos en las particiones de una cola FIFO, independientemente del número de particiones. AWS

recomienda utilizar un grupo de mensajes IDs que pueda tener un gran número de valores distintos.

Optimización de la utilización de particiones

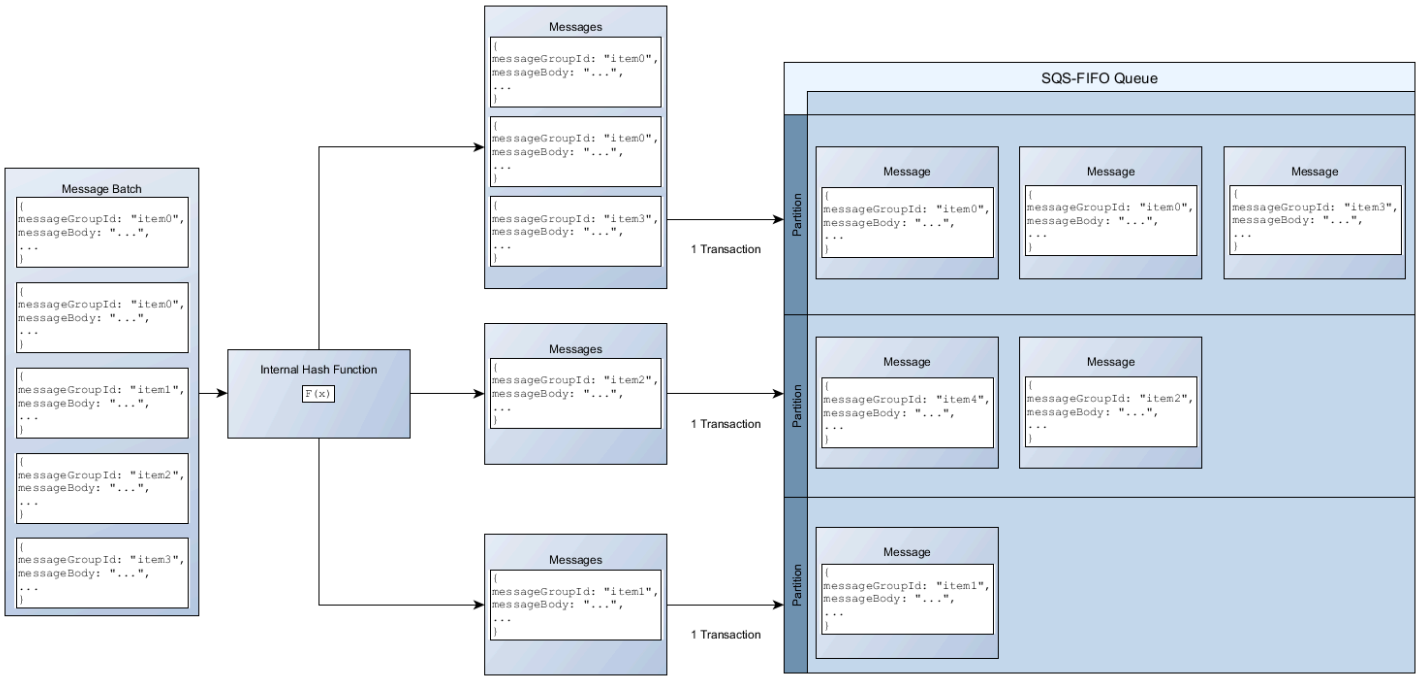
Cada partición admite hasta 3000 mensajes por segundo con procesamiento por lotes o hasta 300 mensajes por segundo para operaciones de envío, recepción y eliminación en las regiones compatibles. Para obtener más información sobre las cuotas de mensajes de alto rendimiento, consulte Service Quotas de Amazon SQS en la Referencia general de Amazon Web Services.

Cuando se usa por lotes APIs, cada mensaje se enruta según el proceso descrito enDistribuir los datos por grupo de mensajes IDs. Los mensajes que se enrutan a la misma partición se agrupan y procesan en una única transacción.

Para optimizar el uso de las particiones de la `SendMessageBatch` API, IDs se AWS recomienda agrupar los mensajes por lotes con el mismo grupo de mensajes siempre que sea posible.

Para optimizar el uso de las particiones en `DeleteMessageBatch` y `ChangeMessageVisibilityBatch` APIs, se AWS recomienda utilizar `ReceiveMessage` las solicitudes con el `MaxNumberOfMessages` parámetro establecido en 10 y agrupar en lotes los identificadores de recepción devueltos por una sola solicitud. `ReceiveMessage`

En el siguiente ejemplo, se envía un lote de mensajes con varios grupos de mensajes. IDs El lote se divide en tres grupos, cada uno de los cuales cuenta para la cuota de la partición.



> **Note**
>
> Amazon SQS solo garantiza que los mensajes con la misma función hash interna de ID de grupo de mensajes se agrupen en una solicitud por lotes. Según el resultado de la función hash interna y el número de particiones, es IDs posible que se agrupen los mensajes con grupos de mensajes diferentes. Dado que la función hash o el número de particiones pueden cambiar en cualquier momento, los mensajes agrupados en un momento dado pueden no estarlo más adelante.

Habilitación del alto rendimiento para las colas FIFO en Amazon SQS

Puede habilitar el alto rendimiento para cualquier cola FIFO nueva o existente. Esta característica incluye tres nuevas opciones a la hora de crear y editar colas FIFO:

- Habilitar FIFO de alto rendimiento: aumenta el rendimiento de los mensajes en la cola FIFO actual.
- Ámbito de desduplicación: especifica si la desduplicación se produce en el nivel de cola o de grupo de mensajes.
- Límite de rendimiento FIFO: especifica si la cuota de rendimiento de los mensajes en la cola FIFO se establece en el nivel de cola o de grupo de mensajes.

Habilitación de un alto rendimiento en una cola FIFO (consola)

1. Inicie la creación o edición de una cola FIFO.
2. Cuando especifique las opciones para la cola, elija Habilitar FIFO de alto rendimiento.

   La activación del alto rendimiento para las colas FIFO establece las opciones relacionadas de la siguiente manera:

   - Ámbito de desduplicación se establece a Grupo de mensajes, la configuración necesaria para utilizar alto rendimiento en las colas FIFO.
   - Límite de rendimiento FIFO se establece a Por ID de grupo de mensajes, la configuración necesaria para utilizar alto rendimiento en las colas FIFO.

   Si cambia alguna de las configuraciones necesarias para utilizar colas FIFO de alto rendimiento, el rendimiento normal estará en vigor para la cola y la desduplicación se producirá según lo especificado.
3. Continúe con la especificación de todas las opciones para la cola. Cuando termine, elija Crear cola o Guardar.

Tras crear o editar la cola FIFO, puede enviar mensajes a ella y recibir y eliminar mensajes, todo ello en un TPS superior. Para obtener información de las cuotas de alto rendimiento, consulte Rendimiento de mensajes en Cuotas de mensajes de Amazon SQS.

Identificadores de colas FIFO y mensajes en Amazon SQS

En esta sección se describen los identificadores de las colas FIFO. Estos identificadores pueden ayudarle a encontrar y manipular colas y mensajes específicos.

Identificadores de colas FIFO de Amazon SQS

Para obtener más información sobre los siguientes identificadores, consulte la Referencia de la API de Amazon Simple Queue Service.

Nombre de cola y URL

Cuando se crea una nueva cola, debe especificar un nombre de cola único para su cuenta y región de AWS . Amazon SQS asigna a cada cola que crea un identificador llamado una URL de cola que

incluye el nombre de la cola y otros componentes de Amazon SQS. Siempre que desee realizar una acción en una cola, tiene que proporcionar su URL de cola.

La cola FIFO debe finalizar con el sufijo `.fifo`. El sufijo cuenta para la cuota de nombre de cola de 80 caracteres. Para determinar si una cola es FIFO, puede comprobar si el nombre de la cola termina con el sufijo.

A continuación se muestra la URL de una cola FIFO denominada `MyQueue` que pertenece a un usuario que tiene el número de cuenta de AWS `123456789012`.

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

Puede recuperar la dirección URL de una cola mediante programación enumerando las colas y analizando la cadena que sigue al número de cuenta. Para obtener más información, consulte `ListQueues`.

Message ID

Cada mensaje recibe un ID de mensaje asignado por el sistema que Amazon SQS le devuelve en la respuesta `SendMessage`. Este identificador es útil para identificar los mensajes. La longitud máxima de un ID de mensaje es 100 caracteres.

Identificador de recepción

Cada vez que recibe un mensaje de una cola, recibe un identificador de recepción para ese mensaje. Este controlador está asociado a la acción de recepción del mensaje, no al propio mensaje. Para eliminar el mensaje o cambiar la visibilidad de los mensajes, debe proporcionar el identificador de recepción (no el ID de mensaje). Por tanto, siempre debe recibir un mensaje para poder eliminarlo (no puede poner un mensaje en la cola y, a continuación, recuperarlo). La longitud máxima de un identificador de recepción es 1024 caracteres.

> **Important**
>
> Si recibe un mensaje más de una vez, cada vez que lo reciba, obtendrá un identificador de recepción diferente. Cuando solicite eliminar el mensaje, debe proporcionar el identificador de recepción recibido más recientemente (de lo contrario, el mensaje podría no eliminarse).

A continuación se muestra un ejemplo de un identificador de recepción (dividido en tres líneas).

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


Identificadores adicionales para las colas FIFO de Amazon SQS

Para obtener más información sobre los siguientes identificadores, consulte Procesamiento una sola vez en Amazon SQS y la Referencia de la API de Amazon Simple Queue Service.

ID de desduplicación de mensajes

Un token que se utiliza en las colas FIFO de Amazon SQS para identificar de forma exclusiva los mensajes y evitar la duplicación. Si se envían varios mensajes con el mismo ID de desduplicación en un intervalo de desduplicación de 5 minutos, se tratan como duplicados y solo se entrega una copia. Si no especifica un ID de desduplicación y está habilitada la desduplicación basada en contenido, Amazon SQS genera un ID de desduplicación mediante un hash del cuerpo del mensaje. Este mecanismo garantiza la entrega una sola vez al eliminar los mensajes duplicados dentro del período de tiempo especificado.

ID de grupo de mensajes

`MessageGroupId`Es un atributo que solo se utiliza en las colas FIFO (primero en entrar, primero en salir) de Amazon SQS para organizar los mensajes en grupos distintos. Los mensajes del mismo grupo de mensajes siempre se procesan de uno en uno, siguiendo un orden estricto, lo que garantiza que no se procesen dos mensajes del mismo grupo simultáneamente. Las colas estándar no utilizan `MessageGroupId` ni ofrecen garantías de pedido. Si se requiere un orden estricto, utilice una cola FIFO en su lugar.

Número de secuencia

El número grande y no consecutivo que Amazon SQS asigna a cada mensaje.

Cuotas de Amazon SQS

En este tema se explican las cuotas y limitaciones de las colas FIFO y estándar de Amazon SQS y se detalla cómo afectan a la creación, la configuración y la gestión de mensajes de las colas. Obtenga información sobre las limitaciones, como los límites de retención de mensajes, los límites de mensajes en curso y los umbrales de rendimiento, así como sobre las estrategias para maximizar la eficiencia mediante el procesamiento por lotes, la optimización de las llamadas a las API y las encuestas prolongadas. En este tema también se abordan las convenciones de nomenclatura, las reglas de etiquetado y los métodos para solicitar aumentos de cuota a fin de satisfacer las cargas de trabajo de alta demanda, garantizando una gestión eficaz de las colas y un rendimiento óptimo.

Cuotas de colas FIFO de Amazon SQS

Cuotas de Amazon SQS

En la siguiente tabla se muestran las cuotas relacionadas con las colas FIFO.

Cuota	Descripción
Cola con retraso	El retraso predeterminado (mínimo) de una cola es de 0 segundos. El valor máximo es de 15 minutos.
Colas mostradas	1000 colas por cada solicitud de `ListQueues`.
Tiempo de espera de sondeo largo	El tiempo máximo de espera de sondeo es de 20 segundos.
Grupos de mensajes	No hay cuota para el número de grupos de mensajes en una cola FIFO.
Mensajes por cola (pendientes)	El número de mensajes que puede almacenar una cola de Amazon SQS es ilimitado.
Mensajes por cola (en tránsito)	Las colas FIFO admiten un máximo de 120 000 mensajes en vuelo (mensajes recibidos por un consumidor pero que aún no se han eliminado). Si se alcanza este límite, Amazon SQS no devuelve ningún error, pero es posible

Cuota	Descripción
	que el procesamiento se vea afectado. Puedes solicitar un aumento por encima de este límite poniéndote en contacto con AWS Support.
Nombre de la cola	La cola FIFO debe finalizar con el sufijo `.fifo`. El sufijo cuenta para la cuota de nombre de cola de 80 caracteres. Para determinar si una cola es FIFO, puede comprobar si el nombre de la cola termina con el sufijo.
Etiqueta de la cola	No recomendamos agregar más de 50 etiquetas a una cola. El etiquetado admite caracteres Unicode en UTF-8.
	Se requiere la etiqueta `Key`, pero la etiqueta `Value` es opcional.
	La etiqueta `Key` y la etiqueta `Value` distinguen entre mayúsculas y minúsculas.
	La pestaña `Key` y la pestaña `Value` pueden incluir caracteres alfanuméricos en UTF-8 y espacios en blanco. Se permiten los siguientes caracteres especiales: `_ . : / = + - @`
	La etiqueta `Key` o `Value` no debe incluir el prefijo reservado `aws:` (con este prefijo, no puede eliminar las claves o los valores de una etiqueta).
	La longitud máxima de la etiqueta `Key` es de 128 caracteres Unicode en UTF-8. La etiqueta `Key` no debe estar vacía o ser nula.
	La longitud máxima de la etiqueta `Value` es de 256 caracteres Unicode en UTF-8. La etiqueta `Value` puede estar vacía o ser nula.

Cuota	Descripción
	Las acciones de etiquetado están limitadas a 30 TPS cada una. Cuenta de AWS Si su aplicación requiere un mayor rendimiento, envíe una solicitud.

Cuotas de colas estándar de Amazon SQS

En la siguiente tabla se muestran las cuotas relacionadas con las colas estándar.

Cuota	Descripción
Cola con retraso	El retraso predeterminado (mínimo) de una cola es de 0 segundos. El valor máximo es de 15 minutos.
Colas mostradas	1000 colas por cada solicitud de `ListQueues` .
Tiempo de espera de sondeo largo	El tiempo máximo de espera de sondeo es de 20 segundos.
Mensajes por cola (pendientes)	El número de mensajes que puede almacenar una cola de Amazon SQS es ilimitado.
Mensajes por cola (en tránsito)	En la mayoría de las colas estándar (según el tráfico de colas y los mensajes atrasados), puede haber un máximo de aproximadamente 120 000 mensajes en tránsito (recibidos de una cola por un consumidor, pero que aún no se han eliminado de la cola). Si alcanza esta cuota mientras utiliza sondeos cortos, Amazon SQS devuelve el mensaje de error `OverLimit` . Si utiliza sondeos largos, Amazon SQS no devuelve ningún mensaje de error. Para evitar llegar a la cuota, conviene eliminar los mensajes de la cola una vez procesados. También puede aumentar el número de las colas que usa para procesar los mensajes. Para solicitar un aumento de la cuota, envíe una solicitud de soporte técnico.

Cuota	Descripción
Nombre de la cola	El nombre de una cola puede tener hasta 80 caracteres. Se aceptan los siguientes caracteres: caracteres alfanuméricos, guiones (`-`) y guiones bajos (`_`). **Note** Los nombres de cola distinguen entre mayúsculas y minúsculas (por ejemplo, `Test-queue` y `test-queue` son colas diferentes).
Etiqueta de la cola	No recomendamos agregar más de 50 etiquetas a una cola. El etiquetado admite caracteres Unicode en UTF-8.
	Se requiere la etiqueta `Key`, pero la etiqueta `Value` es opcional.
	La etiqueta `Key` y la etiqueta `Value` distinguen entre mayúsculas y minúsculas.
	La pestaña `Key` y la pestaña `Value` pueden incluir caracteres alfanuméricos en UTF-8 y espacios en blanco. Se permiten los siguientes caracteres especiales: `_ . : / = + - @`
	La etiqueta `Key` o `Value` no debe incluir el prefijo reservado `aws:` (con este prefijo, no puede eliminar las claves o los valores de una etiqueta).
	La longitud máxima de la etiqueta `Key` es de 128 caracteres Unicode en UTF-8. La etiqueta `Key` no debe estar vacía o ser nula.
	La longitud máxima de la etiqueta `Value` es de 256 caracteres Unicode en UTF-8. La etiqueta `Value` puede estar vacía o ser nula.

Cuota	Descripción
	Las acciones de etiquetado están limitadas a 30 TPS cada una. Cuenta de AWS Si su aplicación requiere un mayor rendimiento, envíe una solicitud.

Cuotas de mensajes de Amazon SQS

En la siguiente tabla se muestran las cuotas relacionadas con los mensajes.

Cuota	Descripción
ID de mensaje por lotes	Un ID de mensaje por lotes puede tener hasta 80 caracteres. Se aceptan los siguientes caracteres: caracteres alfanuméricos, guiones (`-`) y guiones bajos (`_`).
Atributos de mensajes	Un mensaje puede contener hasta 10 atributos de metadatos.
Lote de mensajes	Una única solicitud por lotes de mensajes puede incluir un máximo de 10 mensajes. Para obtener más informaci ón, consulte Configuración de Amazon SQSBuffered AsyncClient en la sección Acciones por lotes de Amazon SQS.
Contenido de los mensajes	Un mensaje solo puede incluir XML, JSON y texto sin formato. Se permiten los siguientes caracteres Unicode: `#x9` | `#xA` | `#xD` | `#x20` a `#xD7FF` | `#xE000` a `#xFFFD` | `#x10000` a `#x10FFFF` Cualquier carácter que no esté incluido en esta lista se rechazará. Para obtener más información, consulte la especificación W3C respecto a los caracteres.
ID de grupo de mensajes	`MessageGroupId` es obligatorio para las colas FIFO. Si no proporciona una `MessageGroupId` al enviar

Cuota	Descripción
	un mensaje a una cola FIFO, la acción fallará. En las colas estándar, el uso `MessageGroupId` permite establecer colas justas. Te recomendamos que incluyas una `MessageGroupId` en todos los mensajes cuando utilices colas justas. La longitud de `MessageGroupId` es de 128 caractere s. Valores válidos: caracteres alfanuméricos y signos de puntuación `(!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)` .
Retención de mensajes	De forma predeterminada, un mensaje se conserva durante 4 días. El mínimo es 60 segundos (1 minuto). El máximo es 1 209 600 segundos (14 días).
Capacidad de procesamiento de mensajes	Colas estándar Las colas estándar admiten un número muy alto, casi ilimitado de llamadas a la API por segundo, por acción (`SendMessage`, `ReceiveMessage` o `DeleteMessage`). Este alto rendimiento las hace ideales para casos de uso que requieren procesar grandes volúmenes de mensajes con rapidez, como el flujo de datos en tiempo real o las aplicaciones a gran escala. Si bien las colas estándar se escalan automáticamente en función de la demanda, es esencial supervisar los patrones de uso para garantizar un rendimiento óptimo, especialm ente en las regiones con mayores cargas de trabajo.

Cuota	Descripción
	Colas FIFO • Cada partición de una cola FIFO está limitada a 300 transacciones por segundo por acción de la API (`SendMessage` , `ReceiveMessage` , y `DeleteMessage`). Este límite se aplica específicamente al modo de rendimiento no alto. Al cambiar al modo de alto rendimiento, puede superar este límite predeterminado. Para activar el modo de alto rendimiento, consulte Habilitación del alto rendimiento para las colas FIFO en Amazon SQS. • Si utiliza procesamiento por lotes, las colas FIFO de rendimiento no alto admiten hasta 3000 mensajes por segundo y por método de la API (`SendMessage` , `ReceiveMessage` y `DeleteMessage`). Los 3000 mensajes por segundo representan 300 llamadas a la API, cada una con un lote de 10 mensajes. Alto rendimiento para las colas FIFO Los límites de FIFO de Amazon SQS se basan en el número de solicitudes de API, no en los límites de mensajes. Para el modo de alto rendimiento, estos límites de solicitudes de API son los siguientes: Límites de rendimiento de las transacciones (llamadas a la API sin procesamiento por lotes) Estos límites definen la frecuencia con la que cada operación de la API (por ejemplo SendMessageReceiveMessage, o DeleteMessage) se puede realizar de forma independiente, lo que garantiza un rendimiento eficiente del sistema dentro de las transacciones por segundo (TPS) permitidas.

Cuota	Descripción
	Los siguientes límites se basan en las llamadas a la API no agrupadas en lotes: • Este de EE. UU. (Norte de Virginia), Oeste de EE. UU. (Oregón) y Europa (Irlanda): hasta 70 000 transacci ones por segundo (TPS). • Este de EE. UU. (Ohio) y Europa (Fráncfort): hasta 19 000 TPS. • Asia-Pacífico (Bombay), Asia-Pacífico (Sídney), Asia-Pacífico (Singapur) y Asia-Pacífico (Tokio): hasta 9000 TPS. • Europa (Londres) y América del Sur (São Paulo): hasta 4500 TPS. • Todos los demás Regiones de AWS: rendimiento predeterminado de 2400 TPS. Maximizar el rendimiento con el procesamiento por lotes Procese varios mensajes en una sola llamada a la API, lo que aumenta considerablemente la eficiencia. En lugar de administrar por lotes, puede enviar, recibir o eliminar hasta diez mensajes en una sola solicitud de la API. Esto reduce el número total de llamadas a la API, lo que le permite procesar más mensajes por segundo y, al mismo tiempo, mantenerse dentro de los límites de transacciones (TPS) de la región, lo que maximiza el rendimiento y el rendimiento del sistema. Para obtener más información, consulte Aumento del rendimiento mediante el escalado horizontal y agrupación en lotes de acciones con Amazon SQS. Los siguientes límites se basan en las llamadas a la API agrupadas en lotes:

Cuota	Descripción
	• Este de EE. UU. (Norte de Virginia), Oeste de EE. UU. (Oregón) y Europa (Irlanda): hasta 700 000 mensajes por segundo (diez veces el límite de 70 000 TPS sin lotes). • Este de EE. UU. (Ohio) y Europa (Fráncfort): hasta 190 000 mensajes por segundo. • Asia-Pacífico (Bombay), Asia-Pacífico (Sídney), Asia-Pacífico (Singapur) y Asia-Pacífico (Tokio): hasta 90 000 mensajes por segundo. • Europa (Londres) y América del Sur (São Paulo): hasta 45 000 mensajes por segundo. • Todos los demás Regiones de AWS: hasta 24 000 mensajes por segundo. Optimización del rendimiento más allá del procesamiento por lotes Si bien el procesamiento por lotes puede aumentar considerablemente el rendimiento, es importante considerar otras estrategias para optimizar el rendimiento de FIFO: • Distribuya los mensajes en varios grupos de mensajes IDs: dado que los mensajes de un solo grupo se procesan de forma secuencial, distribuir la carga de trabajo entre varios grupos de mensajes permite un mejor paralelismo y un mayor rendimiento general. Para obtener más información, consulte Particiones y distribución de datos para obtener alto rendimiento en las colas FIFO de SQS. • Uso eficiente de las llamadas a la API: minimice las llamadas a la API innecesarias, como los cambios frecuentes de visibilidad o la eliminación repetida de

Cuota	Descripción
	mensajes, para optimizar el uso de TPS disponible y mejorar la eficiencia. • Utilice sondeos largos: utilice sondeos largos estableci endo `WaitTimeSeconds` en las solicitudes de recepción para reducir las respuestas vacías cuando no haya mensajes disponibles, reducir las llamadas innecesarias a la API y aprovechar mejor su cuota de TPS. • Solicite aumentos de rendimiento: si su aplicació n requiere un rendimiento superior a los límites predeterminados, solicite un aumento mediante la consola de Service Quotas. Esto puede ser necesario para cargas de trabajo de alta demanda o en regiones con límites predeterminados más bajos. Para activar el modo de alto rendimiento, consulte Habilitación del alto rendimiento para las colas FIFO en Amazon SQS.
Temporizador de mensajes	El retraso predeterminado (mínimo) de un mensaje es de 0 segundos. El valor máximo es de 15 minutos.
Tamaño del mensaje	El tamaño mínimo de los mensajes es de 1 byte (1 carácter). El máximo es de 1.048.576 bytes (1 MiB). Para enviar mensajes de más de 1 MiB, puede utilizar la biblioteca de clientes extendida de Amazon SQS para Java y la biblioteca de clientes extendida de Amazon SQS para Python. Esta biblioteca le permite enviar un mensaje de Amazon SQS que contiene una referencia a una carga de mensajes de Amazon S3. El tamaño de carga máximo es 2 GB. Note Esta biblioteca ampliada solo funciona para clientes síncronos.

Cuota	Descripción
Tiempo de espera de visibilidad de los mensajes	El tiempo de espera de visibilidad predeterminado de un mensaje es de 30 segundos. El mínimo es de 0 segundos. El máximo es de 12 horas.
Información de políticas	La cuota máxima es 8192 bytes, 20 instrucciones, 50 entidades principales o 10 condiciones. Para obtener más información, consulte Cuotas de políticas de Amazon SQS.

Cuotas de políticas de Amazon SQS

En la siguiente tabla se muestran las cuotas relacionadas con las políticas.

Nombre	Máximo
Bytes	8 192
Condiciones	10
Entidades principales	50
Instrucciones	20
Acciones por instrucción	7

Características y capacidades de Amazon SQS

En este tema se proporcionan las características más utilizadas en Amazon SQS para administrar las colas de mensajes, optimizar el rendimiento, garantizar una entrega fiable de los mensajes y administrar el procesamiento de los mensajes de forma eficiente.

Uso de colas de mensajes fallidos en Amazon SQS

Amazon SQS admite colas de letra muerta (DLQs), a las que pueden dirigirse las colas de origen para los mensajes que no se procesan correctamente. DLQs son útiles para depurar la aplicación porque permiten aislar los mensajes no consumidos para determinar por qué el procesamiento no se ha realizado correctamente. Para obtener un rendimiento óptimo, se recomienda mantener la cola de origen y la cola de mensajes fallidos dentro de la misma región y Cuenta de AWS . Una vez que los mensajes están en cola de mensajes fallidos, puede:

- Examinar los registros para ver si hay excepciones que podrían haber causado el traslado de mensajes a una cola de mensajes fallidos.
- Analizar el contenido de los mensajes movidos a la cola de mensajes fallidos para diagnosticar problemas de aplicaciones.
- Determinar si ha concedido al consumidor tiempo suficiente para procesar los mensajes.
- Sacar los mensajes de la cola de mensajes fallidos mediante la redirección de la cola de mensajes fallidos.

Primero deberá crear una nueva cola antes de configurarla como una cola de mensajes fallidos. Para obtener información sobre la configuración de una cola de mensajes fallidos mediante la consola de Amazon SQS, consulte Configure una cola de cartas sin salida mediante la consola Amazon SQS. Para obtener ayuda con las colas de mensajes fallidos, por ejemplo, cómo configurar una alarma para todos los mensajes que se mueven a una cola de mensajes fallidos, consulte Creación de alarmas para colas de cartas sin salida con Amazon CloudWatch.

> **Note**
>
> No utilice una cola de mensajes fallidos con una cola FIFO si no desea interrumpir el orden exacto de los mensajes o las operaciones. Por ejemplo, no utilice una cola de mensajes fallidos con instrucciones de una lista de decisiones de edición (EDL) para un programa de

edición de vídeo, donde al cambiar el orden de las ediciones se cambia el contexto de las ediciones posteriores.

Uso de políticas para colas de mensajes fallidos

Utilice una política de redirección para especificar el `maxReceiveCount`. `maxReceiveCount` es el número de veces que un consumidor puede recibir un mensaje de una cola de origen antes de que se traslade a la cola de mensajes fallidos. Por ejemplo, si `maxReceiveCount` se establece en un valor bajo, como 1, cualquier error en la recepción de un mensaje hará que este se traslade a la cola de mensajes fallidos. Para asegurarse de que su sistema es resiliente frente a los errores, establezca `maxReceiveCount` lo suficientemente alto como para permitir un número suficiente de reintentos.

La política de permiso de redireccionamiento especifica qué colas de origen pueden acceder a la cola de mensajes fallidos. Puede elegir entre permitir todas las colas de origen, permitir colas de origen específicas o denegar que todas las colas de origen usen la cola de mensajes fallidos. De forma predeterminada, se permite que todas las colas de origen utilicen la cola de mensajes fallidos. Si elige permitir colas específicas mediante la opción `byQueue`, puede especificar hasta diez colas de origen con el nombre de recurso de Amazon (ARN) de la cola de origen. Si especifica `denyAll`, la cola no se puede utilizar como una cola de mensajes fallidos.

Descripción de los períodos de retención de mensajes en la cola de mensajes fallidos

En el caso de las colas estándar, la caducidad de un mensaje siempre se basa en su marca temporal original. Cuando un mensaje se mueve a una cola de mensajes fallidos, la marca temporal de la cola no se modifica. La métrica `ApproximateAgeOfOldestMessage` indica cuándo el mensaje pasó a la cola de mensajes fallidos, no cuándo se envió originalmente. Por ejemplo, supongamos que un mensaje pasa un día en la cola original antes de ser trasladado a una cola de mensajes fallidos. Si el periodo de retención de la cola de mensajes fallidos es de cuatro días, el mensaje se elimina de la cola de mensajes fallidos al cabo de tres días y `ApproximateAgeOfOldestMessage` es de tres días. Por lo tanto, se recomienda establecer siempre un periodo de retención de una cola de mensajes fallidos superior al periodo de retención de la cola original.

Para las colas FIFO, la marca temporal de entrada se restablece cuando el mensaje se mueve a una cola de mensajes fallidos. La métrica `ApproximateAgeOfOldestMessage` indica cuándo el mensaje ha pasado a la cola de mensajes fallidos. En el mismo ejemplo anterior, el mensaje

se elimina de la lista de mensajes sin salida al cabo de cuatro días y después de cuatro días. `ApproximateAgeOfOldestMessage`

Configure una cola de cartas sin salida mediante la consola Amazon SQS

Una cola de mensajes sin salida (DLQ) es una cola que recibe mensajes que no se han procesado correctamente desde otra cola, conocida como cola de origen. Amazon SQS no crea la cola de mensajes fallidos automáticamente. Primero deberá crear la cola antes de utilizarla como una cola de mensajes fallidos. Al configurar una DLQ, el tipo de cola debe coincidir con el tipo de cola de origen: una cola FIFO solo puede usar una DLQ FIFO y una cola estándar solo puede usar una DLQ estándar. Puede configurar una cola de letra muerta al crear o editar una cola. Para obtener más información, consulta Uso de colas de mensajes fallidos en Amazon SQS .

Configuración de una cola de mensajes fallidos para una cola existente (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. Seleccione la cola de origen (la cola que enviará los mensajes fallidos a la cola de letra muerta) y, a continuación, seleccione Editar.
4. Desplázate hasta la sección de colas de cartas sin procesar y activa la opción Activar.
5. En Configuración de colas de cartas sin salida, elige el nombre de recurso de Amazon (ARN) de una cola existente que quieras usar como cola de cartas sin salida.
6. Establezca el valor máximo de recepción, que define cuántas veces se puede recibir un mensaje antes de enviarlo a la cola de mensajes sin efecto (rango válido: de 1 a 1000).
7. Seleccione Save.

Obtener información sobre cómo configurar un redireccionamiento de cola de mensajes fallidos en Amazon SQS

Utilice el redireccionamiento de la cola de cartas sin procesar para mover los mensajes no consumidos de una cola de cartas sin salida a otro destino para su procesamiento. De forma predeterminada, el redireccionamiento de cola de mensajes fallidos mueve los mensajes de una cola de mensajes fallidos a una cola de origen. No obstante, también puede configurar cualquier otra cola estándar como destino de redireccionamiento si las colas son del mismo tipo. Por ejemplo, si la cola de mensajes fallidos es una cola FIFO, la cola de destino de redireccionamiento también debe ser

una cola FIFO. Además, puede configurar la velocidad de redireccionamiento para establecer a qué velocidad Amazon SQS mueve los mensajes.

> **Note**
>
> Cuando un mensaje pasa de una cola FIFO a una DLQ FIFO, el ID de desduplicación del mensaje original se reemplazará por el ID del mensaje original. Esto es para asegurarse de que la desduplicación de DLQ no impedirá el almacenamiento de dos mensajes independientes que casualmente comparten un ID de desduplicación.

Las colas de mensajes fallidos redirigen los mensajes en el orden en que se reciben, empezando por el mensaje más antiguo. Sin embargo, la cola de destino ingiere los mensajes redirigidos, así como los mensajes nuevos de otros productores, según el orden en que los recibe. Por ejemplo, si un productor envía mensajes a una cola FIFO de origen cuando de forma simultánea recibe mensajes redirigidos de una cola de mensajes fallidos, los mensajes redirigidos se entrelazarán con los nuevos mensajes del productor.

> **Note**
>
> La tarea de redireccionamiento restablece el periodo de retención. Todos los mensajes redirigidos se consideran mensajes nuevos con un `messageID` y un `enqueueTime` nuevos y se asignan a mensajes redirigidos.

Configuración de un redireccionamiento de cola de mensajes fallidos para una cola estándar existente mediante la API de Amazon SQS

Puede configurar un redireccionamiento cola de mensajes fallidos con las acciones de la API `StartMessageMoveTask`, `ListMessageMoveTasks` y `CancelMessageMoveTask`:

Acción de la API	Descripción
`StartMessageMoveTask`	Inicia una tarea asincrónica para mover mensajes de una cola de origen especificada a una cola de destino especificada.

Acción de la API	Descripción
`ListMessageMoveTasks`	Obtiene las tareas de movimiento de mensajes más recientes (hasta diez) en una cola de origen específica.
`CancelMessageMoveTask`	Cancela una tarea de movimiento de mensajes especificada. Un movimiento de mensajes solo puede cancelarse cuando el estado actual es EN EJECUCIÓN.

Configuración de un redireccionamiento de cola de mensajes fallidos para una cola estándar mediante la consola de Amazon SQS

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. Elija el nombre de la cola que haya configurado como una cola de mensajes fallidos.
4. Seleccione Iniciar el redireccionamiento de la DLQ.
5. En Redireccionar configuración, para Destino del mensaje, realice una de las siguientes acciones:
   - Para redireccionar mensajes a su cola de origen, seleccione Redireccionar a las colas de origen.
   - Para redireccionar mensajes a otra cola, elija Redireccionar a un destino personalizado. A continuación, escriba el nombre de recurso de Amazon (ARN) de una cola de destino existente.
6. En Configuración de control de velocidad, elija una de las siguientes opciones:
   - Sistema optimizado: redireccionar la cola de mensajes fallidos al número máximo de mensajes por segundo.
   - Velocidad máxima personalizada: redireccionar mensajes de la cola de mensajes fallidos con una velocidad máxima personalizada de mensajes por segundo. La velocidad máxima permitida es de 500 mensajes por segundo.

- Se recomienda empezar con un valor pequeño para la velocidad máxima personalizada y comprobar que la cola de origen no se satura de mensajes. A partir de ahí, aumente gradualmente el valor de la velocidad máxima personalizada, sin dejar de monitorear el estado de la cola de origen.

7. Cuando termine de configurar el redireccionamiento de la cola de mensajes fallidos, elija Redireccionar mensajes.

> **Important**
>
> Amazon SQS no admite el filtrado y la modificación de mensajes mientras los redirecciona desde la cola de mensajes fallidos.
> Una tarea de redireccionamiento de cola de mensajes fallidos puede ejecutarse un máximo de 36 horas. Amazon SQS admite un máximo de 100 tareas de redireccionamiento activas por cuenta.

8. Si desea cancelar la tarea de redireccionamiento de mensajes, en la página Detalles de la cola, elija Cancelar redireccionamiento de DLQ. Al cancelar un redireccionamiento de mensajes en curso, los mensajes que ya se hayan movido correctamente a su cola de destino de movimiento permanecerán en ella.

Configuración de los permisos del redireccionamiento de cola de mensajes fallidos

Puede conceder al usuario acceso a acciones específicas de la cola de mensajes fallidos si agrega permisos a la política. Los permisos mínimos necesarios para un redireccionamiento de cola de mensajes fallidos son los siguientes:

Permisos mínimos	Métodos de API necesarios
Inicio de un redirecci onamiento de mensajes	• Agregue `sqs:StartMessageMoveTask`, `sqs:ReceiveMessage`, `sqs:DeleteMessage` y `sqs:GetQueueAttributes` de la cola de mensajes fallidos. Si la cola de mensajes fallidos o la cola de origen están cifradas (conocida como cola SSE), también se necesita `kms:Decrypt` para cualquier clave de KMS que se haya utilizado para cifrar los mensajes. •

Permisos mínimos	Métodos de API necesarios
	Agregue `sqs:SendMessage` de la cola de destino. Si la cola de destino está cifrada, también se requieren `kms:GenerateDataKey` y `kms:Decrypt`.
Cancelación de un redireccionamiento de mensajes en curso	• Agregue `sqs:CancelMessageMoveTask`, `sqs:ReceiveMessage`, `sqs:DeleteMessage` y `sqs:GetQueueAttributes` de la cola de mensajes fallidos. Si la cola de mensajes fallidos está cifrada (conocida como cola SSE), también se requiere `kms:Decrypt`.
Visualización del estado de movimiento de mensajes	• Agregue `sqs:ListMessageMoveTasks` y `sqs:GetQueueAttributes` de la cola de mensajes fallidos.

Configuración de los permisos de un par de colas cifradas (una cola de origen con una cola de mensajes fallidos)

Siga los siguientes pasos para configurar los permisos mínimos para una reunidad de cola de cartas muertas (DLQ):

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, seleccione Policies (Políticas).
3. Cree una nueva política y añada los siguientes permisos. Adjunte la política al usuario o rol de IAM que realizará la operación de redrive.
   - Permisos para la DLQ (cola de fuentes):
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:GetQueueAttributes`

  - `sqs:ListDeadLetterSourceQueues`
  - Especifique el ARN del recurso del DLQ (cola de fuentes) (por ejemplo, «arn:aws:sqs::: «). *<DLQ_region> <DLQ_accountId> <DLQ_name>*
- Permisos para la cola de destino:
  - `sqs:SendMessage`
  - Especifique la cola `Resource` ARN de destino (por ejemplo, «arn:aws:sqs: «). *<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*
- Permisos para las claves KMS:
  - `kms:Decrypt`(Necesario para descifrar los mensajes del DLQ).
  - `kms:GenerateDataKey`(Necesario para cifrar los mensajes de la cola de destino).
    - `Resource` ARNs:
      - El ARN de la clave KMS utilizada para cifrar los mensajes de la DLQ (cola de origen) (por ejemplo, «arn:aws:kms: ::key/ «). *<region> <accountId> <SourceQueueKeyId>*
      - El ARN de la clave KMS utilizada para cifrar los mensajes de la cola de destino (por ejemplo, «arn:aws:kms: ::key/ «). *<region> <accountId> <DestinationQueueKeyId>*

Su política de acceso debe ser similar a la siguiente:

JSON

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

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

Configuración de los permisos mediante un par de colas no cifradas (una cola de origen con una cola de mensajes fallidos)

Siga estos pasos para configurar los permisos mínimos necesarios para gestionar una cola de cartas sin procesar (DLQ) estándar y sin cifrar. Los permisos mínimos requeridos son recibir, eliminar y obtener atributos de la cola de mensajes fallidos, y enviar atributos a la cola de origen.

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, seleccione Policies (Políticas).

3. Cree una nueva política y añada los siguientes permisos. Adjunte la política al usuario o rol de IAM que realizará la operación de redrive.

   - Permisos para la DLQ (cola de fuentes):
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:ListDeadLetterSourceQueues`
     - Especifique el ARN del recurso del DLQ (cola de fuentes) (por ejemplo, «arn:aws:sqs::: «). *<DLQ_region> <DLQ_accountId> <DLQ_name>*
   - Permisos para la cola de destino:
     - `sqs:SendMessage`
     - Especifique la cola `Resource` ARN de destino (por ejemplo, «arn:aws:sqs: «). *<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*

   Su política de acceso debe ser similar a la siguiente:

   JSON

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

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

Uso del redrive de colas de letra muerta con control de acceso a puntos finales de VPC

Si restringes el acceso a las colas a un VPCs uso específico de la `aws:sourceVpc` condición, debes hacer una excepción en el caso de los AWS servicios para habilitar la funcionalidad de redireccionamiento de colas con letra muerta (DLQ). Esto se debe a que el servicio Amazon SQS funciona fuera de la VPC al mover los mensajes.

Para permitir las operaciones de redrive de DLQ, añada la `aws:CalledViaLast` condición a su política de colas. Esto permite a Amazon SQS realizar llamadas a la API en su nombre y, al mismo tiempo, mantener las restricciones de VPC para el acceso directo.

Para permitir tanto el acceso restringido por VPC como la retransmisión de DLQ:

1. Usa la `aws:CalledViaLast` condición en tu política de colas.
2. Aplique la política tanto a la cola de origen como a la DLQ
3. Mantenga las restricciones de VPC para el acceso directo desde otras fuentes

A continuación, se muestra un ejemplo de política que implementa estos requisitos:

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

- Sustituya los valores de los marcadores de posición por sus valores reales
- Esta política utiliza una declaración de «denegar» con condiciones, que es más segura que utilizar declaraciones de «permitir»

- El `StringNotEqualsIfExists` operador gestiona los casos en los que la clave de condición podría no estar presente en el contexto de la solicitud.

Como alternativa, puedes usar la clave de `aws:ViaAWSService` condición para permitir el acceso basado en el servicio y, al mismo tiempo, mantener las restricciones de VPC. Esta clave de condición indica si la solicitud proviene de un AWS servicio. A continuación, se muestra un ejemplo de política que utiliza `aws:ViaAWSService` en lugar de`aws:CalledViaLast`:

```
{
  "Version": "2012-10-17",
  "Id": "SQSRedriveWithVpcRestriction",
  "Statement": [
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DestQueue",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:DestQueue",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    },
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DLQ",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:Dlq",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    }
```

```
  ]
}
```

El BoolIfExists operador con la `aws:ViaAWSService` condición garantiza que se permitan las solicitudes cuando provienen de servicios y, al mismo tiempo, mantiene las restricciones de VPC para el acceso directo. Esto puede ser más fácil de entender y mantener, ya que comprueba directamente si la solicitud la ha realizado un AWS servicio en lugar de comprobar qué servicio realizó la última llamada.

Para obtener más información sobre las claves de condición utilizadas en las políticas de recursos y de IAM, consulte Elementos de la política JSON de IAM: condición.

CloudTrail requisitos de actualización y permiso para la reactivación de colas de cartas muertas de Amazon SQS

El 8 de junio de 2023, Amazon SQS introdujo el redrive de cola de letras muertas (DLQ) para AWS el SDK y (CLI). AWS Command Line Interface Esta capacidad se suma al redrive DLQ ya compatible con la consola. AWS Si ya utilizaste la AWS consola para reimprimir los mensajes en cola con letra muerta, es posible que te afecten los siguientes cambios:

CloudTrail cambio de nombre de eventos

El 15 de octubre de 2023, cambiarán los nombres de los CloudTrail eventos de redrive de colas con letra muerta en la consola Amazon SQS. Si ha configurado alarmas para estos CloudTrail eventos, debe actualizarlas ahora. Los nuevos nombres de CloudTrail eventos de DLQ redrive son los siguientes:

Nombre de evento anterior	Nombre de evento nuevo
`CreateMoveTask`	`StartMessageMoveTask`
`CancelMoveTask`	`CancelMessageMoveTask`

Permisos actualizados

Incluido con el lanzamiento del SDK y CLI, Amazon SQS también ha actualizado los permisos de cola para el redireccionamiento de DLQ a fin de cumplir con las prácticas recomendadas de seguridad. Utilice los siguientes tipos de permisos de cola para redirigir los mensajes de su. DLQs

1. Permisos basados en acciones (actualización de las acciones de la API de DLQ)
2. Permisos de política de Amazon SQS
3. Política de permisos que utiliza caracteres comodín sqs:*

> **Important**
>
> Para utilizar el redireccionamiento de DLQ para SDK o CLI, es necesario disponer de una política de permisos de redireccionamiento de DLQ que coincida con una de las opciones anteriores.

Si sus permisos de cola para el redireccionamiento de DLQ no coinciden con alguna de las opciones anteriores, deberá actualizarlos antes del 31 de agosto de 2023. Desde ahora hasta el 31 de agosto de 2023, su cuenta podrá redireccionar mensajes con los permisos que haya configurado a través de la consola de AWS solo en las regiones en las que haya utilizado anteriormente el redireccionamiento de DLQ. Por ejemplo, supongamos que tiene “Cuenta A” tanto en us-east-1 como en eu-west-1. La «Cuenta A» se utilizó para redirigir los mensajes de la AWS consola en us-east-1 antes del 8 de junio de 2023, pero no en eu-west-1. Entre el 8 de junio de 2023 y el 31 de agosto de 2023, si los permisos de la política de la «Cuenta A» no coinciden con una de las opciones anteriores, solo se pueden usar para redirigir los mensajes de la AWS consola en us-east-1 y no en eu-west-1.

> **Important**
>
> Si sus permisos de redireccionamiento de DLQ no coinciden con una de estas opciones después del 31 de agosto de 2023, su cuenta ya no podrá redireccionar mensajes de DLQ mediante la consola de AWS .
> Sin embargo, si utilizaste la función de reaccionamiento de DLQ en la AWS consola durante agosto de 2023, dispones de una prórroga hasta el 15 de octubre de 2023 para adoptar los nuevos permisos según una de estas opciones.
> Para obtener más información, consulte the section called “Identificación de las políticas afectadas”.

A continuación, se muestran ejemplos de permisos de cola para cada opción de redireccionamiento de DLQ. Cuando se utilizan colas cifradas del lado del servidor (SSE), se requiere el permiso de clave correspondiente AWS KMS .

Basado en acciones

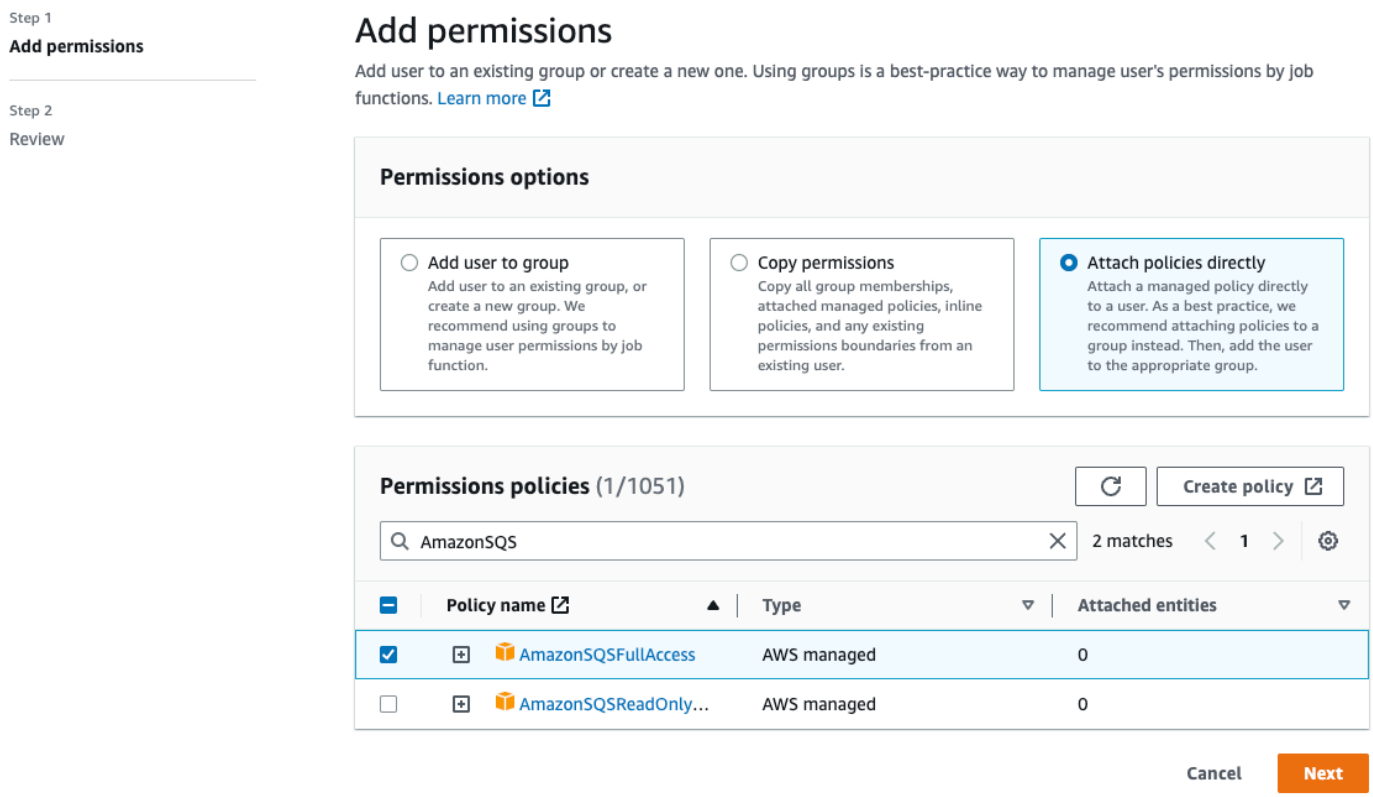
JSON

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

Política administrada

Las siguientes políticas administradas contienen los permisos actualizados necesarios:

- Amazon SQSFull Access: incluye las siguientes tareas de reactivación de colas con letra muerta: iniciar, cancelar y publicar listas.
- Amazon SQSRead OnlyAccess: proporciona acceso de solo lectura e incluye la tarea de reconducir la lista de colas de letras muertas.



Política de permisos que utiliza caracteres comodín sqs*

JSON

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

Identificación de las políticas afectadas

Si utilizas políticas gestionadas por el cliente (CMPs), puedes utilizar un IAM para identificar las políticas AWS CloudTrail afectadas por la actualización de los permisos de las colas.

> **Note**
>
> Si utiliza `AmazonSQSFullAccess` y `AmazonSQSReadOnlyAccess`, no es necesario realizar ninguna otra acción.

1. Inicie sesión en la AWS CloudTrail consola.
2. En la página Historial de eventos, en Buscar atributos, utilice el menú desplegable para seleccionar Nombre del evento. A continuación, busque `CreateMoveTask`.
3. Elija un evento para abrir la página Detalles. En la sección Registros de eventos, recupere `UserName` o `RoleName` del ARN de `userIdentity`.
4. Inicie sesión en la consola de IAM.
   - Para usuarios, seleccione Usuarios. Seleccione el usuario con el `UserName` identificado en el paso anterior.
   - Para roles, seleccione Roles. Busque el usuario con el `RoleName` identificado en el paso anterior.
5. En la página Detalles, en la sección Permisos, revise las políticas que tengan el prefijo `sqs:` en `Action` o revise las políticas que tengan la cola de Amazon SQS definida en `Resource`.

Creación de alarmas para colas de cartas sin salida con Amazon CloudWatch

Configura una CloudWatch alarma para monitorizar los mensajes en una cola de letra muerta utilizando la métrica. `ApproximateNumberOfMessagesVisible` Para obtener instrucciones detalladas, consulta Creación de CloudWatch alarmas para las métricas de Amazon SQS. Cuando se active la alarma, lo que indica que los mensajes se han trasladado a la cola de mensajes sin efecto, puedes sondear la cola para revisarlos y recuperarlos.

Metadatos de mensajes para Amazon SQS

Utilice los atributos de los mensajes para añadir metadatos personalizados a los mensajes de Amazon SQS para sus aplicaciones. Utilice los atributos del sistema de mensajes para almacenar los metadatos y poder integrarlos con otros Servicios de AWS, por ejemplo. AWS X-Ray

Atributos de mensajes de Amazon SQS

Amazon SQS le permite incluir metadatos estructurados (como marcas de tiempo, datos geoespaciales, firmas e identificadores) en los mensajes mediante atributos de mensaje. Cada mensaje puede tener hasta 10 atributos. Los atributos de los mensajes son opcionales y están separados del cuerpo del mensaje (sin embargo, se envían con él). El consumidor puede utilizar los atributos de mensajes para controlar un mensaje de una forma concreta sin tener que procesar el cuerpo del mensaje en primer lugar. Para obtener información sobre cómo enviar mensajes con atributos mediante la consola de Amazon SQS, consulte Envío de un mensaje con atributos mediante Amazon SQS.

> **Note**
>
> No confunda los atributos de los mensajes con los atributos del sistema de mensajes: si bien puede usar los atributos de los mensajes para adjuntar metadatos personalizados a los mensajes de Amazon SQS para sus aplicaciones, puede usar los atributos del sistema de mensajes para almacenar metadatos para otros AWS servicios, como. AWS X-Ray

Temas

- Componentes de atributos de mensajes
- Tipos de datos de atributos de mensajes
- Calcular el resumen del MD5 mensaje para los atributos del mensaje

Componentes de atributos de mensajes

> **Important**
>
> Todos los componentes de un atributo de mensaje se incluyen en la restricción de tamaño de mensaje de 1 MiB.
> `Name`, `Type`, `Value` y el cuerpo del mensaje no deben estar vacíos ni ser null.

Cada atributo de mensaje consta de los siguientes componentes:

- Nombre: el nombre del atributo de mensaje puede contener los siguientes caracteres: `A-Z`, `a-z`, `0-9`, guion bajo (`_`), guion (`-`) y punto (`.`). Se aplican las siguientes restricciones:

  - Puede tener hasta 256 caracteres
  - No puede comenzar por `AWS.` o `Amazon.` (ni ninguna variación de mayúsculas y minúsculas)
  - Distingue entre mayúsculas y minúsculas
  - Debe ser único entre todos los nombres de atributo del mensaje
  - No debe comenzar ni terminar por un punto
  - No debe tener puntos seguidos
- Tipo: el tipo de datos de atributo de mensaje. Los tipos admitidos son `String`, `Number` y `Binary`. También puede agregar información personalizada para cualquier tipo de datos. El tipo de datos tiene las mismas restricciones que el cuerpo del mensaje (para obtener más información, consulte `SendMessage` en la Referencia de la API de Amazon Simple Queue Service). Además, se aplican las siguientes restricciones:
  - Puede tener hasta 256 caracteres
  - Distingue entre mayúsculas y minúsculas
- Valor: el valor del atributo de mensaje. Para los tipos de datos `String`, los valores de atributo tienen las mismas restricciones el cuerpo del mensaje.

Tipos de datos de atributos de mensajes

Los tipos de datos de atributo de mensaje indican a Amazon SQS cómo tratar los correspondientes valores de atributo de mensaje. Por ejemplo, si el tipo es `Number`, Amazon SQS valida valores numéricos.

Amazon SQS admite los tipos de datos lógicos `String`, `Number` y `Binary` con etiquetas de tipo de datos personalizadas opcionales con el formato `.custom-data-type`.

- Cadena: los atributos `String` pueden almacenar texto Unicode mediante cualquier carácter XML válido.
- Número: los atributos `Number` pueden almacenar valores numéricos positivos o negativos. Un número puede tener hasta 38 dígitos de precisión y puede estar comprendido entre 10^-128 y 10^+126.

> **Note**
>
> Amazon SQS quita los ceros al principio y al final.

- Binario: los atributos binarios permiten almacenar datos binarios de cualquier índole, como datos comprimidos, datos cifrados o imágenes.
- Personalizado: para crear un tipo de datos personalizado, añada la etiqueta custom-type a cualquier tipo de datos. Por ejemplo:
  - `Number.byte`, `Number.short`, `Number.int` y `Number.float` pueden ayudarle a distinguir entre tipos de números.
  - `Binary.gif` y `Binary.`png pueden ayudarle a distinguir entre tipos de archivos.

> **Note**
>
> Amazon SQS no interpreta, valida ni utiliza los datos añadidos.
> La etiqueta custom-type tiene las mismas restricciones que el cuerpo del mensaje.

Calcular el resumen del MD5 mensaje para los atributos del mensaje

Si utilizas el AWS SDK para Java, puedes saltarte esta sección. La `MessageMD5ChecksumHandler` clase del SDK para Java admite resúmenes de MD5 mensajes para los atributos de mensajes de Amazon SQS.

Si utiliza la API Query o una de las AWS SDKs que no admite resúmenes de MD5 mensajes para los atributos de mensajes de Amazon SQS, debe seguir las siguientes pautas para realizar MD5 el cálculo del resumen de mensajes.

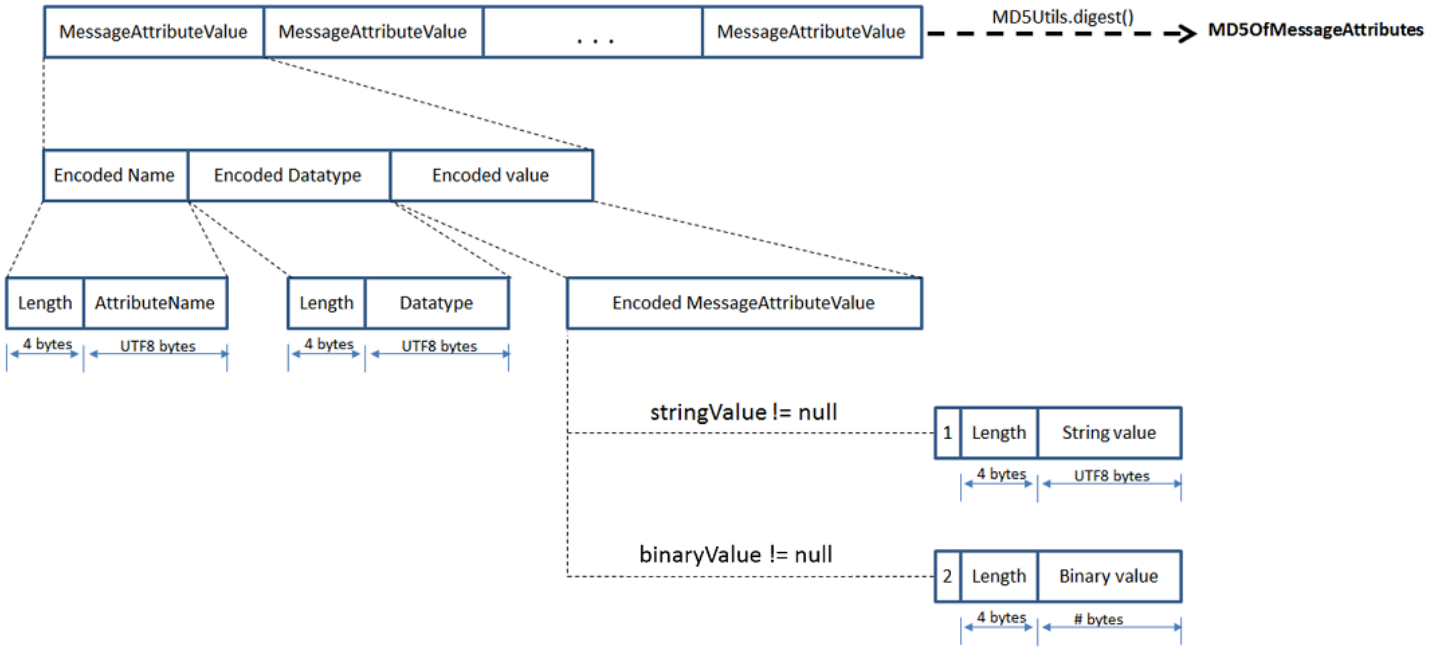
> **Note**
>
> Incluya siempre sufijos de tipos de datos personalizados en el cálculo del resumen de los mensajes MD5 .

Descripción general

A continuación se presenta una descripción general del algoritmo de cálculo del resumen de MD5 mensajes:

1. Ordenar todos los atributos de mensajes por nombre en orden ascendente.
2. Codificar las partes individuales de cada atributo (`Name`, `Type` y `Value`) en un búfer.
3. Calcular el resumen de mensaje de todo el búfer.

El siguiente diagrama muestra la codificación del resumen del MD5 mensaje para un único atributo del mensaje:



Codificación de un único atributo de mensaje de Amazon SQS

1. Codifique el nombre: la longitud (4 bytes) y los bytes UTF-8 del nombre.
2. Codifique el tipo de datos: la longitud (4 bytes) y los bytes UTF-8 del tipo de datos.
3. Codifique el tipo de transporte (`String` o `Binary`) del valor (1 byte).

> **Note**
>
> Los tipos de datos lógicos `String` y `Number` utilizan el tipo de transporte `String`. El tipo de datos lógicos `Binary` utiliza el tipo de transporte `Binary`.

   a. Para el tipo de transporte `String`, codifique 1.
   b. Para el tipo de transporte `Binary`, codifique 2.
4. Codifique el valor del atributo.
   a. Para el tipo de transporte `String`, codifique el valor del atributo: la longitud (4 bytes) y los bytes UTF-8 del valor.
   b. En el tipo de transporte `Binary`, codifique el valor del atributo: la longitud (4 bytes) y los bytes sin formato del valor.

Atributos del sistema de mensajes de Amazon SQS

Aunque puede utilizar atributos de mensajes para adjuntar metadatos personalizados a mensajes de Amazon SQS para sus aplicaciones, puede utilizar atributos del sistema de mensajes para almacenar metadatos para otros servicios de AWS , como AWS X-Ray. Para obtener más información, consulte el parámetro de solicitud `MessageSystemAttribute` de las acciones de la API `SendMessage` y `SendMessageBatch`, el atributo `AWSTraceHeader` de la acción de la API `ReceiveMessage` y el tipo de datos `MessageSystemAttributeValue` en la Referencia de la API de Amazon Simple Queue Service.

Los atributos del sistema de mensajes se estructuran exactamente igual que los atributos del mensaje, con las siguientes excepciones:

- En la actualidad, el único atributo del sistema de mensajes admitido es `AWSTraceHeader`. Su tipo `String` y su valor deben ser una cadena de encabezado de AWS X-Ray rastreo con el formato correcto.
- El tamaño de un atributo del sistema de mensajes no cuenta para el tamaño total de un mensaje.

Recursos necesarios para procesar mensajes de Amazon SQS

Amazon SQS proporciona estimaciones del número aproximado de mensajes retrasados, visibles y no visibles en una cola para ayudarle a evaluar los recursos necesarios para su procesamiento. Para obtener más información acerca de la visibilidad, consulte Tiempo de espera de visibilidad de Amazon SQS.

> **Note**
>
> En el caso de algunas métricas, el resultado es aproximado debido a la arquitectura distribuida de Amazon SQS. En la mayoría de los casos, el recuento debe aproximarse al número real de mensajes de la cola.

En la tabla siguiente se muestra el nombre de atributo que se utilizará con la acción `GetQueueAttributes`:

Tarea	Nombre de atributo
Obtenga el número aproximado de mensajes disponibles para recuperar de la cola.	`ApproximateNumberOfMessagesVisible`
Obtenga el número aproximado de mensajes de la cola que se retrasan y no están disponibles para su lectura inmediata. Esto puede ocurrir cuando la cola está configurada como una cola de retraso o cuando se ha enviado un mensaje con un parámetro de retraso.	`ApproximateNumberOfMessagesDelayed`
Obtenga el número aproximado de mensajes que se encuentran en tránsito. Se considera que los mensajes están en tránsito si se han enviado a un cliente pero aún no se han eliminado o aún no han llegado al final de su período de visibilidad.	`ApproximateNumberOfMessagesNotVisible`

Paginación de colas de listas de Amazon SQS

Los métodos de API `listQueues` y `listDeadLetterQueues` admiten controles de paginación opcionales. De forma predeterminada, estos métodos de API devuelven hasta 1000 colas en el mensaje de respuesta. Puede configurar el parámetro `MaxResults` para que devuelva menos resultados en cada respuesta.

Establezca el parámetro `MaxResults` en la solicitud `listQueues` o `listDeadLetterQueues` para especificar el número máximo de resultados que se deben devolver en la respuesta. Si no establece `MaxResults`, la respuesta incluye un máximo de 1000 resultados y el valor de `NextToken` en la respuesta es nulo.

Si establece `MaxResults`, la respuesta incluye un valor para `NextToken` si hay resultados adicionales que mostrar. Use `NextToken` como parámetro en su próxima solicitud a `listQueues` para recibir la siguiente página de resultados. Si no hay resultados adicionales para mostrar, el valor de `NextToken` de la respuesta es nulo.

Etiquetas de asignación de costos de Amazon SQS

Para organizar e identificar las colas de Amazon SQS para la asignación de costos, puede agregar etiquetas de metadatos que identifiquen el propósito, el propietario o el entorno de una cola. Esto es útil especialmente cuando dispone de muchas colas. Para configurar las etiquetas con la consola de Amazon SQS, consulte the section called “Configuración de etiquetas para una cola”

Puede usar etiquetas de asignación de costos para organizar su AWS factura y reflejar su propia estructura de costos. Para ello, inscríbase para que su Cuenta de AWS factura incluya claves de etiquetas y valores. Para obtener más información, consulte Configuración de un informe de asignación de costos mensual en la Guía del usuario de AWS Billing .

Cada etiqueta consta de un par clave-valor, que usted define. Por ejemplo, puede identificar fácilmente sus colas de producción y prueba si etiqueta sus colas de la siguiente forma:

Cola	Clave	Valor
`MyQueueA`	`QueueType`	`Production`
`MyQueueB`	`QueueType`	`Testing`

> **Note**
>
> Cuando utilice etiquetas de cola, tenga en cuenta las siguientes directrices:
>
> - No recomendamos agregar más de 50 etiquetas a una cola. El etiquetado admite caracteres Unicode en UTF-8.
> - Las etiquetas no tienen ningún significado semántico. Amazon SQS interpreta las etiquetas como cadenas de caracteres.
> - Las etiquetas distinguen entre mayúsculas y minúsculas.
> - Una etiqueta nueva con una clave idéntica a la de una etiqueta existente sobrescribe la existente.
> - Las acciones de etiquetado están limitadas a 30 TPS cada una. Cuenta de AWS Si su aplicación requiere un mayor rendimiento, envíe una solicitud.

Para obtener una lista completa de restricciones de etiqueta, consulte Cuotas de colas estándar de Amazon SQS.

Sondeos cortos y largos de Amazon SQS

Amazon SQS proporciona opciones de sondeo corto y largo para recibir mensajes de una cola. Tenga en cuenta los requisitos de capacidad de respuesta y rentabilidad de su aplicación al elegir entre estas dos opciones de sondeo:

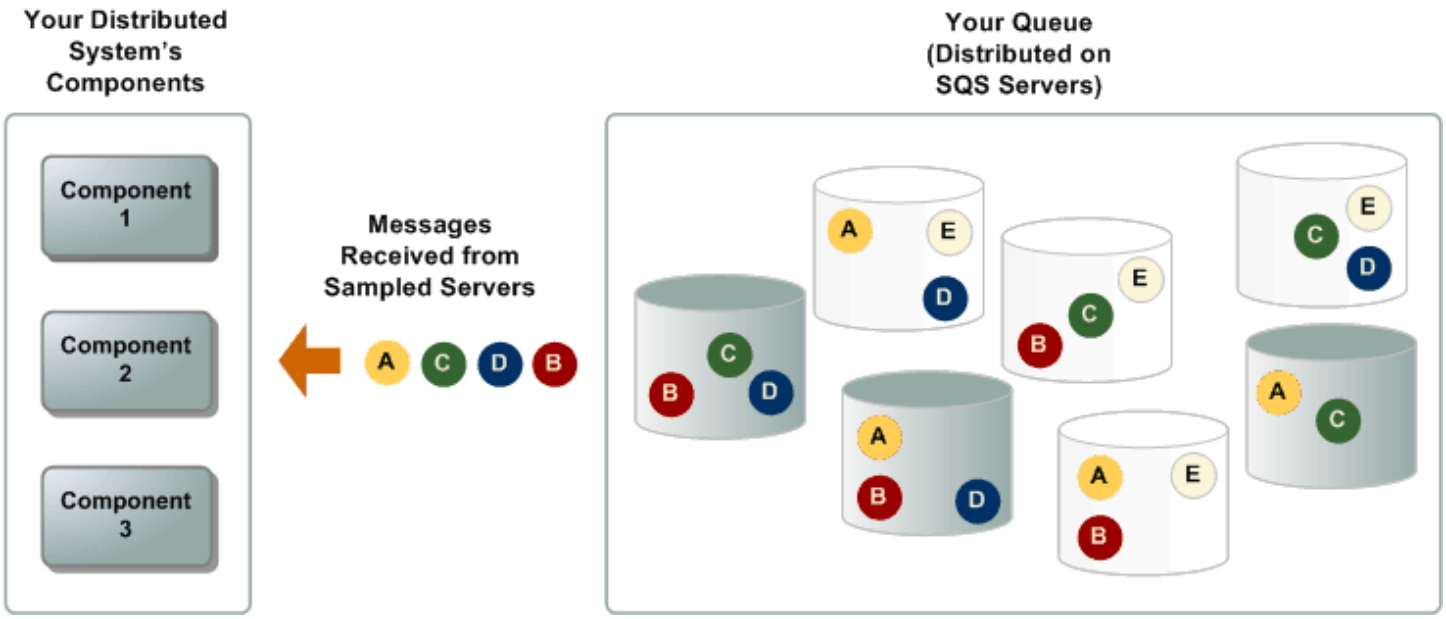
- Sondeo corto (predeterminado): la solicitud `ReceiveMessage` consulta un subconjunto de servidores (según una distribución aleatoria ponderada) para encontrar los mensajes disponibles y envía una respuesta inmediata, incluso si no se encuentra ningún mensaje.
- Sondeo largo: `ReceiveMessage` consulta los mensajes en todos los servidores y envía una respuesta una vez que haya al menos un mensaje disponible, hasta el máximo especificado. Se envía una respuesta vacía solo si caduca el tiempo de espera de sondeo. Esta opción puede reducir el número de respuestas vacías y, potencialmente, los costos.

En las siguientes secciones se explican los detalles de los sondeos cortos y largos.

Consumo de mensajes mediante sondeo corto

Cuando se consumen mensajes de una cola (FIFO o estándar) mediante sondeo corto, Amazon SQS muestrea un subconjunto de sus servidores (en función de una distribución al azar ponderada) y devuelve mensajes solo de esos servidores. Por tanto, una solicitud `ReceiveMessage` determinada podría no devolver todos los mensajes. Sin embargo, si tiene menos de 1 000 mensajes en la cola, una solicitud posterior devolverá sus mensajes. Si sigue consumiendo mensajes de sus colas, Amazon SQS muestrea todos sus servidores y se reciben todos los mensajes.

En el siguiente diagrama se muestra el comportamiento de sondeo corto de los mensajes devueltos de una cola estándar después de que uno de los componentes del sistema realice una solicitud de recepción. Amazon SQS muestrea varios de sus servidores (en gris) y devuelve los mensajes A, C, D y B de estos servidores. El mensaje E no se devuelve en esta solicitud concreta, pero se devuelve en una solicitud posterior.



Consumo de mensajes mediante sondeo largo

Cuando el tiempo de espera de la acción de la API `ReceiveMessage` es superior a 0, se está realizando un sondeo largo. El tiempo máximo de espera de sondeo es de 20 segundos. El sondeo largo ayuda a reducir el costo de uso de Amazon SQS al eliminar el número de respuestas vacías (cuando no hay ningún mensaje disponible para una solicitud `ReceiveMessage`) y las falsas respuestas vacías (cuando los mensajes están disponibles en la cola, pero no se incluyen en una respuesta). Para obtener información sobre cómo habilitar el sondeo largo para una cola nueva o existente mediante la consola de Amazon SQS, consulte Configuración de parámetros de colas mediante la consola de Amazon SQS. Para ver las prácticas recomendadas, consulte Configuración del sondeo largo en Amazon SQS.

El sondeo largo ofrece los siguientes beneficios:

- Reducción de las respuestas vacías al permitir que Amazon SQS espere hasta que haya un mensaje disponible en una cola antes de enviar una respuesta. A menos que la conexión agote el tiempo de espera, la respuesta a la solicitud `ReceiveMessage` contiene al menos uno de los mensajes disponibles, hasta el número máximo de mensajes especificado en la acción `ReceiveMessage`. En contados casos, es posible que reciba respuestas vacías incluso cuando una cola aún contenga mensajes, sobre todo si especifica un valor bajo para el parámetro `ReceiveMessageWaitTimeSeconds`.
- Reduzca las falsas respuestas vacías consultando todos (en lugar de un subconjunto) los servidores de Amazon SQS.

- Devolución de mensajes en cuanto estén disponibles.

Para obtener información sobre cómo confirmar que una cola está vacía, consulte Confirmación de que una cola de Amazon SQS está vacía.

Diferencias entre el sondeo corto y el sondeo largo

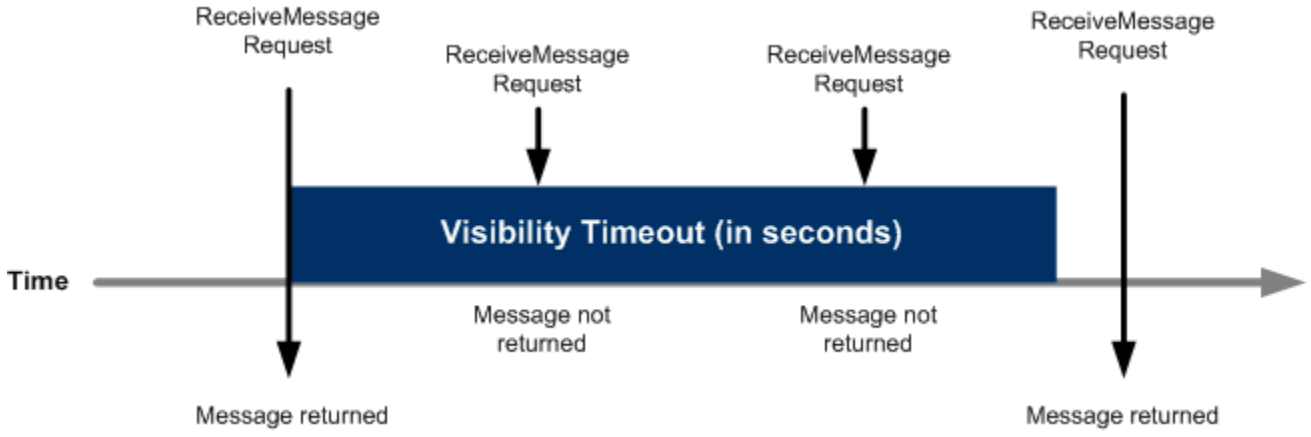
El sondeo corto se produce cuando el parámetro `WaitTimeSeconds` de una solicitud `ReceiveMessage` está establecido en `0` de una de las dos formas siguientes:

- La llamada a `ReceiveMessage` establece `WaitTimeSeconds` en `0`.
- La llamada a `ReceiveMessage` no establece `WaitTimeSeconds` y el atributo de cola `ReceiveMessageWaitTimeSeconds` está establecido en `0`.

Tiempo de espera de visibilidad de Amazon SQS

Cuando recibes un mensaje de una cola de Amazon SQS, permanece en la cola pero pasa a ser invisible temporalmente para otros consumidores. Esta invisibilidad se controla mediante el tiempo de espera de visibilidad, lo que garantiza que otros consumidores no puedan procesar el mismo mensaje mientras usted trabaja en él. Amazon SQS ofrece dos opciones para eliminar los mensajes después del procesamiento:

- Eliminación manual: los mensajes se eliminan de forma explícita mediante esta `DeleteMessage`acción.
- Eliminación automática: en algunos casos AWS SDKs, los mensajes se eliminan automáticamente cuando se procesan correctamente, lo que simplifica los flujos de trabajo.



Casos de uso del tiempo de espera de visibilidad

Gestione las tareas de larga duración: utilice el tiempo de espera de visibilidad para gestionar las tareas que requieren tiempos de procesamiento prolongados. Establezca un tiempo de espera de visibilidad adecuado para los mensajes que requieren un tiempo de procesamiento prolongado. Esto garantiza que otros consumidores no capten el mismo mensaje mientras se está procesando, lo que evita la duplicación de trabajo y mantiene la eficiencia del sistema.

Implemente mecanismos de reintento: amplíe el tiempo de espera de visibilidad mediante programación para las tareas que no se completen dentro del tiempo de espera inicial. Si una tarea no se completa dentro del tiempo de espera de visibilidad inicial, puede ampliarlo mediante programación. Esto permite que el sistema vuelva a intentar procesar el mensaje sin que otros consumidores lo vean, lo que mejora la tolerancia a los errores y la fiabilidad. Combínelo con Dead-Letter Queues (DLQs) para gestionar los errores persistentes.

Coordine los sistemas distribuidos: utilice el tiempo de espera de visibilidad de SQS para coordinar las tareas en todos los sistemas distribuidos. Establezca tiempos de espera de visibilidad que se ajusten a los tiempos de procesamiento esperados para los diferentes componentes. Esto ayuda a mantener la coherencia y evita las condiciones de carrera en arquitecturas distribuidas y complejas.

Optimice la utilización de los recursos: ajuste los tiempos de espera de visibilidad de SQS para optimizar la utilización de los recursos en su aplicación. Al establecer los tiempos de espera adecuados, puede asegurarse de que los mensajes se procesen de manera eficiente sin consumir recursos innecesariamente. Esto conduce a un mejor rendimiento general del sistema y a una mayor rentabilidad.

Establecer y ajustar el tiempo de espera de visibilidad

El tiempo de espera de visibilidad comienza en cuanto recibes un mensaje. Durante este período, se espera que proceses y elimines el mensaje. Si no lo eliminas antes de que se agote el tiempo de espera, el mensaje volverá a estar visible en la cola y otro consumidor podrá recuperarlo. El tiempo de espera de visibilidad predeterminado de una cola es de 30 segundos, pero puedes ajustarlo para que coincida con el tiempo que necesita la aplicación para procesar y eliminar un mensaje. También puedes establecer un tiempo de espera de visibilidad específico para los mensajes individuales sin cambiar la configuración general de la cola. Utilice la `ChangeMessageVisibility`acción para ampliar o acortar el tiempo de espera mediante programación, según sea necesario.

Mensajes y cuotas en vuelo

En Amazon SQS, los mensajes en vuelo son mensajes que ha recibido un consumidor pero que aún no se han eliminado. En el caso de las colas estándar, hay un límite de aproximadamente 120 000 mensajes en vuelo, según el tráfico de las colas y la acumulación de mensajes. Si alcanzas este límite, Amazon SQS devuelve un `OverLimit` error que indica que no se pueden recibir más mensajes hasta que se eliminen algunos mensajes durante el vuelo. En el caso de las colas FIFO, los límites dependen de los grupos de mensajes activos.

- Cuando se utiliza un sondeo breve: si se alcanza este límite mientras se utiliza un sondeo corto, Amazon SQS devolverá un `OverLimit` error que indica que no se pueden recibir más mensajes hasta que se eliminen algunos mensajes en curso.
- Cuando se utiliza un sondeo prolongado: si se utiliza un sondeo prolongado, Amazon SQS no devuelve ningún error cuando se alcanza el límite de mensajes en vuelo. En su lugar, no devolverá ningún mensaje nuevo hasta que el número de mensajes en tránsito caiga por debajo del límite.

Para gestionar los mensajes durante el vuelo de forma eficaz:

1. Eliminación inmediata: elimine los mensajes (manual o automáticamente) después de procesarlos para reducir el número de mensajes durante el vuelo.
2. Monitoree con CloudWatch: configure alarmas para que el número de pasajeros en vuelo sea elevado y así evitar que se alcance el límite.
3. Distribuya la carga: si está procesando un gran volumen de mensajes, utilice colas o consumidores adicionales para equilibrar la carga y evitar cuellos de botella.
4. Solicita un aumento de cuota: envía una solicitud a AWS Support si se requieren límites más altos.

Descripción del tiempo de espera de visibilidad en colas estándar y FIFO

Tanto en las colas estándar como en las FIFO (primero en entrar, primero en salir), el tiempo de espera de visibilidad ayuda a evitar que varios consumidores procesen el mismo mensaje simultáneamente. Sin embargo, debido al modelo de at-least-once entrega de Amazon SQS, no hay ninguna garantía absoluta de que un mensaje no se entregue más de una vez durante el período de tiempo de espera de visibilidad.

- Colas estándar: el tiempo de espera de visibilidad en las colas estándar impide que varios consumidores procesen el mismo mensaje al mismo tiempo. Sin embargo, debido al modelo de at-

least-once entrega, Amazon SQS no garantiza que un mensaje no se entregue más de una vez dentro del período de tiempo de espera de visibilidad.

- Colas FIFO: en el caso de las colas FIFO, los mensajes con el mismo ID de grupo de mensajes se procesan siguiendo una secuencia estricta. Cuando un mensaje con un identificador de grupo de mensajes está en movimiento, los mensajes subsiguientes de ese grupo no estarán disponibles hasta que se elimine el mensaje en movimiento o se agote el tiempo de espera de visibilidad. Sin embargo, esto no «bloquea» el grupo indefinidamente: cada mensaje se procesa en secuencia y los consumidores solo podrán acceder al siguiente mensaje de ese grupo cuando se elimine o vuelva a ser visible. Este enfoque garantiza un procesamiento ordenado dentro del grupo sin impedir innecesariamente que el grupo entregue los mensajes.

Manejar los fallos

Si no procesa ni elimina un mensaje antes de que caduque el tiempo de espera de visibilidad (debido a errores de la aplicación, bloqueos o problemas de conectividad), el mensaje vuelve a estar visible en la cola. Luego, el mismo consumidor o uno diferente pueden recuperarlo para otro intento de procesamiento. Esto garantiza que los mensajes no se pierdan aunque se produzca un error en el procesamiento inicial. Sin embargo, si se establece un tiempo de espera de visibilidad demasiado alto, se puede retrasar la reaparición de los mensajes no procesados, lo que podría ralentizar los reintentos. Es fundamental establecer un tiempo de espera de visibilidad adecuado en función del tiempo de procesamiento previsto para gestionar los mensajes a tiempo.

Cambio y finalización del tiempo de espera de visibilidad

Puedes cambiar o finalizar el tiempo de espera de visibilidad mediante la `ChangeMessageVisibility` siguiente acción:

- Cambiar el tiempo de espera: ajuste el tiempo de espera de visibilidad de forma dinámica utilizando. `ChangeMessageVisibility` Esto le permite ampliar o reducir los tiempos de espera para adaptarlos a las necesidades de procesamiento.
- Finalización del tiempo de espera: si decide no procesar un mensaje recibido, finalice su tiempo de espera de visibilidad configurándolo en 0 segundos `VisibilityTimeout` a lo largo de la acción. `ChangeMessageVisibility` Esto hace que el mensaje esté inmediatamente disponible para que otros consumidores lo procesen.

Prácticas recomendadas

Utilice las siguientes prácticas recomendadas para gestionar los tiempos de espera de visibilidad en Amazon SQS, incluida la configuración, el ajuste y la ampliación de los tiempos de espera, así como la gestión de los mensajes no procesados mediante colas de cartas muertas (). DLQs

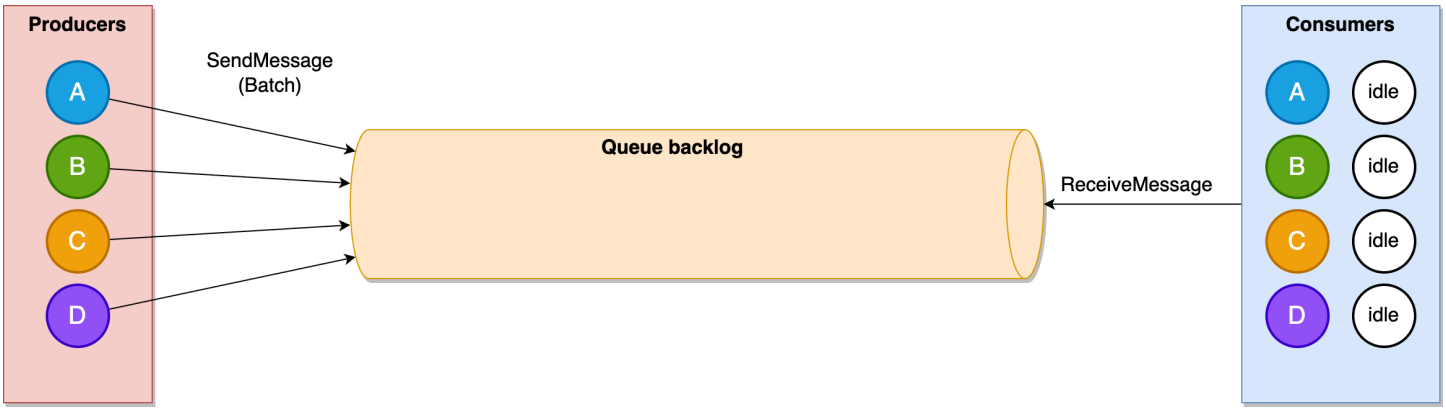
- Configurar y ajustar el tiempo de espera. Comience configurando el tiempo de espera de visibilidad para que coincida con el tiempo máximo que normalmente necesita la aplicación para procesar y eliminar un mensaje. Si no está seguro del tiempo de procesamiento exacto, comience con un tiempo de espera más corto (por ejemplo, 2 minutos) y amplíelo según sea necesario. Implemente un mecanismo de aceleración para extender periódicamente el tiempo de espera de visibilidad y garantizar que el mensaje permanezca invisible hasta que se complete el procesamiento. Esto minimiza los retrasos en el reprocesamiento de los mensajes no administrados y evita una visibilidad prematura.
- Ampliar el tiempo de espera y administrar el límite de 12 horas. Si el tiempo de procesamiento varía o puede superar el tiempo de espera establecido inicialmente, utilice la `ChangeMessageVisibility` acción para extender el tiempo de espera de visibilidad mientras procesa el mensaje. Tenga en cuenta que el tiempo de espera de visibilidad tiene un límite máximo de 12 horas a partir de la recepción del mensaje por primera vez. Ampliar el tiempo de espera no restablece este límite de 12 horas. Si su procesamiento requiere más tiempo que este límite, considere la posibilidad de utilizar AWS Step Functions o dividir la tarea en pasos más pequeños.
- Manejo de mensajes sin procesar. Para administrar los mensajes que fallan en varios intentos de procesamiento, configure una cola de mensajes sin procesar (DLQ). Esto garantiza que los mensajes que no se puedan procesar tras varios reintentos se capturen por separado para su posterior análisis o tratamiento, lo que evita que circulen repetidamente en la cola principal.

Colas de ferias de Amazon SQS

Las colas justas de Amazon SQS mitigan automáticamente el impacto en las colas de varios inquilinos que contienen mensajes de varias entidades lógicas, como clientes, aplicaciones de cliente o tipos de mensajes. En estos entornos de colas compartidas, una métrica de rendimiento fundamental es el tiempo de espera, que mide el tiempo total que los mensajes pasan en una cola desde su llegada hasta su procesamiento. Cuando un inquilino crea un retraso en la cola al publicar más mensajes de los que el sistema puede gestionar, las colas justas minimizan el impacto en el tiempo de espera para los demás inquilinos.

Estado estacionario

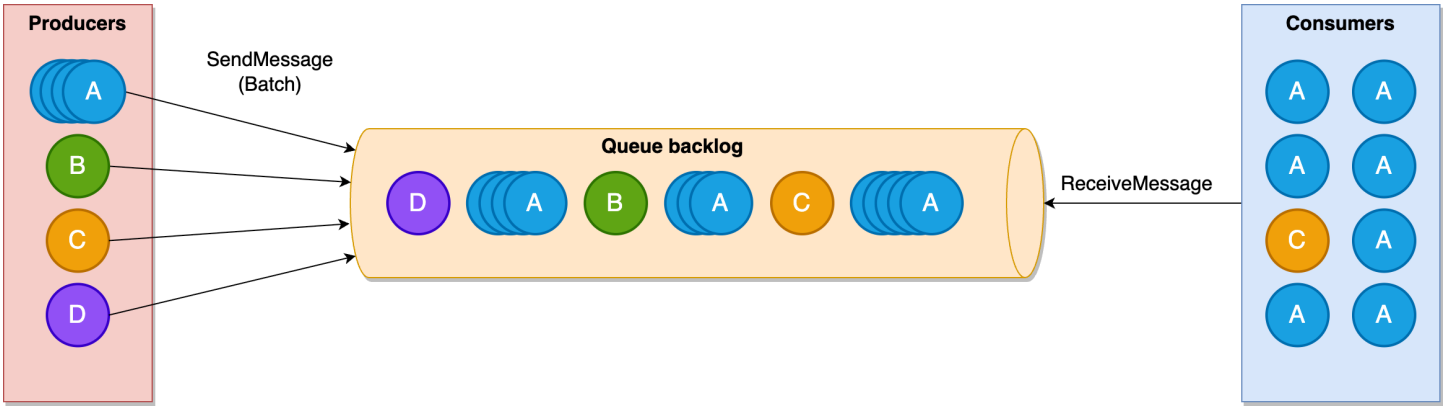
El siguiente diagrama ilustra una cola de varios inquilinos que contiene mensajes de cuatro inquilinos distintos (denominados A, B, C y D). La cola funciona de forma estable y no hay acumulación de mensajes pendientes, ya que los consumidores reciben los mensajes en cuanto aparecen en la cola. Todos los inquilinos tienen tiempos de permanencia bajos. No toda la capacidad de consumo se utiliza plenamente en este estado estable.



Impacto ruidoso en los vecinos

El ruido en los vecinos se produce cuando un inquilino de una cola de varios inquilinos crea un atraso, lo que aumenta el tiempo de espera de los mensajes para todos los demás inquilinos. Un inquilino puede convertirse en un vecino ruidoso si envía un mayor volumen de mensajes que otros inquilinos, o cuando los consumidores tardan más en procesar los mensajes de ese inquilino en particular.

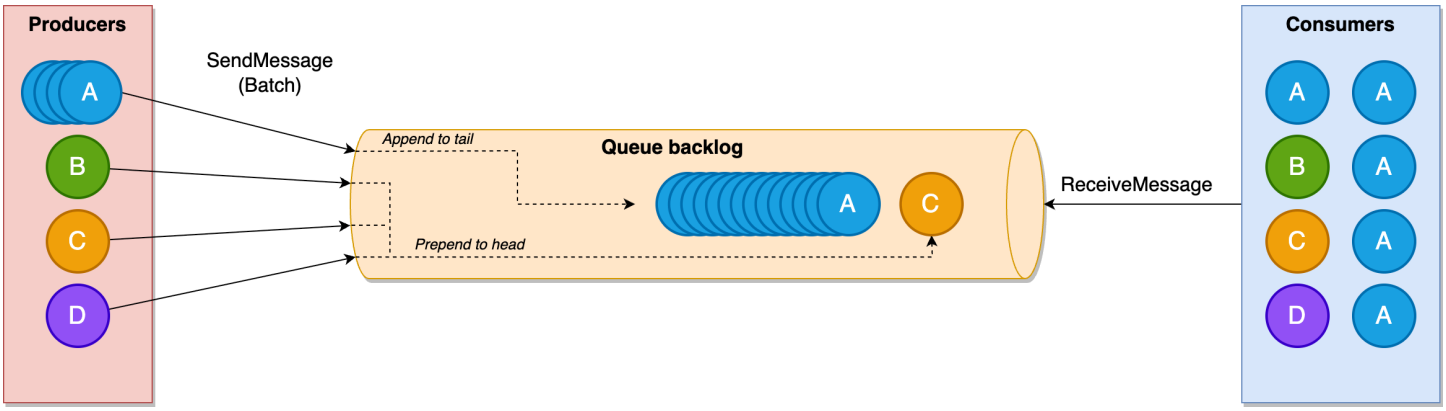
Este diagrama ilustra cómo el aumento del tráfico del inquilino A crea un atraso en la cola. Los consumidores están ocupados procesando solo los mensajes del arrendatario A, mientras que los mensajes de otros inquilinos se quedan pendientes, lo que se traduce en tiempos de espera más prolongados para todos los inquilinos.



Mitigación con colas justas

Amazon SQS detecta vecinos ruidosos mediante la supervisión de la distribución de los mensajes entre los inquilinos durante el procesamiento (estado «en movimiento»). Cuando un inquilino tiene un número desproporcionadamente grande de mensajes durante el vuelo en comparación con otros, Amazon SQS lo identifica como un vecino ruidoso y da prioridad a la entrega de mensajes a los demás inquilinos. Este enfoque reduce el impacto del tiempo de permanencia en los demás inquilinos.

Este diagrama ilustra cómo Amazon SQS Fair Quees aborda el problema de los vecinos ruidosos. Cuando un inquilino (el inquilino A) hace ruido, Amazon SQS prioriza la devolución de los mensajes de otros inquilinos (B, C y D). Esta priorización ayuda a mantener bajos los tiempos de espera de los inquilinos B, C y D, que permanecen silenciosos, mientras que el tiempo de espera de los mensajes del inquilino A aumenta hasta que se agotan las colas acumuladas sin afectar a los demás inquilinos.



> **Note**
>
> Amazon SQS no limita la tasa de consumo por inquilino. Permite a los consumidores recibir mensajes de inquilinos vecinos ruidosos cuando hay capacidad de consumo y la cola no tiene otros mensajes que devolver. Al igual que las colas estándar de Amazon SQS, las colas justas permiten un rendimiento prácticamente ilimitado y no hay límites en cuanto al número de inquilinos que podría tener en la cola.

¿Qué diferencia hay con las colas FIFO

Las colas FIFO mantienen un orden estricto al limitar la cantidad de mensajes a bordo de cada inquilino. Si bien esto evita que los vecinos sean ruidosos, limita el rendimiento de cada inquilino. Las colas justas están diseñadas para escenarios con varios inquilinos en los que se prioriza un alto rendimiento, un tiempo de espera reducido y una asignación justa de los recursos. Las colas justas

permiten que varios consumidores procesen los mensajes del mismo inquilino simultáneamente y, al mismo tiempo, ayudan a todos los inquilinos a mantener tiempos de espera constantes.

Utilizar colas justas

Los productores de sus mensajes pueden añadir un identificador de inquilino configurando un `MessageGroupId` en un mensaje saliente:

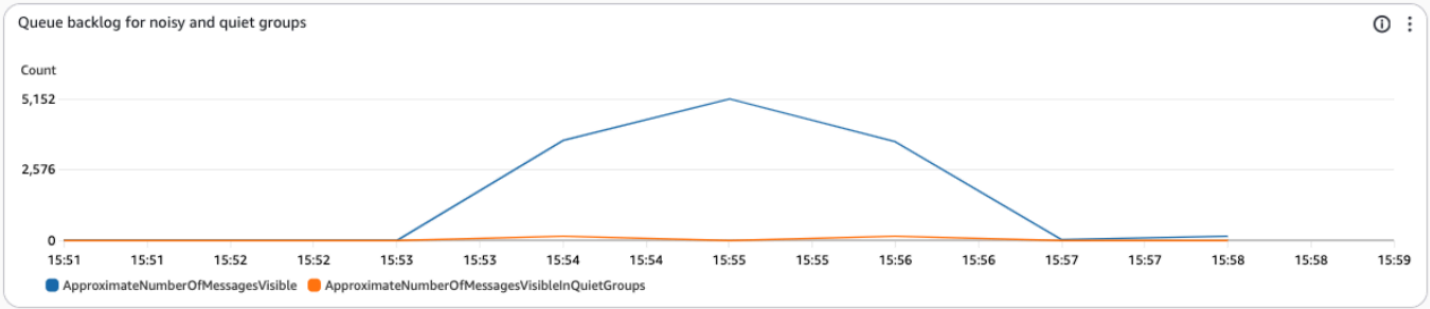
```
// Send message with tenant identifier
SendMessageRequest request = new SendMessageRequest()
    .withQueueUrl(queueUrl)
    .withMessageBody(messageBody)
    .withMessageGroupId("tenant-123"); // Tenant identifier
sqs.sendMessage(request);
```

La función de imparcialidad se aplicará automáticamente en todas las colas estándar de Amazon SQS para los mensajes con la propiedad. MessageGroupId No requiere ningún cambio en el código de consumo, no afecta a la latencia de la API y no implica ninguna limitación de rendimiento.

Métricas de colas CloudWatch justas

Amazon SQS proporciona CloudWatch métricas adicionales para ayudarle a supervisar la mitigación del impacto en los vecinos ruidosos. Por ejemplo, puede comparar `Approximate..InQuietGroups` las métricas con las métricas estándar a nivel de cola. Durante los picos de tráfico de un usuario específico, las métricas generales a nivel de cola pueden revelar el aumento de los atrasos o la antigüedad de los mensajes. Sin embargo, si analizas los grupos silenciosos de forma aislada, puedes identificar que la mayoría de los grupos de mensajes o inquilinos que no son ruidosos no se ven afectados.

A continuación, puede ver un ejemplo en el que la métrica estándar de acumulación de colas (ApproximateNumberOfMessagesVisible) aumenta debido a un inquilino ruidoso, mientras que la acumulación de pendientes en el caso de los inquilinos no ruidosos () sigue siendo baja. ApproximateNumberOfMessagesVisibleInQuietGroups



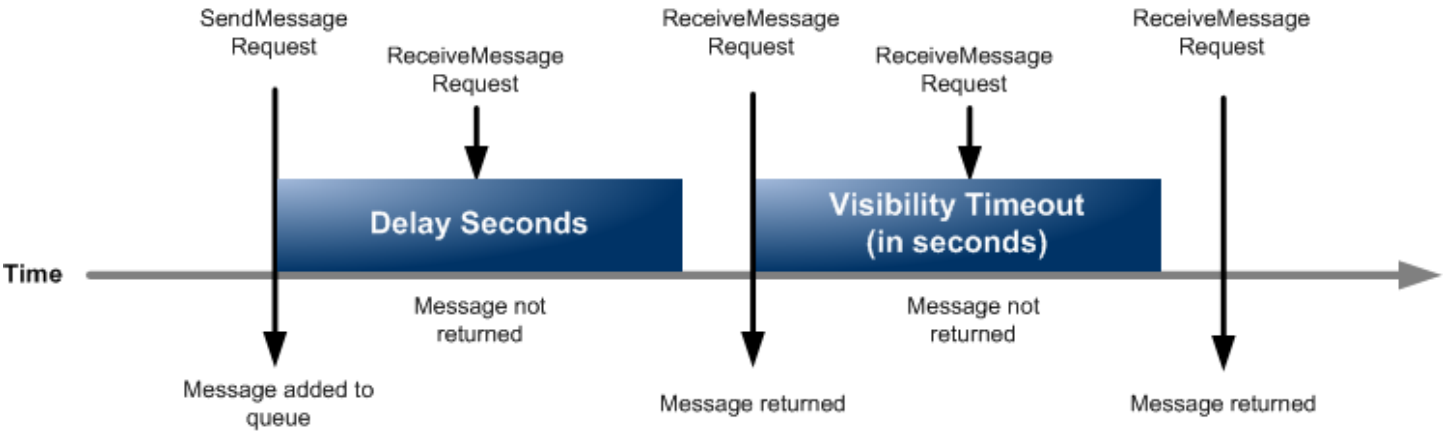
Para obtener una lista completa de las CloudWatch métricas de Amazon SQS y sus descripciones, consulte las CloudWatch métricas de Amazon SQS.

Colas con retraso de Amazon SQS

Las colas con retraso le permiten posponer la entrega de nuevos mensajes a los consumidores durante un número de segundos, por ejemplo, cuando su aplicación consumidora necesita tiempo adicional para procesar los mensajes. Si crea una cola con retraso, los mensajes que envíe a la cola permanecerán invisible para los consumidores mientras dure el período de retraso. El retraso predeterminado (mínimo) de una cola es de 0 segundos. El valor máximo es de 15 minutos. Para obtener información sobre la configuración las colas con retraso mediante la consola, consulte Configuración de parámetros de colas mediante la consola de Amazon SQS.

> **Note**
>
> En las colas estándar, la configuración de retraso por cola no es retroactiva, es decir, el cambio de la configuración no afecta al retraso de los mensajes que ya están en la cola.
> En las colas FIFO, la configuración de retraso por cola es retroactiva, es decir, el cambio de la configuración afecta al retraso de los mensajes que ya están en la cola.

Las colas con retraso son similares a los tiempos de espera de visibilidad porque ambos hacen que los mensajes no estén disponibles para los consumidores durante un periodo de tiempo determinado. La diferencia es que para las colas con retraso un mensaje está oculto cuando es la primera vez que se añade a la cola, mientras que para los tiempos de espera de visibilidad un mensaje está oculto solo después de que se consuma un mensaje de la cola. El siguiente diagrama ilustra la relación existente entre las colas con retraso y los tiempos de espera de visibilidad.



Opciones de programación ampliadas

Si bien las colas de demora y los temporizadores de mensajes de Amazon SQS permiten programar la entrega de mensajes hasta 15 minutos en el futuro, es posible que necesite capacidades de programación más flexibles. En esos casos, considere la posibilidad de utilizar EventBridge Scheduler, que le permite programar miles de millones de acciones únicas o recurrentes de la API sin limitaciones de tiempo. EventBridge Scheduler es la solución recomendada para los casos de uso avanzados de la programación de mensajes.

Para configurar los segundos de retraso en mensajes individuales, en lugar de en una cola completa, utilice temporizadores de mensajes para permitir que Amazon SQS utilice el valor del temporizador de mensajes en lugar `DelaySeconds` del valor de la cola de retraso. `DelaySeconds` EventBridge El programador también permite programar mensajes individuales.

Colas temporales de Amazon SQS

Las colas temporales le ayudan a ahorrar tiempo de desarrollo y costes de implementación al utilizar patrones de mensajes comunes como, por ejemplo, solicitud-respuesta. Puede utilizar el Cliente de colas temporales para crear colas temporales de alto rendimiento, rentables y administradas por la aplicación.

El cliente asigna automáticamente varias colas temporales (colas administradas por la aplicación creadas bajo demanda para un proceso concreto) a una única cola de Amazon SQS. De este modo, la aplicación puede realizar menos llamadas y tiene un rendimiento superior cuando el tráfico de cada cola temporal es bajo. Cuando una cola temporal ya no está en uso, el cliente la limpia automáticamente, incluso aunque algunos procesos que utilicen el cliente no se hayan cerrado correctamente.

Estas son las ventajas que brindan las colas temporales:

- Actúan como canales de comunicación ligeros para procesos o subprocesos específicos.
- Se pueden crear y eliminar sin generar costos adicionales.
- Son compatibles con las colas estáticas (normales) de Amazon SQS a través de una API. Esto significa que el código existente que envía y recibe los mensajes puede enviar mensajes a colas virtuales y recibir mensajes de estas.

Colas virtuales

Las colas virtuales son estructuras de datos locales que el Cliente de colas temporales crea. Las colas virtuales permiten combinar varios destinos con poco tráfico en una única cola de Amazon SQS. Para obtener información sobre prácticas recomendadas, consulte Evitar reutilizar el mismo ID de grupo de mensajes con colas virtuales.

> **Note**
>
> - Cuando se crea una cola virtual, solo se crean estructuras de datos temporales en las que los consumidores reciben mensajes. Como las colas virtuales no hacen llamadas de API a Amazon SQS, no generan costo alguno.
> - Las cuotas de TPS se aplican a todas las colas virtuales en una sola cola de host. Para obtener más información, consulte Cuotas de mensajes de Amazon SQS.

La clase de encapsulamiento `AmazonSQSVirtualQueuesClient` proporciona compatibilidad con atributos relacionados con las colas virtuales. Para crear una cola virtual, debe llamar a la acción `CreateQueue` de la API con el atributo `HostQueueURL`. Este atributo especifica la cola existente que hospeda las colas virtuales.

La dirección URL de una cola virtual tiene el siguiente formato.

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

Cuando un productor llama a la acción `SendMessage` o `SendMessageBatch` de la API en una dirección URL de una cola virtual, el Cliente de colas temporales hace lo siguiente:

1. Extrae el nombre de la cola virtual.

2. Lo adjunta como un atributo más del mensaje.
3. Envía el mensaje a la cola del host.

Mientras el productor envía mensajes, un subproceso en segundo plano sondea la cola del host y envía los mensajes recibidos a las colas virtuales en función de los atributos del mensaje correspondiente.

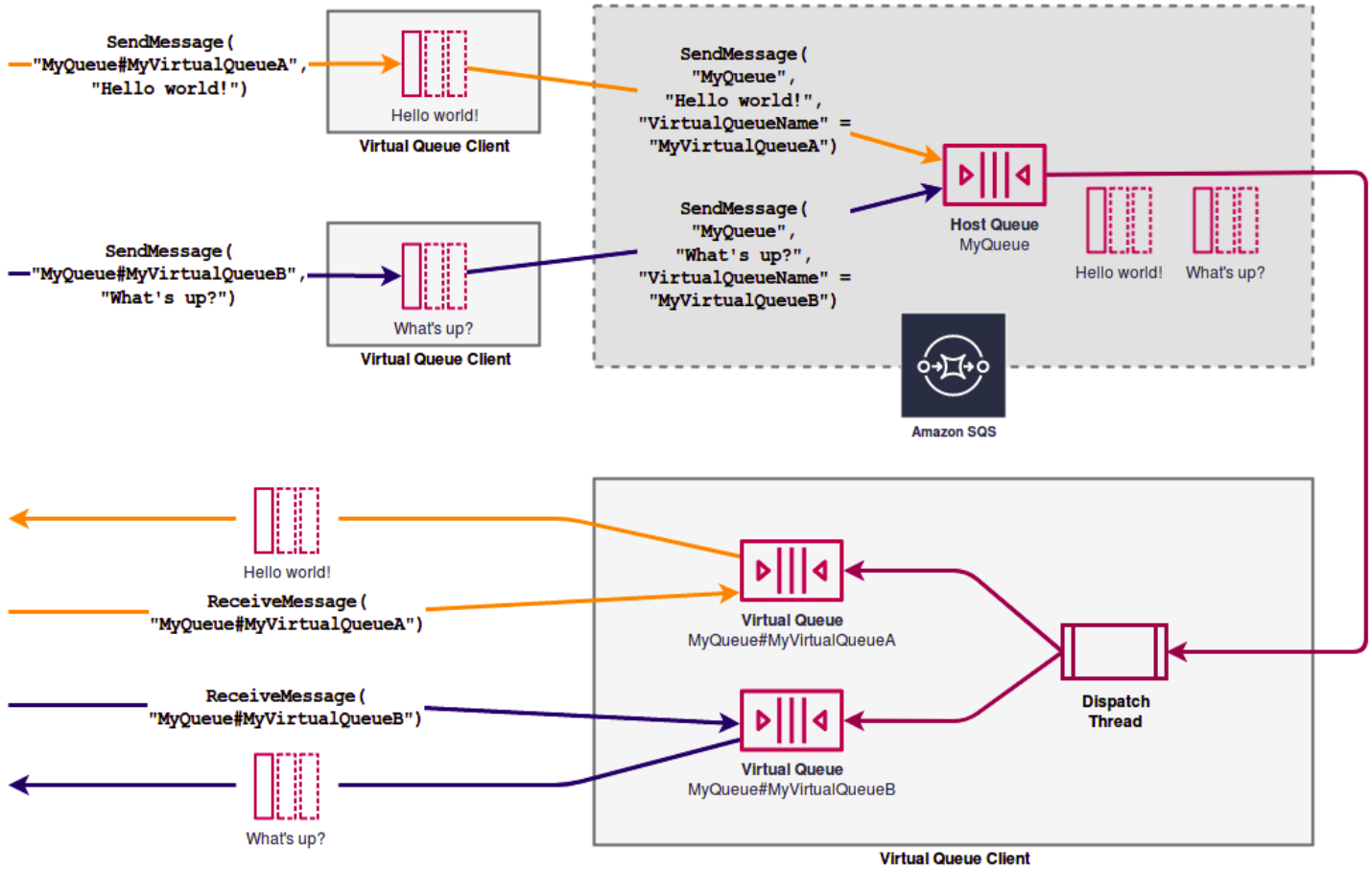
Mientras el consumidor llama a la acción `ReceiveMessage` de la API en una dirección URL de una cola virtual, el Cliente de colas temporales bloquea la llamada de este cliente localmente hasta que el subproceso en segundo plano envíe un mensaje a la cola virtual. (Este proceso es similar a la recopilación previa de mensajes del Cliente asíncrono en búfer: una sola acción de la API puede proporcionar mensajes en hasta 10 colas virtuales). Al eliminar una cola virtual, se elimina cualquier recurso del lado del cliente sin necesidad de llamar al propio servicio Amazon SQS.

La clase `AmazonSQSTemporaryQueuesClient` convierte automáticamente todas las colas que crea en colas temporales. También crea colas de host de forma automática con los mismos atributos, bajo demanda. Los nombres de estas colas comparten un prefijo común que se puede configurar (de forma predeterminada, `__RequesterClientQueues__`) y que las identifica como colas temporales. De este modo, el cliente puede actuar como un sustituto inmediato que optimiza el código existente, encargado de crear y eliminar las colas. El cliente también incluye las interfaces `AmazonSQSRequester` y `AmazonSQSResponder`, que posibilitan la comunicación bidireccional entre las colas.

Patrón de mensajes de respuesta a solicitudes (colas virtuales)

El caso de uso más común de las colas temporales es el patrón de mensajes de respuesta a solicitudes, donde un solicitante crea una cola temporal para recibir cada mensaje de respuesta. Para no tener que crear una cola de Amazon SQS por cada mensaje de respuesta, el Cliente de colas temporales permite crear y eliminar varias colas temporales sin necesidad de hacer llamadas a la API de Amazon SQS. Para obtener más información, consulte el tema Implementación de sistemas de solicitud-respuesta.

En el siguiente diagrama, se muestra una configuración frecuente con este patrón.



Situación de ejemplo: procesamiento de una solicitud de inicio de sesión

En el siguiente ejemplo, se muestra cómo puede utilizar las interfaces `AmazonSQSRequester` y `AmazonSQSResponder` para procesar una solicitud de inicio de sesión del usuario.

En el lado del cliente

```
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();
    }
}
```

Al enviar una solicitud de inicio de sesión, ocurre lo siguiente:

1. Se crea una tabla temporal.
2. Se adjunta la URL de la cola temporal al mensaje como un atributo.
3. Se envía el mensaje.
4. Se recibe una respuesta de la cola temporal.
5. Se elimina la cola temporal.
6. Se devuelve la respuesta.

En el lado del servidor

En el siguiente ejemplo se presupone que, en el momento de la construcción, se crea un subproceso que sondea la cola y llama al método `handleLoginRequest()` en cada mensaje. Además, se presupone que se utiliza el método `doLogin()`.

```
public class LoginServer {

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

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

```
            AmazonSQSResponderClientBuilder.defaultClient();

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

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

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

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

Limpieza de colas

Para asegurarse de que Amazon SQS recupera los recursos en memoria utilizados por las colas virtuales, cuando la aplicación ya no necesite el Cliente de colas temporales, debería llamar al método `shutdown()`. También se puede utilizar el método `shutdown()` de la interfaz `AmazonSQSRequester`.

El Cliente de colas temporales también cuenta con un mecanismo para eliminar las colas huérfanas del host. En cada una de las colas que recibe llamadas de la API durante un determinado periodo de tiempo (de forma predeterminada, cinco minutos), el cliente utiliza la acción `TagQueue` de la API para etiquetar las colas que permanecen en uso.

> **Note**
>
> Cualquier acción de la API que se realice en una cola marcará dicha cola como no inactiva, incluidas las acciones `ReceiveMessage` que no devuelven mensajes.

El subproceso en segundo plano utiliza las acciones `ListQueues` y `ListTags` de la API para comprobar todas las colas con el prefijo configurado y eliminar las colas que no se han etiquetado en los últimos cinco minutos. De esta forma, si un cliente no se cierra de forma limpia, el resto de los

clientes activos limpiarán después. Para reducir las cargas de trabajo duplicadas, todos los clientes con el mismo prefijo se comunican a través de una cola de trabajo interna y compartida.

Temporizadores de mensajes de Amazon SQS

Los temporizadores de mensajes te permiten establecer un período de invisibilidad inicial para un mensaje cuando se añade a una cola. Por ejemplo, si envías un mensaje con un temporizador de 45 segundos, permanecerá oculto para los consumidores durante los primeros 45 segundos. El retraso predeterminado (mínimo) de un mensaje es de 0 segundos. El valor máximo es de 15 minutos. Para obtener información sobre cómo enviar mensajes con temporizadores mediante la consola, consulte Enviar un mensaje mediante una cola estándar.

> **Note**
>
> Las colas FIFO no admiten temporizadores en los mensajes individuales.

Para establecer un período de retraso en una cola completa, en lugar de en mensajes individuales, usa las colas de demora. La configuración de un temporizador de mensajes para un mensaje individual anula cualquier valor `DelaySeconds` en una cola con retraso de Amazon SQS.

Opciones de programación ampliadas

Si bien las colas de demora y los temporizadores de mensajes de Amazon SQS permiten programar la entrega de mensajes hasta 15 minutos en el futuro, es posible que necesite capacidades de programación más flexibles. En esos casos, considere la posibilidad de utilizar EventBridge Scheduler, que le permite programar miles de millones de acciones únicas o recurrentes de la API sin limitaciones de tiempo. EventBridge Scheduler es la solución recomendada para los casos de uso avanzados de la programación de mensajes.

Acceso a Amazon EventBridge Pipes a través de la consola Amazon SQS

Amazon EventBridge Pipes conecta las fuentes con los objetivos. Los tubos están diseñados para la point-to-point integración entre las fuentes y los objetivos compatibles, y permiten realizar transformaciones y enriquecimientos avanzados. EventBridge Los tubos proporcionan una forma altamente escalable de conectar tu cola de Amazon SQS con AWS servicios como Step Functions,

Amazon SQS y API Gateway, así como con aplicaciones de software como servicio (SaaS) de terceros, como Salesforce.

Para configurar una canalización, elija el origen, agregue filtros opcionales, defina el enriquecimiento opcional y elija el destino de los datos del evento.

En la página de detalles de una cola de Amazon SQS, puede ver las canalizaciones que utilizan ese clúster como origen. Desde allí, también puede hacer lo siguiente:

- Inicie la EventBridge consola para ver los detalles de la canalización.
- Inicie la EventBridge consola para crear una nueva tubería con la cola como fuente.

Para obtener más información sobre la configuración de una cola de Amazon SQS como fuente de canalización, consulte Amazon SQS cola como fuente en la Guía del usuario de Amazon. EventBridge Para obtener más información sobre EventBridge Pipes en general, consulte Pipes. EventBridge

Para acceder a EventBridge las canalizaciones de una cola de Amazon SQS determinada

1. En la consola de Amazon SQS, abra la página Colas.
2. Seleccione una cola.
3. En la página de detalles de la cola, seleccione la EventBridge pestaña Pipes.

   La pestaña EventBridge Tuberías incluye una lista de todas las tuberías actualmente configuradas para usar la cola seleccionada como fuente, que incluye:

   - nombre de la canalización
   - estado actual
   - destino de la canalización
   - cuándo se modificó la canalización por última vez
4. Consulte más detalles de la canalización o cree una nueva, si lo desea:

   - Para acceder a más detalles sobre una canalización:

     Elija el nombre de la canalización.

     Esto abre la página de detalles de Pipe de la EventBridge consola.
   - Para crear una nueva canalización:

Elija Conectar la cola de Amazon SQS a la canalización.

Esto abre la página Crear canalización de la EventBridge consola, con la cola Amazon SQS especificada como fuente de canalización. Para obtener más información, consulta Cómo crear una EventBridge tubería en la Guía del EventBridge usuario de Amazon.

> **Important**
>
> Una sola canalización lee un mensaje de una cola de Amazon SQS y, después, se elimina de la cola una vez procesado, independientemente de que el mensaje coincida o no con el filtro que puede configurar para esa canalización. Continúe con precaución al configurar varias canalizaciones para que utilicen la misma cola como origen.

Administración de mensajes de Amazon SQS de gran tamaño con biblioteca de clientes ampliada y Amazon Simple Storage Service

Utilice la biblioteca de clientes extendida de Amazon SQS para Java y la biblioteca de clientes extendida de Amazon SQS para Python para enviar mensajes de gran tamaño, especialmente para cargas útiles de entre 256 KB y 2 GB. Estas bibliotecas almacenan la carga útil del mensaje en un bucket de Amazon S3 y envían una referencia al objeto almacenado en la cola de Amazon SQS.

> **Note**
>
> Las bibliotecas de clientes ampliadas de Amazon SQS son compatibles con las colas estándar y FIFO.

Administración de mensajes de Amazon SQS grandes mediante Java y Amazon S3

Utilice la biblioteca de clientes extendida Amazon SQS para Java con Amazon S3 para gestionar mensajes de Amazon SQS de gran tamaño, especialmente para cargas útiles que oscilan entre 256 KB y 2 GB. La biblioteca almacena la carga útil del mensaje en un bucket de Amazon S3 y envía un mensaje que contiene una referencia al objeto almacenado en la cola de Amazon SQS.

Con la biblioteca de clientes extendida Amazon SQS para Java, puede:

- Especificar si los mensajes se almacenan siempre en Amazon S3 o solo cuando su tamaño supera los 256 KB
- Enviar un mensaje que hace referencia a un único objeto de mensaje almacenado en un bucket de S3
- Recuperar el objeto de mensaje desde un bucket de Amazon S3
- Eliminar el objeto de mensaje de un bucket de Amazon S3

Requisitos previos

En el siguiente ejemplo, se utiliza el SDK de AWS Java. Para instalar y configurar el SDK, consulte Configurar el AWS SDK para Java en la Guía para AWS SDK para Java desarrolladores.

Antes de ejecutar el código de ejemplo, configure sus AWS credenciales. Para obtener más información, consulte Configurar AWS las credenciales y la región para el desarrollo en la Guía para AWS SDK para Java desarrolladores.

El SDK para Java y la biblioteca de clientes ampliada de Amazon SQS para Java requieren J2SE Development Kit 8.0 o una versión posterior.

> **Note**
>
> Puede utilizar la biblioteca de clientes ampliada de Amazon SQS para Java a fin de administrar los mensajes de Amazon SQS mediante Amazon S3 solo con AWS SDK para Java. No puede hacerlo con la AWS CLI consola Amazon SQS, la API HTTP de Amazon SQS ni ninguna otra. AWS SDKs

AWS SDK for Java 2.x Ejemplo: uso de Amazon S3 para gestionar mensajes de Amazon SQS de gran tamaño

En el siguiente ejemplo de SDK for SDK for Java 2.x, se utiliza la biblioteca de cliente extendida para Java para trabajar con mensajes de gran tamaño. En el constructor, el siguiente código:

- Crea un bucket de Amazon S3 con un nombre aleatorio
- Crea una cola de SQS que comienza con `MyQueue`
- Envuelve un cliente Amazon S3 estándar de Java SDK en una instancia de `AmazonSQSExtendedClient`

En el `sendAnReceiveMessage` método, el ejemplo envía un mensaje aleatorio que se almacena en un bucket de Amazon S3 porque pesa más de 256 KB (el tamaño máximo de mensaje estándar). Por último, el método recupera el mensaje y muestra información sobre el mismo en la consola.

Puede ver el ejemplo completo enhttps://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());
    }
}
```

Puede utilizar Apache Maven para configurar y compilar proyectos de cliente extendido de Amazon SQS para su proyecto de Java o para compilar el propio SDK. Especifique los módulos individuales del SDK que desea usar en su aplicación.

```
<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>
```

Administración de mensajes de Amazon SQS grandes mediante Python y Amazon S3

Utilice la biblioteca de clientes Amazon SQS Extended Client para Python con Amazon S3 para gestionar mensajes de Amazon SQS de gran tamaño, especialmente para cargas útiles de entre 256 KB y 2 GB. La biblioteca almacena la carga útil del mensaje en un bucket de Amazon S3 y envía un mensaje que contiene una referencia al objeto almacenado en la cola de Amazon SQS.

Con la biblioteca de clientes extendida Amazon SQS para Python, puede:

- Especifique si las cargas útiles se almacenan siempre en Amazon S3 o solo se almacenan en Amazon S3 cuando el tamaño de la carga supera los 256 KB
- Enviar un mensaje que hace referencia a un único objeto de mensaje almacenado en un bucket de S3
- Recuperar el objeto de carga útil correspondiente de un bucket de Amazon S3
- Eliminar el objeto de carga útil correspondiente de un bucket de Amazon S3

Requisitos previos

A continuación, se indican los requisitos previos para utilizar la biblioteca de clientes ampliada de Amazon SQS para Python:

- Una AWS cuenta con las credenciales necesarias. Para crear una AWS cuenta, vaya a la página de AWS inicio y, a continuación, seleccione Crear una AWS cuenta. Siga las instrucciones. Para obtener información sobre las credenciales, consulte Credenciales.
- Un AWS SDK: el ejemplo de esta página usa Boto3 del SDK de AWS Python. Para instalar y configurar el SDK, consulte la documentación del AWS SDK para Python en la Guía del desarrollador del AWS SDK para Python
- Python 3.x (o posterior) y `pip`.
- La biblioteca de clientes ampliada de Amazon SQS para Python, disponible en PyPI

> **Note**
>
> Puede utilizar la biblioteca de clientes extendida Amazon SQS para Python para gestionar los mensajes de Amazon SQS mediante Amazon S3 únicamente con el AWS SDK para Python. No puede hacerlo con la AWS CLI, la consola Amazon SQS, la API HTTP de Amazon SQS ni ninguna otra opción. AWS SDKs

Configuración del almacenamiento de mensajes

El cliente ampliado de Amazon SQS utiliza los siguientes atributos de mensaje para configurar las opciones de almacenamiento de mensajes de Amazon S3:

- `large_payload_support`: el nombre del bucket de Amazon S3 para almacenar los mensajes de gran tamaño.
- `always_through_s3`: si es `True`, todos los mensajes se almacenan en Amazon S3. Si es `False`, los mensajes de menos de 256 KB no se serializarán en el bucket de S3. El valor predeterminado es `False`.
- `use_legacy_attribute`: si es `True`, todos los mensajes publicados utilizan el atributo de mensaje reservado heredado (`SQSLargePayloadSize`) en lugar del atributo de mensaje reservado actual (`ExtendedPayloadSize`).

Administración de mensajes grandes de Amazon SQS con la biblioteca de clientes ampliada para Python

En el siguiente ejemplo se crea un bucket de Amazon S3 con un nombre aleatorio. Después crea una cola de Amazon SQS denominada `MyQueue` y envía a la cola un mensaje almacenado en un bucket de S3 y de más de 256 KB. Por último, el código recupera el mensaje, devuelve información sobre él y, a continuación, elimina el mensaje, la cola y el bucket.

```
import boto3
import sqs_extended_client

#Set the Amazon SQS extended client configuration with large payload.
sqs_extended_client = boto3.client("sqs", region_name="us-east-1")
sqs_extended_client.large_payload_support = "amzn-s3-demo-bucket"
sqs_extended_client.use_legacy_attribute = False
```

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


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

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

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

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

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

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

# Deleting the queue
delete_queue_response = sqs_extended_client.delete_queue(
```

```
    QueueUrl=queue_url
)

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

Configuración de colas de Amazon SQS mediante la consola de Amazon SQS

Utilice la consola de Amazon SQS para configurar y gestionar las colas y las funciones de Amazon SQS. También puede:

- Habilite el cifrado del lado del servidor para mejorar la seguridad.
- Asocie una cola de mensajes sin procesar para gestionar los mensajes no procesados.
- Configure un disparador para invocar una función Lambda para el procesamiento basado en eventos.

Control de acceso basado en atributos para Amazon SQS

¿Qué es ABAC?

El control de acceso basado en atributos (ABAC) es un proceso de autorización que define los permisos en función de las etiquetas que se adjuntan a los usuarios y los recursos. AWS ABAC proporciona un control de acceso detallado y flexible basado en atributos y valores, reduce el riesgo de seguridad relacionado con las políticas basadas en roles reconfiguradas y centraliza la auditoría y la administración de las políticas de acceso. Para obtener más información sobre ABAC, consulte Qué es ABAC para AWS en la Guía del usuario de IAM.

Amazon SQS admite ABAC al permitirle controlar el acceso a las colas de Amazon SQS en función de las etiquetas y los alias asociados a una cola de Amazon SQS. Las claves de condición de etiqueta y alias que habilitan ABAC en Amazon SQS autorizan a las entidades principales de IAM a utilizar las colas de Amazon SQS sin necesidad de editar políticas ni administrar concesiones. Para obtener más información acerca de las claves de condición de ABAC, consulte Claves de estado de Amazon SQS en la Referencia de autorizaciones de servicio.

Con ABAC, puede utilizar etiquetas para configurar los permisos y las políticas de acceso de IAM para sus colas de Amazon SQS, lo que lo ayudará a escalar la administración de permisos. Puede crear una única política de permisos en IAM mediante etiquetas que agregue a cada rol empresarial, sin tener que actualizar la política cada vez que agregue un nuevo recurso. También puede asociar etiquetas a entidades principales de IAM para crear una política de ABAC. Puede diseñar políticas de ABAC para permitir las operaciones de Amazon SQS cuando la etiqueta del rol de usuario de

IAM que está realizando la llamada coincida con la etiqueta de cola de Amazon SQS. Para obtener más información sobre el etiquetado AWS, consulte AWS Estrategias de etiquetado y. Etiquetas de asignación de costos de Amazon SQS

> **Note**
>
> ABAC para Amazon SQS está disponible actualmente en AWS todas las regiones comerciales en las que Amazon SQS está disponible, con las siguientes excepciones:
>
> - Asia-Pacífico (Hyderabad)
> - Asia-Pacífico (Melbourne)
> - Europa (España)
> - Europa (Zúrich)

¿Por qué debo utilizar ABAC en Amazon SQS?

He aquí algunos beneficios de utilizar ABAC en Amazon SQS:

- ABAC para Amazon SQS requiere menos políticas de permisos. No tiene que crear diferentes políticas para diferentes funciones de trabajo. Puede utilizar etiquetas de recursos y solicitudes que se apliquen a más de una cola, lo que reduce la sobrecarga operativa.
- Utilizar ABAC para escalar equipos rápidamente. Los permisos para nuevos recursos se conceden automáticamente en función de las etiquetas cuando los recursos se etiquetan adecuadamente durante su creación.
- Utilizar permisos en la entidad principal de IAM para restringir el acceso a los recursos. Puede crear etiquetas para la entidad principal de IAM y utilizarlas a fin de restringir el acceso a acciones específicas que coincidan con las etiquetas de la entidad principal de IAM. Esto lo ayuda a automatizar el proceso de concesión de permisos de solicitud.
- Realizar un seguimiento de quién accede a sus recursos. Puede determinar la identidad de una sesión si consulta los atributos de usuario en AWS CloudTrail.

Temas

- Etiquetado para el control de acceso en Amazon SQS
- Creación de usuarios de IAM y colas de Amazon SQS
- Pruebas de control de acceso basado en atributos de Amazon SQS

Etiquetado para el control de acceso en Amazon SQS

El siguiente es un ejemplo del uso de etiquetas para el control de acceso en Amazon SQS. La política de IAM restringe a un usuario de IAM todas las acciones de Amazon SQS para todas las colas que incluyan una etiqueta de recurso con la clave environment y el valor production. Para obtener más información, consulte Control de acceso basado en atributos con etiquetas y Organizations AWS.

JSON

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

Creación de usuarios de IAM y colas de Amazon SQS

En los siguientes ejemplos se explica cómo crear una política ABAC para controlar el acceso a Amazon SQS mediante AWS Management Console y. AWS CloudFormation

Mediante el AWS Management Console

Crear un usuario de IAM

1. Inicie sesión en la consola de IAM AWS Management Console y ábrala en. https://console.aws.amazon.com/iam/
2. Desde el panel de navegación izquierdo, elija Usuario.

3. Elija Agregar usuarios e introduzca un nombre en el cuadro de texto Nombre de usuario.
4. Seleccione (Clave de acceso: acceso mediante programación) y elija Siguiente: permisos.
5. Elija Siguiente:Etiquetas.
6. Agregue la clave de la etiqueta como `environment` y el valor de la etiqueta como `beta`.
7. Seleccione Siguiente: revisión y, después, elija Crear usuario.
8. Copie y guarde el ID de clave de acceso y la clave de acceso secreta en un lugar seguro.

Agregar permisos de usuario de IAM

1. Seleccione el usuario de IAM que ha creado.
2. Elija Agregar política insertada.
3. En la pestaña JSON, pegue la política siguiente:

   JSON

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

```
        "sqs:SetQueueAttributes",
        "sqs:tagqueue"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/environment": "${aws:PrincipalTag/environment}"
        }
      }
    },
  ]
}
```

4. Elija Revisar política.
5. Elija Crear política.

Usando AWS CloudFormation

Utilice la siguiente AWS CloudFormation plantilla de ejemplo para crear un usuario de IAM con una política en línea adjunta y una cola de 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"
```

Pruebas de control de acceso basado en atributos de Amazon SQS

En los siguientes ejemplos se muestra cómo probar el control de acceso basado en atributos en Amazon SQS.

Cree una cola con la clave de etiqueta establecida a environment y el valor de etiqueta establecido a prod

Ejecute este comando AWS CLI para probar la creación de la cola con la clave de etiqueta establecida en environment y el valor de la etiqueta establecido en prod. Si no tiene AWS CLI, puede descargarla y configurarla para su máquina.

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

Recibe un error `AccessDenied` del punto de conexión de Amazon SQS:

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

Esto se debe a que el valor de etiqueta en el usuario de IAM no coincide con la etiqueta pasada en la llamada a la API `CreateQueue`. Recuerde que aplicamos una etiqueta al usuario de IAM con la clave establecida a `environment` y el valor establecido a `beta`.

Cree una cola con la clave de etiqueta establecida a environment y el valor de etiqueta establecido a beta

Ejecute este comando de la CLI para probar la creación de una cola con la clave de etiqueta establecida a `environment` y el valor de etiqueta establecido a `beta`.

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

Recibe un mensaje que confirma la creación correcta de la cola, similar al que se muestra a continuación.

```
{
"QueueUrl": "<queueUrl>“
}
```

Envío de un mensaje a una cola

Ejecute este comando de la CLI para probar el envío de un mensaje a una cola.

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

La respuesta muestra una entrega correcta del mensaje a la cola de Amazon SQS. El permiso de usuario de IAM le permite enviar un mensaje a una cola que tenga una etiqueta `beta`. La respuesta incluye `MD5OfMessageBody` y `MessageId` con el mensaje.

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

Configuración de parámetros de colas mediante la consola de Amazon SQS

Al crear o editar una cola, puede configurar los siguientes parámetros:

- Tiempo de espera de visibilidad: el tiempo durante el que un mensaje recibido de una cola (por un consumidor) no estará visible para los demás consumidores de mensajes. Para obtener más información, consulte Tiempo de espera de visibilidad.

  > **Note**
  >
  > Utilizar la consola para configurar el tiempo de espera de visibilidad configura el valor del tiempo de espera para todos los mensajes de la cola. Para configurar el tiempo de espera de uno o varios mensajes, debe usar uno de los. AWS SDKs

- Periodo de retención del mensaje: el tiempo que Amazon SQS retiene los mensajes que permanecen en la cola. De forma predeterminada, la cola retiene los mensajes durante cuatro días. Puede configurar una cola para que conserve los mensajes durante un máximo de 14 días. Para obtener más información, consulte Periodo de retención de mensajes.

- Retraso de entrega: el tiempo que Amazon SQS tardará en entregar un mensaje que se agregue a la cola. Para obtener más información, consulte Retraso de entrega.
- Tamaño máximo del mensaje: el tamaño máximo de mensaje para esta cola. Para obtener más información, consulte Tamaño máximo de los mensajes.
- Tiempo de espera de recepción del mensaje: el tiempo máximo que Amazon SQS espera a que los mensajes estén disponibles después de que la cola reciba una solicitud de recepción. Para obtener más información, consulte Sondeos cortos y largos de Amazon SQS.
- Habilite la deduplicación basada en el contenido: Amazon SQS puede crear automáticamente la IDs deduplicación en función del cuerpo del mensaje. Para obtener más información, consulte Colas FIFO de Amazon SQS.
- Habilitar FIFO de alto rendimiento: se utiliza para permitir un alto rendimiento de los mensajes en la cola. Al elegir esta opción, las opciones relacionadas (Ámbito de desduplicación y Límite de rendimiento FIFO) cambian a la configuración necesaria para habilitar un alto rendimiento para las colas FIFO. Para obtener más información, consulte Alto rendimiento para las colas FIFO en Amazon SQS y Cuotas de mensajes de Amazon SQS.
- Política de permiso de redireccionamiento: define qué colas de origen pueden utilizar esta cola como cola de mensajes fallidos. Para obtener más información, consulte Uso de colas de mensajes fallidos en Amazon SQS .

Configuración de los parámetros de una cola existente (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas. Elija una cola y, a continuación, Editar.
3. Desplácese a la sección Configuración.
4. En Tiempo de espera de visibilidad, introduzca la duración y las unidades. El intervalo es de 0 segundos a 12 horas. El valor predeterminado es de 30 segundos.
5. En Periodo de retención del mensaje, introduzca la duración y las unidades. El intervalo va de 1 minuto a 14 días. El valor predeterminado es 4 días.
6. Para una cola estándar, introduzca un valor en Tiempo de espera de recepción del mensaje. El intervalo es de 0 a 20 segundos. El valor predeterminado es 0 segundos, lo que establece un sondeo corto. Cualquier valor distinto de cero establece un sondeo largo.
7. En Retraso de entrega, introduzca la duración y las unidades. El intervalo es de 0 segundos a 15 minutos. El valor predeterminado es 0 segundos.

8. En Tamaño máximo del mensaje, introduzca un valor. El rango va de 1 KiB a 1024 KiB. El valor predeterminado es 1024 KiB.
9. Para una cola FIFO, elija Habilitar la desduplicación basada en el contenido para activar la desduplicación basada en el contenido. La configuración predeterminada está desactivada.
10. (Opcional) Para que una cola FIFO permita un mayor rendimiento en el envío y recepción de mensajes en la cola, elija Habilitar FIFO de alto rendimiento.

    Al elegir esta opción, las opciones relacionadas (Ámbito de desduplicación y Límite de rendimiento FIFO) cambian a la configuración necesaria para habilitar un alto rendimiento para las colas FIFO. Si cambia alguna de las configuraciones necesarias para utilizar FIFO de alto rendimiento, el rendimiento normal estará en vigor para la cola y la desduplicación se producirá según lo especificado. Para obtener más información, consulte Alto rendimiento para las colas FIFO en Amazon SQS y Cuotas de mensajes de Amazon SQS.
11. En Política de permiso de redireccionamiento, elija Habilitado. Seleccione una de las siguientes opciones: Permitir todo (predeterminada), Por cola o Denegar todo. Al elegir Por cola, especifique una lista de hasta diez colas de origen por nombre de recurso de Amazon (ARN).
12. Cuando termine de configurar los parámetros de la cola, elija Guardar.

Configuración de una política de acceso en Amazon SQS

Al editar una cola, puede configurar su política de acceso para controlar quién puede interactuar con ella.

- La política de acceso define qué cuentas, usuarios y roles tienen permisos para acceder a la cola.
- Especifica las acciones permitidas, como `SendMessageReceiveMessage`, o `DeleteMessage`.
- De forma predeterminada, solo el propietario de la cola tiene permiso para enviar y recibir mensajes.

Para configurar la política de acceso para una cola existente (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. Elija una cola y, a continuación, Editar.
4. Desplácese hasta la sección Política de acceso.

5. Edite las declaraciones de política de acceso en el cuadro de entrada. Para obtener más información sobre las instrucciones de política de acceso, consulte Identity and Access Management en Amazon SQS.
6. Cuando termine de configurar la política de acceso, seleccione Guardar.

Configuración del cifrado del servidor para una cola mediante claves de cifrado administradas por SQS

Además de la opción predeterminada de cifrado del servidor (SSE) administrado por Amazon SQS, el SSE administrado por Amazon SQS (SSE-SQS) le permite crear un cifrado del servidor administrado personalizado que utiliza claves de cifrado administradas por SQS para proteger los datos confidenciales enviados a través de las colas de mensajes. Con SSE-SQS, no necesita crear ni administrar claves de cifrado, ni modificar su código para cifrar sus datos. SSE-SQS le permite transmitir datos de forma segura y le ayuda a cumplir con los estrictos requisitos de cifrado y normativos sin costo adicional.

SSE-SQS protege los datos en reposo mediante el cifrado AES-256 (estándar de cifrado avanzado de 256 bits). SSE cifra los mensajes en cuanto Amazon SQS los recibe. Amazon SQS almacena los mensajes de forma cifrada y los descifra solo cuando los envía a un consumidor autorizado.

> **Note**
>
> - La opción SSE predeterminada solo es efectiva cuando se crea una cola sin especificar atributos de cifrado.
> - Amazon SQS le permite desactivar todo el cifrado de las colas. Por lo tanto, desactivar KMS-SSE no activará automáticamente SQS-SSE. Si desea activar SQS-SSE después de desactivar KMS-SSE, deberá agregar un cambio de atributo en la solicitud.

Configuración del cifrado SSE-SQS en una cola (consola)

> **Note**
>
> Las colas nuevas que se creen mediante el punto de conexión HTTP (no TLS) no habilitarán el cifrado SSE-SQS de forma predeterminada. Una práctica recomendada en materia de

seguridad consiste en crear colas de Amazon SQS mediante puntos de conexión HTTPS o Signature Version 4.

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. Elija una regla y, a continuación, elija Editar.
4. Expanda Cifrado.
5. Para Cifrado del servidor, elija Activado (predeterminado).

> **Note**
>
> Con SSE activado, se rechazarán las solicitudes `SendMessage` y `ReceiveMessage` anónimas a la cola cifrada. Las prácticas recomendadas de seguridad de Amazon SQS desaconsejan utilizar solicitudes anónimas. Si desea enviar solicitudes anónimas a una cola de Amazon SQS, asegúrese de desactivar SSE.

6. Seleccione Clave de Amazon SQS (SSE-SQS). No hay ningún costo adicional por utilizar esta opción.
7. Seleccione Save.

Configuración del cifrado del servidor para una cola mediante la consola de Amazon SQS

Para proteger los datos de los mensajes de una cola, Amazon SQS tiene habilitado de forma predeterminada el cifrado del servidor (SSE) para todas las colas de nueva creación. Amazon SQS se integra con Amazon Web Services Key Management Service (Amazon Web Services KMS) para administrar las claves de KMS para el cifrado del servidor (SSE). Para obtener información acerca del uso de SSE, consulte Cifrado en reposo en Amazon SQS.

La clave de KMS que asigne a su cola debe tener una política de claves que incluya permisos para todas las entidades principales que estén autorizadas a utilizar la cola. Para obtener más información, consulte Administración de claves.

Si no es el propietario de la clave de KMS o si ha iniciado sesión con una cuenta que no tiene los permisos `kms:ListAliases` y `kms:DescribeKey`, no podrá ver la información sobre la clave de

KMS en la consola de Amazon SQS. Pida al propietario de la clave de KMS que le conceda estos permisos. Para obtener más información, consulte Administración de claves.

Al crear o editar una cola, puede configurar SSE-KMS.

Configuración de SSE-KMS para una cola existente (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. Elija una regla y, a continuación, elija Editar.
4. Expanda Cifrado.
5. Para Cifrado del servidor, elija Activado (predeterminado).

> **Note**
>
> Con SSE activado, se rechazarán las solicitudes `SendMessage` y `ReceiveMessage` anónimas a la cola cifrada. Las prácticas recomendadas de seguridad de Amazon SQS desaconsejan utilizar solicitudes anónimas. Si desea enviar solicitudes anónimas a una cola de Amazon SQS, asegúrese de desactivar SSE.

6. Seleccione Clave de AWS Key Management Service (SSE-KMS).

   La consola muestra la Descripción, la Cuenta y el ARN de la clave de KMS de la clave de KMS.

7. Especifique el ID de clave de KMS de la cola. Para obtener más información, consulte Términos clave.

   a. Elija la opción Elegir un alias de clave de KMS.

   b. La clave predeterminada es la clave de KMS administrada por Amazon Web Services para Amazon SQS. Para usar esta clave, selecciónela de la lista Clave de KMS.

   c. Para usar una clave de KMS personalizada de su cuenta de Amazon Web Services, selecciónela de la lista Clave de KMS. Para obtener instrucciones sobre la creación de claves de KMS personalizadas, consulte Creación de claves en la Guía para desarrolladores de Amazon Web Services Key Management Service.

   d. Para utilizar una clave de KMS personalizada que no esté en la lista o una clave de KMS personalizada de otra cuenta de Amazon Web Services, elija Introducir el alias de la clave de KMS e introduzca el Nombre de recurso de Amazon (ARN) de la clave de KMS.

8. (Opcional) Para Periodo de reutilización de la clave de datos, especifique un valor entre 1 minuto y 24 horas. El valor predeterminado es 5 minutos. Para obtener más información, consulte Descripción del período de reutilización de la clave de datos.
9. Cuando termine de configurar SSE-KMS, elija Guardar.

Configuración de etiquetas de asignación de costos para una cola mediante la consola de Amazon SQS

Para organizar e identificar sus colas de Amazon SQS, puede añadir etiquetas de asignación de costes. Para obtener más información, consulte Etiquetas de asignación de costos de Amazon SQS.

- La pestaña Etiquetado de la página de detalles muestra las etiquetas de la cola.
- Puede añadir o modificar etiquetas al crear o editar una cola.

Configuración de etiquetas para una cola existente (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. Elija una cola y, a continuación, Editar.
4. Desplácese hacia abajo hasta la sección Etiquetas.
5. Agregue, modifique o elimine etiquetas de cola:
   a. Para agregar una etiqueta, elija Agregar nueva etiqueta, introduzca una Clave y un Valor y, a continuación, elija Agregar nueva etiqueta.
   b. Para actualizar una etiqueta, cambie su Clave y Valor.
   c. Para eliminar una etiqueta, elija Eliminar junto a su par clave-valor.
6. Cuando termine de configurar las etiquetas, elija Guardar.

Suscripción de una cola a un tema de Amazon SNS mediante la consola de Amazon SQS

Puede suscribir una o más colas de Amazon SQS a un tema de Amazon SNS. Cuando publicas un mensaje en un tema, Amazon SNS envía el mensaje a cada cola suscrita. Amazon SQS administra la suscripción y gestiona los permisos necesarios. Para obtener más información sobre

Amazon SNS, consulte ¿Qué es Amazon SNS? en la Guía para desarrolladores de Amazon Simple Notification Service.

Al suscribir una cola de Amazon SQS a un tema de Amazon SNS, Amazon SNS utiliza HTTPS para reenviar los mensajes a Amazon SQS. Para obtener información sobre el uso de Amazon SNS con colas cifradas de Amazon SQS, consulte Configure los permisos de KMS para los servicios AWS.

> **Important**
>
> Amazon SQS admite un máximo de 20 declaraciones para cada política de acceso. La suscripción a un tema de Amazon SNS agrega una instrucción de este tipo. Si se supera esta cantidad, se producirá un error en la entrega de la suscripción al tema.

Para suscribir una cola a un tema de Amazon SNS (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. En la lista de colas, elija la cola para suscribirse al tema de Amazon SNS.
4. En el menú Actions (Acciones), elija Subscribe to Amazon SNS topic (Suscribirse a tema de Amazon SNS).
5. En el menú Especifique un tema de Amazon SNS disponible para esta cola, elija el tema de Amazon SNS para su cola.

   Si el tema de SNS no aparece en la lista, elija Introducir el ARN del tema de Amazon SNS y, a continuación, introduzca el nombre de recurso de Amazon (ARN) del tema.
6. Seleccione Save.
7. Para verificar la suscripción, publique un mensaje en el tema y visualice el mensaje en la cola. Para obtener más información, consulte Publicación de mensajes en Amazon SNS en la Guía para desarrolladores de Amazon Simple Notification Service.

Suscripciones multicuenta

Si la cola de Amazon SQS y el tema de Amazon SNS son diferentes Cuentas de AWS, se requieren permisos adicionales.

Propietario del tema (cuenta A)

Modifique la política de acceso del tema de Amazon SNS para permitir que las colas Cuenta de AWS de Amazon SQS se suscriban. Ejemplo de declaración de política:

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

Esta política permite suscribirse `111122223333` a la cuenta`MyTopic`.

Propietario de la cola (cuenta B)

Modifique la política de acceso de la cola de Amazon SQS para permitir que el tema Amazon SNS envíe mensajes. Ejemplo de declaración de política:

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

Esta política permite `MyTopic` enviar mensajes a`MyQueue`.

Suscripciones entre regiones

Para suscribirse a un tema de Amazon SNS en otro sitio Región de AWS, asegúrese de lo siguiente:

- La política de acceso al tema de Amazon SNS permite suscripciones entre regiones.
- La política de acceso a la cola de Amazon SQS permite que el tema Amazon SNS envíe mensajes entre regiones.

Para obtener más información, consulte Enviar mensajes de Amazon SNS a una cola AWS Lambda o función de Amazon SQS en una región diferente en la Guía para desarrolladores de Amazon Simple Notification Service.

Configuración de una cola de Amazon SQS para activar una función AWS Lambda

Puede utilizar una función Lambda para procesar los mensajes de una cola de Amazon SQS. Lambda sondea la cola e invoca la función de forma sincrónica, pasando un lote de mensajes como un evento.

Configurar el tiempo de espera de visibilidad

Establezca el tiempo de espera de visibilidad de la cola en al menos seis veces el tiempo de espera de la función. Esto garantiza que Lambda tenga tiempo suficiente para volver a intentarlo si una función está restringida durante el procesamiento de un lote anterior.

Uso de una cola de cartas sin salida (DLQ)

Especifique una cola de letra muerta para capturar los mensajes que la función Lambda no pueda procesar.

Gestión de múltiples colas y funciones

Una función Lambda puede procesar varias colas mediante la creación de una fuente de eventos independiente para cada cola. También puede asociar varias funciones de Lambda a la misma cola.

Permisos para colas cifradas

Si asocia una cola cifrada a una función de Lambda pero Lambda no sondea los mensajes, agregue el permiso `kms:Decrypt` a su rol de ejecución de Lambda.

Restricciones

La cola y la función Lambda deben estar en la misma posición. Región de AWS

Una cola cifrada que usa la clave predeterminada (clave de KMS AWS administrada para Amazon SQS) no puede invocar una función de Lambda en otra. Cuenta de AWS

Para obtener información detallada sobre la implementación, consulte Uso AWS Lambda con Amazon SQS en la Guía para AWS Lambda desarrolladores.

Requisitos previos

Para configurar desencadenadores de funciones de Lambda, debe cumplir los siguientes requisitos:

- Si es un usuario, su rol de Amazon SQS debe incluir los siguientes permisos:
  - `lambda:CreateEventSourceMapping`
  - `lambda:ListEventSourceMappings`
  - `lambda:ListFunctions`
- El rol de ejecución de Lambda debe incluir los permisos siguientes:
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `sqs:ReceiveMessage`
- Si asocia una cola cifrada a una función de Lambda, agregue el permiso `kms:Decrypt` a la función de ejecución de Lambda.

Para obtener más información, consulte Información general sobre la administración del acceso en Amazon SQS.

Configuración de una cola para desencadenar una función de Lambda (consola)

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. En la página Colas, elija la cola que desee configurar.
4. En la página de la cola, elija la pestaña Desencadenadores de Lambda.
5. En la página Desencadenadores de Lambda, elija un desencadenador de Lambda.

   Si la lista no incluye el desencadenador de Lambda que necesita, elija Configurar desencadenador de función de Lambda. Introduzca el nombre de recurso de Amazon (ARN) de la función de Lambda o elija un recurso existente. A continuación, elija Guardar.
6. Seleccione Save. La consola guarda la configuración y muestra la página Detalles de la cola.

   En la página Detalles, la pestaña Desencadenadores de Lambda muestra la función de Lambda y su estado. Se tarda aproximadamente un minuto en asociar la función de Lambda a la cola.
7. Para verificar los resultados de la configuración, envíe un mensaje a la cola y, a continuación, consulte la función de Lambda desencadenada en la consola de Lambda.

Automatización de las notificaciones de AWS los servicios a Amazon SQS mediante Amazon EventBridge

Amazon EventBridge le permite automatizar Servicios de AWS y responder a eventos, como problemas de aplicaciones o cambios en los recursos, prácticamente en tiempo real.

- Puede crear reglas para filtrar eventos específicos y definir acciones automatizadas cuando una regla coincida con un evento.
- EventBridge admite varios destinos, incluidas las colas FIFO y estándar de Amazon SQS, que reciben eventos en formato JSON.

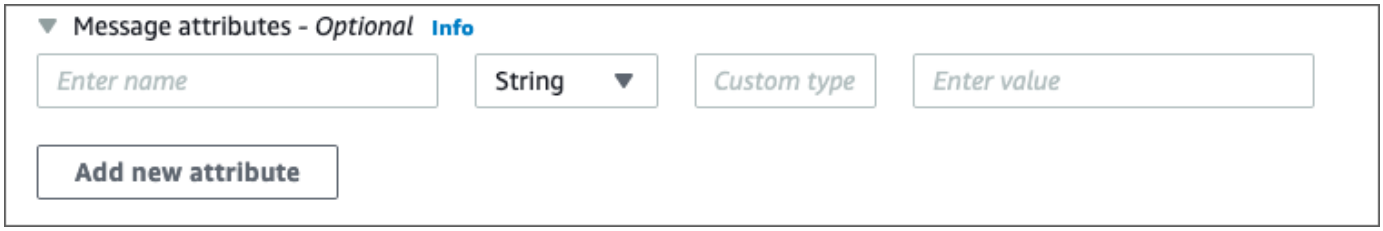
Para obtener más información, consulta EventBridge los objetivos de Amazon en la Guía del EventBridge usuario de Amazon.

Envío de un mensaje con atributos mediante Amazon SQS

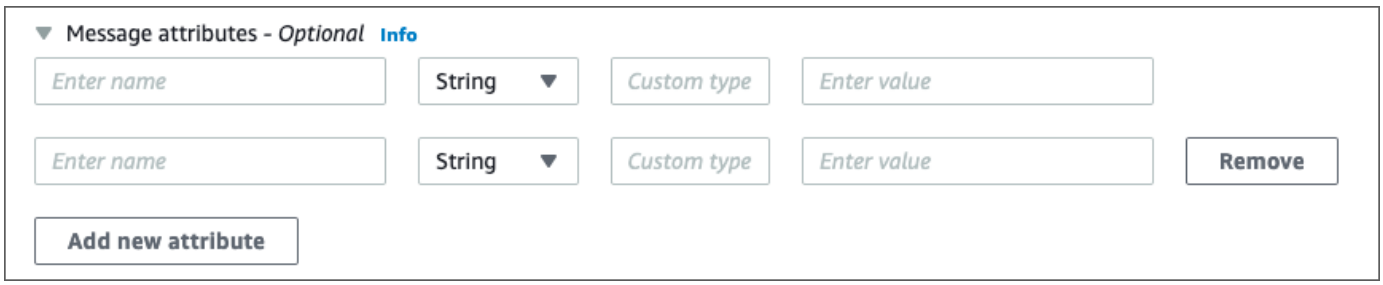
Para las colas estándar y FIFO, puede incluir metadatos estructurados en los mensajes, incluidas las marcas de tiempo, los datos geoespaciales, las firmas y los identificadores. Para obtener más información, consulte Atributos de mensajes de Amazon SQS.

Envío de mensajes con atributos a una cola mediante la consola de Amazon SQS

1. Abra la consola Amazon SQS en. https://console.aws.amazon.com/sqs/
2. En el panel de navegación, elija Colas.
3. En la página Colas, elija una cola.
4. Seleccione Enviar y recibir mensajes.
5. Introduzca los parámetros de los atributos del mensaje.
   a. En el cuadro de texto del nombre, introduzca un nombre único de 256 caracteres como máximo.
   b. Para el tipo de atributo, elija Cadena, Número o Binario.
   c. (Opcional) Introduzca un tipo de datos personalizado. Por ejemplo, puede agregar **`byte`**, **`int`** o **`float`** como tipos de datos personalizados para Número.
   d. En el cuadro de texto del valor, introduzca el valor de atributo del mensaje.



6. Para agregar otro atributo de mensaje, elija Agregar nuevo atributo.



7. Puede modificar los valores de atributo en cualquier momento antes de enviar el mensaje.
8. Para eliminar un atributo, elija Eliminar. Para eliminar el primer atributo, cierre Atributos del mensaje.
9. Cuando termine de agregar atributos al mensaje, elija Enviar mensaje. Se envía el mensaje y la consola muestra un mensaje de confirmación. Para ver información sobre los atributos del mensaje enviado, seleccione Ver detalles. Seleccione Listo para cerrar el cuadro de diálogo Detalles del mensaje.

Prácticas recomendadas para Amazon SQS

Amazon SQS gestiona y procesa las colas de mensajes, lo que permite que diferentes partes de una aplicación intercambien mensajes de forma fiable y a escala. En este tema se describen las mejores prácticas operativas clave, como el uso de sondeos prolongados para reducir las respuestas vacías, la implementación de colas con letra muerta para gestionar los errores de procesamiento y la optimización de los permisos de las colas por motivos de seguridad.

Temas

- Gestión de errores y mensajes problemáticos en Amazon SQS
- Desduplicación y agrupación de mensajes de Amazon SQS
- Procesamiento y temporización de mensajes de Amazon SQS

Gestión de errores y mensajes problemáticos en Amazon SQS

Este tema proporciona instrucciones detalladas sobre la gestión y la mitigación de errores en Amazon SQS, incluidas las técnicas para administrar los errores de las solicitudes, capturar los mensajes problemáticos y configurar la retención de colas de mensajes fallidos para garantizar la fiabilidad de los mensajes.

Temas

- Gestión de errores de solicitudes en Amazon SQS
- Captura de mensajes problemáticos en Amazon SQS
- Configuración de la retención de la cola de mensajes fallidos en Amazon SQS

Gestión de errores de solicitudes en Amazon SQS

Para gestionar los errores de solicitud, utilice una de las siguientes estrategias:

- Si utilizas un AWS SDK, ya tienes a tu disposición una lógica automática de reintentos y retrocesos. Para obtener más información, consulte Reintentos de error y retroceso exponencial en AWS en la Referencia general de Amazon Web Services.
- Si no utiliza las funciones del AWS SDK para volver a intentarlo y retrasar, espere una pausa (por ejemplo, 200 ms) antes de volver a intentar la ReceiveMessageacción si no recibe ningún mensaje, si se agota el tiempo de espera o si recibe un mensaje de error de Amazon SQS. Si

quiere utilizar posteriormente `ReceiveMessage` con los mismos resultados, haga una pausa mayor (por ejemplo, 400 ms).

Captura de mensajes problemáticos en Amazon SQS

Para capturar todos los mensajes que no se pueden procesar y recopilar CloudWatch métricas precisas, configura una cola de mensajes sin procesar.

- La política de redireccionamiento redirige los mensajes a una cola de mensajes fallidos después de que la cola de origen no puede procesar un mensaje un número especificado de veces.
- El uso de una cola de mensajes fallidos reduce el número de mensajes y la posibilidad de exponer el sistema a mensajes de píldoras venenosas (mensajes que se reciben pero no se pueden procesar).
- Incluir un mensaje sobre una píldora venenosa en una lista puede distorsionar la `ApproximateAgeOfOldestMessage` CloudWatch métrica al indicar una antigüedad incorrecta del mensaje sobre la píldora venenosa. Cuando se utiliza esta métrica, la configuración de una cola de mensajes fallidos ayuda a evitar falsas alarmas.

Configuración de la retención de la cola de mensajes fallidos en Amazon SQS

En el caso de las colas estándar, la caducidad de un mensaje siempre se basa en su marca temporal original. Cuando un mensaje se mueve a una cola de mensajes fallidos, la marca temporal de la cola no se modifica. La métrica `ApproximateAgeOfOldestMessage` indica cuándo el mensaje pasó a la cola de mensajes fallidos, no cuándo se envió originalmente. Por ejemplo, supongamos que un mensaje pasa un día en la cola original antes de ser trasladado a una cola de mensajes fallidos. Si el periodo de retención de la cola de mensajes fallidos es de cuatro días, el mensaje se elimina de la cola de mensajes fallidos al cabo de tres días y `ApproximateAgeOfOldestMessage` es de tres días. Por lo tanto, se recomienda establecer siempre un periodo de retención de una cola de mensajes fallidos superior al periodo de retención de la cola original.

Para las colas FIFO, la marca temporal de entrada se restablece cuando el mensaje se mueve a una cola de mensajes fallidos. La métrica `ApproximateAgeOfOldestMessage` indica cuándo el mensaje ha pasado a la cola de mensajes fallidos. En el mismo ejemplo anterior, el mensaje se elimina de la cola de mensajes fallidos al cabo de cuatro días y `ApproximateAgeOfOldestMessage` es de cuatro días.

Desduplicación y agrupación de mensajes de Amazon SQS

En este tema se describen las prácticas recomendadas para garantizar un procesamiento coherente de los mensajes en Amazon SQS. En él se explica cómo usar:

- `MessageDeduplicationId`para evitar la duplicación de mensajes en las colas FIFO.
- `MessageGroupId`para gestionar el orden de los mensajes dentro de distintos grupos de mensajes.

Temas

- Cómo evitar el procesamiento incoherente de los mensajes en Amazon SQS
- Uso del ID de deduplicación de mensajes en Amazon SQS
- Uso del ID de grupo de mensajes con las colas FIFO de Amazon SQS
- Uso del ID de intento de solicitud de recepción de Amazon SQS

Cómo evitar el procesamiento incoherente de los mensajes en Amazon SQS

Debido a que Amazon SQS es un sistema distribuido, es posible que un consumidor no reciba un mensaje aunque lo marque como entregado al regresar correctamente de una llamada de método de API `ReceiveMessage`. En este caso, Amazon SQS registra el mensaje como entregado al menos una vez, aunque el consumidor no lo haya recibido. Dado que no se realizan intentos adicionales de entregar mensajes en estas condiciones, no recomendamos establecer el número máximo de recepciones en 1 para una cola de mensajes fallidos.

Uso del ID de deduplicación de mensajes en Amazon SQS

`MessageDeduplicationId`es un token que se utiliza únicamente en las colas FIFO de Amazon SQS para evitar la entrega de mensajes duplicados. Garantiza que, en un período de deduplicación de 5 minutos, solo se procese y entregue una instancia de un mensaje con el mismo ID de deduplicación.

Si Amazon SQS ya ha aceptado un mensaje con un ID de deduplicación específico, se confirmará cualquier mensaje posterior con el mismo ID, pero no se entregará a los consumidores.

> **Note**
>
> Amazon SQS sigue rastreando el ID de deduplicación incluso después de recibir y eliminar el mensaje.

Temas

- Cuándo proporcionar un ID de deduplicación de mensajes en Amazon SQS
- Habilitación de la desduplicación para un sistema de un solo productor y un solo consumidor en Amazon SQS
- Escenarios de recuperación ante interrupciones en Amazon SQS
- Configuración de los tiempos de espera de visibilidad en Amazon SQS

Cuándo proporcionar un ID de deduplicación de mensajes en Amazon SQS

El productor debe especificar un ID de deduplicación de mensajes en los siguientes escenarios:

- Al enviar cuerpos de mensajes idénticos, deben tratarse como únicos.
- Al enviar mensajes con el mismo contenido pero con atributos de mensaje diferentes, asegúrese de que cada mensaje se procese por separado.
- Cuando se envían mensajes con contenido diferente (por ejemplo, un contador de reintentos en el cuerpo del mensaje) pero que requieren que Amazon SQS los reconozca como duplicados.

Habilitación de la desduplicación para un sistema de un solo productor y un solo consumidor en Amazon SQS

Si tiene un único productor y un solo consumidor, y los mensajes son únicos porque incluyen un identificador de mensaje específico de la aplicación en el cuerpo, siga estas prácticas recomendadas:

- Habilite la desduplicación basada en el contenido para la cola (cada uno de sus mensajes tiene un cuerpo único). El productor puede omitir el ID de desduplicación de mensajes.
- Cuando la desduplicación basada en contenido está habilitada para una cola FIFO de Amazon SQS y se envía un mensaje con un identificador de desduplicación, el identificador de desduplicación `SendMessage` invalida el identificador de desduplicación basado en contenido generado.

- Aunque no es necesario que el consumidor proporcione un ID de intento de solicitud de recepción para cada solicitud, es recomendable hacerlo porque permite que las secuencias de reintento tras un error se ejecuten con mayor rapidez.
- Puede reintentar las solicitudes de envío o recepción, ya que no interfieren con la ordenación de los mensajes en las colas FIFO.

Escenarios de recuperación ante interrupciones en Amazon SQS

El proceso de desduplicación en las colas FIFO está sujeto a limitación temporal. Al diseñar la aplicación, asegúrese de que tanto el productor como el consumidor puedan recuperarse de las interrupciones en los clientes o la red sin introducir duplicados ni errores de procesamiento.

Consideraciones sobre el productor

- Amazon SQS aplica un plazo de deduplicación de 5 minutos.
- Si un productor vuelve a intentar una `SendMessage`solicitud transcurridos 5 minutos, Amazon SQS la trata como un mensaje nuevo, lo que podría crear duplicados.

Consideraciones de los consumidores

- Si un consumidor no procesa un mensaje antes de que caduque el tiempo de espera de visibilidad, es posible que otro consumidor lo reciba y procese, lo que provocará una duplicación del procesamiento.
- Ajusta el tiempo de espera de visibilidad en función del tiempo de procesamiento de tu solicitud.
- Usa la `ChangeMessageVisibility`API para extender el tiempo de espera mientras se está procesando un mensaje.
- Si un mensaje no se procesa repetidamente, diríjalo a una cola de mensajes sin procesar (DLQ) en lugar de permitir que se vuelva a procesar indefinidamente.

- El productor debe estar al tanto del intervalo de desduplicación de la cola. Amazon SQS tiene un intervalo mínimo de desduplicación de cinco minutos. El reintento de solicitudes `SendMessage` después de que finalice el intervalo de desduplicación puede introducir mensajes duplicados en la cola. Por ejemplo, un dispositivo móvil en un automóvil envía mensajes cuyo orden es importante. Si el automóvil pierde la conectividad móvil durante un periodo de tiempo antes de recibir un reconocimiento, el reintento de la solicitud después de recuperar la conectividad móvil puede crear un duplicado.

- El consumidor debe tener un tiempo de espera de visibilidad que minimice el riesgo de no poder procesar los mensajes antes de que finalice el tiempo de espera de visibilidad. Para ampliar el tiempo de espera de visibilidad mientras se procesan los mensajes, llame a la acción `ChangeMessageVisibility`. Sin embargo, si el tiempo de espera de visibilidad finaliza, otro consumidor puede comenzar inmediatamente a procesar los mensajes, lo que hará que un mensaje se procese varias veces. Para evitar esta situación, configure una cola de mensajes fallidos.

Configuración de los tiempos de espera de visibilidad en Amazon SQS

Para garantizar un procesamiento fiable de los mensajes, establece el tiempo de espera de visibilidad para que sea superior al tiempo de espera de lectura del AWS SDK. Esto se aplica cuando se utiliza la `ReceiveMessage`API tanto con sondeos cortos como largos. Un tiempo de espera de visibilidad más prolongado evita que los mensajes estén disponibles para otros consumidores antes de que se complete la solicitud original, lo que reduce el riesgo de que se duplique el procesamiento.

Uso del ID de grupo de mensajes con las colas FIFO de Amazon SQS

En las colas FIFO (primero en entrar, primero en salir), es un atributo que organiza los mensajes en grupos distintos. `MessageGroupId` Los mensajes del mismo grupo de mensajes siempre se procesan de uno en uno, siguiendo un orden estricto, lo que garantiza que no se procesen dos mensajes del mismo grupo simultáneamente. En las colas estándar, el uso `MessageGroupId` permite hacer colas justas. Si se requiere un orden estricto, utilice una cola FIFO.

Temas

- Intercalación de varios grupos de mensajes ordenados en Amazon SQS
- Evitar el procesamiento duplicado en un sistema de varios productores o consumidores en Amazon SQS
- Evite grandes acumulaciones de mensajes con el mismo ID de grupo de mensajes en Amazon SQS
- Cómo evitar reutilizar el mismo ID de grupo de mensajes con colas virtuales en Amazon SQS

Intercalación de varios grupos de mensajes ordenados en Amazon SQS

Para intercalar varios grupos de mensajes ordenados en una sola cola FIFO, asigne una `MessageGroupId`a cada grupo (por ejemplo, datos de sesión de diferentes usuarios). Esto permite

que varios consumidores lean de la cola simultáneamente y, al mismo tiempo, garantiza que los mensajes del mismo grupo se procesen en orden.

Cuando se `MessageGroupId` está procesando un mensaje con una información específica y es invisible, ningún otro consumidor puede procesar los mensajes de ese mismo grupo hasta que caduque el tiempo de espera de visibilidad o se elimine el mensaje.

Evitar el procesamiento duplicado en un sistema de varios productores o consumidores en Amazon SQS

En un sistema de alto rendimiento y baja latencia en el que el orden de los mensajes no es una prioridad, los productores pueden asignar un mensaje único a cada mensaje. `MessageGroupId` Esto garantiza que las colas FIFO de Amazon SQS eliminen los duplicados, incluso en una configuración con varios productores o varios consumidores. Si bien este enfoque evita la duplicación de mensajes, no garantiza el orden de los mensajes, ya que cada mensaje se trata como un grupo independiente.

En cualquier sistema con varios productores y consumidores, siempre existe el riesgo de duplicar la entrega. Si un consumidor no procesa un mensaje antes de que caduque el tiempo de espera de visibilidad, Amazon SQS vuelve a hacer que el mensaje esté disponible, lo que podría permitir que otro consumidor lo recoja. Para mitigar esta situación, asegúrese de configurar correctamente el reconocimiento del mensaje y el tiempo de espera de visibilidad en función del tiempo de procesamiento.

Evite grandes acumulaciones de mensajes con el mismo ID de grupo de mensajes en Amazon SQS

Las colas FIFO admiten un máximo de 120 000 mensajes en movimiento (mensajes recibidos por un consumidor pero que aún no se han eliminado). Si se alcanza este límite, Amazon SQS no devuelve ningún error, pero el procesamiento puede verse afectado. Puedes solicitar un aumento por encima de este límite poniéndote en contacto con AWS Support.

Las colas FIFO analizan los primeros 120 000 mensajes para determinar los grupos de mensajes disponibles. Si se acumula una gran cantidad de mensajes pendientes en un solo grupo de mensajes, los mensajes de otros grupos que se envíen posteriormente permanecerán bloqueados hasta que se procese la acumulación.

> **Note**
>
> Se puede producir una acumulación de mensajes cuando un consumidor deja de procesar un mensaje repetidamente. Esto puede deberse a problemas con el contenido de los mensajes o a errores por parte del consumidor. Para evitar demoras en el procesamiento de los mensajes, configura una cola de texto sin procesar para mover los mensajes no procesados después de varios intentos fallidos. Esto garantiza que se puedan procesar otros mensajes del mismo grupo de mensajes, lo que evita los cuellos de botella en el sistema.

Cómo evitar reutilizar el mismo ID de grupo de mensajes con colas virtuales en Amazon SQS

Cuando utilice colas virtuales con una cola de host compartida, evite reutilizar las mismas colas virtuales en distintas colas virtuales. `MessageGroupId` Si varias colas virtuales comparten la misma cola de hosts y contienen mensajes con el mismo servidor`MessageGroupId`, esos mensajes pueden bloquearse entre sí, lo que impide un procesamiento eficiente. Para garantizar un procesamiento fluido de los mensajes, asigne `MessageGroupId` valores únicos a los mensajes de las diferentes colas virtuales.

Uso del ID de intento de solicitud de recepción de Amazon SQS

El ID de intento de solicitud de recepción es un token único que se utiliza para deduplicar `ReceiveMessage`llamadas en Amazon SQS. Durante una interrupción de la red o un problema de conectividad entre la aplicación y Amazon SQS, se recomienda:

- Proporcione un identificador de intento de solicitud de recepción al realizar una `ReceiveMessage` llamada.
- Vuelva a intentarlo con el mismo ID de intento de solicitud de recepción si la operación falla.

Procesamiento y temporización de mensajes de Amazon SQS

En este tema se proporciona una guía completa sobre cómo optimizar la velocidad y la eficiencia de la gestión de mensajes en Amazon SQS, incluidas las estrategias para el procesamiento puntual de los mensajes, la selección del mejor modo de sondeo y la configuración del sondeo largo para mejorar el rendimiento.

Temas

- Procesamiento de los mensajes a tiempo en Amazon SQS
- Configuración del sondeo largo en Amazon SQS
- Uso del modo de sondeo apropiado en Amazon SQS

Procesamiento de los mensajes a tiempo en Amazon SQS

La configuración del tiempo de espera de visibilidad depende de lo que tarde la aplicación en procesar y eliminar un mensaje. Por ejemplo, si la aplicación necesita 10 segundos para procesar un mensaje y establece el tiempo de espera de visibilidad en 15 minutos, debe esperar un tiempo relativamente largo para intentar procesar el mensaje de nuevo si el intento de procesamiento anterior produce un error. Asimismo, si la aplicación requiere 10 segundos para procesar un mensaje pero el usuario establece el tiempo de espera de visibilidad en solo 2 segundos, otro consumidor recibe un mensaje duplicado mientras que el consumidor original sigue trabajando en el mensaje.

Para asegurarse de que hay tiempo suficiente para procesar los mensajes, utilice una de las siguientes estrategias:

- Si sabe (o puede calcular razonablemente) el tiempo que se tarda en procesar un mensaje, amplíe el tiempo de espera de visibilidad del mensaje al tiempo máximo que se tarda en procesar y eliminar el mensaje. Para obtener más información, consulte Configuración del tiempo de espera de visibilidad.
- Si no sabe cuánto tiempo tarda en procesarse un mensaje, cree un latido para su proceso consumidor: especifique el tiempo de espera de visibilidad inicial (por ejemplo, dos minutos) y, a continuación, mientras su consumidor siga trabajando en el mensaje, continúe ampliando el tiempo de espera de visibilidad en dos minutos cada minuto.

> **Important**
>
> El tiempo máximo de visibilidad es de 12 horas desde el momento en que Amazon SQS recibe la solicitud `ReceiveMessage`. La ampliación del tiempo de espera de visibilidad no restablece el máximo de 12 horas.
> Además, es posible que no pueda establecer el tiempo de espera de un mensaje individual en 12 horas completas (por ejemplo, 43 200 segundos), ya que la solicitud `ReceiveMessage` inicia el temporizador. Por ejemplo, si recibe un mensaje e inmediatamente establece el máximo de 12 horas mediante el envío de una llamada `ChangeMessageVisibility` con `VisibilityTimeout` igual a 43 200 segundos, es

probable que se produzca un error. No obstante, utilizar un valor de 43 195 segundos funcionará a menos que exista un retraso significativo entre la solicitud del mensaje a través de `ReceiveMessage` y la actualización del tiempo de espera de visibilidad. Si su consumidor necesita más de 12 horas, considere usar Step Functions.

Configuración del sondeo largo en Amazon SQS

Cuando el tiempo de espera de la acción de la API `ReceiveMessage` es superior a 0, se está realizando un sondeo largo. El tiempo máximo de espera de sondeo es de 20 segundos. El sondeo largo ayuda a reducir el costo de uso de Amazon SQS al eliminar el número de respuestas vacías (cuando no hay ningún mensaje disponible para una solicitud `ReceiveMessage`) y las falsas respuestas vacías (cuando los mensajes están disponibles en la cola, pero no se incluyen en una respuesta). Para obtener más información, consulte Sondeos cortos y largos de Amazon SQS.

Para obtener un procesamiento óptimo de los mensajes, utilice las siguientes estrategias:

- En la mayoría de los casos, puede establecer el tiempo de espera de `ReceiveMessage` en 20 segundos. Si 20 segundos es demasiado tiempo para su aplicación, establezca un tiempo de espera de `ReceiveMessage` más corto (1 segundo como mínimo). Si no utiliza un AWS SDK para acceder a Amazon SQS o si configura un AWS SDK para que tenga un tiempo de espera más corto, es posible que tenga que modificar su cliente de Amazon SQS para permitir solicitudes más largas o utilizar un tiempo de espera más corto para sondeos prolongados.
- Si implementa el sondeo largo para varias colas, utilice un subproceso para cada cola en lugar de un solo subproceso para todas las colas. El uso de un único subproceso para cada cola permite que la aplicación procese los mensajes de cada una de las colas en cuanto estén disponibles, mientras que el uso de un único subproceso para sondear varias colas podría impedir que su aplicación procesara los mensajes disponibles en otras colas mientras la aplicación espera (hasta 20 segundos) para la cola que no tiene ningún mensaje disponible.

> **Important**
>
> Para evitar errores HTTP, asegúrese de que el tiempo de espera de respuesta HTTP de las solicitudes `ReceiveMessage` sea mayor que el parámetro `WaitTimeSeconds`. Para obtener más información, consulte ReceiveMessage.

Uso del modo de sondeo apropiado en Amazon SQS

- El sondeo largo le permite consumir mensajes de la cola de Amazon SQS tan pronto como estén disponibles.
  - Para reducir el costo derivado del uso de Amazon SQS y reducir el número de recepciones vacías en una cola vacía (respuestas a la acción `ReceiveMessage` que no devuelven ningún mensaje), habilite el sondeo largo. Para obtener más información, consulte Sondeo largo de Amazon SQS.
  - Para aumentar la eficacia cuando se sondean varios subprocesos con varias recepciones, reduzca el número de procesos.
  - En la mayoría de los casos, el sondeo largo es preferible al sondeo corto.
- El sondeo corto devuelve respuestas inmediatamente, incluso aunque la cola de Amazon SQS sondeada esté vacía.
  - Para satisfacer los requisitos de una aplicación que espera respuestas inmediatas a la solicitud `ReceiveMessage`, utilice el sondeo corto.
  - El sondeo corto se factura igual que el sondeo largo.

Ejemplos de SDK de Java de Amazon SQS

AWS SDK para Java Le permite crear aplicaciones Java que interactúan con Amazon SQS y otros. Servicios de AWS

- Para instalar y configurar el SDK, consulte Introducción en la Guía para desarrolladores de AWS SDK for Java 2.x .
- Para obtener información sobre las operaciones de cola básicas, como la creación de una cola o el envío de un mensaje, consulte Uso de colas de mensajes de Amazon SQS en la Guía para desarrolladores.AWS SDK for Java 2.x
- Esta guía también incluye ejemplos de funciones adicionales de Amazon SQS, como:
  - Uso del cifrado del servidor con colas de Amazon SQS
  - Configuración de etiquetas para una cola de Amazon SQS
  - Envío de un mensaje con atributos a una cola de Amazon SQS

Uso del cifrado del servidor con colas de Amazon SQS

Úselo AWS SDK para Java para añadir el cifrado del lado del servidor (SSE) a una cola de Amazon SQS. Cada cola utiliza una clave KMS de AWS Key Management Service (AWS KMS) para generar las claves de cifrado de datos. Este ejemplo utiliza la clave de KMS administrada por AWS para Amazon SQS.

Para obtener más información sobre cómo utilizar SSE y el rol de la clave de KMS, consulte Cifrado en reposo en Amazon SQS.

Agregar SSE a una cola existente

Para activar el cifrado del servidor en una cola existente, utilice el método `SetQueueAttributes` para establecer el atributo `KmsMasterKeyId`.

El siguiente ejemplo de código establece la AWS KMS key clave de KMS AWS administrada para Amazon SQS. En el ejemplo también se establece el periodo de reutilización de AWS KMS key a 140 segundos.

Antes de ejecutar el código de ejemplo, asegúrese de haber establecido sus AWS credenciales. Para obtener más información, consulte Configurar AWS las credenciales y la región para el desarrollo en la Guía para AWS SDK for Java 2.x desarrolladores.

```
    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();
        }
    }
```

Desactivación de SSE para una cola

Para desactivar el cifrado del servidor para una cola existente, establezca el atributo `KmsMasterKeyId` a una cadena vacía mediante el método `SetQueueAttributes`.

> **Important**
>
> `null` no es un valor válido para `KmsMasterKeyId`.

Creación de una cola con SSE

Para activar SSE al crear la cola, agregue el atributo `KmsMasterKeyId` al método de la API `CreateQueue`.

En el siguiente ejemplo se crea una nueva cola con SSE activado. La cola utiliza la clave de KMS administrada por AWS para Amazon SQS. En el ejemplo también se establece el periodo de reutilización de AWS KMS key a 160 segundos.

Antes de ejecutar el código de ejemplo, asegúrese de haber establecido sus AWS credenciales. Para obtener más información, consulte Configurar AWS las credenciales y la región para el desarrollo en la Guía para AWS SDK for Java 2.x desarrolladores.

```
// 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);
```

Recuperación de atributos de SSE

Para obtener información sobre la recuperación de atributos de cola, consulte Ejemplos en la Referencia de la API de Amazon Simple Queue Service.

Para recuperar el ID de la clave de KMS o el periodo de reutilización de la clave de datos para una cola en concreto, ejecute el método `GetQueueAttributes` y recupere los valores `KmsMasterKeyId` y `KmsDataKeyReusePeriodSeconds`.

Configuración de etiquetas para una cola de Amazon SQS

Utilice etiquetas de asignación de costos para organizar e identificar sus colas de Amazon SQS. En los siguientes ejemplos se muestra cómo configurar etiquetas mediante la función AWS SDK para Java. Para obtener más información, consulte Etiquetas de asignación de costos de Amazon SQS.

Antes de ejecutar el código de ejemplo, asegúrese de haber establecido sus AWS credenciales. Para obtener más información, consulte Configurar AWS las credenciales y la región para el desarrollo en la Guía para AWS SDK for Java 2.x desarrolladores.

Enumeración de etiquetas

Para obtener una lista de las etiquetas de una cola, utilice el método `ListQueueTags`.

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

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

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

// Create the ListQueueTagsRequest.
final ListQueueTagsRequest listQueueTagsRequest =
```

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

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

Adición o actualización de etiquetas

Para agregar o actualizar valores de etiqueta para una cola, utilice el método `TagQueue`.

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

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

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

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

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

Eliminación de etiquetas

Para eliminar una o varias etiquetas de la cola, utilice el método `UntagQueue`. En el siguiente ejemplo se elimina la etiqueta `Accounting ID`.

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

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

Envío de un mensaje con atributos a una cola de Amazon SQS

Puede incluir metadatos estructurados (como marcas temporales, datos geoespaciales, firmas e identificadores) con los mensajes mediante atributos de mensaje. Para obtener más información, consulte Atributos de mensajes de Amazon SQS.

Antes de ejecutar el código de ejemplo, asegúrese de haber establecido sus AWS credenciales. Para obtener más información, consulte Configurar AWS las credenciales y la región para el desarrollo en la Guía para AWS SDK for Java 2.x desarrolladores.

Definición de atributos

Para definir un atributo para un mensaje, agregue el siguiente código, que utiliza el tipo de datos `MessageAttributeValue`. Para obtener más información, consulte Componentes de atributos de mensajes y Tipos de datos de atributos de mensajes.

Calcula AWS SDK para Java automáticamente las sumas de comprobación del cuerpo y los atributos del mensaje y las compara con los datos que devuelve Amazon SQS. Para obtener más información, consulte la Guía para desarrolladores de AWS SDK for Java 2.x y Calcular el resumen del MD5 mensaje para los atributos del mensaje para otros lenguajes de programación.

String

En este ejemplo se define un atributo `String` denominado `Name` con el valor `Jane`.

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

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

Number

En este ejemplo se define un atributo `Number` denominado `AccurateWeight` con el valor `230.000000000000000001`.

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

Binary

En este ejemplo se define un atributo `Binary` denominado `ByteArray` con el valor de una matriz de 10 bytes sin inicializar.

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

String (custom)

En este ejemplo se define el atributo personalizado `String.EmployeeId` denominado `EmployeeId` con el valor `ABC123456`.

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

Number (custom)

En este ejemplo se define el atributo personalizado `Number.AccountId` denominado `AccountId` con el valor `000123456`.

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

> **Note**
>
> Dado que el tipo de datos base es `Number`, el método `ReceiveMessage` devuelve `123456`.

Binary (custom)

En este ejemplo se define el atributo personalizado `Binary.JPEG` denominado `ApplicationIcon` con el valor de una matriz de 10 bytes sin inicializar.

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

Envío de un mensaje con atributos

En este ejemplo se agregan los atributos a `SendMessageRequest` antes de enviar el mensaje.

```
// 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**
>
> Si envía un mensaje a una cola First-In-First-Out (FIFO), asegúrese de que el `sendMessage` método se ejecute después de proporcionar el ID del grupo de mensajes.
> Si utiliza el método `SendMessageBatch` en lugar de `SendMessage`, debe especificar los atributos de los mensajes del lote.

Uso APIs con Amazon SQS

En este tema se proporciona información sobre la creación de puntos de enlace de Amazon SQS, la realización de solicitudes de API de consulta mediante los métodos GET y POST y el uso de acciones de API por lotes. Para obtener información detallada sobre las acciones de Amazon SQS (como parámetros, errores, ejemplos y tipos de datos), consulte la Referencia de la API de Amazon Simple Queue Service.

Para acceder a Amazon SQS mediante diversos lenguajes de programación, también puede utilizar AWS SDKs, que contienen las siguientes funciones automáticas:

- Firmar criptográficamente sus solicitudes de servicio
- Reintentar solicitudes
- Tratar las respuestas a errores

Para obtener más información, consulte the section called “Trabajando con AWS SDKs”.

Para obtener información sobre la herramienta de línea de comandos, consulte las secciones de Amazon SQS en la Referencia de comandos de AWS CLI y la Referencia de cmdlets de Herramientas de AWS para PowerShell .

Amazon SQS APIs con protocolo JSON AWS

Amazon SQS utiliza el protocolo AWS JSON como mecanismo de transporte para todos los Amazon APIs SQS de las versiones del AWS SDK especificadas. AWS El protocolo JSON proporciona un mayor rendimiento, menor latencia y una comunicación más rápida. application-to-application AWS El protocolo JSON es más eficiente en serialization/deserialization las solicitudes y respuestas en comparación con el protocolo de AWS consulta. Si aún prefiere utilizar el protocolo de AWS consulta con SQS APIs, consulte ¿Qué idiomas son compatibles con el protocolo AWS JSON que se utiliza en Amazon SQS? APIs las versiones del AWS SDK que admiten el protocolo de consulta Amazon AWS SQS.

Amazon SQS utiliza el protocolo AWS JSON para comunicarse entre los clientes AWS del SDK (por ejemplo, Java, Python, Golang JavaScript) y el servidor Amazon SQS. Una solicitud HTTP de una operación de la API de Amazon SQS acepta la entrada con formato JSON. La operación de Amazon SQS se ejecuta y la respuesta de la ejecución se comparte de nuevo con el cliente del SDK en formato JSON. En comparación con las AWS consultas, AWS JSON es más simple, rápido y eficiente para transportar datos entre el cliente y el servidor.

- AWS El protocolo JSON actúa como mediador entre el cliente y el servidor de Amazon SQS.
- El servidor no entiende el lenguaje de programación en el que se crea la operación de Amazon SQS, pero entiende el protocolo AWS JSON.
- El protocolo AWS JSON utiliza la serialización (convierte el objeto a formato JSON) y la deserialización (convierte el formato JSON en objeto) entre el cliente y el servidor de Amazon SQS.

Para obtener más información sobre el protocolo AWS JSON con Amazon SQS, consulte. Protocolo Amazon SQS JSON AWS FAQs

AWS El protocolo JSON está disponible en la versión del AWS SDK especificada. Para revisar la versión y las fechas de lanzamiento del SDK en todas las variantes de idioma, consulta la matriz de compatibilidad de versiones AWS SDKs y herramientas en la Guía de referencia de AWS SDKs and Tools

Realizar solicitudes de API de consulta mediante el protocolo AWS JSON en Amazon SQS

En este tema se explica cómo construir un punto final de Amazon SQS, realizar solicitudes POST e interpretar las respuestas.

> **Note**
>
> AWS El protocolo JSON es compatible con la mayoría de las variantes de idioma. Para ver una lista completa de las variantes de lenguaje admitidas, consulte ¿Qué idiomas son compatibles con el protocolo AWS JSON que se utiliza en Amazon SQS? APIs.

Construcción de un punto de enlace

Para trabajar con colas de Amazon SQS, debe crear un punto de conexión. Para obtener información sobre los puntos de conexión de Amazon SQS, consulte las páginas siguientes en Referencia general de Amazon Web Services:

- Puntos de conexión regionales
- Puntos de conexión y cuotas de Amazon Simple Queue Service

Cada punto de conexión de Amazon SQS es independiente. Por ejemplo, si dos colas se denominan MyQueue y una tiene el punto de enlace `sqs.us-east-2.amazonaws.com` mientras que la otra tiene el punto de enlace `sqs.eu-west-2.amazonaws.com`, no comparten ningún dato entre ellas.

A continuación, se muestra un ejemplo de un punto de conexión que realiza una solicitud para crear una cola.

```
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**
>
> Los nombres de las colas y las colas URLs distinguen mayúsculas de minúsculas.
> La estructura de *AUTHPARAMS* depende de la firma de la solicitud de API. Para obtener más información, consulte Firmar solicitudes de AWS API en la Referencia general de Amazon Web Services.

Realizar una solicitud POST

Las solicitudes POST de Amazon SQS envían parámetros de consulta como un formulario en el cuerpo de una solicitud HTTP.

A continuación, se muestra un ejemplo de un encabezado HTTP con `X-Amz-Target` establecido a `AmazonSQS.<operationName>` y de un encabezado HTTP con `Content-Type` establecido a `application/x-amz-json-1.0`.

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

Esta solicitud HTTP POST envía un mensaje a una cola de Amazon SQS.

> **Note**
>
> Ambos encabezados HTTP `X-Amz-Target` y `Content-Type` son obligatorios.
> El cliente HTTP puede añadir otros elementos a la solicitud HTTP, según la versión de HTTP del cliente.

Interpretación de las respuestas de la API JSON de Amazon SQS

Cuando envía una solicitud a Amazon SQS, devuelve una respuesta JSON con los resultados. La estructura de respuesta depende de la acción de API que haya utilizado.

Para entender los detalles de estas respuestas, consulta:

- La acción de API específica en las acciones de API de la referencia de la API de Amazon Simple Queue Service
- la Protocolo Amazon SQS JSON AWS FAQs,

Estructura de una respuesta JSON correcta

Si la solicitud se realiza correctamente, el elemento de respuesta principal es `x-amzn-RequestId`, que contiene el identificador único universal (UUID) de la solicitud, así como otros campos de respuesta añadidos. Por ejemplo, la siguiente respuesta de `CreateQueue` contiene el campo `QueueUrl` que, a su vez, contiene la URL de la cola creada.

```
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"
}
```

Estructura de una respuesta de error JSON

Si una solicitud no se realiza correctamente, Amazon SQS devuelve la respuesta principal, incluidos el encabezado HTTP y el cuerpo.

En el encabezado HTTP, `x-amzn-RequestId` contiene el UUID de la solicitud. `x-amzn-query-error` contiene dos informaciones: el tipo de error y si se trata de un error del productor o del consumidor.

En el cuerpo de la respuesta, `"__type"` indica otros detalles del error y `Message` señala la condición del error en un formato legible.

El siguiente es un ejemplo de respuesta de error en formato JSON:

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

Protocolo Amazon SQS JSON AWS FAQs

En este tema se tratan las preguntas más frecuentes sobre el uso del protocolo AWS JSON con Amazon SQS.

¿Qué es el protocolo AWS JSON y en qué se diferencia de las solicitudes y respuestas de la API Amazon SQS existentes?

JSON es uno de los métodos de conexión más utilizados y aceptados para la comunicación entre sistemas heterogéneos. Amazon SQS utiliza JSON como medio de comunicación entre un cliente AWS del SDK (por ejemplo, Java, Python, Golang JavaScript) y el servidor Amazon SQS. Una solicitud HTTP de una operación de la API de Amazon SQS acepta entradas en formato JSON. La operación de Amazon SQS se ejecuta y la respuesta de la ejecución se comparte de nuevo con el cliente del SDK en formato JSON. En comparación con las consultas de AWS , JSON es más eficiente a la hora de transportar datos entre el cliente y el servidor.

- El protocolo Amazon SQS AWS JSON actúa como mediador entre el cliente y el servidor de Amazon SQS.
- El servidor no entiende el lenguaje de programación en el que se crea la operación de Amazon SQS, pero entiende el protocolo AWS JSON.
- El protocolo Amazon SQS AWS JSON utiliza la serialización (convierte el objeto a formato JSON) y la deserialización (convierte el formato JSON en objeto) entre el cliente y el servidor de Amazon SQS.

¿Cómo puedo empezar a utilizar los protocolos AWS JSON para Amazon SQS?

Para empezar a utilizar la última versión del AWS SDK y lograr una mensajería más rápida para Amazon SQS, actualice el AWS SDK a la versión especificada o a cualquier versión posterior. Para obtener más información sobre los clientes del SDK, consulte la columna Guía de la tabla siguiente.

La siguiente es una lista de las versiones del SDK en todas las variantes de idioma del protocolo AWS JSON para su uso con Amazon SQS: APIs

Idioma	Repositorio de clientes del SDK	Versión de cliente del SDK requerida	Guía
C++	aws/ aws-sdk-cpp	1.11.98	AWS SDK para C++
Golang 1.x	aws/ aws-sdk-go	v1.47.7	AWS SDK para Go
Golang 2.x	aws/ 2 aws-sdk-go-v	v1.28.0	

Idioma	Repositorio de clientes del SDK	Versión de cliente del SDK requerida	Guía
			AWS SDK para Go V2
Java 1.x	aws/ aws-sdk-java	1.12.585	AWS SDK para Java
Java 2.x	aws/ 2 aws-sdk-java-v	2.21.19	AWS SDK para Java
JavaScript v2.x	aws/ aws-sdk-js	JavaScript en AWS	
JavaScript v3.x	aws/ 3 aws-sdk-js-v	v3.447.0	JavaScript en AWS
.NET	aws/ aws-sdk-net	3.7.681.0	AWS SDK para .NET
PHP	aws/ aws-sdk-php	3.285.2	AWS SDK para PHP
Python-boto3	boto/boto3	1.28,82	AWS SDK para Python (Boto3)
Python-botocore	boto/botocore	1,31,82	AWS SDK para Python (Boto3)
awscli	CLI de AWS	1.29.82	Interfaz de la línea de comandos de AWS
Ruby	aws/ aws-sdk-ruby	1.67.0	AWS SDK para Ruby

¿Cuáles son los riesgos de habilitar el protocolo JSON para mis cargas de trabajo de Amazon SQS?

Si utiliza una implementación personalizada del AWS SDK o una combinación de clientes personalizados y un AWS SDK para interactuar con Amazon SQS que genera respuestas basadas en AWS consultas (también conocidas como basadas en XML), es posible que no sea compatible con el protocolo JSON. AWS Si tiene algún problema, póngase en contacto con AWS Support.

¿Qué sucede si ya tengo la última versión del AWS SDK, pero mi solución de código abierto no es compatible con JSON?

Debe cambiar la versión del SDK por la anterior a la que esté utilizando. Consulte ¿Cómo puedo empezar a utilizar los protocolos AWS JSON para Amazon SQS? para obtener más información. AWS Las versiones del SDK que aparecen en ¿Cómo puedo empezar a utilizar los protocolos AWS JSON para Amazon SQS? utilizan el protocolo JSON wire para Amazon SQS APIs. Si cambia el AWS SDK a la versión anterior, Amazon SQS APIs utilizará la AWS consulta.

¿Qué idiomas son compatibles con el protocolo AWS JSON que se utiliza en Amazon SQS? APIs

Amazon SQS admite todas las variantes de idioma donde AWS SDKs estén disponibles de forma general (GA). Actualmente, no se admite Kotlin, Rust ni Swift. Para obtener más información sobre otras variantes de lenguaje, consulte Herramientas para crear en AWS.

¿Qué regiones son compatibles con el protocolo AWS JSON utilizado en Amazon SQS? APIs

Amazon SQS admite el protocolo AWS JSON en todas AWS las regiones en las que Amazon SQS está disponible.

¿Qué mejoras de latencia puedo esperar al actualizar a las versiones de AWS SDK especificadas para Amazon SQS mediante el protocolo AWS JSON?

AWS El protocolo JSON es más eficiente en la serialización y deserialización de solicitudes y respuestas en comparación con el protocolo de consultas. AWS Según las pruebas de AWS rendimiento para una carga útil de mensajes de 5 KB, el protocolo JSON para Amazon SQS end-to-end reduce la latencia del procesamiento de mensajes hasta un 23% y reduce el uso de CPU y memoria del lado del cliente de la aplicación.

¿Quedará AWS obsoleto el protocolo de consultas?

AWS el protocolo de consulta seguirá siendo compatible. Puede seguir utilizando el protocolo de AWS consultas siempre que la versión AWS del SDK esté configurada en una versión anterior distinta de la que se indica en Cómo empezar con los protocolos AWS JSON para Amazon SQS.

¿Dónde puedo encontrar más información sobre el protocolo AWS JSON?

Puede encontrar más información sobre el protocolo JSON en Protocolo AWS JSON 1.0 en la documentación de Smithy. Para obtener más información sobre las solicitudes de la API de Amazon SQS mediante el protocolo AWS JSON, consulte Realizar solicitudes de API de consulta mediante el protocolo AWS JSON en Amazon SQS.

Realizar solicitudes a la API de AWS consultas mediante el protocolo de consultas en Amazon SQS

En este tema se explica cómo construir un punto final de Amazon SQS, realizar solicitudes GET y POST e interpretar las respuestas.

Construcción de un punto de enlace

Para trabajar con colas de Amazon SQS, debe crear un punto de conexión. Para obtener información sobre los puntos de conexión de Amazon SQS, consulte las páginas siguientes en Referencia general de Amazon Web Services:

- Puntos de conexión regionales
- Puntos de conexión y cuotas de Amazon Simple Queue Service

Cada punto de conexión de Amazon SQS es independiente. Por ejemplo, si dos colas se denominan MyQueue y una tiene el punto de enlace `sqs.us-east-2.amazonaws.com` mientras que la otra tiene el punto de enlace `sqs.eu-west-2.amazonaws.com`, no comparten ningún dato entre ellas.

A continuación se muestra un ejemplo de un punto de enlace que realiza una solicitud para crear una cola.

```
https://sqs.eu-west-2.amazonaws.com/
?Action=CreateQueue
```

```
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> Los nombres de las colas y las colas URLs distinguen mayúsculas de minúsculas.
> La estructura de *AUTHPARAMS* depende de la firma de la solicitud de API. Para obtener más información, consulte Firmar solicitudes de AWS API en la Referencia general de Amazon Web Services.

Realizar una solicitud GET

Una solicitud GET de Amazon SQS se estructura como una URL que consta de los siguientes elementos:

- Punto de conexión: el recurso sobre el que actúa la solicitud (el nombre y la URL de la cola); por ejemplo, `https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`
- Acción: la acción que se desea realizar en el punto de conexión. Se utiliza un signo de interrogación (`?`) para separar el punto de enlace de la acción; por ejemplo, `?Action=SendMessage&MessageBody=Your%20Message%20Text`
- Parámetros: los parámetros de la solicitud. Cada parámetro está separado del siguiente por el signo `&`; por ejemplo, `&Version=2012-11-05&AUTHPARAMS`

A continuación se muestra un ejemplo de una solicitud GET que envía un mensaje a una cola de Amazon SQS.

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

> **Note**
>
> Los nombres de las colas y las colas URLs distinguen mayúsculas de minúsculas.

Como las solicitudes GET lo son URLs, debe codificar en URL todos los valores de los parámetros. Como no se permite la entrada de espacios URLs, cada espacio está codificado como URL. `%20` El resto del ejemplo no se ha codificado como URL para facilitar su lectura.

Realizar una solicitud POST

Las solicitudes POST de Amazon SQS envían parámetros de consulta como un formulario en el cuerpo de una solicitud HTTP.

A continuación, se muestra un ejemplo de encabezado HTTP con `Content-Type` establecido a `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
```

Al encabezado le sigue una solicitud GET de `form-urlencoded` que envía un mensaje a una cola de Amazon SQS. Cada parámetro está separado del siguiente por un signo ampersand (&).

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

> **Note**
>
> Solo es obligatorio el encabezado de HTTP `Content-Type`. El parámetro *AUTHPARAMS* es el mismo que para la solicitud GET.
> El cliente HTTP puede añadir otros elementos a la solicitud HTTP, según la versión de HTTP del cliente.

Interpretación de las respuestas de la API XML de Amazon SQS

Cuando envía una solicitud a Amazon SQS, devuelve una respuesta XML que contiene los resultados de la solicitud. Para comprender la estructura y los detalles de estas respuestas, consulte las acciones específicas de la API en la referencia de la API de Amazon Simple Queue Service.

Estructura de una respuesta XML correcta

Si la solicitud se realiza correctamente, el elemento principal de la respuesta tiene el mismo nombre que la acción, pero se le añade `Response` (por ejemplo, *`ActionName`*`Response`).

Este elemento contiene los siguientes elementos secundarios:

- ***`ActionName`*`Result`**: contiene un elemento específico de la acción. Por ejemplo, el elemento `CreateQueueResult` contiene el elemento `QueueUrl` que, a su vez, contiene la URL de la cola que se ha creado.
- **`ResponseMetadata`**: contiene el `RequestId` que, a su vez, contiene el identificador único universal (UUID) de la solicitud.

A continuación se muestra un ejemplo de una respuesta correcta en formato XML:

```
<CreateQueueResponse
   xmlns=https://sqs.us-east-2.amazonaws.com/doc/2012-11-05/
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:type=CreateQueueResponse>
   <CreateQueueResult>
      <QueueUrl>https://sqs.us-east-2.amazonaws.com/770098461991/queue2</QueueUrl>
   </CreateQueueResult>
   <ResponseMetadata>
      <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
   </ResponseMetadata>
</CreateQueueResponse>
```

Estructura de una respuesta de error XML

Si una solicitud no se realiza correctamente, Amazon SQS siempre devuelve el elemento principal de la respuesta `ErrorResponse`. Este elemento contiene un elemento `Error` y un elemento `RequestId`.

El elemento `Error` contiene los siguientes elementos secundarios:

- **`Type`**: especifica si se trata de un error del productor o del consumidor.
- **`Code`**: especifica el tipo de error.
- **`Message`**: especifica la condición de error en formato legible.
- **`Detail`**: (opcional) especifica información adicional sobre el error.

El elemento `RequestId` contiene el UUID de la solicitud.

A continuación se muestra un ejemplo de una respuesta de error en formato XML:

```
<ErrorResponse>
   <Error>
      <Type>Sender</Type>
      <Code>InvalidParameterValue</Code>
      <Message>
         Value (quename_nonalpha) for parameter QueueName is invalid.
         Must be an alphanumeric String of 1 to 80 in length.
      </Message>
   </Error>
   <RequestId>42d59b56-7407-4c4a-be0f-4c88daeea257</RequestId>
</ErrorResponse>
```

Autenticación de solicitudes para Amazon SQS

La autenticación es un proceso para identificar y verificar a la parte que envía una solicitud. Durante la primera fase de la autenticación, AWS verifica la identidad del productor y si está registrado para utilizar AWS (para obtener más información, consulte Paso 1: Cree un usuario Cuenta de AWS de IAM). A continuación AWS , sigue el siguiente procedimiento:

1. El productor (remitente) obtiene la credencial necesaria.
2. El productor envía una solicitud y la credencial al consumidor (receptor).
3. El consumidor utiliza la credencial para verificar si el productor envió la solicitud.
4. Se produce una de las circunstancias siguientes:
   - Si la autenticación se realiza correctamente, el consumidor procesa la solicitud.
   - Si se produce un error de autenticación, el consumidor rechaza la solicitud y devuelve un error.

Proceso básico de autenticación con HMAC-SHA

Cuando se obtiene acceso a Amazon SQS con la API de consultas, se deben proporcionar los siguientes elementos para autenticar la solicitud:

- El identificador de clave de AWS acceso que lo identifica Cuenta de AWS y que se AWS utiliza para buscar su clave de acceso secreta.

- La firma de la solicitud HMAC-SHA, calculada con tu clave de acceso secreta (un secreto compartido que solo tú conoces AWS; para obtener más información, consulta 04). RFC21 El AWS SDK se encarga del proceso de firma; sin embargo, si se realiza una solicitud de consulta a través de HTTP o HTTPS, se deberá incluir una firma en cada solicitud de consulta.

  1. Genere una clave firma Signature Version 4. Para obtener más información, consulte Generación de la clave de firma con Java.

     > **Note**
     >
     > Amazon SQS es compatible con la versión 4 de Signature, que proporciona seguridad y rendimiento SHA256 mejorados en comparación con las versiones anteriores. Cuando cree aplicaciones nuevas que usen Amazon SQS, utilice Signature Version 4.

  2. codifique en formato base64 la firma de la solicitud. El siguiente ejemplo se de código Java lo hace:

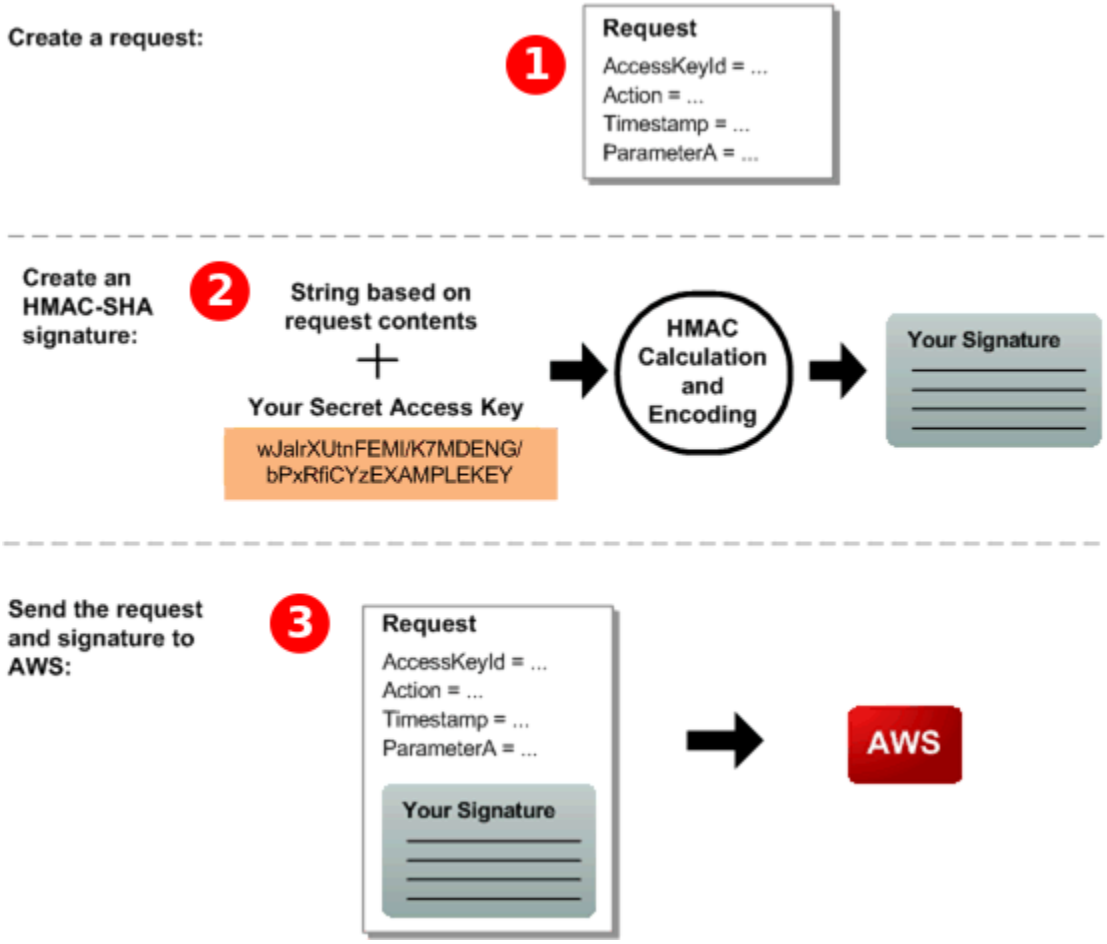
```
package amazon.webservices.common;

// 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);
    }
}
```

- La marca temporal (o vencimiento) de la solicitud. La marca temporal que utilice en la solicitud debe ser un objeto `dateTime`, con la fecha completa, incluidas las horas, los minutos y los segundos. Por ejemplo: `2007-01-31T23:59:59Z` Aunque no es necesario, le recomendamos que proporcione el objeto utilizando la zona horaria de la hora universal coordinada (hora del meridiano de Greenwich).

> **Note**
>
> Asegúrese de que la hora del servidor esté ajustada correctamente. Si especifica una marca de tiempo (en lugar de una fecha de caducidad), la solicitud caduca automáticamente 15 minutos después de la hora especificada (AWS no procesa las solicitudes con marcas de tiempo más de 15 minutos antes de la hora actual en los servidores). AWS
> Si utiliza .NET, no debe enviar marcas temporales demasiado específicas (debido a las diferentes interpretaciones con respecto a cómo se debe descartar la precisión adicional del tiempo). En este caso, debe crear manualmente objetos `dateTime` con una precisión de no más de un milisegundo.

Parte 1: la solicitud del usuario

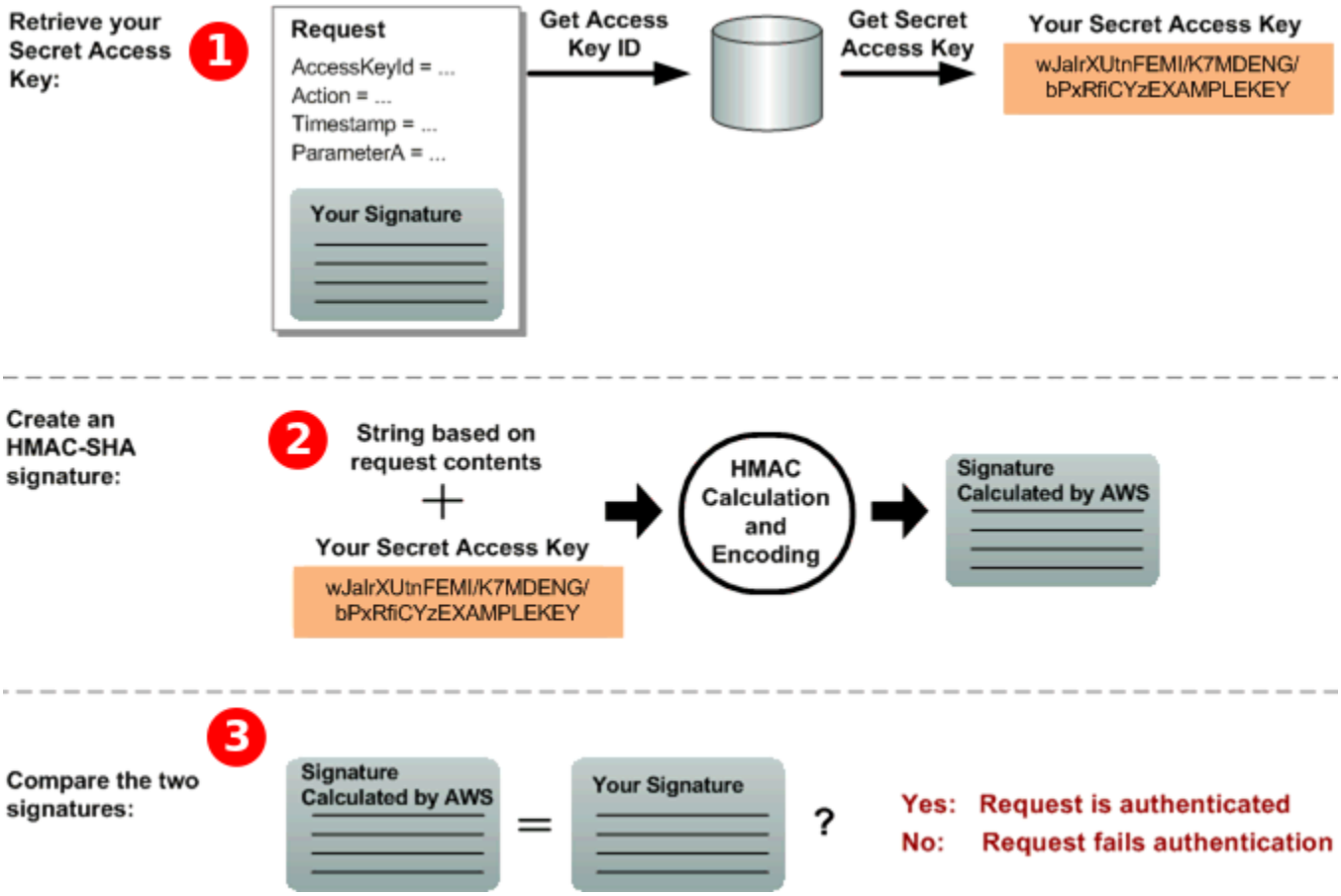
El siguiente es el proceso que debe seguir para autenticar las AWS solicitudes mediante una firma de solicitud del HMAC-SHA.



1. Cree una solicitud para. AWS
2. Calcule la firma de código de autenticación de mensajes mediante algoritmos hash con clave (HMAC-SHA) utilizando su clave de acceso secreta.
3. Incluye la firma y tu ID de clave de acceso en la solicitud y, a continuación, envía la solicitud a AWS.

Parte 2: La respuesta de AWS

AWS comienza el siguiente proceso en respuesta.



1. AWS utiliza el identificador de clave de acceso para buscar su clave de acceso secreta.
2. AWS genera una firma a partir de los datos de la solicitud y la clave de acceso secreta, utilizando el mismo algoritmo que utilizó para calcular la firma que envió en la solicitud.
3. Se produce una de las circunstancias siguientes:
   - Si la firma que se AWS genera coincide con la que se envió en la solicitud, se considerará AWS que la solicitud es auténtica.
   - Si la comparación falla, la solicitud se descarta y AWS devuelve un error.

Acciones por lotes de Amazon SQS

Amazon SQS permite realizar acciones por lotes para ayudarle a reducir los costos y manipular hasta 10 mensajes con una sola acción. Estas acciones por lotes incluyen:

- `SendMessageBatch`
- `DeleteMessageBatch`

- `ChangeMessageVisibilityBatch`

Al utilizar acciones por lotes, puede realizar varias operaciones en una sola llamada a la API, lo que ayuda a optimizar el rendimiento y reducir los costos. Puede aprovechar la funcionalidad por lotes mediante la API de consultas o cualquier AWS SDK que admita las acciones por lotes de Amazon SQS.

Detalles importantes

- Límite de tamaño de los mensajes: el tamaño total de todos los mensajes enviados en una sola `SendMessageBatch` llamada no puede superar los 1.048.576 bytes (1 MiB)
- Permisos: no se pueden establecer permisos para `SendMessageBatch`, `DeleteMessageBatch` ni `ChangeMessageVisibilityBatch` de forma explícita. Sin embargo, al establecer permisos para `SendMessage`, `DeleteMessage` o `ChangeMessageVisibility`, se establecen también permisos para las versiones por lotes correspondientes de esas acciones.
- Soporte de consola: la consola de Amazon SQS no es compatible con las acciones de procesamiento por lotes. Debe usar la API de consultas o un AWS SDK para realizar operaciones por lotes.

Procesamiento por lotes de las acciones de los mensajes

Para optimizar aún más los costos y la eficiencia, tenga en cuenta las siguientes prácticas recomendadas para acciones de mensajes de procesamiento por lotes:

- Acciones de la API de procesamiento por lotes: use las acciones de la API de procesamiento por lotes de Amazon SQS para enviar, recibir y eliminar mensajes, y para cambiar el tiempo de espera de visibilidad de varios mensajes con una sola acción. Esto reduce el número de llamadas a la API y los costos asociados.
- Almacenamiento en búfer del cliente y sondeos largos: combine el almacenamiento en búfer del cliente con el procesamiento de solicitudes por lotes mediante un sondeo largo junto con el cliente asíncrono con almacenamiento en búfer incluido en el AWS SDK para Java. Este enfoque ayuda a minimizar el número de solicitudes y optimiza la gestión de grandes volúmenes de mensajes.

> **Note**
>
> El cliente asíncrono con búfer de Amazon SQS no admite actualmente las colas FIFO.

Habilitación del almacenamiento en búfer del cliente y del agrupamiento en lotes de solicitudes con Amazon SQS

AWS SDK para Java incluye `AmazonSQSBufferedAsyncClient` que tiene acceso a Amazon SQS. Este cliente permite un procesamiento sencillo de solicitudes por lotes mediante el almacenamiento en búfer del lado del cliente. Las llamadas realizadas desde el cliente se almacenan primero en búfer y, a continuación, se envían como una solicitud por lotes a Amazon SQS.

El almacenamiento en búfer en el cliente permite almacenar en búfer hasta diez solicitudes y enviarlas como una solicitud por lotes, lo que disminuye el costo de uso de Amazon SQS y el número de solicitudes enviadas. `AmazonSQSBufferedAsyncClient` almacena en búfer tanto las llamadas sincrónicas como las asincrónicas. Las solicitudes por lotes y la compatibilidad con los sondeos largos también pueden contribuir a mejorar el rendimiento. Para obtener más información, consulte Aumento del rendimiento mediante el escalado horizontal y agrupación en lotes de acciones con Amazon SQS.

Dado que `AmazonSQSBufferedAsyncClient` implementa la misma interfaz que `AmazonSQSAsyncClient`, la migración de `AmazonSQSAsyncClient` a `AmazonSQSBufferedAsyncClient` solo suele requerir cambios mínimos en el código.

> **Note**
>
> El cliente asíncrono con búfer de Amazon SQS no admite actualmente las colas FIFO.

Uso de Amazon SQSBuffered AsyncClient

Antes de comenzar, complete los pasos de Configuración de Amazon SQS.

AWS SDK para Java 1.x

Para el AWS SDK para Java 1.x, puedes crear uno nuevo `AmazonSQSBufferedAsyncClient` basado en el siguiente ejemplo:

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

Después de crear el nuevo `AmazonSQSBufferedAsyncClient`, puede utilizarlo para enviar varias solicitudes a Amazon SQS (del mismo modo que con `AmazonSQSAsyncClient`), por ejemplo:

```
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);
```

Configuración de Amazon SQSBuffered AsyncClient

`AmazonSQSBufferedAsyncClient` está preconfigurado con ajustes válidos para la mayoría de los casos de uso. Se pueden configurar ajustes adicionales de `AmazonSQSBufferedAsyncClient`; por ejemplo:

1. Crear una instancia de la clase `QueueBufferConfig` con los parámetros de configuración necesarios.
2. Proporcionar la instancia al constructor `AmazonSQSBufferedAsyncClient`.

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

final QueueBufferConfig config = new QueueBufferConfig()
```

```
    .withMaxInflightReceiveBatches(5)
    .withMaxDoneReceiveBatches(15);

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync, config);
```

QueueBufferConfig parámetros de configuración

Parámetro	Valor predeterminado	Descripción
`longPoll`	`true`	Cuando `longPoll` se establece en `true`, `AmazonSQSBufferedAsyncClient` intenta utilizar el sondeo largo a la hora de consumir mensajes.
`longPollWaitTimeoutSeconds`	20 s	El tiempo máximo, en segundos, que una llamada a `ReceiveMessage` se bloquea en el servidor a la espera de que aparezcan mensajes en la cola antes de devolver un resultado de recepción vacío. **Note** Este parámetro no tiene ningún efecto cuando el sondeo largo está deshabilitado.
`maxBatchOpenMs`	200ms	

Parámetro	Valor predeterminado	Descripción
		El tiempo máximo, en milisegundos, que una llamada saliente espera otras llamadas para procesar por lotes mensajes del mismo tipo. Cuanto mayor sea el valor, menos lotes son necesario s para realizar la misma cantidad de trabajo (no obstante, la primera llamada de un lote tiene que esperar más tiempo). Cuando se establece este parámetro en `0`, las solicitud es enviadas no esperan a otras solicitudes, lo que en la práctica deshabilita el procesamiento por lotes.

Parámetro	Valor predeterminado	Descripción
`maxBatchSize`	10 solicitudes por lote	El número máximo de mensajes que se procesan juntos por lotes en una sola solicitud. Cuanto mayor sea la configuración, menos lotes serán necesarios para llevar a cabo el mismo número de solicitudes. **Note** El valor máximo permitido para Amazon SQS es de diez solicitudes por lote.
`maxBatchSizeBytes`	1 MiB	El tamaño máximo de un lote de mensajes, en bytes, que el cliente intenta enviar a Amazon SQS. **Note** 1 MiB es el valor máximo permitido para Amazon SQS.

Parámetro	Valor predeterminado	Descripción
`maxDoneReceiveBatches`	10 lotes	El número máximo de lotes de recepción que `AmazonSQSBufferedAsyncClient` captura previamente y almacena en el lado del cliente. Cuanto mayor sea el valor, más solicitudes de recepción podrán satisfacerse sin tener que realizar una llamada a Amazon SQS (sin embargo, cuantos más mensajes se capturen previamente, más tiempo permanecerán en el búfer, lo que hará que caduque su propio tiempo de espera de visibilidad). **Note** `0` indica que se ha deshabilitado la captura previa de mensajes y que estos solo se consumen bajo demanda.

Parámetro	Valor predeterminado	Descripción
`maxInflightOutboundBatches`	5 lotes	El número máximo de lotes salientes activos que se pueden procesar al mismo tiempo. Cuanto mayor sea el valor, más rápido se podrán enviar los lotes salientes (sujeto a cuotas como la CPU o el ancho de banda) y más subprocesos podrá consumir `AmazonSQSBufferedAsyncClient` .

Parámetro	Valor predeterminado	Descripción
`maxInflightReceiveBatches`	10 lotes	El número máximo de lotes de recepción activos que se pueden procesar al mismo tiempo. Cuanto mayor sea el valor, más mensajes de podrán recibir (sujeto a cuotas como la CPU o el ancho de banda) y más subprocesos podrá consumir `AmazonSQSBufferedAsyncClient` . **Note** `0` indica que se ha deshabilitado la captura previa de mensajes y que estos solo se consumen bajo demanda.

Parámetro	Valor predeterminado	Descripción
`visibilityTimeoutSeconds`	-1	Cuando este parámetro se establece en un valor positivo distinto de cero, el tiempo de espera de visibilidad que se establece aquí anula el tiempo de espera de visibilidad definido en la cola desde la que se consumen los mensajes. **Note** `-1` indica que está seleccionada la configuración predeterminada de la cola. No se puede establecer el tiempo de espera de visibilidad en `0`.

AWS SDK para Java 2.x

Para el AWS SDK para Java 2.x, puedes crear uno nuevo `SqsAsyncBatchManager` basado en el siguiente ejemplo:

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
```

```
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

Después de crear el nuevo `SqsAsyncBatchManager`, puede utilizarlo para enviar varias solicitudes a Amazon SQS (del mismo modo que con `SqsAsyncClient`), por ejemplo:

```
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);
```

Configuración de SqsAsyncBatchManager

`SqsAsyncBatchManager` está preconfigurado con ajustes válidos para la mayoría de los casos de uso. Se pueden configurar ajustes adicionales de `SqsAsyncBatchManager`; por ejemplo:

Crear una configuración personalizada mediante`SqsAsyncBatchManager.Builder`:

```
SqsAsyncBatchManager customizedBatchManager = SqsAsyncBatchManager.builder()
    .client(sqs)
    .scheduledExecutor(Executors.newScheduledThreadPool(5))
    .overrideConfiguration(b -> b
        .maxBatchSize(10)
        .sendRequestFrequency(Duration.ofMillis(200))
        .receiveMessageMinWaitDuration(Duration.ofSeconds(10))
        .receiveMessageVisibilityTimeout(Duration.ofSeconds(20))
        .receiveMessageAttributeNames(Collections.singletonList("*"))

 .receiveMessageSystemAttributeNames(Collections.singletonList(MessageSystemAttributeName.ALL))
    .build();
```

Parámetros **`BatchOverrideConfiguration`**

Parámetro	Valor predeterminado	Descripción
`maxBatchSize`	10 solicitudes por lote	El número máximo de mensajes que se procesan juntos por lotes en una sola solicitud. Cuanto mayor sea la configuración, menos lotes serán necesarios para llevar a cabo el mismo número de solicitudes. **Note** El valor máximo permitido para Amazon SQS es de 10 solicitudes por lote.
`sendRequestFrequency`	200ms	El tiempo máximo, en milisegundos, que una llamada saliente espera otras

Parámetro	Valor predeterminado	Descripción
		llamadas para procesar por lotes mensajes del mismo tipo. Cuanto mayor sea el valor, menos lotes son necesarios para realizar la misma cantidad de trabajo (no obstante, la primera llamada de un lote tiene que esperar más tiempo). Cuando se establece este parámetro en `0`, las solicitudes enviadas no esperan a otras solicitudes, lo que en la práctica deshabilita el procesamiento por lotes.

Parámetro	Valor predeterminado	Descripción
`receiveMessageVisibilityTimeout`	-1	Cuando este parámetro se establece en un valor positivo distinto de cero, el tiempo de espera de visibilidad que se establece aquí anula el tiempo de espera de visibilidad definido en la cola desde la que se consumen los mensajes. **Note** 1 indica que está seleccionada la configuración predeterminada de la cola. No se puede establecer el tiempo de espera de visibilidad en `0`.
`receiveMessageMinWaitDuration`	50 ms	El tiempo mínimo (en milisegundos) que espera una `receiveMessage` llamada a que se recuperen los mensajes disponibles. Cuanto más alta sea la configuración, menos lotes se necesitarán para llevar a cabo el mismo número de solicitudes.

Aumento del rendimiento mediante el escalado horizontal y agrupación en lotes de acciones con Amazon SQS

Amazon SQS admite mensajería de alto rendimiento. Para obtener más información sobre los límites de rendimiento, consulte. Cuotas de mensajes de Amazon SQS

Para maximizar el rendimiento:

- Amplíe a los productores y consumidores de forma horizontal añadiendo más instancias de cada uno de ellos.
- Utilice el procesamiento por lotes de acciones para enviar o recibir varios mensajes en una sola solicitud, lo que reduce la sobrecarga de llamadas a la API.

Escalado horizontal

Dado que se accede a Amazon SQS a través de un protocolo HTTP de solicitud-respuesta, la latencia de solicitudes (el intervalo de tiempo que transcurre entre que se inicia una solicitud y se recibe una respuesta) limita el rendimiento que se puede alcanzar con un solo subproceso a través de una única conexión. Por ejemplo, si la latencia de un cliente EC2 basado en Amazon a Amazon SQS de la misma región tiene un promedio de 20 ms, el rendimiento máximo de un solo hilo a través de una sola conexión tiene un promedio de 50 TPS.

El escalado horizontal supone aumentar el número de productores de mensajes (que realizan solicitudes `SendMessage`) y consumidores de mensajes (que realizan solicitudes `ReceiveMessage` y `DeleteMessage`) para aumentar el desempeño general de la cola. Existen tres métodos para realizar el escalado horizontal:

- Aumentar el número de subprocesos por cliente
- Añadir más clientes
- Aumentar el número de subprocesos por cliente y añadir más clientes

Cuando se añaden más clientes, se consiguen ganancias prácticamente lineales en el desempeño de la cola. Por ejemplo, si se duplica el número de clientes, también se obtiene el doble de desempeño.

Agrupación en lotes de acciones

La agrupación por lotes realiza más trabajo durante cada ciclo de ida y vuelta al servicio (por ejemplo, al enviar varios mensajes en una única solicitud `SendMessageBatch`). Las acciones de procesamiento por lotes de Amazon SQS son `SendMessageBatch`, `DeleteMessageBatch` y `ChangeMessageVisibilityBatch`. Para aprovechar el procesamiento por lotes sin modificar los productores ni los consumidores, puede utilizar el Cliente asincrónico en búfer de Amazon SQS.

> **Note**
>
> Debido a que `ReceiveMessage` puede procesar 10 mensajes a la vez, no hay ninguna acción `ReceiveMessageBatch`.

La agrupación en lotes distribuye la latencia de la acción por lotes entre varios mensajes en una solicitud por lotes, en lugar de aceptar toda la latencia para un único mensaje (por ejemplo, una solicitud `SendMessage`). Como en cada ciclo de ida y vuelta se realiza más trabajo, las solicitudes por lotes hacen un uso más eficaz de los subprocesos y las conexiones, por lo que se mejora el desempeño.

Puede combinar la agrupación en lotes con el escalado horizontal para proporcionar un desempeño con menos subprocesos, conexiones y solicitudes de los que serían necesarios en el caso de utilizar solicitudes de mensajes individuales. Puede utilizar acciones de Amazon SQS por lotes para enviar, recibir o eliminar hasta diez mensajes a la vez. Dado que el uso de Amazon SQS se factura por solicitudes, el procesamiento por lotes puede reducir significativamente los costos.

La agrupación en lotes puede introducir cierta complejidad en una aplicación (por ejemplo, la aplicación debe acumular los mensajes antes de enviarlos, o a veces debe esperar más para recibir una respuesta). Sin embargo, la agrupación en lotes puede resultar eficaz en los casos siguientes:

- Cuando la aplicación genera muchos mensajes en poco tiempo, por lo que el retraso nunca es muy largo.
- Cuando un consumidor de mensajes busca mensajes en una cola a discreción, a diferencia de los productores de mensajes típicos que tienen que enviar mensajes como respuesta a eventos que no controlan.

> **Important**
>
> Una solicitud por lotes puede realizarse correctamente aunque se hayan producido errores en mensajes individuales del lote. Después de una solicitud por lotes, compruebe siempre si hay errores en mensajes individuales y vuelva a intentar la acción si es necesario.

Ejemplo funcional en Java de solicitudes de una sola operación y por lotes

Requisitos previos

Añada los paquetes `aws-java-sdk-sqs.jar`, `aws-java-sdk-ec2.jar` y `commons-logging.jar` a la ruta de clases de compilación Java. Los siguientes ejemplos muestran estas dependencias en el archivo `pom.xml` de un proyecto Maven.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-ec2</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>LATEST</version>
    </dependency>
</dependencies>
```

SimpleProducerConsumer.java

En el siguiente ejemplo de código Java se implementa un patrón productor-consumidor sencillo. El subproceso principal genera una serie de subprocesos productores y consumidores que procesan mensajes de 1 KB durante el tiempo especificado. Este ejemplo incluye productores y consumidores que realizan solicitudes de una única operación y otros que realizan solicitudes por lotes.

```
/*
```

```
 * 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.
            }
        }
    }
}
```

Monitoreo de las métricas de volumen de la ejecución del ejemplo

Amazon SQS genera automáticamente métricas de volumen para los mensajes enviados, recibidos y eliminados. Puede tener acceso a estas y otras métricas a través de la pestaña Monitoring (Monitoreo) de la cola o en la CloudWatch consola.

> **Note**
>
> Las métricas pueden tardar en estar disponibles hasta 15 minutos después del inicio de la cola.

Uso de Amazon SQS con un SDK AWS

AWS Los kits de desarrollo de software (SDKs) están disponibles para muchos lenguajes de programación populares. Cada SDK proporciona una API, ejemplos de código y documentación que facilitan a los desarrolladores la creación de aplicaciones en su lenguaje preferido.

Documentación de SDK	Ejemplos de código
AWS SDK para C++	AWS SDK para C++ ejemplos de código

Documentación de SDK	Ejemplos de código
AWS CLI	AWS CLI ejemplos de código
AWS SDK para Go	AWS SDK para Go ejemplos de código
AWS SDK para Java	AWS SDK para Java ejemplos de código
AWS SDK para JavaScript	AWS SDK para JavaScript ejemplos de código
AWS SDK para Kotlin	AWS SDK para Kotlin ejemplos de código
AWS SDK para .NET	AWS SDK para .NET ejemplos de código
AWS SDK para PHP	AWS SDK para PHP ejemplos de código
Herramientas de AWS para PowerShell	Herramientas de AWS para PowerShell ejemplos de código
AWS SDK para Python (Boto3)	AWS SDK para Python (Boto3) ejemplos de código
AWS SDK para Ruby	AWS SDK para Ruby ejemplos de código
AWS SDK para Rust	AWS SDK para Rust ejemplos de código
AWS SDK para SAP ABAP	AWS SDK para SAP ABAP ejemplos de código
AWS SDK para Swift	AWS SDK para Swift ejemplos de código

Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicite un ejemplo de código a través del enlace de Enviar comentarios que se encuentra al final de esta página.

Uso de JMS con Amazon SQS

La Biblioteca de mensajes Java de Amazon SQS es una interfaz de Servicio de mensajes de Java (JMS) para Amazon SQS que le permite aprovechar las ventajas de Amazon SQS en aplicaciones que ya utilizan JMS. La interfaz le permite utilizar Amazon SQS como proveedor JMS sin tener que realizar apenas cambios en el código. Junto con AWS SDK para Java, la Biblioteca de mensajes Java de Amazon SQS le permite crear conexiones y sesiones de JMS, así como productores y consumidores que envían y reciben mensajes hacia colas de Amazon SQS y desde ellas.

La biblioteca admite el envío y la recepción de mensajes a una cola (el point-to-point modelo JMS) según la especificación JMS 1.1. La biblioteca permite enviar mensajes de texto, bytes u objetos de forma sincrónica a colas de Amazon SQS. La biblioteca también admite la recepción de objetos de forma sincrónica o asincrónica.

Para obtener información sobre las características de la biblioteca de mensajería Java de Amazon SQS que admiten la especificación JMS 1.1, consulte Implementaciones de JMS 1.1 admitidas por Amazon SQS Amazon SQS. FAQs

Requisitos previos para trabajar con JMS y Amazon SQS

Antes de empezar, debe disponer de los siguientes requisitos previos:

- SDK para Java

  Hay dos formas de incluir el SDK para Java en su proyecto:

  - Descargue e instale el SDK para Java.
  - Utilice Maven para obtener la Biblioteca de mensajes Java de Amazon SQS.

  > **Note**
  >
  > El SDK para Java se incluye como dependencia.
  > El SDK para Java y la biblioteca de clientes ampliada de Amazon SQS para Java requieren J2SE Development Kit 8.0 o una versión posterior.

  Para obtener información acerca de la descarga de SDK para Java, consulte SDK para Java.

- Biblioteca de mensajes Java de Amazon SQS

Si no utiliza Maven, debe añadir el paquete `amazon-sqs-java-messaging-lib.jar` a la ruta de clases de Java. Para obtener información sobre la descarga de la biblioteca, consulte Biblioteca de mensajes Java de Amazon SQS.

> **Note**
>
> La Biblioteca de mensajes Java de Amazon SQS incluye compatibilidad con Maven y Spring Framework.
> Para obtener ejemplos de código en los que se utiliza Maven, el marco Spring y la Biblioteca de mensajes Java de Amazon SQS, consulte Ejemplos funcionales en Java del uso de JMS con colas estándar de Amazon SQS.
>
> ```
> <dependency>
>  <groupId>com.amazonaws</groupId>
>  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
>  <version>1.0.4</version>
>  <type>jar</type>
> </dependency>
> ```

- Cola de Amazon SQS

  Cree una cola mediante Amazon SQS, la AWS Management Console API o `CreateQueue` el cliente Amazon SQS empaquetado incluido en la biblioteca de mensajería Java de Amazon SQS.

  - Para obtener información acerca de cómo crear una cola con Amazon SQS a través de la AWS Management Console o la API `CreateQueue`, consulte Creación de colas.
  - Para obtener información acerca del uso de la Biblioteca de mensajes Java de Amazon SQS, consulte Uso de la Biblioteca de mensajes Java de Amazon SQS.

Uso de la Biblioteca de mensajes Java de Amazon SQS

Para comenzar a utilizar el Servicio de mensajes de Java (JMS) con Amazon SQS, utilice los ejemplos de código de esta sección. En las próximas secciones se muestra cómo crear una conexión JMS y una sesión y cómo enviar y recibir un mensaje.

El objeto cliente de Amazon SQS encapsulado incluido en la Biblioteca de mensajes Java de Amazon SQS comprueba si existe una cola de Amazon SQS. Si la cola no existe, el cliente la crea.

Creación de una conexión JMS

Antes de comenzar, consulte los requisitos previos de Requisitos previos para trabajar con JMS y Amazon SQS.

1. Cree una fábrica de conexiones y llame al método `createConnection` de la fábrica.

```
// Create a new connection factory with all defaults (credentials and region) set
 automatically
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.defaultClient()
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

   La clase `SQSConnection` amplía `javax.jms.Connection`. Junto con los métodos de conexión estándar de JMS, `SQSConnection` ofrece métodos adicionales como, por ejemplo, `getAmazonSQSClient` y `getWrappedAmazonSQSClient`. Ambos métodos le permiten llevar a cabo operaciones administrativas no incluidas en la especificación JMS, como la creación de nuevas colas. Sin embargo, el método `getWrappedAmazonSQSClient` también ofrece una versión encapsulada del cliente de Amazon SQS que utiliza la conexión actual. El contenedor transforma cada excepción del cliente en una `JMSException`, por lo que es más fácil que la utilice el código existente, que espera encontrar `JMSException`.

2. Puede utilizar los objetos de cliente devueltos por `getAmazonSQSClient` y `getWrappedAmazonSQSClient` para realizar operaciones administrativas que no están incluidas en la especificación JMS (por ejemplo, puede crear una cola de Amazon SQS).

   Si dispone de código que espera encontrar excepciones de JMS, debería utilizar `getWrappedAmazonSQSClient`:

   - Si utiliza `getWrappedAmazonSQSClient`, el objeto de cliente devuelto transforma todas las excepciones en las excepciones de JMS.
   - Si utiliza `getAmazonSQSClient`, todas las excepciones serán excepciones de Amazon SQS.

Creación de una cola de Amazon SQS

El objeto de cliente encapsulado comprueba si existe una cola de Amazon SQS.

Si no existe una cola, el cliente la crea. Si la cola existe, la función no devuelve nada. Para obtener más información, consulte la sección "Create the queue if needed" en el ejemplo TextMessageSender.java.

Para crear una cola estándar

```
// 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");
}
```

Creación de una cola 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**
>
> La cola FIFO debe finalizar con el sufijo `.fifo`.
> Para obtener más información acerca del atributo `ContentBasedDeduplication`, consulte Procesamiento una sola vez en Amazon SQS.

Envío de mensajes de forma sincrónica

1. Cuando la conexión y la cola de Amazon SQS subyacente estén listas, cree una sesión de JMS sin transacciones con el modo `AUTO_ACKNOWLEDGE`.

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. Para enviar un mensaje de texto a la cola, cree una identidad de cola de JMS y un productor de mensajes.

```
// 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. Cree un mensaje de texto y envíelo a la cola.

   - Para enviar un mensaje a una cola estándar, no es necesario configurar ningún parámetro adicional.

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

   - Para enviar un mensaje a una cola FIFO, debe establecer el ID de grupo de mensajes. También puede establecer un ID de desduplicación de mensajes. Para obtener más información, consulte Términos clave de cola FIFO de Amazon SQS.

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Set the message group ID
message.setStringProperty("JMSXGroupID", "Default");

// You can also set a custom message deduplication ID
// message.setStringProperty("JMS_SQS_DeduplicationId", "hello");
```

```
// Here, it's not needed because content-based deduplication is enabled for the
 queue

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
System.out.println("JMS Message Sequence Number " +
 message.getStringProperty("JMS_SQS_SequenceNumber"));
```

Recepción de mensajes de forma sincrónica

1. Para recibir mensajes, cree un consumidor para la misma cola e invoque el método `start`.

   Puede llamar al método `start` de la conexión en cualquier momento. Sin embargo, el consumidor no empieza a recibir mensajes hasta que no se le llama.

   ```
   // Create a consumer for the 'MyQueue'
   MessageConsumer consumer = session.createConsumer(queue);
   // Start receiving incoming messages
   connection.start();
   ```

2. Llame al método `receive` del consumidor con un tiempo de espera establecido en 1 segundo y, a continuación, imprima el contenido del mensaje recibido.

   - Cuando se recibe un mensaje de una cola estándar, se puede obtener acceso al contenido del mensaje.

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

   - Tras recibir un mensaje de una cola FIFO, puede obtener acceso al contenido del mensaje y otros atributos del mensaje específicos de FIFO, como el ID del grupo de mensajes, el ID de desduplicación de mensajes y el número de secuencia. Para obtener más información, consulte Términos clave de cola FIFO de Amazon SQS.

```
// Receive a message from 'MyQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and display the text
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    System.out.println("Group id: " +
 receivedMessage.getStringProperty("JMSXGroupID"));
    System.out.println("Message deduplication id: " +
 receivedMessage.getStringProperty("JMS_SQS_DeduplicationId"));
    System.out.println("Message sequence number: " +
 receivedMessage.getStringProperty("JMS_SQS_SequenceNumber"));
}
```

3. Cierre la conexión y la sesión.

```
// Close the connection (and the session).
connection.close();
```

El resultado tiene un aspecto similar al siguiente:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

> **Note**
>
> Puede utilizar el marco Spring para inicializar estos objetos.
> Para obtener más información, consulte `SpringExampleConfiguration.xml`, `SpringExample.java` y el resto de clases auxiliares en `ExampleConfiguration.java` y `ExampleCommon.java` en la sección Ejemplos funcionales en Java del uso de JMS con colas estándar de Amazon SQS.

Para obtener ejemplos completos del envío y la recepción de objetos, consulte TextMessageSender.java y SyncMessageReceiver.java.

Recepción de mensajes de forma asincrónica

En el ejemplo de Uso de la Biblioteca de mensajes Java de Amazon SQS, se envía un mensaje a `MyQueue` y se recibe de forma sincrónica.

El siguiente ejemplo muestra cómo recibir los mensajes de forma asíncrona a través de un agente de escucha.

1. Implemente la interfaz `MessageListener`.

```
class MyListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to
 screen.
            System.out.println("Received: " + ((TextMessage) message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

   Se llama al método `onMessage` de la interfaz `MessageListener` cuando recibe un mensaje. En esta implementación del agente de escucha, se imprime el texto almacenados en el mensaje.

2. En lugar de llamar explícitamente al método `receive` del consumidor, establezca el agente de escucha de mensajes del consumidor en una instancia de la implementación de `MyListener`. El subproceso principal espera un segundo.

```
// 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);
```

Los pasos restantes son idénticos a los del ejemplo Uso de la Biblioteca de mensajes Java de Amazon SQS. Para obtener un ejemplo completo de un consumidor asíncrono, consulte `AsyncMessageReceiver.java` Ejemplos funcionales en Java del uso de JMS con colas estándar de Amazon SQS.

El resultado de este ejemplo tiene un aspecto similar al siguiente:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Uso del modo de reconocimiento del cliente

En el ejemplo de Uso de la Biblioteca de mensajes Java de Amazon SQS, se utiliza el modo `AUTO_ACKNOWLEDGE`, en el que cada mensaje recibido se confirma automáticamente (y, por tanto, se elimina de la cola subyacente de Amazon SQS).

1. Para reconocer explícitamente los mensajes una vez procesados, debe crear la sesión con el modo `CLIENT_ACKNOWLEDGE`.

   ```
   // Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
   Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
   ```

2. Cuando se reciba el mensaje, muéstrelo y acéptelo explícitamente.

   ```
   // 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**
   >
   > En este modo, cuando un mensaje se reconoce, todos los mensajes recibidos antes que este mensaje también se reconocen implícitamente. Por ejemplo, si se reciben 10

mensajes y solo se reconoce el décimo mensaje (en el orden en que los mensajes se reciben), los nueve mensajes anteriores también se reconocen.

Los pasos restantes son idénticos a los del ejemplo Uso de la Biblioteca de mensajes Java de Amazon SQS. Para ver un ejemplo completo de un consumidor sincrónico con el modo de reconocimiento del cliente, consulte `SyncMessageReceiverClientAcknowledge.java` en Ejemplos funcionales en Java del uso de JMS con colas estándar de Amazon SQS.

El resultado de este ejemplo tiene un aspecto similar al siguiente:

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

Uso del modo de reconocimiento sin orden

Cuando se utiliza el modo `CLIENT_ACKNOWLEDGE`, todos los mensajes recibidos antes que un mensaje reconocido explícitamente se reconocen automáticamente. Para obtener más información, consulte Uso del modo de reconocimiento del cliente.

La Biblioteca de mensajes Java de Amazon SQS proporciona otro modo de confirmación. Cuando se utiliza el modo `UNORDERED_ACKNOWLEDGE`, el cliente debe reconocer individualmente y de forma explícita todos los mensajes recibidos, independientemente del orden de recepción. Para ello, cree una sesión con el modo `UNORDERED_ACKNOWLEDGE`.

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

Los pasos restantes son idénticos a los del ejemplo Uso del modo de reconocimiento del cliente. Para obtener un ejemplo completo de un consumidor síncrono con el modo `UNORDERED_ACKNOWLEDGE`, consulte `SyncMessageReceiverUnorderedAcknowledge.java`.

En este ejemplo, el resultado tiene un aspecto similar al siguiente:

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Uso del Servicio de mensajes de Java con otros clientes de Amazon SQS

El uso del cliente Java Message Service (JMS) de Amazon SQS con el AWS SDK limita el tamaño de los mensajes de Amazon SQS a 256 KB. No obstante, puede crear un proveedor de JMS mediante cualquier cliente de Amazon SQS. Por ejemplo, puede utilizar el cliente de JMS con la biblioteca de clientes ampliada de Amazon SQS para Java a fin de enviar un mensaje de Amazon SQS que contenga una referencia a una carga de mensaje (de hasta 2 GB) en Amazon S3. Para obtener más información, consulte Administración de mensajes de Amazon SQS grandes mediante Java y Amazon S3.

En el siguiente ejemplo de código Java se crea el proveedor de JMS para la biblioteca de clientes ampliada.

Consulte los requisitos previos en Requisitos previos para trabajar con JMS y Amazon SQS antes de probar este ejemplo.

```
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);
```

El siguiente ejemplo de código de Java crea la fábrica de conexiones:

```
// 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();
```

Ejemplos funcionales en Java del uso de JMS con colas estándar de Amazon SQS

En los siguientes ejemplos de código se muestra cómo utilizar el Servicio de mensajes de Java (JMS) con las colas estándar de Amazon SQS. Para obtener más información sobre cómo trabajar con colas FIFO, consulte Creación de una cola FIFO, Envío de mensajes de forma sincrónica y Recepción de mensajes de forma sincrónica. (La recepción de mensajes de forma sincrónica es la misma para las colas estándar y FIFO. No obstante, los mensajes de las colas FIFO contienen más atributos).

Consulte los requisitos previos en Requisitos previos para trabajar con JMS y Amazon SQS antes de probar los siguientes ejemplos.

ExampleConfiguration.java

El siguiente ejemplo de código de Java SDK v 1.x establece el nombre de cola predeterminado, la región y las credenciales que se van a utilizar con los demás ejemplos de Java.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
```

```
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleConfiguration {
    public static final String DEFAULT_QUEUE_NAME = "SQSJMSClientExampleQueue";

    public static final Region DEFAULT_REGION = Region.getRegion(Regions.US_EAST_2);

    private static String getParameter( String args[], int i ) {
        if( i + 1 >= args.length ) {
            throw new IllegalArgumentException( "Missing parameter for " + args[i] );
        }
        return args[i+1];
    }

    /**
     * Parse the command line and return the resulting config. If the config parsing
 fails
     * print the error and the usage message and then call System.exit
     *
     * @param app the app to use when printing the usage string
     * @param args the command line arguments
     * @return the parsed config
     */
    public static ExampleConfiguration parseConfig(String app, String args[]) {
        try {
            return new ExampleConfiguration(args);
        } catch (IllegalArgumentException e) {
            System.err.println( "ERROR: " + e.getMessage() );
            System.err.println();
            System.err.println( "Usage: " + app + " [--queue <queue>] [--region
 <region>] [--credentials <credentials>] ");
            System.err.println( "  or" );
            System.err.println( "       " + app + " <spring.xml>" );
            System.exit(-1);
            return null;
        }
    }
```

```
    private ExampleConfiguration(String args[]) {
        for( int i = 0; i < args.length; ++i ) {
            String arg = args[i];
            if( arg.equals( "--queue" ) ) {
                setQueueName(getParameter(args, i));
                i++;
            } else if( arg.equals( "--region" ) ) {
                String regionName = getParameter(args, i);
                try {
                    setRegion(Region.getRegion(Regions.fromName(regionName)));
                } catch( IllegalArgumentException e ) {
                    throw new IllegalArgumentException( "Unrecognized region " +
 regionName );
                }
                i++;
            } else if( arg.equals( "--credentials" ) ) {
                String credsFile = getParameter(args, i);
                try {
                    setCredentialsProvider( new
 PropertiesFileCredentialsProvider(credsFile) );
                } catch (AmazonClientException e) {
                    throw new IllegalArgumentException("Error reading credentials from
 " + credsFile, e );
                }
                i++;
            } else {
                throw new IllegalArgumentException("Unrecognized option " + arg);
            }
        }
    }

    private String queueName = DEFAULT_QUEUE_NAME;
    private Region region = DEFAULT_REGION;
    private AWSCredentialsProvider credentialsProvider = new
 DefaultAWSCredentialsProviderChain();

    public String getQueueName() {
        return queueName;
    }

    public void setQueueName(String queueName) {
        this.queueName = queueName;
    }
```

```
    public Region getRegion() {
        return region;
    }

    public void setRegion(Region region) {
        this.region = region;
    }

    public AWSCredentialsProvider getCredentialsProvider() {
        return credentialsProvider;
    }

    public void setCredentialsProvider(AWSCredentialsProvider credentialsProvider) {
        // Make sure they're usable first
        credentialsProvider.getCredentials();
        this.credentialsProvider = credentialsProvider;
    }
}
```

TextMessageSender.java

El siguiente ejemplo de código de Java crea un productor de mensajes de texto.

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

public class TextMessageSender {
    public static void main(String args[]) throws JMSException {
```

```
        ExampleConfiguration config =
ExampleConfiguration.parseConfig("TextMessageSender", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer =
session.createProducer( session.createQueue( config.getQueueName() ) );

        sendMessages(session, producer);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static void sendMessages( Session session, MessageProducer producer ) {
        BufferedReader inputReader = new BufferedReader(
            new InputStreamReader( System.in, Charset.defaultCharset() ) );

        try {
            String input;
            while( true ) {
                System.out.print( "Enter message to send (leave empty to exit): " );
                input = inputReader.readLine();
                if( input == null || input.equals("" ) ) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
```

```
                System.out.println( "Send message " + message.getJMSMessageID() );
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
            System.err.println( "Failed reading input: " + e.getMessage() );
        } catch (JMSException e) {
            System.err.println( "Failed sending message: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

SyncMessageReceiver.java

El siguiente ejemplo de código de Java crea un consumidor de mensajes sincrónico.

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

public class SyncMessageReceiver {
public static void main(String args[]) throws JMSException {
    ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

    ExampleCommon.setupLogging();

    // Create the connection factory based on the config
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
```

```
            AmazonSQSClientBuilder.standard()
                    .withRegion(config.getRegion().getName())
                    .withCredentials(config.getCredentialsProvider())
            );

    // Create the connection
    SQSConnection connection = connectionFactory.createConnection();

    // Create the queue if needed
    ExampleCommon.ensureQueueExists(connection, config.getQueueName());

    // Create the session
    Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
    MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

    connection.start();

    receiveMessages(session, consumer);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println( "Connection closed" );
}

private static void receiveMessages( Session session, MessageConsumer consumer ) {
    try {
        while( true ) {
            System.out.println( "Waiting for messages");
            // Wait 1 minute for a message
            Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        }
    } catch (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
```

```
}
```

AsyncMessageReceiver.java

El siguiente ejemplo de código de Java crea un consumidor de mensajes asincrónico.

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

public class AsyncMessageReceiver {
    public static void main(String args[]) throws JMSException, InterruptedException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());
```

```
        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

        // No messages are processed until this is called
        connection.start();

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }


    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();

        public void waitForOneMinuteOfSilence() throws InterruptedException {
            for(;;) {
                long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
                long remainingTillOneMinuteOfSilence =
                    TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
                if( remainingTillOneMinuteOfSilence < 0 ) {
                    break;
                }
                TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
            }
        }


        @Override
        public void onMessage(Message message) {
            try {
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message " +
 message.getJMSMessageID() );
```

```
                timeOfLastMessage = System.nanoTime();
            } catch (JMSException e) {
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}
```

SyncMessageReceiverClientAcknowledge.java

El siguiente ejemplo de código de Java crea un consumidor sincrónico con el modo de reconocimiento del cliente.

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

/**
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 attempt since in CLIENT_ACKNOWLEDGE mode,
```

```
 * as expected, all the messages prior to the acknowledged messages are also
 acknowledged.
 *
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with client acknowledge mode
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
 session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
 session.createConsumer(session.createQueue(config.getQueueName()));
```

```
        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving no messages since
        // we have acknowledged the second message. Although we didn't explicitly
acknowledge the first message,
        // in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the
explicitly acknowledged message
        // are also acknowledged. Therefore, we have implicitly acknowledged the first
message.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
        // Create a text message and send it
```

```
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
 "Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

SyncMessageReceiverUnorderedAcknowledge.java

El siguiente ejemplo de código de Java crea un consumidor sincrónico con el modo de reconocimiento sin orden.

```
/*
 * 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

El siguiente ejemplo de código XML es un archivo de configuración bean para SpringExample.java.

```
<!--
    Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.

    Licensed under the Apache License, Version 2.0 (the "License").
    You may not use this file except in compliance with the License.
    A copy of the License is located at

    https://aws.amazon.com/apache2.0

    or in the "license" file accompanying this file. This file is distributed
    on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
    express or implied. See the License for the specific language governing
    permissions and limitations under the License.
-->

<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
schema/util/spring-util-3.0.xsd
```

```
    ">

    <bean id="CredentialsProviderBean"
 class="com.amazonaws.auth.DefaultAWSCredentialsProviderChain"/>

    <bean id="ClientBuilder" class="com.amazonaws.services.sqs.AmazonSQSClientBuilder"
 factory-method="standard">
        <property name="region" value="us-east-2"/>
        <property name="credentials" ref="CredentialsProviderBean"/>
    </bean>

    <bean id="ProviderConfiguration"
 class="com.amazon.sqs.javamessaging.ProviderConfiguration">
        <property name="numberOfMessagesToPrefetch" value="5"/>
    </bean>

    <bean id="ConnectionFactory"
 class="com.amazon.sqs.javamessaging.SQSConnectionFactory">
        <constructor-arg ref="ProviderConfiguration" />
        <constructor-arg ref="ClientBuilder" />
    </bean>

    <bean id="Connection" class="javax.jms.Connection"
        factory-bean="ConnectionFactory"
        factory-method="createConnection"
        init-method="start"
        destroy-method="close" />

    <bean id="QueueName" class="java.lang.String">
        <constructor-arg value="SQSJMSClientExampleQueue"/>
    </bean>
</beans>
```

SpringExample.java

El siguiente ejemplo de código Java utiliza el archivo de configuración bean para inicializar sus objetos.

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
```

```
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SpringExample {
    public static void main(String args[]) throws JMSException {
        if( args.length != 1 || !args[0].endsWith(".xml")) {
            System.err.println( "Usage: " + SpringExample.class.getName() + " <spring
 config.xml>" );
            System.exit(1);
        }

        File springFile = new File( args[0] );
        if( !springFile.exists() || !springFile.canRead() ) {
            System.err.println( "File " + args[0] + " doesn't exist or isn't
 readable.");
            System.exit(2);
        }

        ExampleCommon.setupLogging();

        FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext( "file://" +
 springFile.getAbsolutePath() );

        Connection connection;
        try {
            connection = context.getBean(Connection.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the JMS connection to use: " +
 e.getMessage() );
            System.exit(3);
            return;
        }

        String queueName;
        try {
```

```
            queueName = context.getBean("QueueName", String.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the name of the queue to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        if( connection instanceof SQSConnection ) {
            ExampleCommon.ensureQueueExists( (SQSConnection) connection, queueName );
        }

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
session.createConsumer( session.createQueue( queueName) );

        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                    break;
                }
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message" );
            }
        } catch (JMSException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
```

```
}
```

ExampleCommon.java

En el siguiente ejemplo de código Java, se comprueba si existe una cola de Amazon SQS y, si no existe, se crea. También incluye ejemplos de código de registro.

```
/*
 * 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() );
        }
    }
}
```

Implementaciones de JMS 1.1 admitidas por Amazon SQS

La Biblioteca de mensajes Java de Amazon SQS es compatible con las siguientes implementaciones de JMS 1.1. Para obtener más información sobre las características y capacidades compatibles de la Biblioteca de mensajes Java de Amazon SQS, consulte las preguntas frecuentes de Amazon SQS.

Interfaces comunes admitidas

- `Connection`
- `ConnectionFactory`

- `Destination`
- `Session`
- `MessageConsumer`
- `MessageProducer`

Tipos de mensajes admitidos

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

Modos de confirmación de mensajes admitidos

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

> **Note**
>
> El modo `UNORDERED_ACKNOWLEDGE` no forma parte de la especificación JMS 1.1. Este modo ayuda a Amazon SQS a permitir que un cliente de JMS confirme explícitamente un mensaje.

Propiedades reservadas y encabezados definidos por JMS

Para enviar mensajes

Al enviar mensajes, puede definir los siguientes encabezados y propiedades para cada mensaje:

- `JMSXGroupID` (necesario para las colas FIFO, no permitido para las colas estándar)
- `JMS_SQS_DeduplicationId` (opcional para las colas FIFO, no permitido para las colas estándar)

Después de enviar mensajes, Amazon SQS establece los siguientes encabezados y propiedades para cada mensaje:

- `JMSMessageID`
- `JMS_SQS_SequenceNumber` (solo para colas FIFO)

Para recibir mensajes

Después de recibir mensajes, Amazon SQS establece los siguientes encabezados y propiedades para cada mensaje:

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID` (solo para colas FIFO)
- `JMS_SQS_DeduplicationId` (solo para colas FIFO)
- `JMS_SQS_SequenceNumber` (solo para colas FIFO)

Tutoriales de Amazon SQS

En este tema se proporcionan tutoriales que le ayudarán a explorar las características y la funcionalidad de Amazon SQS.

Tutoriales

- Creación de una cola de Amazon SQS mediante AWS CloudFormation
- Tutorial: Envío de un mensaje a una cola de Amazon SQS desde Amazon Virtual Private Cloud

Creación de una cola de Amazon SQS mediante AWS CloudFormation

Utilice la AWS CloudFormation consola junto con una plantilla JSON o YAML para crear una cola de Amazon SQS. Para obtener más información, consulte Cómo trabajar con AWS CloudFormation las plantillas y el *`AWS::SQS::Queue`*recurso en la Guía del AWS CloudFormation usuario.

Para usar AWS CloudFormation para crear una cola de Amazon SQS.

1. Copie el siguiente código JSON a un archivo denominado `MyQueue.json`. Para crear una cola estándar, omita las propiedades `FifoQueue` y `ContentBasedDeduplication`. Para obtener más información sobre la desduplicación basada en el contenido, consulte Procesamiento una sola vez en Amazon SQS.

   > **Note**
   >
   > La cola FIFO debe finalizar con el sufijo `.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. Inicie sesión en la consola de AWS CloudFormation y seleccione Create Stack (Crear pila).
3. En el panel Specify Template (Especificar plantilla), elija Upload a template file (Cargar un archivo de plantilla), elija el archivo `MyQueue.json` y, a continuación, elija Next (Siguiente).
4. En la página Specify Details, escriba `MyQueue` en Stack Name y, a continuación, elija Next.
5. En la página Opciones, seleccione Siguiente.
6. En la página Review (Revisar), elija Create (Crear).

   AWS CloudFormation comienza a crear la `MyQueue` pila y muestra el estado CREATE_IN_PROGRESS. Cuando el proceso se haya completado, AWS CloudFormation mostrará el estado CREATE_COMPLETE.

Filter: Active By Stack Name Showing 1 stack

	Stack Name	Created Time	Status	Description
☑	MyQueue	2017-02-20 11:39:47 UTC-0800	CREATE_COMPLETE	

7. (Opcional) Para mostrar el nombre, la URL y el ARN de la cola, elija el nombre de la pila y, a continuación, en la página siguiente, expanda la sección Outputs.

Tutorial: Envío de un mensaje a una cola de Amazon SQS desde Amazon Virtual Private Cloud

En este tutorial se muestra cómo enviar mensajes a una cola de Amazon SQS a través de una red privada y segura. La red incluye:

- Una VPC que contiene una instancia de Amazon EC2 .
- Un punto de enlace de VPC de interfaz, que permite a la EC2 instancia de Amazon conectarse a Amazon SQS sin utilizar la Internet pública.

Incluso en una red totalmente privada, puede conectarse a la EC2 instancia de Amazon y enviar mensajes a la cola de Amazon SQS. Para obtener más información, consulte Puntos de conexión de Amazon Virtual Private Cloud para Amazon SQS.

> **Important**
>
> - Puede usar Amazon Virtual Private Cloud solo con puntos de conexión HTTPS de Amazon SQS.
> - Al configurar Amazon SQS para enviar mensajes desde Amazon VPC, debe habilitar el DNS privado y especificar los puntos de enlace en el formato `sqs.us-east-2.amazonaws.com` o `sqs.us-east-2.api.aws` para el punto de enlace de doble pila.
> - Amazon SQS también admite puntos de enlace FIPS mediante el PrivateLink uso del servicio de puntos de enlace. `com.amazonaws.region.sqs-fips` Puede conectarse a los puntos de conexión FIPS en este formato. `sqs-fips.region.amazonaws.com`
> - Al utilizar el punto de conexión de doble pila en Amazon Virtual Private Cloud, las solicitudes se enviarán mediante IPv4 y IPv6.

- Los DNS privados no admiten los puntos de enlace heredados, como `queue.amazonaws.com` o `us-east-2.queue.amazonaws.com`.

Paso 1: Crear un EC2 key pair de Amazon

Un key pair te permite conectarte a una EC2 instancia de Amazon. Se componen de una clave pública que cifra la información de inicio de sesión y de una clave privada que la descifra.

1. Inicia sesión en la EC2consola de Amazon.
2. En el menú de navegación, en Network & Security (Red y seguridad), seleccione Key Pairs (Pares de claves).
3. Seleccione Create Key Pair.
4. En el cuadro de diálogo Create Key Pair (Crear par de claves), en Key pair name (Nombre del par de claves), escriba `SQS-VPCE-Tutorial-Key-Pair` y haga clic en Create (Crear).
5. El navegador descarga el archivo de clave privada `SQS-VPCE-Tutorial-Key-Pair.pem` automáticamente.

   > **Important**
   >
   > Guarda este archivo en un lugar seguro. EC2 no genera un `.pem` archivo para el mismo par de claves por segunda vez.

6. Para permitir que un cliente SSH se conecte a tu EC2 instancia, configura los permisos de tu archivo de clave privada de forma que solo tu usuario pueda tener permisos de lectura para él, por ejemplo:

   ```
   chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
   ```

Paso 2: Crea recursos AWS

Para configurar la infraestructura necesaria, debe usar una AWS CloudFormation plantilla, que es un modelo para crear una pila compuesta de AWS recursos, como EC2 instancias de Amazon y colas de Amazon SQS.

La pila de este tutorial incluye los siguientes recursos:

- Una VPC y los recursos de red asociados, como una subred, un grupo de seguridad, una gateway de Internet y una tabla de ruteo.
- Una EC2 instancia de Amazon lanzada en la subred de VPC
- Una cola de Amazon SQS

1. Descarga la AWS CloudFormation plantilla nombrada `SQS-VPCE-Tutorial-CloudFormation.yaml`desde. GitHub
2. Inicie sesión en la consola de AWS CloudFormation.
3. Elija Crear pila.
4. En la página Select Template (Seleccionar plantilla), seleccione Upload a template to Amazon S3 (Cargar una plantilla en Amazon S3), elija el archivo `SQS-VPCE-SQS-Tutorial-CloudFormation.yaml` y haga clic en Next (Siguiente).
5. En la página Specify Details (Especificar detalles), haga lo siguiente:
   a. En Nombre de pila, escriba `SQS-VPCE-Tutorial-Stack`.
   b. Para KeyName, elija SQS-VPCE-Tutorial-Key-Pair.
   c. Elija Siguiente.
6. En la página Opciones, seleccione Siguiente.
7. En la página de revisión, en la sección Capacidades, elija Acepto que AWS CloudFormation podría crear recursos de IAM con nombres personalizados. y, a continuación, elija Crear.

AWS CloudFormation comienza a crear la pila y muestra el estado CREATE_IN_PROGRESS. Cuando el proceso se haya completado, AWS CloudFormation mostrará el estado CREATE_COMPLETE.

Paso 3: Confirma que tu EC2 instancia no es de acceso público

La AWS CloudFormation plantilla lanza una EC2 instancia con un nombre `SQS-VPCE-Tutorial-EC2-Instance` en la VPC. Esta EC2 instancia no permite el tráfico saliente y no puede enviar mensajes a Amazon SQS. Para comprobarlo, debe conectarse a la instancia, intentar conectarse a un punto de conexión público y tratar de enviar un mensaje a Amazon SQS.

1. Inicia sesión en la EC2consola de Amazon.
2. En el menú de navegación, en Instances (Instancias), haga clic en Instances (Instancias).

3. Seleccione SQS-VPCE-. Tutorial-EC2Instance
4. Copie el nombre de host en Public DNS, por ejemplo, ec2-203-0-113-0.us-west-2.compute.amazonaws.com.
5. En el directorio que contiene el par de claves que creó anteriormente, conéctese a la instancia utilizando el comando siguiente; por ejemplo:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. Intente conectarse a un punto de enlace público; por ejemplo:

```
ping amazon.com
```

   Tal y como se esperaba, se produce un error en el intento de conexión.

7. Inicie sesión en la consola de Amazon SQS.
8. En la lista de colas, selecciona la cola creada por tu plantilla, por ejemplo, VPCE-SQS-Tutorial-Stack- -1 IJK. AWS CloudFormation CFQueue ABCDEFGH2
9. En la tabla de detalles, copie la URL, por ejemplo,. https://sqs.us-east-2.amazonaws.com/123456789012/
10. Desde tu EC2 instancia, intenta publicar un mensaje en la cola mediante el siguiente comando, por ejemplo:

```
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."
```

   Tal y como se esperaba, se produce un error al enviar el mensaje.

> **Important**
>
> Posteriormente, cuando cree un punto de conexión de VPC para Amazon SQS, el envío se realizará correctamente.

Paso 4: Crear un punto de conexión de VPC para Amazon SQS

Para conectar su VPC a Amazon SQS, debe definir un punto de conexión de VPC de interfaz. Tras añadir el punto de enlace, puede utilizar la API Amazon SQS de la EC2 instancia de su VPC. Esto le permite enviar mensajes a una cola de la AWS red sin tener que cruzar la Internet pública.

> **Note**
>
> La EC2 instancia sigue sin tener acceso a otros AWS servicios y puntos finales de Internet.

1. Inicie sesión en la consola de Amazon VPC.
2. En el menú de navegación, seleccione Endpoints (Puntos de enlace).
3. Seleccione Crear punto de conexión.
4. En la página Crear punto de conexión, en Nombre del servicio, elija el nombre del servicio para Amazon SQS.

   > **Note**
   >
   > Los nombres de los servicios varían según la AWS región actual. Por ejemplo, si se encuentra en el este de EE. UU. (Ohio), el nombre del servicio es com.amazonaws. ***us-east-2***.sqs.

5. En VPC, seleccione SQS-VPCE-Tutorial-VPC.
6. En Subnets (Subredes), seleccione la red cuyo ID de subred contiene SQS-VPCE-Tutorial-Subnet.
7. En Security group (Grupo de seguridad), haga clic en Select security groups (Seleccionar grupos de seguridad) y elija el grupo de seguridad cuyo nombre de grupo contiene SQS VPCE Tutorial Security Group.
8. Elija Crear punto de conexión.

   Se crea el punto de enlace de la VPC de interfaz y se muestra su ID; por ejemplo, vpce-0ab1cdef2ghi3j456k.
9. Seleccione Cerrar.

   En la consola de Amazon VPC, se abre la página Puntos de enlace.

Amazon VPC comienza a crear el punto de conexión y muestra el estado pendiente. Cuando el proceso se haya completado, Amazon VPC mostrará el estado disponible.

Paso 5: enviar un mensaje a la cola de Amazon SQS

Ahora que su VPC incluye un punto de enlace para Amazon SQS, puede conectarse a EC2 su instancia y enviar mensajes a su cola.

1. Vuelva a conectarse a su EC2 instancia, por ejemplo:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

2. Intente publicar de nuevo un mensaje en la cola utilizando el comando siguiente; por ejemplo:

```
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."
```

   El intento de envío se realiza correctamente y se muestran el MD5 resumen del cuerpo del mensaje y el identificador del mensaje, por ejemplo:

```
{
 "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
 "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

Para obtener información sobre cómo recibir y eliminar el mensaje de la cola creada por la AWS CloudFormation plantilla (por ejemplo, VPCE-SQS-Tutorial-Stack- -1 IJK), consulte. CFQueue ABCDEFGH2 Recepción y eliminación de mensajes en Amazon SQS

Para obtener más información sobre la eliminación de recursos, consulte los siguientes temas:

- Eliminación de un punto de conexión de VPC en la Guía del usuario de Amazon VPC
- Eliminación de una cola de Amazon SQS
- Finaliza tu instancia en la Guía del EC2 usuario de Amazon
- Eliminación de la VPC en la Guía del usuario de Amazon VPC
- Eliminar una pila de la AWS CloudFormation consola en la guía AWS CloudFormation del usuario

- Eliminar tu par de claves en la Guía del EC2 usuario de Amazon

Ejemplos de código para Amazon SQS mediante AWS SDKs

Los siguientes ejemplos de código muestran cómo utilizar Amazon SQS con un kit de desarrollo de AWS software (SDK).

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica a través de llamadas a varias funciones dentro del servicio o combinado con otros Servicios de AWS.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Introducción

Introducción a Amazon SQS

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon SQS.

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
        }
    }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Código para el CMake archivo 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})
```

Código del archivo de origen 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;
}
```

- Para obtener más información sobre la API, consulte ListQueuesla Referencia AWS SDK para C++ de la API.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
   }
  }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para Go de la API.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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);
        }
    }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Inicializar un cliente de Amazon SQS y enumerar las colas.

```
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"));
};
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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")
            }
    }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla referencia sobre el AWS SDK para la API de Kotlin.

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

El archivo `Package.swift`.

```
import PackageDescription

let package = Package(
    name: "sqs-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/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")

    ]
)
```

El código fuente de 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)
        }
    }
}
```

- Para obtener detalles sobre la API, consulte ListQueuesla referencia sobre la API de AWS SDK para Swift.

Ejemplos de código

- Ejemplos básicos de Amazon SQS mediante AWS SDKs
  - Introducción a Amazon SQS
  - Acciones para Amazon SQS mediante AWS SDKs
    - Utilizar AddPermission con una CLI

- Úselo ChangeMessageVisibility con un AWS SDK o CLI
- Utilizar ChangeMessageVisibilityBatch con una CLI
- Úselo CreateQueue con un AWS SDK o CLI
- Úselo DeleteMessage con un AWS SDK o CLI
- Úselo DeleteMessageBatch con un AWS SDK o CLI
- Úselo DeleteQueue con un AWS SDK o CLI
- Úselo GetQueueAttributes con un AWS SDK o CLI
- Úselo GetQueueUrl con un AWS SDK o CLI
- Utilizar ListDeadLetterSourceQueues con una CLI
- Úselo ListQueues con un AWS SDK o CLI
- Utilizar PurgeQueue con una CLI
- Úselo ReceiveMessage con un AWS SDK o CLI
- Utilizar RemovePermission con una CLI
- Úselo SendMessage con un AWS SDK o CLI
- Úselo SendMessageBatch con un AWS SDK o CLI
- Úselo SetQueueAttributes con un AWS SDK o CLI

- Escenarios para el uso de Amazon SQS AWS SDKs
  - Crear una aplicación web que envíe y recupere mensajes mediante Amazon SQS
  - Creación de una aplicación de mensajería con Step Functions
  - Creación de una aplicación de exploración de Amazon Textract
  - Cree y publique en un tema FIFO de Amazon SNS mediante un SDK AWS
  - Detecte personas y objetos en un vídeo con Amazon Rekognition AWS mediante un SDK
  - Gestione mensajes de Amazon SQS de gran tamaño mediante Amazon S3 con un SDK AWS
  - Reciba y procese notificaciones de eventos de Amazon S3 mediante un AWS SDK
  - Publique mensajes de Amazon SNS en las colas de Amazon SQS mediante un SDK AWS
  - Envíe y reciba lotes de mensajes con Amazon SQS mediante un SDK AWS
  - Utilice el marco de procesamiento de AWS mensajes para.NET para publicar y recibir mensajes de Amazon SQS
  - Utilice la biblioteca de mensajería Java de Amazon SQS para trabajar con la interfaz Java Message Service (JMS) para Amazon SQS
  - Trabaje con etiquetas de cola y Amazon SQS mediante un SDK AWS

- Ejemplos de sistemas sin servidor para Amazon SQS
  - Invocar una función de Lambda desde un desencadenador de Amazon SQS
  - Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.

Ejemplos básicos de Amazon SQS mediante AWS SDKs

Los siguientes ejemplos de código muestran cómo utilizar los conceptos básicos de Amazon Simple Queue Service con AWS SDKs.

Ejemplos

- Introducción a Amazon SQS
- Acciones para Amazon SQS mediante AWS SDKs
  - Utilizar AddPermission con una CLI
  - Úselo ChangeMessageVisibility con un AWS SDK o CLI
  - Utilizar ChangeMessageVisibilityBatch con una CLI
  - Úselo CreateQueue con un AWS SDK o CLI
  - Úselo DeleteMessage con un AWS SDK o CLI
  - Úselo DeleteMessageBatch con un AWS SDK o CLI
  - Úselo DeleteQueue con un AWS SDK o CLI
  - Úselo GetQueueAttributes con un AWS SDK o CLI
  - Úselo GetQueueUrl con un AWS SDK o CLI
  - Utilizar ListDeadLetterSourceQueues con una CLI
  - Úselo ListQueues con un AWS SDK o CLI
  - Utilizar PurgeQueue con una CLI
  - Úselo ReceiveMessage con un AWS SDK o CLI
  - Utilizar RemovePermission con una CLI
  - Úselo SendMessage con un AWS SDK o CLI
  - Úselo SendMessageBatch con un AWS SDK o CLI
  - Úselo SetQueueAttributes con un AWS SDK o CLI

Introducción a Amazon SQS

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Amazon SQS.

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
```

```
        }
    }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Código para el CMake archivo 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})
```

Código del archivo de origen 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;
}
```

- Para obtener más información sobre la API, consulte ListQueuesla Referencia AWS SDK para C++ de la API.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
  }
 }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para Go de la API.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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);
        }
    }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Inicializar un cliente de Amazon SQS y enumerar las colas.

```
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"));
};
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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")
```

```
            }
    }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla referencia sobre el AWS SDK para la API de Kotlin.

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

El archivo `Package.swift`.

```
import PackageDescription

let package = Package(
    name: "sqs-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/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")

    ]
)
```

El código fuente de 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)
        }
    }
}
```

- Para obtener detalles sobre la API, consulte ListQueuesla referencia sobre la API de AWS SDK para Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Acciones para Amazon SQS mediante AWS SDKs

Los siguientes ejemplos de código muestran cómo realizar acciones individuales de Amazon SQS con. AWS SDKs Cada ejemplo incluye un enlace a GitHub, donde puede encontrar instrucciones para configurar y ejecutar el código.

Estos fragmentos llaman a la API de Amazon SQS y son fragmentos de código de programas más grandes que se deben ejecutar en contexto. Puede ver las acciones en contexto en Escenarios para el uso de Amazon SQS AWS SDKs .

Los siguientes ejemplos incluyen solo las acciones que se utilizan con mayor frecuencia. Para obtener una lista completa, consulte la Referencia de la API de Amazon Simple Queue Service.

Ejemplos

- Utilizar AddPermission con una CLI
- Úselo ChangeMessageVisibility con un AWS SDK o CLI
- Utilizar ChangeMessageVisibilityBatch con una CLI
- Úselo CreateQueue con un AWS SDK o CLI
- Úselo DeleteMessage con un AWS SDK o CLI
- Úselo DeleteMessageBatch con un AWS SDK o CLI
- Úselo DeleteQueue con un AWS SDK o CLI
- Úselo GetQueueAttributes con un AWS SDK o CLI
- Úselo GetQueueUrl con un AWS SDK o CLI
- Utilizar ListDeadLetterSourceQueues con una CLI
- Úselo ListQueues con un AWS SDK o CLI
- Utilizar PurgeQueue con una CLI
- Úselo ReceiveMessage con un AWS SDK o CLI
- Utilizar RemovePermission con una CLI
- Úselo SendMessage con un AWS SDK o CLI
- Úselo SendMessageBatch con un AWS SDK o CLI

- Úselo SetQueueAttributes con un AWS SDK o CLI

Utilizar **`AddPermission`** con una CLI

Los siguientes ejemplos de código muestran cómo utilizar `AddPermission`.

CLI

AWS CLI

Para añadir un permiso a una cola.

Este ejemplo permite a la AWS cuenta especificada enviar mensajes a la cola especificada.

Comando:

```
aws sqs add-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue --aws-
account-ids 12345EXAMPLE --actions SendMessage
```

Salida:

```
None.
```

- Para obtener más información sobre la API, consulte AddPermissionla Referencia de AWS CLI comandos.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: Este ejemplo permite Cuenta de AWS a la persona especificada enviar mensajes desde la cola especificada.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte AddPermission Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: Este ejemplo permite al especificado Cuenta de AWS enviar mensajes desde la cola especificada.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte AddPermission Herramientas de AWS para PowerShellCmdlet Reference (V5).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte. Uso de Amazon SQS con un SDK AWS En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`ChangeMessageVisibility`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `ChangeMessageVisibility`.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener más información sobre la API, consulta ChangeMessageVisibilityla Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Cambio de la visibilidad del tiempo de espera de un mensaje

En este ejemplo, se cambia la visibilidad del tiempo de espera del mensaje especificado a 10 horas (10 horas * 60 minutos * 60 segundos).

Comando:

```
aws sqs change-message-visibility --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --
visibility-timeout 36000
```

Salida:

```
None.
```

- Para obtener más información sobre la API, consulta ChangeMessageVisibilityla Referencia de AWS CLI comandos.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir un mensaje de Amazon SQS y cambiar la visibilidad del tiempo de espera.

```
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;
};
```

- Para obtener más información sobre la API, consulta ChangeMessageVisibilityla Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir un mensaje de Amazon SQS y cambiar la visibilidad del tiempo de espera.

```
// 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");
    }
  }
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta ChangeMessageVisibilityla Referencia AWS SDK para JavaScript de la API.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo, se cambia el tiempo de espera de visibilidad del mensaje con el identificador de recibo especificado en la cola especificada a 10 horas (10 horas * 60 minutos * 60 segundos = 36000 segundos).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Para obtener más información sobre la API, consulte ChangeMessageVisibility Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se cambia el tiempo de espera de visibilidad del mensaje con el identificador de recibo especificado en la cola especificada a 10 horas (10 horas * 60 minutos * 60 segundos = 36000 segundos).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Para obtener más información sobre la API, consulte ChangeMessageVisibilityla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Ruby

SDK para Ruby

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta ChangeMessageVisibilityla Referencia AWS SDK para Ruby de la API.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilizar **`ChangeMessageVisibilityBatch`** con una CLI

Los siguientes ejemplos de código muestran cómo utilizar `ChangeMessageVisibilityBatch`.

CLI

AWS CLI

Para cambiar la visibilidad del tiempo de espera de varios mensajes en un lote

En este ejemplo, se cambia la visibilidad del tiempo de espera de dos mensajes especificados a 10 horas (10 horas * 60 minutos * 60 segundos).

Comando:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

Archivo de entrada (change-message-visibility-batch.json):

```
[
  {
    "Id": "FirstMessage",
        "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
        "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBkTUH...HifSnw==",
        "VisibilityTimeout": 36000
  }
]
```

Salida:

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

- Para obtener más información sobre la API, consulte ChangeMessageVisibilityBatchla Referencia de AWS CLI comandos.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo, se cambia el tiempo de espera de visibilidad de 2 mensajes con los identificadores de recibo especificado en la cola especificada. El valor de visibilidad del tiempo de espera del primer mensaje se cambia a 10 horas (10 horas * 60 minutos * 60 segundos = 36000 segundos). El valor de visibilidad del tiempo de espera del segundo mensaje se cambia a 5 horas (5 horas * 60 minutos * 60 segundos = 18000 segundos).

```
$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
```

Salida:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- Para obtener más información sobre la API, consulte ChangeMessageVisibilityBatch Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se cambia el tiempo de espera de visibilidad de 2 mensajes con los identificadores de recibo especificado en la cola especificada. El valor de visibilidad del tiempo de espera del primer mensaje se cambia a 10 horas (10 horas * 60 minutos * 60 segundos = 36000 segundos). El valor de visibilidad del tiempo de espera del segundo mensaje se cambia a 5 horas (5 horas * 60 minutos * 60 segundos = 18000 segundos).

```
$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
```

Salida:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- Para obtener más información sobre la API, consulte ChangeMessageVisibilityBatchla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte. Uso de Amazon SQS con un SDK AWS En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`CreateQueue`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `CreateQueue`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Publicación de mensajes en colas
- Enviar y recibir lotes de mensajes
- Utilice la biblioteca de mensajería Java de Amazon SQS para trabajar con la interfaz JMS

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una cola con un nombre específico.

```
    /// <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;
    }
```

Crear una cola de Amazon SQS y enviarle un mensaje.

```
    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;
        }
```

```
    }
```

- Para obtener más información sobre la API, consulta CreateQueuela Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener más información sobre la API, consulta CreateQueuela Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Para crear una cola

En este ejemplo, se crea una cola con el nombre especificado, se establece el período de retención de mensajes en 3 días (3 días * 24 horas * 60 minutos * 60 segundos) y se establece la cola de mensajes fallidos en la cola especificada con un recuento máximo de 1000 mensajes recibidos.

Comando:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

Archivo de entrada (create-queue.json):

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}
```

Salida:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
```

```
}
```

- Para obtener más información sobre la API, consulta CreateQueuela Referencia de AWS CLI comandos.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
}
```

- Para obtener más información sobre la API, consulta CreateQueuela Referencia AWS SDK para Go de la API.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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);
        }
    }
}
```

- Para obtener más información sobre la API, consulta CreateQueuela Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una cola estándar de 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;
};
```

Crear una cola de Amazon SQS con sondeo largo.

```
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;
};
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta CreateQueuela Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una cola estándar de 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);
  }
});
```

Crear una cola de Amazon SQS que espere a que llegue un mensaje.

```
// 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);
  }
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta CreateQueuela Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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()
    }
}
```

- Para obtener más información sobre la API, consulta CreateQueuela referencia sobre el AWS SDK para la API de Kotlin.

PowerShell

Herramientas para la versión PowerShell 4

Ejemplo 1: en este ejemplo se crea una cola con el nombre especificado.

```
New-SQSQueue -QueueName MyQueue
```

Salida:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte CreateQueue Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo se crea una cola con el nombre especificado.

```
New-SQSQueue -QueueName MyQueue
```

Salida:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte CreateQueuela referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta CreateQueuela AWS Referencia de API de SDK for Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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__
```

- Para obtener más información sobre la API, consulta CreateQueuela Referencia AWS SDK para Ruby de la API.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una cola estándar de 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.
```

Crear una cola de Amazon SQS que espere a que llegue un mensaje.

```
    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.
```

- Para obtener más información sobre la API, consulte CreateQueuela referencia sobre la API ABAP del AWS SDK para SAP.

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
        }
```

- Para obtener más información sobre la API, consulta CreateQueuela referencia sobre la API de AWS SDK for Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`DeleteMessage`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteMessage`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Enviar y recibir lotes de mensajes

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Reciba un mensaje de una cola de Amazon SQS y, acto seguido, elimínelo.

```
        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;
        }
    }
```

- Para obtener más información sobre la API, consulta DeleteMessagela Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener más información sobre la API, consulta DeleteMessagela Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Eliminación de un mensaje

En este ejemplo, se elimina el mensaje especificado.

Comando:

```
aws sqs delete-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

Salida:

```
None.
```

- Para obtener más información sobre la API, consulta DeleteMessagela Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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);
        }
```

- Para obtener más información sobre la API, consulta DeleteMessagela Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir y eliminar mensajes de 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,
        })),
      }),
    );
  }
};
```

- Para obtener más información sobre la API, consulta DeleteMessagela Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir y eliminar mensajes de 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);
      }
```

```
    });
  }
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta DeleteMessagela Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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!")
    }
}
```

- Para obtener más información sobre la API, consulta DeleteMessagela referencia sobre el AWS SDK para la API de Kotlin.

PowerShell

Herramientas para la versión PowerShell 4

Ejemplo 1: en este ejemplo, se elimina el mensaje con el identificador de recibo especificado de la cola especificada.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Para obtener más información sobre la API, consulte DeleteMessage Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se elimina el mensaje con el identificador de recibo especificado de la cola especificada.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Para obtener más información sobre la API, consulte DeleteMessagela referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta DeleteMessagela AWS Referencia de API de SDK for Python (Boto3).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte. Uso de Amazon SQS con un SDK AWS En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`DeleteMessageBatch`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteMessageBatch`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Procese notificaciones de eventos de S3
- Publicación de mensajes en colas
- Enviar y recibir lotes de mensajes

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
    }
```

- Para obtener más información sobre la API, consulta DeleteMessageBatchla Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;
            }
```

- Para obtener más información sobre la API, consulta DeleteMessageBatchla Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Eliminación de varios mensajes en un lote

En este ejemplo, se eliminan los mensajes especificados.

Comando:

```
aws sqs delete-message-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-
batch.json
```

Archivo de entrada (delete-message-batch.json):

```
[
  {
        "Id": "FirstMessage",
        "ReceiptHandle": "AQEB1mgl...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
```

```
]
```

Salida:

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- Para obtener más información sobre la API, consulte DeleteMessageBatchla Referencia de AWS CLI comandos.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
}
```

- Para obtener más información sobre la API, consulta DeleteMessageBatchla Referencia AWS SDK para Go de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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,
        })),
      }),
    );
  }
};
```

- Para obtener más información sobre la API, consulta DeleteMessageBatchla Referencia AWS SDK para JavaScript de la API.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo, se eliminan 2 mensajes con los identificadores de recibo especificado de la cola especificada.

```
$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
```

Salida:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- Para obtener más información sobre la API, consulte DeleteMessageBatch Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se eliminan 2 mensajes con los identificadores de recibo especificado de la cola especificada.

```
$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
```

Salida:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- Para obtener más información sobre la API, consulte DeleteMessageBatchla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta DeleteMessageBatchla AWS Referencia de API de SDK for Python (Boto3).

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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.")
        }
```

- Para obtener más información sobre la API, consulta DeleteMessageBatchla referencia sobre la API de AWS SDK for Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`DeleteQueue`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteQueue`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Publicación de mensajes en colas
- Enviar y recibir lotes de mensajes
- Utilice la biblioteca de mensajería Java de Amazon SQS para trabajar con la interfaz JMS

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Eliminar una cola mediante su URL.

```
/// <summary>
/// Delete a queue by its URL.
/// </summary>
/// <param name="queueUrl">The url of the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteQueueByUrl(string queueUrl)
{
    var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
        new DeleteQueueRequest()
        {
            QueueUrl = queueUrl
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para obtener más información sobre la API, consulta DeleteQueuela Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener más información sobre la API, consulta DeleteQueuela Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Para eliminar una cola

En este ejemplo, se elimina la cola especificada.

Comando:

```
aws sqs delete-queue --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

Salida:

```
None.
```

- Para obtener más información sobre la API, consulta DeleteQueuela Referencia de AWS CLI comandos.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
}
```

- Para obtener más información sobre la API, consulta DeleteQueuela Referencia AWS SDK para Go de la API.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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);
        }
    }
}
```

- Para obtener más información sobre la API, consulta DeleteQueuela Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Eliminar una cola de 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;
};
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta DeleteQueuela Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Eliminar una cola de 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);
  }
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta DeleteQueuela Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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!")
    }
}
```

- Para obtener más información sobre la API, consulta DeleteQueuela referencia sobre el AWS SDK para la API de Kotlin.

PowerShell

Herramientas para la versión PowerShell 4

Ejemplo 1: en este ejemplo, se elimina la cola especificada.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Para obtener más información sobre la API, consulte DeleteQueue Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se elimina la cola especificada.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Para obtener más información sobre la API, consulte DeleteQueuela referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta DeleteQueuela AWS Referencia de API de SDK for Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
```

- Para obtener más información sobre la API, consulta DeleteQueuela Referencia AWS SDK para Ruby de la API.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
        MESSAGE 'SQS queue deleted' TYPE 'I'.
    ENDTRY.
```

- Para obtener más información sobre la API, consulte DeleteQueuela referencia sobre la API ABAP del AWS SDK para SAP.

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
    }
```

- Para obtener más información sobre la API, consulta DeleteQueuela referencia sobre la API de AWS SDK for Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`GetQueueAttributes`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetQueueAttributes`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Procese notificaciones de eventos de S3
- Publicación de mensajes en colas

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;
    }
```

- Para obtener más información sobre la API, consulta GetQueueAttributesla Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;


            }
```

- Para obtener más información sobre la API, consulta GetQueueAttributesla Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Obtención de los atributos de una cola

En este ejemplo, se obtienen todos los atributos de la cola especificada.

Comando:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

Salida:

```
{
  "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"
  }
}
```

En este ejemplo, se obtienen únicamente los atributos de tamaño máximo de mensaje y tiempo de espera de visibilidad de la cola especificada.

Comando:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-
names MaximumMessageSize VisibilityTimeout
```

Salida:

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- Para obtener más información sobre la API, consulta GetQueueAttributesla Referencia de AWS CLI comandos.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
}
```

- Para obtener más información sobre la API, consulta GetQueueAttributesla Referencia AWS SDK para Go de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;
};
```

- Para obtener más información sobre la API, consulta GetQueueAttributesla Referencia AWS SDK para JavaScript de la API.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo se enumeran todos los atributos de la cola especificada.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Salida:

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
```

```
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                          [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

Ejemplo 2: en este ejemplo se enumeran de forma separada solo los atributos de la cola especificada.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Salida:

```
VisibilityTimeout                       : 30
DelaySeconds                            : 0
MaximumMessageSize                      : 262144
MessageRetentionPeriod                  : 345600
ApproximateNumberOfMessages             : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed      : 0
CreatedTimestamp                        : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                   : 12/29/2015 2:23:17 PM
QueueARN                                : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                  :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":
```

```
 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- Para obtener más información sobre la API, consulte GetQueueAttributes Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo se enumeran todos los atributos de la cola especificada.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Salida:

```
VisibilityTimeout                       : 30
DelaySeconds                            : 0
MaximumMessageSize                      : 262144
MessageRetentionPeriod                  : 345600
ApproximateNumberOfMessages             : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed      : 0
CreatedTimestamp                        : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                   : 12/29/2015 2:23:17 PM
QueueARN                                : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                  :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
```

```
                                        arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                        [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

Ejemplo 2: en este ejemplo se enumeran de forma separada solo los atributos de la cola especificada.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Salida:

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                        398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                        arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- Para obtener más información sobre la API, consulte GetQueueAttributesla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

```
            }
        }
```

- Para obtener más información sobre la API, consulta GetQueueAttributesla referencia sobre la API de AWS SDK for Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`GetQueueUrl`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetQueueUrl`.

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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.");
            }
        }
    }
```

- Para obtener más información sobre la API, consulta GetQueueUrlla Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener más información sobre la API, consulta GetQueueUrlla Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Obtención de la URL de una cola

En este ejemplo, se obtiene la URL de la cola especificada.

Comando:

```
aws sqs get-queue-url --queue-name MyQueue
```

Salida:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- Para obtener más información sobre la API, consulta GetQueueUrlla Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
            GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

- Para obtener más información sobre la API, consulta GetQueueUrlla Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Obtener la URL de una cola de 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;
};
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta GetQueueUrlla Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Obtener la URL de una cola de 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);
  }
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta GetQueueUrlla Referencia AWS SDK para JavaScript de la API.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: En este ejemplo se muestra la URL de la cola con el nombre especificado.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Salida:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte GetQueueUrl Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: En este ejemplo se muestra la URL de la cola con el nombre especificado.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Salida:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte GetQueueUrlla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta GetQueueUrlla AWS Referencia de API de SDK for Python (Boto3).

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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.
```

- Para obtener más información sobre la API, consulte GetQueueUrlla referencia sobre la API ABAP del AWS SDK para SAP.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilizar **`ListDeadLetterSourceQueues`** con una CLI

Los siguientes ejemplos de código muestran cómo utilizar `ListDeadLetterSourceQueues`.

CLI

AWS CLI

Para enumerar las colas de origen de mensajes fallidos

En este ejemplo, se muestran las colas asociadas a la cola de origen de mensajes fallidos especificada.

Comando:

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Salida:

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- Para obtener más información sobre la API, consulte ListDeadLetterSourceQueuesla Referencia de AWS CLI comandos.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: En este ejemplo se URLs enumeran las colas que dependen de la cola especificada como cola de cartas muertas.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Salida:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Para obtener más información sobre la API, consulte Herramientas de AWS para PowerShell Cmdlet ListDeadLetterSourceQueuesReference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: En este ejemplo se enumeran URLs las colas que se basan en la cola especificada como cola de cartas muertas.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Salida:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Para obtener más información sobre la API, consulte Herramientas de AWS para PowerShell Cmdlet ListDeadLetterSourceQueuesReference (V5).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte. Uso de Amazon SQS con un SDK AWS En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`ListQueues`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `ListQueues`.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;
```

```
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Creación de una lista de las colas

En este ejemplo, se enumeran todas las colas.

Comando:

```
aws sqs list-queues
```

Salida:

```
{
  "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"
  ]
}
```

En este ejemplo, solo se muestran las colas que comienzan por "My".

Comando:

```
aws sqs list-queues --queue-name-prefix My
```

Salida:

```
{
  "QueueUrls": [
```

```
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia de AWS CLI comandos.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
  }
 }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para Go de la API.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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);
        }
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enumerar las colas de Amazon SQS.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
```

```
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enumerar las colas de Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
```

```
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
        }
    }
}
```

- Para obtener más información sobre la API, consulta ListQueuesla referencia sobre el AWS SDK para la API de Kotlin.

PowerShell

Herramientas para la versión PowerShell 4

Ejemplo 1: en este ejemplo, se enumeran todas las colas.

```
Get-SQSQueue
```

Salida:

```
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
```

Ejemplo 2: en este ejemplo se enumeran todas las colas que empiezan con el nombre especificado.

```
Get-SQSQueue -QueueNamePrefix My
```

Salida:

```
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
```

- Para obtener más información sobre la API, consulte ListQueues Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se enumeran todas las colas.

```
Get-SQSQueue
```

Salida:

```
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
```

Ejemplo 2: en este ejemplo se enumeran todas las colas que empiezan con el nombre especificado.

```
Get-SQSQueue -QueueNamePrefix My
```

Salida:

```
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
```

- Para obtener más información sobre la API, consulte ListQueuesla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta ListQueuesla AWS Referencia de API de SDK for Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta ListQueuesla Referencia AWS SDK para Ruby de la API.

Rust

SDK para Rust

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recupere la primera cola de Amazon SQS de la lista en la región.

```
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())
}
```

- Para obtener más información sobre la API, consulta ListQueuesla referencia sobre la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        oo_result = lo_sqs->listqueues( ).        " oo_result is returned for
testing purposes. "
        MESSAGE 'Retrieved list of queues.' TYPE 'I'.
    ENDTRY.
```

- Para obtener más información sobre la API, consulte ListQueuesla referencia sobre la API ABAP del AWS SDK para SAP.

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
            }
        }
```

- Para obtener más información sobre la API, consulta ListQueuesla referencia sobre la API de AWS SDK for Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilizar **`PurgeQueue`** con una CLI

Los siguientes ejemplos de código muestran cómo utilizar `PurgeQueue`.

CLI

AWS CLI

Para purgar una cola

En este ejemplo, se eliminan los mensajes de la cola especificada.

Comando:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyNewQueue
```

Salida:

```
None.
```

- Para obtener más información sobre la API, consulte PurgeQueuela Referencia de AWS CLI comandos.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo, se eliminan todos los mensajes de la cola especificada.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte PurgeQueue Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se eliminan todos los mensajes de la cola especificada.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte PurgeQueuela referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte. Uso de Amazon SQS con un SDK AWS En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`ReceiveMessage`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `ReceiveMessage`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Gestione mensajes de gran tamaño con S3
- Procese notificaciones de eventos de S3
- Publicación de mensajes en colas
- Enviar y recibir lotes de mensajes

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir mensajes de una cola mediante su URL

```
    /// <summary>
    /// Receive messages from a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>The list of messages.</returns>
    public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
 maxMessages)
    {
        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
            new ReceiveMessageRequest()
            {
                QueueUrl = queueUrl,
                MaxNumberOfMessages = maxMessages,
                WaitTimeSeconds = 1
            });
        return messageResponse.Messages;
    }
```

Reciba un mensaje de una cola de Amazon SQS y, acto seguido, elimínelo.

```
        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;
        }
    }
```

- Para obtener más información sobre la API, consulta ReceiveMessagela Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener más información sobre la API, consulta ReceiveMessagela Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Recepción de un mensaje

En este ejemplo, se reciben hasta 10 mensajes disponibles y se devuelven todos los atributos disponibles.

Comando:

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All --message-
attribute-names All --max-number-of-messages 10
```

Salida:

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

En este ejemplo, se recibe el siguiente mensaje disponible y se devuelven solo los SentTimestamp atributos SenderId y, además, el atributo del PostalCode mensaje.

Comando:

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-
names SenderId SentTimestamp --message-attribute-names PostalCode
```

Salida:

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

```
}
```

- Para obtener más información sobre la API, consulte ReceiveMessagela Referencia de AWS CLI comandos.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
}
```

- Para obtener más información sobre la API, consulta ReceiveMessagela Referencia AWS SDK para Go de la API.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;
```

- Para obtener más información sobre la API, consulta ReceiveMessagela Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir un mensaje de una cola de 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,
        })),
      }),
    );
  }
};
```

Recibir un mensaje de una cola de Amazon SQS mediante el soporte de sondeos largos.

```
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;
};
```

- Para obtener más información sobre la API, consulta ReceiveMessagela Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir un mensaje de una cola de Amazon SQS mediante el soporte de sondeos largos.

```
// 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);
  }
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta ReceiveMessagela Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
        }
    }
}
```

- Para obtener más información sobre la API, consulta ReceiveMessagela referencia sobre el AWS SDK para la API de Kotlin.

PowerShell

Herramientas para la versión PowerShell 4

Ejemplo 1: en este ejemplo se muestra la información de los 10 mensajes siguientes que se van a recibir en la cola especificada. La información contendrá valores para los atributos del mensaje especificado, si existen.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Salida:

```
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==
```

- Para obtener más información sobre la API, consulte ReceiveMessage Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo se muestra la información de los 10 mensajes siguientes que se van a recibir en la cola especificada. La información contendrá valores para los atributos del mensaje especificado, si existen.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Salida:

```
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==
```

- Para obtener más información sobre la API, consulte ReceiveMessagela referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta ReceiveMessagela AWS Referencia de API de SDK for Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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__
```

- Para obtener más información sobre la API, consulta ReceiveMessagela Referencia AWS SDK para Ruby de la API.

Rust

SDK para Rust

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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(())
}
```

- Para obtener más información sobre la API, consulta ReceiveMessagela referencia sobre la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Recibir un mensaje de una cola de 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.
```

Recibir un mensaje de una cola de Amazon SQS mediante el soporte de sondeos largos.

```
   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.
```

- Para obtener más información sobre la API, consulte ReceiveMessagela referencia sobre la API ABAP del AWS SDK para SAP.

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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("---")
        }
```

- Para obtener más información sobre la API, consulta ReceiveMessagela referencia sobre la API de AWS SDK for Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilizar **`RemovePermission`** con una CLI

Los siguientes ejemplos de código muestran cómo utilizar `RemovePermission`.

CLI

AWS CLI

Eliminación de un permiso

En este ejemplo, se quita el permiso con la etiqueta especificada de la cola indicada.

Comando:

```
aws sqs remove-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue
```

Salida:

```
None.
```

- Para obtener más información sobre la API, consulte RemovePermissionla Referencia de AWS CLI comandos.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo, se quita el permiso con la etiqueta especificada de la cola indicada.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte RemovePermission Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se quita el permiso con la etiqueta especificada de la cola indicada.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte RemovePermissionla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte. Uso de Amazon SQS con un SDK AWS En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`SendMessage`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `SendMessage`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Gestione mensajes de gran tamaño con S3
- Enviar y recibir lotes de mensajes

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una cola de Amazon SQS y enviarle un mensaje.

```
    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;
        }
    }
```

- Para obtener más información sobre la API, consulta SendMessagela Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener más información sobre la API, consulta SendMessagela Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Para enviar un mensaje

En este ejemplo, se envía un mensaje con el cuerpo del mensaje, el período de retraso y los atributos del mensaje especificados a la cola especificada.

Comando:

```
aws sqs send-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --message-body "Information about the
 largest city in Any Region." --delay-seconds 10 --message-attributes file://
send-message.json
```

Archivo de entrada (send-message.json):

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

Salida:

```
{
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- Para obtener más información sobre la API, consulta SendMessagela Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

A continuación se muestran dos ejemplos de la `SendMessage` operación:

- Envía un mensaje con un cuerpo y un retraso

- Envía un mensaje con un cuerpo y los atributos del mensaje

Envía un mensaje con un cuerpo y un retraso.

```
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);
        }
    }
}
```

Envía un mensaje con el cuerpo y los atributos del mensaje.

```
    /**
     * <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);
```

```
        }
    }
```

- Para obtener más información sobre la API, consulte SendMessagela Referencia AWS SDK for Java 2.x de la API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enviar un mensaje a una cola de 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;
};
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta SendMessagela Referencia AWS SDK para JavaScript de la API.

SDK para JavaScript (v2)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Enviar un mensaje a una cola de 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);
  }
});
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener más información sobre la API, consulta SendMessagela Referencia AWS SDK para JavaScript de la API.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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.")
```

```
    }
}
```

- Para obtener más información sobre la API, consulta SendMessagela referencia sobre el AWS SDK para la API de Kotlin.

PowerShell

Herramientas para la versión PowerShell 4

Ejemplo 1: en este ejemplo se envía un mensaje con los atributos y el cuerpo del mensaje especificados a la cola especificada, con un retraso de 10 segundos en la entrega del mensaje.

```
$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
```

Salida:

```
MD5OfMessageAttributes             MD5OfMessageBody                   MessageId

----------------------             ----------------                   ---------

1d3e51347bc042efbdf6dda31EXAMPLE   51b0a3256d59467f973009b73EXAMPLE   c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- Para obtener más información sobre la API, consulte SendMessage Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo se envía un mensaje con los atributos y el cuerpo del mensaje especificados a la cola especificada, con un retraso de 10 segundos en la entrega del mensaje.

```
$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
```

Salida:

```
MD5OfMessageAttributes              MD5OfMessageBody                   MessageId

----------------------              ----------------                   ---------

1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- Para obtener más información sobre la API, consulte SendMessagela referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta SendMessagela AWS Referencia de API de SDK for Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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__
```

- Para obtener más información sobre la API, consulta SendMessagela Referencia AWS SDK para Ruby de la API.

Rust

SDK para Rust

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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(())
}
```

- Para obtener más información sobre la API, consulta SendMessagela referencia sobre la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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.
```

- Para obtener más información sobre la API, consulte SendMessagela referencia sobre la API ABAP del AWS SDK para SAP.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`SendMessageBatch`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `SendMessageBatch`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Enviar y recibir lotes de mensajes

CLI

AWS CLI

Envío de varios mensajes por lotes

En este ejemplo, se envían 2 mensajes con los cuerpos de los mensajes, los períodos de retraso y los atributos de los mensajes especificados a la cola especificada.

Comando:

```
aws sqs send-message-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-
batch.json
```

Archivo de entrada (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"
      }
        }
  }
]
```

Salida:

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

- Para obtener más información sobre la API, consulte SendMessageBatchla Referencia de AWS CLI comandos.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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);
```

- Para obtener más información sobre la API, consulta SendMessageBatchla Referencia AWS SDK for Java 2.x de la API.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo, se envían 2 mensajes con los atributos y los cuerpos de los mensajes especificados a la cola especificada. La entrega se retrasa 15 segundos para el primer mensaje y 10 segundos para el segundo mensaje.

```
$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
```

Salida:

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- Para obtener más información sobre la API, consulte SendMessageBatch Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo, se envían 2 mensajes con los atributos y los cuerpos de los mensajes especificados a la cola especificada. La entrega se retrasa 15 segundos para el primer mensaje y 10 segundos para el segundo mensaje.

```
$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
```

Salida:

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- Para obtener más información sobre la API, consulte SendMessageBatchla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta SendMessageBatchla AWS Referencia de API de SDK for Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta SendMessageBatchla Referencia AWS SDK para Ruby de la API.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Úselo **`SetQueueAttributes`** con un AWS SDK o CLI

Los siguientes ejemplos de código muestran cómo utilizar `SetQueueAttributes`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Publicación de mensajes en colas

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Establecer el atributo de política de una cola para un tema.

```
    /// <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;
    }
```

- Para obtener más información sobre la API, consulta SetQueueAttributesla Referencia AWS SDK para .NET de la API.

C++

SDK para C++

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

Configurar una cola de mensajes fallidos.

```
        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();
}
```

Configurar una cola de Amazon SQS para utilizar sondeos largos.

```
        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();
}
```

- Para obtener más información sobre la API, consulta SetQueueAttributesla Referencia AWS SDK para C++ de la API.

CLI

AWS CLI

Fijación de los atributos de una cola

Este ejemplo establece la cola especificada en un retraso de entrega de 10 segundos, un tamaño máximo de mensaje de 128 KB (128 KB * 1024 bytes), un período de retención de mensajes de 3 días (3 días * 24 horas * 60 minutos * 60 segundos), un tiempo de espera de recepción de mensajes de 20 segundos y un tiempo de espera de visibilidad predeterminado de 60 segundos. En este ejemplo, también se asocia la cola de mensajes fallidos especificada a un recuento máximo de 1000 mensajes recibidos.

Comando:

```
aws sqs set-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-
attributes.json
```

Archivo de entrada (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"
}
```

Salida:

```
None.
```

- Para obtener más información sobre la API, consulte SetQueueAttributesla Referencia de AWS CLI comandos.

Go

SDK para Go V2

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
```

- Para obtener más información sobre la API, consulta SetQueueAttributesla Referencia AWS SDK para Go de la API.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Configure un Amazon SQS para que utilice el cifrado del lado del servidor (SSE) mediante una clave KMS personalizada.

```
    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();
        }
    }
```

- Para obtener más información sobre la API, consulte la referencia de SetQueueAttributesla AWS SDK for Java 2.x API.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;
};
```

Configurar una cola de Amazon SQS para utilizar sondeos largos.

```
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;
};
```

Configurar una cola de mensajes fallidos.

```
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;
};
```

- Para obtener más información sobre la API, consulta SetQueueAttributesla Referencia AWS SDK para JavaScript de la API.

PowerShell

Herramientas para la PowerShell versión 4

Ejemplo 1: en este ejemplo se muestra cómo establecer una política para suscribirse una cola a un tema de SNS. Cuando se publica un mensaje en un tema, se envía a la cola suscritas.

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
```

```
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
          }
      }
    }
  ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

Ejemplo 2: en este ejemplo se establecen los atributos especificados de la cola especificada.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte SetQueueAttributes Herramientas de AWS para PowerShellCmdlet Reference (V4).

Herramientas para la versión 5 PowerShell

Ejemplo 1: en este ejemplo se muestra cómo establecer una política para suscribirse una cola a un tema de SNS. Cuando se publica un mensaje en un tema, se envía a la cola suscritas.

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
```

```
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
          }
      }
    }
  ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

Ejemplo 2: en este ejemplo se establecen los atributos especificados de la cola especificada.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Para obtener más información sobre la API, consulte SetQueueAttributesla referencia de Herramientas de AWS para PowerShell cmdlets (V5).

Swift

SDK para Swift

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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.")
        }
```

- Para obtener más información sobre la API, consulta SetQueueAttributesla referencia sobre la API de AWS SDK for Swift.

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Escenarios para el uso de Amazon SQS AWS SDKs

Los siguientes ejemplos de código muestran cómo implementar escenarios comunes en Amazon SQS con. AWS SDKs Estos escenarios muestran cómo llevar a cabo tareas específicas con

llamadas a varias funciones dentro de Amazon SQS o en combinación con otros Servicios de AWS. En cada escenario se incluye un enlace al código fuente completo, con instrucciones de configuración y ejecución del código.

Los escenarios requieren un nivel intermedio de experiencia para ayudarlo a entender las acciones de servicio en su contexto.

Ejemplos

- Crear una aplicación web que envíe y recupere mensajes mediante Amazon SQS
- Creación de una aplicación de mensajería con Step Functions
- Creación de una aplicación de exploración de Amazon Textract
- Cree y publique en un tema FIFO de Amazon SNS mediante un SDK AWS
- Detecte personas y objetos en un vídeo con Amazon Rekognition AWS mediante un SDK
- Gestione mensajes de Amazon SQS de gran tamaño mediante Amazon S3 con un SDK AWS
- Reciba y procese notificaciones de eventos de Amazon S3 mediante un AWS SDK
- Publique mensajes de Amazon SNS en las colas de Amazon SQS mediante un SDK AWS
- Envíe y reciba lotes de mensajes con Amazon SQS mediante un SDK AWS
- Utilice el marco de procesamiento de AWS mensajes para.NET para publicar y recibir mensajes de Amazon SQS
- Utilice la biblioteca de mensajería Java de Amazon SQS para trabajar con la interfaz Java Message Service (JMS) para Amazon SQS
- Trabaje con etiquetas de cola y Amazon SQS mediante un SDK AWS

Crear una aplicación web que envíe y recupere mensajes mediante Amazon SQS

En los siguientes ejemplos de código se muestra cómo crear una aplicación de mensajería con Amazon SQS.

Java

SDK para Java 2.x

Muestra cómo usar la API de Amazon SQS para desarrollar una API de REST de Spring que envíe y recupere mensajes.

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK para Kotlin

Muestra cómo usar la API de Amazon SQS para desarrollar una API de REST de Spring que envíe y recupere mensajes.

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Amazon SQS

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una aplicación de mensajería con Step Functions

El siguiente ejemplo de código muestra cómo crear una aplicación de AWS Step Functions mensajería que recupere los registros de mensajes de una tabla de base de datos.

Python

SDK para Python (Boto3)

Muestra cómo utilizar AWS SDK para Python (Boto3) with AWS Step Functions para crear una aplicación de mensajería que recupere los registros de mensajes de una tabla de Amazon DynamoDB y los envíe con Amazon Simple Queue Service (Amazon SQS). La máquina de estados se integra con una AWS Lambda función para escanear la base de datos en busca de mensajes no enviados.

- Crear una máquina de estado que recupere y actualice los registros de mensajes de una tabla de Amazon DynamoDB.
- Actualizar la definición de la máquina de estado para que también envíe mensajes a Amazon Simple Queue Service (Amazon SQS).
- Iniciar y detener las ejecuciones de la máquina de estado.
- Conectar con Lambda, DynamoDB y Amazon SQS desde una máquina de estado mediante integraciones de servicio.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una aplicación de exploración de Amazon Textract

Los siguientes ejemplos de código indican cómo explorar la salida de Amazon Textract mediante una aplicación interactiva.

JavaScript

SDK para JavaScript (v3)

Muestra cómo utilizarla AWS SDK para JavaScript para crear una aplicación de React que utilice Amazon Textract para extraer datos de la imagen de un documento y mostrarlos en una página web interactiva. Este ejemplo se ejecuta en un navegador web y requiere una identidad autenticada de Amazon Cognito para las credenciales. Para el almacenamiento utiliza Amazon Simple Storage Service (Amazon S3) y para las notificaciones consulta una cola de Amazon Simple Queue Service (Amazon SQS) que está suscrita a un tema de Amazon Simple Notification Service (Amazon SNS).

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK para Python (Boto3)

Muestra cómo utilizar Amazon Textract para detectar elementos de texto, formulario y tabla en una imagen de documento. AWS SDK para Python (Boto3) La imagen de entrada y la salida de Amazon Textract aparecen en una aplicación Tkinter que permite explorar los elementos detectados.

- Envía la imagen de un documento a Amazon Textract y explora el resultado de los elementos detectados.
- Envía imágenes directamente a Amazon Textract o mediante un bucket de Amazon Simple Storage Service (Amazon S3).
- Utilice la función asincrónica APIs para iniciar un trabajo que publique una notificación en un tema del Amazon Simple Notification Service (Amazon SNS) cuando se complete el trabajo.
- Consulta una cola de Amazon Simple Queue Service (Amazon SQS) en busca de un mensaje de finalización de trabajo y muestra los resultados.

Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en. GitHub

Servicios utilizados en este ejemplo

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS

- Amazon Textract

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Cree y publique en un tema FIFO de Amazon SNS mediante un SDK AWS

Los siguientes ejemplos de código muestran cómo crear y publicar en un tema FIFO de Amazon SNS.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo

- crea un tema FIFO de Amazon SNS, dos colas FIFO de Amazon SQS y una cola estándar.
- suscribe las colas al tema y publica un mensaje en el tema.

La prueba verifica la recepción del mensaje en cada cola. El ejemplo completo también muestra la incorporación de políticas de acceso y, al final, elimina los recursos.

```
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);
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
  - CreateTopic
  - Publish
  - Subscribe

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crea un tema FIFO de Amazon SNS, suscribe una cola FIFO de Amazon SQS al tema y publica un mensaje en el tema.

```
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
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - CreateTopic
  - Publish

- Subscribe

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crea un tema de FIFO, suscribe una cola FIFO de Amazon SQS al tema y publica un mensaje en un tema de 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.
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para SAP ABAP.
  - CreateTopic
  - Publish
  - Subscribe

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Detecte personas y objetos en un vídeo con Amazon Rekognition AWS mediante un SDK

Los siguientes ejemplos de código indican cómo detectar personas y objetos en un video con Amazon Rekognition.

Java

SDK para Java 2.x

Muestra cómo utilizar la API Java de Amazon Rekognition para crear una aplicación que detecte rostros y objetos en vídeos ubicados en un bucket de Amazon Simple Storage Service (Amazon S3). La aplicación envía al administrador una notificación por correo electrónico con los resultados mediante Amazon Simple Email Service (Amazon SES).

Para ver el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulta el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS

- Amazon SQS

Python

SDK para Python (Boto3)

Utilice Amazon Rekognition para detectar caras, objetos y personas en videos iniciando trabajos de detección asíncronos. Este ejemplo también configura Amazon Rekognition para que notifique un tema de Amazon Simple Notification Service (Amazon SNS) cuando se finalicen los trabajos y suscriba una cola de Amazon Simple Queue Service (Amazon SQS) al tema. Cuando la cola recibe un mensaje sobre un trabajo, se recupera el trabajo y se muestran los resultados

Este ejemplo se ve mejor en GitHub. Para obtener el código fuente completo y las instrucciones sobre cómo configurarlo y ejecutarlo, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Gestione mensajes de Amazon SQS de gran tamaño mediante Amazon S3 con un SDK AWS

El siguiente ejemplo de código muestra cómo utilizar la biblioteca de clientes extendida de Amazon SQS para trabajar con mensajes de Amazon SQS de gran tamaño.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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;
        }
    }
```

- Para obtener más información, consulte la Guía para desarrolladores de AWS SDK for Java 2.x.
- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
  - CreateBucket
  - PutBucketLifecycleConfiguration
  - ReceiveMessage
  - SendMessage

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Reciba y procese notificaciones de eventos de Amazon S3 mediante un AWS SDK

El siguiente ejemplo de código muestra cómo trabajar con notificaciones de eventos de S3 de una forma orientada a objetos.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo muestra cómo procesar un evento de notificación de S3 utilizando 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);
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
  - DeleteMessageBatch
  - GetQueueAttributes
  - PutBucketNotificationConfiguration
  - ReceiveMessage

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Publique mensajes de Amazon SNS en las colas de Amazon SQS mediante un SDK AWS

En el siguiente ejemplo de código, se muestra cómo:

- Crear un tema (FIFO o no FIFO)
- Suscribir varias colas al tema con la opción de aplicar un filtro
- Publicar mensajes en el tema
- Sondear las colas en busca de los mensajes recibidos

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar un escenario interactivo en un símbolo del sistema

```
/// <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;
    }
}
```

Crear una clase que encapsule las operaciones de 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;
    }
}
```

Crear una clase que encapsule las operaciones de 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;
    }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para .NET .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage

- SetQueueAttributes
- Subscribe
- Unsubscribe

C++

SDK para C++

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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();
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para C++ .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe

- Unsubscribe

Go

SDK para Go V2

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar un escenario interactivo en un símbolo del sistema.

```
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))
}
```

Defina una estructura que encapsule las acciones de Amazon SNS utilizadas en este ejemplo.

```
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
}
```

Defina una estructura que encapsule las acciones de Amazon SQS utilizadas en este ejemplo.

```
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
}
```

Eliminación de recursos.

```
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)
  }
 }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Go .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish

- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Java

SDK para Java 2.x

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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 "";
    }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este es el punto de partida para este escenario.

```
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();
};
```

El código anterior proporciona las dependencias necesarias e inicia el escenario. La siguiente sección contiene la mayor parte del ejemplo.

```
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();
    }
  }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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
    }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Kotlin.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic

- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Swift

SDK para Swift

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de 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)
        }
    }
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Swift.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Envíe y reciba lotes de mensajes con Amazon SQS mediante un SDK AWS

En el siguiente ejemplo de código, se muestra cómo:

- Crear una cola de Amazon SQS.
- Enviar lotes de mensajes a la cola.

- Recibir lotes de mensajes de la cola.
- Eliminar lotes de mensajes de una cola.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Como se muestra en los siguientes ejemplos, puede gestionar las operaciones de mensajes por lotes con Amazon SQS mediante dos enfoques diferentes: AWS SDK for Java 2.x

SendRecvBatch.java utiliza operaciones por lotes explícitas. Los lotes de mensajes se crean manualmente y se llaman directamente`sendMessageBatch()`. `deleteMessageBatch()` También gestionas las respuestas por lotes, incluidos los mensajes fallidos. Este enfoque le proporciona un control total sobre el tamaño de los lotes y la gestión de errores. Sin embargo, requiere más código para gestionar la lógica de procesamiento por lotes.

SimpleProducerConsumer.java usa la `SqsAsyncBatchManager` biblioteca de alto nivel para el procesamiento automático de solicitudes por lotes. Las llamadas individuales `sendMessage()` y las `deleteMessage()` llamadas se realizan con las mismas firmas de métodos que en el cliente estándar. El SDK almacena automáticamente estas llamadas en búfer y las envía como operaciones por lotes. Este enfoque requiere cambios mínimos en el código y, al mismo tiempo, proporciona beneficios en el rendimiento del procesamiento por lotes.

Utilice el procesamiento por lotes explícito cuando necesite un control detallado sobre la composición de los lotes y la gestión de errores. Utilice el procesamiento por lotes automático cuando desee optimizar el rendimiento con cambios mínimos en el código.

SendRecvBatch.java: utiliza operaciones por lotes explícitas con los mensajes.

```
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: utiliza el procesamiento automático de mensajes por lotes.

```
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.
                }
            }
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear funciones para contener funciones de mensajes de 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
```

Utilizar las funciones de contenedor para enviar y recibir mensajes por lotes.

```
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)
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilice el marco de procesamiento de AWS mensajes para.NET para publicar y recibir mensajes de Amazon SQS

El siguiente ejemplo de código muestra cómo crear aplicaciones que publiquen y reciban mensajes de Amazon SQS mediante el AWS Message Processing Framework para.NET.

.NET

SDK para .NET

Proporciona un tutorial sobre el marco de procesamiento de AWS mensajes para .NET. El tutorial crea una aplicación web que permite al usuario publicar un mensaje de Amazon SQS y una aplicación de línea de comandos que recibe el mensaje.

Para obtener el código fuente completo e instrucciones sobre cómo configurarlo y ejecutarlo, consulte el tutorial completo en la Guía para AWS SDK para .NET desarrolladores y el ejemplo correspondiente GitHub.

Servicios utilizados en este ejemplo

- Amazon SQS

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilice la biblioteca de mensajería Java de Amazon SQS para trabajar con la interfaz Java Message Service (JMS) para Amazon SQS

El siguiente ejemplo de código muestra cómo utilizar la biblioteca de mensajería Java de Amazon SQS para trabajar con la interfaz JMS.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Los siguientes ejemplos funcionan con colas estándar de Amazon SQS e incluyen:

- Envío de un mensaje de texto.
- Recibir mensajes de forma sincrónica.
- Recibir mensajes de forma asíncrona.
- Recibir mensajes mediante el modo CLIENT_ACNOWLEDGE.
- Recibir mensajes mediante el modo UNORDERED_ACNOWLEDGE.
- Uso de Spring para inyectar dependencias.
- Una clase de utilidad que proporciona los métodos comunes utilizados en los otros ejemplos.

Para obtener más información sobre el uso de JMS con Amazon SQS, consulte la Guía para desarrolladores de Amazon SQS.

Envío de un mensaje de texto.

```
    /**
     * 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);
        }
    }
```

Recibir mensajes de forma sincrónica.

```
    /**
     * 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);
        }
    }
```

Recibir mensajes de forma asíncrona.

```
    /**
     * 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" );
    }
```

Recibir mensajes mediante el modo CLIENT_ACNOWLEDGE.

```
    /**
     * 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();
       }
   }
```

Recibir mensajes mediante el modo UNORDERED_ACNOWLEDGE.

```
   /**
    * 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();
        }
    }
```

Uso de Spring para inyectar dependencias.

```
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);
        }
    }
}
```

Definiciones de Spring Bean.

```
<?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>
```

Clase de utilidad que proporciona los métodos comunes utilizados en los demás ejemplos.

```
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);
            }
        };
    }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
  - CreateQueue
  - DeleteQueue

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Trabaje con etiquetas de cola y Amazon SQS mediante un SDK AWS

El siguiente ejemplo de código muestra cómo realizar una operación de etiquetado con Amazon SQS.

Java

SDK para Java 2.x

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

En el siguiente ejemplo, se crean etiquetas para una cola, se enumeran las etiquetas y se las quita.

```
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);
        }
    }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x .
  - ListQueueTags
  - TagQueue
  - UntagQueue

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Ejemplos de sistemas sin servidor para Amazon SQS

Los siguientes ejemplos de código muestran cómo utilizar Amazon SQS con. AWS SDKs

Ejemplos

- Invocar una función de Lambda desde un desencadenador de Amazon SQS
- Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.

Invocar una función de Lambda desde un desencadenador de Amazon SQS

En los siguientes ejemplos de código, se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir mensajes de una cola de SQS. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Uso de un evento de SQS con Lambda mediante .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 para Go V2

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Uso de un evento de SQS con Lambda mediante 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 para Java 2.x

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Uso de un evento de SQS con Lambda mediante 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 para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumir un evento de SQS con JavaScript Lambda mediante.

```
// 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;
  }
}
```

Consumir un evento de SQS con TypeScript Lambda mediante.

```
// 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 para PHP

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Uso de un evento de SQS con Lambda mediante 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 para Python (Boto3)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Uso de un evento de SQS con Lambda mediante 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 para Ruby

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Uso de un evento de SQS con Lambda mediante 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 para Rust

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Uso de un evento de SQS con Lambda mediante 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
}
```

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulte Uso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.

En los siguientes ejemplos de código, se muestra cómo implementar una respuesta parcial por lotes para las funciones de Lambda que reciben eventos de una cola de SQS. La función informa los

errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

.NET

SDK para .NET

> **Note**
>
> Hay más información al respecto GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de SQS con Lambda mediante .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 para Go V2

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
```

```
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {

  if /* Your message processing condition here */ {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK para Java 2.x

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de SQS con Lambda mediante 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>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(message.getMessageId()));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más información. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Informe de errores de artículos de lotes de SQS con JavaScript Lambda mediante.

```
// 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}`);
}
```

Informe de errores de artículos de lotes de SQS con TypeScript Lambda mediante.

```
// 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 para PHP

> **Note**
>
> Hay más información al respecto. GitHub Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de SQS con Lambda mediante 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 para Python (Boto3)

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de SQS con Lambda mediante 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:
                # process message
            except Exception as e:
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

Ruby

SDK para Ruby

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de SQS con Lambda mediante 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 para Rust

> **Note**
>
> Hay más información GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de SQS con Lambda mediante 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
}
```

Para obtener una lista completa de guías para desarrolladores del AWS SDK y ejemplos de código, consulteUso de Amazon SQS con un SDK AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Solución de problemas de Amazon SQS

En este tema se proporcionan consejos de solución de problemas y errores comunes que pueden surgir al utilizar la consola Amazon SQS, la API de Amazon SQS u otras herramientas con Amazon SQS. Si se encuentra con un problema que no aparezca en esta lista, puede utilizar el botón Comentarios de esta página para notificarlo.

Para obtener más consejos sobre la resolución de problemas y respuestas a preguntas comunes de soporte, visite el Centro de conocimientos de AWS.

Temas

- Solución de problemas de error de acceso rechazado en Amazon SQS
- Solución de errores de API de Amazon SQS
- Solución de problemas relacionados con la cola de mensajes fallidos y la redireccionamiento de DLQ
- Solución de problemas de limitación de FIFO en Amazon SQS
- Solucionar problemas de mensajes no devueltos en una llamada a la API de Amazon ReceiveMessage SQS
- Solucionar problemas de errores de red en Amazon SQS
- Resolución de problemas de las colas de Amazon Simple Queue Service mediante AWS X-Ray

Solución de problemas de error de acceso rechazado en Amazon SQS

En los siguientes temas se describen las causas más comunes de los errores `AccessDenied` o `AccessDeniedException` en las llamadas a la API de Amazon SQS. Para obtener más información sobre cómo solucionar estos errores, consulte ¿Cómo soluciono los errores «» o «AccessDenied» en las llamadas a la API de Amazon SQS? AccessDeniedException en la Guía del Centro de AWS Conocimiento.

Ejemplos de mensajes de error:

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to
        the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

-o bien-

```
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.
```

Política de colas y política de IAM de Amazon SQS

Para comprobar si el solicitante tiene los permisos adecuados para realizar una operación de Amazon SQS, haga lo siguiente:

- Identifique la entidad principal de IAM que realiza la llamada a la API de Amazon SQS. Si la entidad principal de IAM es de la misma cuenta, la política de colas de Amazon SQS o la política de AWS Identity and Access Management (IAM) deben incluir permisos que permitan explícitamente el acceso a la acción.
- Si la entidad principal es una entidad de IAM:
  - Para comprobar su rol o usuario de IAM, consulte la esquina superior derecha de la AWS Management Console o utilice el comando `aws sts get-caller-identity`.
  - Compruebe las políticas de IAM asociadas al usuario o rol de IAM. Puede usar uno de los métodos siguientes:
    - Pruebe las políticas de IAM con el simulador de políticas de IAM.
    - Revisar los diferentes tipos de políticas de IAM.
  - Si es necesario, edite la política de usuario de IAM.
  - Compruebe la política de colas y edítela si es necesario.
- Si el principal es un AWS servicio, la política de colas de Amazon SQS debe permitir el acceso de forma explícita.
- Si la entidad principal es una entidad principal multicuenta, tanto la política de colas de Amazon SQS como la política de IAM deben permitir el acceso de forma explícita.
- Si la política utiliza un elemento de condición, compruebe que la condición restrinja el acceso.

> **Important**
>
> Una denegación explícita en cualquier política invalida todos los permisos concedidos. Estos son algunos ejemplos básicos de políticas de Amazon SQS.

AWS Key Management Service permisos

Si su cola de Amazon SQS tiene el cifrado del lado del servidor (SSE) activado y una opción gestionada por el cliente AWS KMS key, se deben conceder permisos tanto a los productores como a los consumidores. Para confirmar si una cola está cifrada, puede utilizar el atributo `KmsMasterKeyId` de la API `GetQueueAttributes` o la consola de colas, en Cifrado.

- Permisos necesarios para los productores:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

- Permisos necesarios para los consumidores:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- Permisos necesarios para el acceso entre cuentas:

```
{
"Effect": "Allow",
"Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
```

```
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

Elija una de las siguientes opciones para habilitar el cifrado de una cola de Amazon SQS:

- SSE-Amazon SQS (clave de cifrado creada y administrada por el servicio de Amazon SQS).
- AWS clave predeterminada administrada (alias/aws/sqs)
- Clave administrada por el cliente

Sin embargo, si utiliza una clave KMS AWS administrada, no puede modificar la política de claves predeterminada. Por lo tanto, para proporcionar acceso a otros servicios y entre cuentas, utilice la clave administrada por el cliente. Esto le permite editar la política de claves.

Política de punto de conexión de VPC

Si accede a Amazon SQS a través de un punto de conexión de Amazon Virtual Private Cloud (Amazon VPC), la política de puntos de conexión de VPC de Amazon SQS debe permitir el acceso. Puede crear una política para los puntos de conexión de Amazon VPC correspondiente a Amazon SQS, en la que podrá especificar lo siguiente:

1. La entidad principal que puede realizar acciones.
2. Las acciones que se pueden realizar.
3. Los recursos en los que se pueden llevar a cabo las acciones.

En el siguiente ejemplo, la política de puntos finales de la VPC especifica que el usuario *MyUser* de IAM puede enviar mensajes a la cola de Amazon SQS. *MyQueue* A otras acciones, a los usuarios de IAM y a los recursos de Amazon SQS se les niega el acceso a través del punto de conexión de VPC.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
```

```
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

Política de control de servicios de la organización

Si Cuenta de AWS pertenece a una organización, AWS Organizations las políticas pueden impedir que acceda a sus colas de Amazon SQS. De forma predeterminada, AWS Organizations las políticas no bloquean ninguna solicitud a Amazon SQS. Sin embargo, asegúrese de que sus AWS Organizations políticas no se hayan configurado para bloquear el acceso a las colas de Amazon SQS. Para obtener instrucciones sobre cómo comprobar sus AWS Organizations políticas, consulte Enumerar todas las políticas en la Guía del AWS Organizations usuario.

Solución de errores de API de Amazon SQS

En los siguientes temas se describen los errores más comunes que se devuelven al realizar llamadas a la API de Amazon SQS y cómo solucionarlos.

QueueDoesNotExist error

Este error se devolverá cuando el servicio de Amazon SQS no encuentre la cola mencionada para la acción de Amazon SQS.

Posibles causas y soluciones:

- Región incorrecta: revise la configuración del cliente de Amazon SQS para confirmar que ha configurado la región correcta en el cliente. Si no configuras una región en el cliente, el SDK o AWS CLI elige la región en el archivo de configuración o en la variable de entorno. Si el SDK no encuentra ninguna región en el archivo de configuración, establece la región en us-east-1 de forma predeterminada.
- Es posible que la cola se haya eliminado recientemente: si la cola se eliminó antes de realizar la llamada a la API, la llamada a la API devolverá este error. Comprueba si hay CloudTrail alguna DeleteQueueoperación antes de que se produzca el error.
- Problemas con los permisos: si el usuario o rol de AWS Identity and Access Management (IAM) solicitante no tiene los permisos necesarios, es posible que reciba el siguiente error:

  ```
  The specified queue does not exist or you do not have access to it.
  ```

Compruebe los permisos y realice la llamada a la API con los permisos correctos.

Para obtener más información sobre cómo solucionar el `QueueDoesNotExist` error, consulte ¿Cómo soluciono el QueueDoesNotExist error cuando realizo llamadas a la API a mi cola de Amazon SQS? en la Guía del Centro de AWS Conocimiento.

InvalidAttributeValue error

Este error se devolverá al actualizar la política de recursos de colas de Amazon SQS o las propiedades con una política o una entidad principal incorrectas.

Posibles causas y soluciones:

- Política de recursos no válida: compruebe que la política de recursos contenga todos los campos obligatorios. Para obtener más información, consulte Referencia de elementos de la política JSON de IAM y Validar políticas de IAM. También puede usar el generador de políticas de IAM para crear y probar una política de recursos de Amazon SQS. Asegúrese de que la política esté en formato JSON.
- Entidad principal no válida: asegúrese de que el elemento `Principal` existe en la política de recursos y de que el valor es válido. Si su elemento `Principal` de política de recursos de Amazon SQS incluye una entidad de IAM, asegúrese de que la entidad existe antes de usar la política. Amazon SQS valida la política de recursos y comprueba la entidad de IAM. Si la entidad de IAM no existe, se producirá un error. Para confirmar las entidades de IAM, utilice los caracteres `GetRole`y `GetUser` APIs.

Para obtener información adicional sobre cómo solucionar un `InvalidAttributeValue` error, consulte ¿Cómo soluciono el QueueDoesNotExist error cuando realizo llamadas a la API a mi cola de Amazon SQS? en la Guía del Centro de Conocimiento AWS .

ReceiptHandle error

Al realizar una llamada a la API `DeleteMessage`, el error `ReceiptHandleIsInvalid` o `InvalidParameterValue` podría devolverse si el controlador de recepción es incorrecto o ha caducado.

- ReceiptHandleIsInvalid error: si el identificador del recibo es incorrecto, recibirás un error similar al de este ejemplo:

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
 The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- InvalidParameterValue error: si el identificador del recibo está caducado, recibirás un error similar al de este ejemplo:

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:
 Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The
 receipt handle has expired.
```

Posibles causas y soluciones:

El controlador de recepción se crea para cada mensaje recibido y solo es válido durante el período de tiempo de espera de visibilidad. Cuando vence el tiempo de espera de visibilidad, el mensaje pasa a ser visible en la cola para los consumidores. Cuando vuelva a recibir el mensaje del consumidor, recibirá un nuevo controlador de recepción. Para evitar errores en el controlador de recepción de vencimiento o valor incorrecto, utilice el controlador de recepción correcto para eliminar el mensaje dentro del período de espera de visibilidad de las colas de Amazon SQS.

Para obtener información adicional sobre cómo solucionar un `ReceiptHandle` error, consulte ¿Cómo se solucionan los errores «» y ReceiptHandleIsInvalid «» cuando uso la llamada a la API de Amazon DeleteMessage SQS? InvalidParameterValue en la Guía del Centro de AWS Conocimiento.

Solución de problemas relacionados con la cola de mensajes fallidos y la redireccionamiento de DLQ

En los siguientes temas se describen las causas más comunes de los problemas de DLQ y redireccionamiento de DLQ y cómo solucionarlos. Para obtener más información, consulte How do I troubleshoot Amazon SQS DLQ redrive issues? en la Guía del Centro de conocimiento de AWS .

Problemas de DLQ

Obtenga información sobre los problemas más comunes de DLQ y cómo resolverlos.

La visualización de mensajes mediante la consola puede hacer que los mensajes se muevan a una cola de mensajes fallidos

Amazon SQS cuenta la visualización de un mensaje en la consola; para la política de redireccionamiento de la cola correspondiente. Por tanto, si ve un mensaje en la consola el número de veces especificado en la política de redireccionamiento de la cola correspondiente, el mensaje se enviará a la cola de mensajes fallidos de dicha cola.

Para configurar este comportamiento, puede elegir una de las siguientes opciones:

- Aumentar la configuración Maximum Receives de la política de redireccionamiento de la cola correspondiente.
- Evitar visualizar los mensajes de la cola correspondiente en la consola.

Los valores de **`NumberOfMessagesSent`** y **`NumberOfMessagesReceived`** para una cola de mensajes fallidos no coinciden

Si envía manualmente un mensaje a una cola de mensajes fallidos, la métrica `NumberOfMessagesSent` lo captura. Sin embargo, si se envía un mensaje a una cola de mensajes fallidos como consecuencia de un intento fallido de procesamiento, esta métrica no lo captura. Por tanto, es posible que los valores de `NumberOfMessagesSent` y `NumberOfMessagesReceived` sean diferentes.

Creación y configuración de un redireccionamiento de cola de mensajes fallidos

El redireccionamiento de cola de mensajes fallidos requiere que establezca los permisos adecuados para que Amazon SQS reciba mensajes de la cola de mensajes fallidos y envíe mensajes a la cola de destino. Si no tiene los permisos correctos, la tarea de redireccionamiento de cola de mensajes fallidos puede fallar. Puede ver el estado de su tarea de redireccionamiento de mensajes para solucionar los problemas e intentarlo de nuevo.

Administración de errores de mensajes en cola estándar y FIFO

Las colas estándar siguen procesando mensajes hasta que termina el período de retención. Este procesamiento continuo minimiza las posibilidades de que la cola quede bloqueada por mensajes no consumidos. El hecho de tener un gran número de mensajes que el consumidor no elimina de forma reiterada podría aumentar los costos y suponer una carga adicional para el hardware. Para mantener bajos los costos, mueva los mensajes fallidos a la cola de mensajes fallidos.

Las colas estándar también permiten un número elevado de mensajes en tránsito. Si la mayoría de los mensajes no se pueden consumir y no se envían a una cola de mensajes fallidos, la tasa de procesamiento de mensajes se puede ralentizar. Para mantener la eficacia de la cola, asegúrese de que su aplicación se ocupa correctamente del procesamiento de mensajes.

Las colas FIFO proporcionan un procesamiento único al consumir mensajes en secuencia de un grupo de mensajes. Aunque el consumidor puede seguir recuperando mensajes ordenados de otro grupo de mensajes, el primer grupo de mensajes sigue sin estar disponible hasta que el mensaje que bloquea la cola se procese correctamente o se traslade a una cola de mensajes fallidos.

Además, las colas FIFO permiten que haya menos mensajes en tránsito. Para evitar que su cola FIFO se bloquee por un mensaje, asegúrese de que su aplicación se encarga correctamente del procesamiento de mensajes.

Para obtener más información, consulte Cuotas de mensajes de Amazon SQS y Prácticas recomendadas para Amazon SQS.

Problemas de redireccionamiento de DLQ

Obtenga información sobre los problemas más comunes de redireccionamiento de DLQ y cómo resolverlos.

AccessDenied problema de permisos

El error `AccessDenied` se produce cuando el redireccionamiento de DLQ falla porque la entidad de AWS Identity and Access Management (IAM) no tiene los permisos necesarios.

Ejemplos de mensajes de error:

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

Se requieren los siguientes permisos de API para realizar solicitudes de redireccionamiento de DLQ:

Inicio de un redireccionamiento de mensajes:

- Permisos de la cola de mensajes fallidos:
  - `sqs:StartMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`

  - `kms:Decrypt`: cuando la cola de mensajes fallidos o la cola de origen original están cifradas.
- Permisos de cola de destino:
  - `sqs:SendMessage`
  - `kms:GenerateDataKey`: cuando la cola de destino está cifrada.
  - `kms:Decrypt`: cuando la cola de destino está cifrada.

Cancelación de un redireccionamiento de mensajes en curso:

- Permisos de la cola de mensajes fallidos:
  - `sqs:CancelMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt`: cuando la cola de mensajes fallidos o la cola de origen original están cifradas.

Visualización del estado de movimiento de mensajes:

- Permisos de la cola de mensajes fallidos:
  - `sqs:ListMessageMoveTasks`
  - `sqs:GetQueueAttributes`

Error **`NonExistentQueue`**

El error `NonExistentQueue` se produce cuando la cola de origen de Amazon SQS no existe o se ha eliminado. Compruebe y redireccione a una cola de Amazon SQS que esté presente.

Ejemplos de mensajes de error:

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

Error CouldNotDetermineMessageSource

El error `CouldNotDetermineMessageSource` se produce al intentar iniciar un redireccionamiento de DLQ en las siguientes situaciones:

- Un mensaje de Amazon SQS enviado directamente al DLQ con la API `SendMessage`.
- Un mensaje del tema AWS Lambda o función del Amazon Simple Notification Service (Amazon SNS) con el DLQ configurado.

Para resolver este error, seleccione Redireccionar a un destino personalizado al iniciar el redireccionamiento. A continuación, introduzca el ARN de la cola de Amazon SQS para mover todos los mensajes del DLQ a la cola de destino.

Ejemplos de mensajes de error:

```
Failed: CouldNotDetermineMessageSource
```

Solución de problemas de limitación de FIFO en Amazon SQS

De forma predeterminada, las colas FIFO admiten 300 transacciones por segundo y por acción de la API para `SendMessage`, `ReceiveMessage` y `DeleteMessage`. Las solicitudes de más de 300 TPS reciben el error `ThrottlingException` incluso si los mensajes en la cola están disponibles. Para mitigar esto, puede usar los siguientes métodos:

- Habilitación del alto rendimiento para las colas FIFO en Amazon SQS.
- Utilice las acciones por lotes de la API Amazon SQS `SendMessageBatch`, `DeleteMessageBatch` y `ChangeMessageVisibilityBatch` para aumentar el límite de TPS hasta 3000 mensajes por segundo por acción de la API y reducir los costos. En el caso de la API de `ReceiveMessage`, defina el parámetro `MaxNumberofMessages` para recibir hasta diez mensajes por transacción. Para obtener más información, consulte Acciones por lotes de Amazon SQS.
- En el caso de las colas FIFO con un alto rendimiento, siga las recomendaciones para optimizar la utilización de las particiones. Envía mensajes con el mismo grupo de mensajes IDs en lotes. Elimine los mensajes o cambie los valores de tiempo de espera de visibilidad de los mensajes en lotes con identificadores de recepción procedentes de las mismas solicitudes de API `ReceiveMessage`.
- Aumente el número de valores únicos de `MessageGroupId`. Esto permite una distribución uniforme entre las particiones de cola FIFO. Para obtener más información, consulte Uso del ID de grupo de mensajes de Amazon SQS.

Para obtener más información, consulte ¿Por qué mi cola FIFO de Amazon SQS no devuelve todos los mensajes o los mensajes de otros grupos? en la Guía del Centro de conocimiento de AWS .

Solucionar problemas de mensajes no devueltos en una llamada a la API de Amazon ReceiveMessage SQS

En los siguientes temas se describen las causas más comunes por las que no se puede devolver un mensaje de Amazon SQS a los consumidores y cómo solucionarlas. Para obtener más información, consulte Why can't I receive messages from my Amazon SQS queue? en la Guía del Centro de conocimiento de AWS .

Cola vacía

Para determinar si una cola está vacía, utilice el sondeo largo para llamar a la API `ReceiveMessage`. También puedes usar las `ApproximateNumberOfMessagesDelayed` CloudWatch métricas `ApproximateNumberOfMessagesVisible`, `ApproximateNumberOfMessagesNotVisible`, y. Si todos los valores de la métrica se establecen en 0 durante varios minutos, la cola se considera vacía.

Se ha superado el límite de tránsito

Si utiliza sondeos largos y si se supera el límite de tránsito de la cola (20 000 en el caso de FIFO y 120 000 en estándar de forma predeterminada), Amazon SQS no devolverá los mensajes de error que superen los límites de la cuota.

Retraso de mensajes

Si la cola de Amazon SQS está configurada como una cola de retraso o los mensajes se enviaron con temporizadores de mensajes, los mensajes no estarán visibles hasta que finalice el tiempo de retraso. Para comprobar si una cola está configurada como cola de retrasos, utilice el atributo `DelaySeconds` de la API `GetQueueAttributes` o desde la consola de colas situada en Plazo de entrega. Comprueba la `ApproximateNumberOfMessagesDelayed` CloudWatch métrica para saber si algún mensaje se retrasa.

El mensaje está en tránsito

Si un consumidor diferente ha sondeado el mensaje, el mensaje estará en movimiento o será invisible durante el período de tiempo de espera de visibilidad. Es posible que los sondeos

adicionales arrojen una recepción vacía. Comprueba la ApproximateNumberOfMessagesVisible CloudWatchmétrica para entender la cantidad de mensajes que están disponibles para ser recibidos. En el caso de las colas FIFO, si un mensaje con el ID de grupo de mensajes está en tránsito, no se devolverán más mensajes a menos que elimine el mensaje o este se haga visible. Esto se debe a que el orden de los mensajes se mantiene a nivel de grupo de mensajes en una cola FIFO.

Método de sondeo

Si utiliza un sondeo corto, (WaitTimeSecondses 0) Amazon SQS toma una muestra de un subconjunto de sus servidores y devuelve los mensajes únicamente de esos servidores. Por lo tanto, es posible que no reciba los mensajes aunque estén disponibles para recibirlos. Las solicitudes de sondeo posteriores devolverán los mensajes.

Si utiliza un sondeo largo, Amazon SQS sondea todos los servidores y envía una respuesta después de recopilar al menos un mensaje disponible y hasta el número máximo especificado. Si el valor de ReceiveMessage WaitTimeSecondses demasiado bajo, es posible que no reciba todos los mensajes disponibles.

Solucionar problemas de errores de red en Amazon SQS

En los siguientes temas se describen las causas más comunes de los problemas de red en Amazon SQS y cómo solucionarlos.

ETIMEOUT error

El error ETIMEOUT se produce cuando el cliente no puede establecer una conexión TCP con un punto de conexión de Amazon SQS.

Solución de problemas:

- Comprobar la conexión de la red

  Pruebe su conexión de red a Amazon SQS ejecutando comandos como `telnet`.

  Example: telnet sqs.us-east-1.amazonaws.com 443
- Comprobar la configuración de red
  - Asegúrese de que las reglas, las rutas y las listas de control de acceso (ACLs) del firewall local permiten el tráfico en el puerto que utiliza.
  - Las reglas de salida de los grupos de seguridad deben permitir el tráfico al puerto 80 o 443.

- Las reglas de salida de la ACL de red deben permitir el tráfico al puerto TCP 80 o 443.
- Las reglas de entrada de la ACL de la red deben permitir el tráfico en los puertos TCP 1024-65535.
- Las instancias de Amazon Elastic Compute Cloud (Amazon EC2) que se conectan a la Internet pública deben tener conectividad a Internet.

- Puntos de conexión de Amazon Virtual Private Cloud (Amazon VPC)

  Si accede a Amazon SQS a través de un punto de conexión de Amazon VPC, el grupo de seguridad del punto de conexión debe permitir el tráfico entrante al grupo de seguridad del cliente en el puerto 443. La ACL de red asociada a la subred del punto de conexión de VPC debe tener esta configuración:

  - Las reglas de salida de la ACL de red deben permitir el tráfico en los puertos TCP 1024-65535 (puertos efímeros).
  - Las reglas de entrada de la ACL de la red deben permitir el tráfico en el puerto 443.

Además, la política de puntos finales de VPC AWS Identity and Access Management (IAM) de Amazon SQS debe permitir el acceso. El siguiente ejemplo de política de punto final de VPC especifica que el usuario *MyUser* de IAM puede enviar mensajes a la cola de Amazon SQS. *MyQueue* A otras acciones, a los usuarios de IAM y a los recursos de Amazon SQS se les niega el acceso a través del punto de conexión de 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

El error UnknownHostException se produce cuando no se ha podido determinar la dirección IP del host.

Solución de problemas:

Utilice la utilidad nslookup para devolver la dirección IP asociada al nombre de host:

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI o SDK para puntos finales heredados de Python:

```
nslookup <region>.queue.amazonaws.com
```

Si ha recibido un resultado incorrecto, siga las instrucciones de How does DNS work and how do I troubleshoot partial or intermittent DNS failures? en la Guía del Centro de conocimiento de AWS .

Si ha recibido un resultado válido, es probable que se trate de un problema de aplicación. Para resolver problemas de nivel de aplicación, pruebe los siguientes métodos:

- Reinicie la aplicación.
- Confirme que la aplicación Java no tenga una caché de DNS defectuosa. Si es posible, configure la aplicación para que cumpla con el TTL del DNS. Para obtener más información, consulte Configuración del TTL de la JVM para las búsquedas de nombres de DNS.

Para obtener información adicional sobre cómo solucionar errores de red, consulte ¿Cómo soluciono los errores de conexión «ETIMEOUT» y «» de Amazon SQS? UnknownHostException en la Guía del Centro de Conocimiento.AWS

Resolución de problemas de las colas de Amazon Simple Queue Service mediante AWS X-Ray

AWS X-Ray recopila datos sobre las solicitudes que atiende su aplicación y le permite ver y filtrar los datos para identificar posibles problemas y oportunidades de optimización. Para cualquier solicitud rastreada hasta su aplicación, puede ver información detallada sobre la solicitud, la respuesta y las llamadas que la aplicación realiza a los AWS recursos intermedios, los microservicios, las bases de datos y la web HTTP. APIs

Para enviar encabezados de AWS X-Ray seguimiento a través de Amazon SQS, puede realizar una de las siguientes acciones:

- Utilizar el encabezado de rastreo `X-Amzn-Trace-Id`.
- Utilizar el atributo de sistema de mensajes `AWSTraceHeader`.

Para recopilar datos sobre errores y latencia, debe instrumentar el cliente de `AmazonSQS` mediante el SDK de AWS X-Ray.

Puede utilizar la AWS X-Ray consola para ver el mapa de conexiones entre Amazon SQS y otros servicios que utiliza su aplicación. También puede utilizar la consola para ver métricas como la latencia media y las tasas de errores. Para obtener más información, consulte Amazon SQS y AWS X-Ray en la Guía para desarrolladores de AWS X-Ray .

Seguridad en Amazon SQS

En esta sección se proporciona información sobre la seguridad, la autenticación y el control de acceso de Amazon SQS, así como sobre el lenguaje de la política de acceso de Amazon SQS.

Temas

- Protección de datos en Amazon SQS
- Identity and Access Management en Amazon SQS
- Registro y monitoreo en Amazon SQS
- Validación de la conformidad de Amazon SQS
- Resiliencia en Amazon SQS
- Seguridad de la infraestructura en Amazon SQS
- Prácticas recomendadas de seguridad para Amazon SQS

Protección de datos en Amazon SQS

El modelo de se aplica a protección de datos en Amazon Simple Queue Service. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global en la que se ejecutan todos los Nube de AWS. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulta las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulta la publicación de blog sobre el Modelo de responsabilidad compartida de AWS y GDPR en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Se utiliza SSL/TLS para comunicarse con AWS los recursos. Se recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.

- Configure la API y el registro de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre el uso de CloudTrail senderos para capturar AWS actividades, consulte Cómo trabajar con CloudTrail senderos en la Guía del AWS CloudTrail usuario.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utiliza servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-3 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulta Estándar de procesamiento de la información federal (FIPS) 140-3.

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabaja con Amazon SQS u otro dispositivo Servicios de AWS mediante la consola, la API o. AWS CLI AWS SDKs Cualquier dato que ingrese en etiquetas o campos de texto de formato libre utilizados para nombres se puede emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

En las siguientes secciones se proporciona información sobre la protección de datos en Amazon SQS.

Cifrado de datos en Amazon SQS

La protección de datos se refiere a salvaguardarlos en tránsito (al desplazarse desde y hacia Amazon SQS) y en reposo (almacenados en discos en centros de datos de Amazon SQS). Puede proteger los datos en tránsito con una Capa de sockets seguros (SSL, Secure Sockets Layer) o con el cifrado del lado del cliente. De forma predeterminada, Amazon SQS almacena los mensajes y archivos mediante el cifrado de disco. Puede proteger los datos en reposo solicitando a Amazon SQS que cifre los mensajes antes de guardarlos en el sistema de archivos cifrados en sus centros de datos. Amazon SQS recomienda usar SSE para optimizar el cifrado de datos.

Temas

- Cifrado en reposo en Amazon SQS

- Administración de claves de Amazon SQS

Cifrado en reposo en Amazon SQS

El cifrado del servidor (SSE) le permite transferir información confidencial en colas cifradas. El SSE protege el contenido de los mensajes en las colas mediante claves de cifrado administradas por SQL (SSE-SQS) o claves administradas en el (SSE-KMS). AWS Key Management Service Para obtener información sobre cómo administrar el SSE mediante el, consulte lo siguiente: AWS Management Console

- Configuración de SSE-SQS para una cola (consola)
- Configuración de SSE-KMS para una cola (consola)

Para obtener información sobre la gestión de la ESS mediante las AWS SDK para Java (y `GetQueueAttributes` las acciones `CreateQueueSetQueueAttributes`, y), consulte los siguientes ejemplos:

- Uso del cifrado del servidor con colas de Amazon SQS
- Configuración de los permisos de KMS para Servicios de AWS

SSE cifra los mensajes en cuanto Amazon SQS los recibe. Los mensajes se almacenan cifrados y Amazon SQS los descifra únicamente cuando se envían a un consumidor autorizado.

> **Important**
>
> Todas las solicitudes hechas a las colas con SSE habilitado deben usar HTTPS y Signature Version 4.
>
> Una cola cifrada que usa la clave predeterminada (clave de KMS AWS administrada para Amazon SQS) no puede invocar una función de Lambda en otra. Cuenta de AWS
>
> Algunas funciones de AWS los servicios que pueden enviar notificaciones a Amazon SQS mediante la AWS Security Token Service `AssumeRole` acción son compatibles con SSE, pero solo funcionan con colas estándar:
>
> - Enlaces de ciclo de vida de escalado automático
> - AWS Lambda Colas de mensajes fallidos

Para obtener información acerca de la compatibilidad de otros servicios con temas de cifrado, consulte Configure los permisos de KMS para los servicios AWS y la documentación de los servicios.

AWS KMS combina hardware y software seguros y de alta disponibilidad para proporcionar un sistema de administración de claves adaptado a la nube. Cuando utiliza Amazon SQS con AWS KMS, las claves de datos que cifran los datos de sus mensajes también se cifran y almacenan con los datos que protegen.

A continuación, se describen los beneficios de usar AWS KMS:

- Puede crear y administrar AWS KMS keys usted mismo.
- También puede usar la clave de KMS AWS administrada para Amazon SQS, que es única para cada cuenta y región.
- Los estándares AWS KMS de seguridad pueden ayudarle a cumplir los requisitos de conformidad relacionados con el cifrado.

Para obtener más información, consulte ¿Qué es AWS Key Management Service? en la Guía para desarrolladores de AWS Key Management Service .

Ámbito de cifrado

SSE cifra el cuerpo de un mensaje en una cola de Amazon SQS.

SSE no cifra lo siguiente:

- Metadatos de la cola (atributos y nombre de la cola)
- Metadatos del mensaje (ID de mensaje, marca temporal y atributos)
- Métricas por cola

El cifrado de un mensaje evita que usuarios no autorizados o anónimos obtengan acceso a su contenido. Con SSE activado, se rechazarán las solicitudes `SendMessage` y `ReceiveMessage` anónimas a la cola cifrada. Las prácticas recomendadas de seguridad de Amazon SQS desaconsejan utilizar solicitudes anónimas. Si desea enviar solicitudes anónimas a una cola de

Amazon SQS, asegúrese de desactivar SSE. Esto no afecta al funcionamiento normal de Amazon SQS:

- Un mensaje se cifra únicamente si se envía con posterioridad a la habilitación del cifrado de una cola. Amazon SQS no cifra mensajes atrasados.
- Cualquier mensaje cifrado permanece en dicho estado aunque el cifrado de su cola esté deshabilitado.

Mover un mensaje a una cola de mensajes fallidos no afecta a su cifrado:

- Cuando Amazon SQS mueve un mensaje de una cola de origen cifrada a una cola de mensajes fallidos sin cifrar, el mensaje permanece cifrado.
- Cuando Amazon SQS mueve un mensaje de una cola de origen sin cifrar a una cola de mensajes fallidos cifrada, el mensaje permanece sin cifrar.

Términos clave

Los siguientes términos clave pueden ayudarle a comprender mejor la funcionalidad de SSE. Para obtener descripciones detalladas, consulte la Referencia de la API de Amazon Simple Queue Service.

Clave de datos

La clave (DEK) es responsable de cifrar el contenido de los mensajes de Amazon SQS.

Para obtener más información, consulte Claves de datos en la Guía para desarrolladores de AWS Key Management Service en la Guía para desarrolladores de AWS Encryption SDK .

Periodo de reutilización de la clave de datos

El tiempo, en segundos, durante el que Amazon SQS puede reutilizar una clave de datos para cifrar o descifrar los mensajes antes de volver a llamar. AWS KMS Un entero que representa segundos, entre 60 segundos (1 minuto) y 86 400 segundos (24 horas). El valor predeterminado es 300 (5 minutos). Para obtener más información, consulte Descripción del período de reutilización de la clave de datos.

> **Note**
>
> En el improbable caso de que no pueda conectarse AWS KMS, Amazon SQS seguirá utilizando la clave de datos en caché hasta que se restablezca la conexión.

ID de clave de KMS

El alias, el ARN del alias, el ID de clave o el ARN de clave de una clave de KMS AWS administrada o una clave de KMS personalizada, en su cuenta o en otra cuenta. Si bien el alias de la clave de KMS AWS administrada para Amazon SQS es siempre`alias/aws/sqs`, el alias de una clave de KMS personalizada puede, por ejemplo, ser. `alias/MyAlias` Puede utilizar estas claves de KMS para proteger los mensajes que se encuentran en las colas de Amazon SQS.

> **Note**
>
> Tenga en cuenta lo siguiente:
>
> - Si no especifica una clave de KMS personalizada, Amazon SQS utilizará la clave de KMS AWS gestionada para Amazon SQS.
> - La primera vez que utilice AWS Management Console para especificar la clave de KMS AWS gestionada de Amazon SQS para una cola, AWS KMS crea la clave de KMS AWS gestionada para Amazon SQS.
> - Como alternativa, la primera vez que utilice la `SendMessageBatch` acción `SendMessage` o en una cola con SSE activado, AWS KMS creará la clave de KMS AWS gestionada para Amazon SQS.

Puede crear claves de KMS, definir las políticas que controlan cómo se pueden usar las claves de KMS y auditar el uso de las claves de KMS mediante la sección de claves administradas por el cliente de la AWS KMS consola o la `CreateKey` AWS KMS acción. Para obtener más información, consulte Claves de KMS y Creación de claves en la Guía para desarrolladores de AWS Key Management Service . Para ver más ejemplos de identificadores clave de KMS, consulta KeyIdla Referencia de la AWS Key Management Service API. Para obtener información sobre la búsqueda de identificadores de claves de KMS, consulte Encontrar el ID y el ARN de la clave en la Guía para desarrolladores de AWS Key Management Service .

> **Important**
>
> Su uso AWS KMS conlleva cargos adicionales. Para obtener más información, consulte Estimación de costos AWS KMS y Precios de AWS Key Management Service.

Cifrado de sobre

La seguridad de los datos cifrados depende en parte de la protección de la clave de datos que permite descifrarlos. Amazon SQS utiliza la clave de KMS para cifrar la clave de datos y, a continuación, la clave de datos cifrada se almacena con el mensaje cifrado. Esta práctica de utilizar una clave de KMS para cifrar las claves de datos se denomina cifrado de sobre.

Para obtener más información, consulte Cifrado de envoltura en la Guía del desarrollador de AWS Encryption SDK .

Administración de claves de Amazon SQS

Amazon SQS se integra con el AWS Key Management Service (KMS) para administrar las claves de KMS para el cifrado del lado del servidor (SSE). Consulte Cifrado en reposo en Amazon SQS para obtener información sobre SSE y las definiciones de administración de claves. Amazon SQS utiliza las claves de KMS para validar y proteger las claves de datos que cifran y descifran los mensajes. En las siguientes secciones se proporciona información sobre cómo trabajar con las claves de KMS y las claves de datos en el servicio de Amazon SQS.

Configuración de los permisos de AWS KMS

Cada clave de KMS debe tener una política de claves. Tenga en cuenta que no puede modificar la política de claves de una clave de KMS AWS gestionada para Amazon SQS. La política de esta clave de KMS incluye permisos para que todas las entidades principales de la cuenta (que tienen permiso para usar Amazon SQS) utilicen colas cifradas.

Amazon SQS distingue a las personas que llaman en función de sus AWS credenciales, ya sea que utilicen AWS cuentas, usuarios de IAM o funciones de IAM diferentes. Además, el mismo rol de IAM con diferentes políticas de alcance se considerará como solicitantes distintos. Sin embargo, cuando se utilizan sesiones de roles de IAM, `RoleSessionName` si se cambian únicamente las mismas políticas de rol y alcance de IAM, no se crearán solicitantes distintos. Por lo tanto, al especificar los principios de la política AWS KMS clave, evite basarse únicamente en `RoleSessionName` las diferencias, ya que estas sesiones se tratarán como el mismo solicitante.

De forma alternativa, puede especificar los permisos necesarios en una política de IAM asignada a las entidades principales que producen y consumen mensajes cifrados. Para obtener más información, consulte Uso de políticas de IAM con AWS KMS en la Guía para desarrolladores de AWS Key Management Service .

> **Note**
>
> Si bien puede configurar los permisos globales para enviar y recibir desde Amazon SQS, es AWS KMS necesario nombrar explícitamente el ARN completo de las claves de KMS en regiones específicas en la `Resource` sección de una política de IAM.

Configure los permisos de KMS para los servicios AWS

Varios AWS servicios actúan como fuentes de eventos que pueden enviar eventos a las colas de Amazon SQS. Para permitir que estas fuentes de eventos funcionen con colas cifradas, debe crear una clave de KMS administrada por el cliente y añadir permisos en la política de claves para que el servicio utilice los métodos de API necesarios AWS KMS . Realice los siguientes pasos para configurar los permisos.

> **Warning**
>
> Al cambiar la clave de KMS para cifrar los mensajes de Amazon SQS, tenga en cuenta que los mensajes cifrados con la clave de KMS anterior permanecerán cifrados con esa clave. Para descifrar estos mensajes, debe conservar la antigua clave de KMS y asegurarse de que su política de claves concede a Amazon SQS los permisos para `kms:Decrypt` y `kms:GenerateDataKey`. Tras actualizar a una nueva clave de KMS para cifrar los mensajes nuevos, asegúrese de que todos los mensajes cifrados con la antigua clave de KMS se procesen y se eliminen de la cola antes de eliminar o deshabilitar la antigua clave de KMS.

1. Cree una clave de KMS administrada por el cliente. Para obtener más información, consulte Creación de claves en la Guía para desarrolladores de AWS Key Management Service .
2. Para permitir que la fuente de eventos del AWS servicio utilice los métodos `kms:Decrypt` y `kms:GenerateDataKey` API, añada la siguiente declaración a la política de claves de KMS.

JSON

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": "service.amazonaws.com"
         },
         "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
         ],
         "Resource": "*"
       }]
}
```

Reemplace "service" (servicio) en el ejemplo anterior por Service name (Nombre del servicio) del origen del evento. Los orígenes de eventos incluyen los siguientes servicios.

Origen del evento	Nombre del servicio
CloudWatch Eventos de Amazon	`events.amazonaws.com`
Notificaciones de eventos de Amazon S3	`s3.amazonaws.com`
Suscripciones a temas de Amazon SNS	`sns.amazonaws.com`

3. Configure una cola SSE existente mediante el ARN de su clave de KMS.
4. Proporcione el ARN de la cola cifrada al origen de eventos.

Configure AWS KMS los permisos para los productores

Cuando caduca el periodo de reutilización de la clave de datos, la siguiente llamada del productor a `SendMessage` o `SendMessageBatch` también desencadena llamadas a `kms:Decrypt` y `kms:GenerateDataKey`. La llamada a `kms:Decrypt` es para verificar la integridad de la nueva

clave de datos antes de usarla. Por lo tanto, el productor debe tener los permisos `kms:Decrypt` y `kms:GenerateDataKey` para la clave de KMS.

Agregue la siguiente instrucción a la política de IAM del productor. Recuerde utilizar los valores de ARN correctos para el recurso de clave y el recurso de cola.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
    ]
}
```

Configure AWS KMS los permisos para los consumidores

Cuando caduca el periodo de reutilización de la clave de datos, la siguiente llamada del cliente a `ReceiveMessage` también desencadena una llamada a `kms:Decrypt`, para verificar la integridad de la nueva clave de datos antes de usarla. Por lo tanto, el cliente debe tener el permiso `kms:Decrypt` para cualquier clave de KMS que se utilice para cifrar los mensajes en la cola especificada. Si la cola funciona como una cola de mensajes fallidos, el consumidor también debe tener el permiso `kms:Decrypt` para todas las clave de KMS que se utilicen para cifrar los mensajes en la cola de origen. Agregue la siguiente instrucción a la política de IAM del cliente. Recuerde utilizar los valores de ARN correctos para el recurso de clave y el recurso de cola.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
    ]
}
```

Configuración de permisos de AWS KMS con una protección de suplente confuso

Cuando la entidad principal de una instrucción de política de claves es una entidad principal de servicio de AWS, puede utilizar las claves de condición global `aws:SourceArn` o `aws:SourceAccount` para protegerse del escenario del suplente confuso. Para utilizar estas claves de condición, establezca el valor al nombre de recurso de Amazon (ARN) o la cuenta del recurso que se está cifrando. Si no conoce el ARN del recurso, utilice `aws:SourceAccount` en su lugar.

En esta política de claves de KMS, un recurso específico de un servicio que sea propiedad de la cuenta `111122223333` puede llamar a KMS para las acciones `Decrypt` y `GenerateDataKey`, que se producen durante el uso de SSE de Amazon SQS.

JSON

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

```
            "Effect": "Allow",
            "Principal": {
                "Service": "<replaceable>service</replaceable>.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:<replaceable>service<::111122223333:resource"
                    ]
                }
            }
        }
    ]
}
```

Cuando se utilizan colas Amazon SQS habilitadas para SSE, los siguientes servicios admiten `aws:SourceArn`:

- Amazon SNS
- Amazon S3
- CloudWatch Eventos
- AWS Lambda
- CodeBuild
- Perfiles de clientes de Amazon Connect
- AWS Auto Scaling
- Amazon Chime

Descripción del período de reutilización de la clave de datos

El periodo de reutilización de la clave de datos define la duración máxima de Amazon SQS para reutilizar la misma clave de datos. Cuando finaliza el periodo de reutilización de la clave de datos, Amazon SQS genera una nueva clave de datos. Tenga en cuenta las siguientes directrices sobre el periodo de reutilización.

- Un período de reutilización más corto proporciona una mayor seguridad, pero se traduce en más llamadas AWS KMS, lo que puede conllevar gastos superiores al nivel gratuito.
- Aunque la clave de datos se almacena en caché de forma independiente para el cifrado y el descifrado, el periodo de reutilización se aplica a ambas copias de la clave de datos.
- Cuando finaliza el período de reutilización de la clave de datos, se realiza la siguiente llamada al método `SendMessage` o `SendMessageBatch` normalmente se activa una llamada al AWS KMS `GenerateDataKey` método para obtener una nueva clave de datos. Además, las siguientes llamadas a, `SendMessage` y cada una de ellas `ReceiveMessage` activará una llamada AWS KMS `Decrypt` a, para comprobar la integridad de la clave de datos antes de utilizarla.
- Los directores (Cuentas de AWS o los usuarios) no comparten claves de datos (los mensajes enviados por directores únicos siempre reciben claves de datos únicas). Por lo tanto, el volumen de llamadas a AWS KMS es un múltiplo del número de principales únicos que se utilizan durante el período de reutilización de la clave de datos.

Estimación de costos AWS KMS

Para predecir los costes y comprender mejor su AWS factura, quizá le interese saber con qué frecuencia Amazon SQS utiliza su clave de KMS.

> **Note**
>
> Si bien la siguiente fórmula puede brindarle una muy buena idea de los costos esperados, los costos reales podrían ser más elevados debido a la naturaleza distribuida de Amazon SQS.

Para calcular el número de solicitudes de la API (`R`) por cola, utilice la siguiente fórmula:

```
R = (B / D) * (2 * P + C)
```

`B` es el período de facturación (en segundos).

`D` es el período de reutilización de claves de datos (en segundos).

`P` es el número de entidades principales de producción que realizan envíos a la cola de Amazon SQS.

`C` es el número de entidades principales de consumo que reciben información desde la cola de Amazon SQS.

> **Important**
>
> En general, las entidades principales de producción generan el doble del costo que las de consumo. Para obtener más información, consulte Descripción del período de reutilización de la clave de datos.
> Si el productor y el consumidor tienen diferentes usuarios de , el costo aumenta.

A continuación se muestran algunos cálculos de ejemplo. Para obtener información exacta sobre precios, consulte Precios de AWS Key Management Service.

Ejemplo 1: Calcular el número de llamadas a la AWS KMS API para 2 directores y 1 cola

En este ejemplo se presupone lo siguiente:

- El período de facturación va del 1 al 31 de enero (2 678 400 segundos).
- El periodo de reutilización de la clave de datos está establecido en 5 minutos (300 segundos).
- Hay una cola.
- Hay una entidad principal de producción y una entidad principal de consumo.

```
(2,678,400 / 300) * (2 * 1 + 1) = 26,784
```

Ejemplo 2: Calcular el número de llamadas a la AWS KMS API para varios productores y consumidores y 2 colas

En este ejemplo se presupone lo siguiente:

- El período de facturación va del 1 al 28 de febrero (2 419 200 segundos).
- El periodo de reutilización de la clave de datos está establecido en 24 horas (86 400 segundos).
- Hay 2 colas.
- La primera cola tiene 3 entidades principales productoras y una entidad principal consumidora.
- La segunda cola tiene 5 entidades principales productoras y 2 entidades principales consumidoras.

```
(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532
```

AWS KMS errores

Cuando trabaja con Amazon SQS AWS KMS, es posible que se produzcan errores. Las siguientes referencias describen los errores y sus posibles soluciones.

- Errores comunes de AWS KMS
- Errores Decrypt de AWS KMS
- AWS KMS GenerateDataKey errores

Privacidad del tráfico entre redes en Amazon SQS

El punto de conexión de Amazon Virtual Private Cloud (Amazon VPC) para Amazon SQS es una entidad lógica en una VPC que permite la conectividad solo a Amazon SQS. La VPC direcciona las solicitudes a Amazon SQS y vuelve a direccionar las respuestas a la VPC. En las siguientes secciones se proporciona información sobre cómo trabajar con puntos de enlace de la VPC y crear políticas de puntos de enlace de la VPC.

Puntos de conexión de Amazon Virtual Private Cloud para Amazon SQS

Si utiliza Amazon VPC para alojar sus AWS recursos, puede establecer una conexión entre su VPC y Amazon SQS. Puede utilizar esta conexión para enviar mensajes a sus colas de Amazon SQS sin atravesar el Internet público.

Amazon VPC le permite lanzar AWS recursos en una red virtual personalizada. Puede utilizar una VPC para controlar la configuración de red, como el intervalo de direcciones IP, las subredes, las tablas de enrutamiento y las puertas de enlace de red. Para obtener más información VPCs, consulte la Guía del usuario de Amazon VPC.

Para conectar la VPC a Amazon SQS, primero debe definir un punto de conexión de VPC de interfaz, lo que le permitirá conectar la VPC a otros servicios de AWS . Con el punto de conexión, se ofrece conectividad escalable y fiable con Amazon SQS sin necesidad de utilizar una puerta de enlace de Internet, una instancia de traducción de direcciones de red (NAT) o una conexión de VPN. Para obtener más información, consulte Tutorial: Envío de un mensaje a una cola de Amazon SQS desde Amazon Virtual Private Cloud y Ejemplo 5: denegar el acceso si no es desde un punto de enlace de la VPC en esta guía y Puntos de conexión de VPC de interfaz (AWS PrivateLink) en la Guía del usuario de Amazon VPC.

> **Important**
>
> - Puede usar Amazon Virtual Private Cloud solo con puntos de conexión HTTPS de Amazon SQS.
> - Al configurar Amazon SQS para enviar mensajes desde Amazon VPC, debe habilitar el DNS privado y especificar los puntos de enlace en el formato `sqs.us-east-2.amazonaws.com` o `sqs.us-east-2.api.aws` para el punto de enlace de doble pila.
> - Amazon SQS también admite puntos de enlace FIPS mediante el PrivateLink uso del servicio de puntos de enlace. `com.amazonaws.region.sqs-fips` Puede conectarse a los puntos de conexión FIPS en este formato. `sqs-fips.region.amazonaws.com`
> - Al utilizar el punto de conexión de doble pila en Amazon Virtual Private Cloud, las solicitudes se enviarán mediante IPv4 y IPv6.
> - Los DNS privados no admiten los puntos de enlace heredados, como `queue.amazonaws.com` o `us-east-2.queue.amazonaws.com`.

Creación de una política de punto de conexión de VPC para Amazon SQS

Puede crear una política para los puntos de conexión de Amazon VPC correspondiente a Amazon SQS y especificar lo siguiente:

- La entidad principal que puede realizar acciones.
- Las acciones que se pueden realizar.
- Los recursos en los que se pueden llevar a cabo las acciones.

Para obtener más información, consulte Control del acceso a servicios con puntos de conexión de VPC en la Guía del usuario de Amazon VPC

En la política de puntos de conexión de VPC del ejemplo siguiente, se especifica que el usuario `MyUser` puede enviar mensajes a la cola `MyQueue` de Amazon SQS.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
```

```
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

Se deniega lo siguiente:

- Otras acciones de la API de Amazon SQS, como `sqs:CreateQueue` y `sqs:DeleteQueue`.
- Otros usuarios y reglas de que intentan utilizar este punto de enlace de la VPC.
- El envío de mensajes por parte de `MyUser` a otra cola de Amazon SQS.

> **Note**
>
> El usuario puede seguir utilizando otras acciones de la API de Amazon SQS desde fuera de la VPC. Para obtener más información, consulte Ejemplo 5: denegar el acceso si no es desde un punto de enlace de la VPC.

Connect a Amazon SQS mediante puntos de enlace de doble pila (IPv4 y) IPv6

Los puntos de enlace de doble pila admiten ambos tipos de tráfico. IPv4 IPv6 Cuando realizas una solicitud a un punto final de doble pila, la URL del punto final se convierte en una IPv4 o una dirección. IPv6 Para obtener más información sobre los puntos finales FIPS y de doble pila, consulta la guía de referencia del SDK.

Amazon SQS admite puntos de enlace regionales de doble pila, lo que significa que debe especificar la AWS región como parte del nombre del punto de enlace. Los nombres de puntos finales de doble pila utilizan la siguiente convención de nomenclatura:. `sqs.`*`Region`*`.amazonaws.com` Por ejemplo, el nombre del punto de conexión de doble pila para la región `eu-west-1` es `sqs.`*`eu-west-1`*`.amazonaws.com`.

Para ver la lista completa de puntos de conexión de Amazon SQS, consulte la AWS Referencia general.

Identity and Access Management en Amazon SQS

AWS Identity and Access Management (IAM) es un sistema Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a los AWS recursos. Los administradores de IAM controlan quién se puede autenticar (iniciar sesión) y autorizar (tener permisos) para utilizar los recursos de Amazon SQS. La IAM es una Servicio de AWS herramienta que puede utilizar sin coste adicional.

Público

La forma de usar AWS Identity and Access Management (IAM) varía según el trabajo que realice en Amazon SQS.

Usuario de servicio: si utiliza el servicio Amazon SQS para realizar el trabajo, el administrador proporciona las credenciales y los permisos que necesita. A medida que utilice más características de Amazon SQS para realizar el trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarle a solicitar los permisos correctos al administrador. Si no puede acceder a una característica de Amazon SQS, consulte Solución de problemas de identidad y acceso de Amazon Simple Queue Service.

Administrador de servicio: si está a cargo de los recursos de Amazon SQS de la empresa, probablemente tenga acceso completo a Amazon SQS. El trabajo consiste en determinar a qué características y recursos de Amazon SQS deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su gestionador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información acerca de cómo la empresa puede utilizar IAM con Amazon SQS, consulte Cómo funciona Amazon Simple Queue Service con IAM.

Administrador de IAM: si es un administrador de IAM, es posible que desee obtener información sobre cómo escribir políticas para administrar el acceso a Amazon SQS. Para consultar ejemplos de políticas de Amazon SQS basadas en identidades que puede utilizar en IAM, consulte Prácticas recomendadas sobre las políticas.

Autenticación con identidades

La autenticación es la forma de iniciar sesión AWS con sus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como usuario de IAM o asumiendo una función de IAM. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (Centro de identidades de IAM), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su gestionador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accedes AWS mediante la federación, estás asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte Cómo iniciar sesión Cuenta de AWS en su Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener más información sobre la firma de solicitudes, consulte AWS Signature Versión 4 para solicitudes API en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactor en la Guía del usuario de AWS IAM Identity Center y Autenticación multifactor AWS en IAM en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utiliza el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren que inicie sesión como usuario raíz, consulta Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos, incluidos los que requieren acceso de administrador, que utilicen la federación con un proveedor de identidades para acceder Servicios de AWS mediante credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidades web AWS Directory Service, el directorio del Centro de Identidad o cualquier usuario al que acceda Servicios de AWS mediante las credenciales proporcionadas a través de una fuente de identidad. Cuando las identidades federadas acceden Cuentas de AWS, asumen funciones y las funciones proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utiliza AWS IAM Identity Center. Puede crear usuarios y grupos en el Centro de identidades de IAM, o puede conectarse y sincronizarse con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus Cuentas de AWS aplicaciones. Para obtener más información, consulta ¿Qué es el Centro de identidades de IAM? en la Guía del usuario de AWS IAM Identity Center .

Usuarios y grupos de IAM

Un usuario de IAM es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puedes iniciar sesión como grupo. Puedes usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos para grandes conjuntos de usuarios. Por ejemplo, puede asignar un nombre a un grupo IAMAdminsy concederle permisos para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales de larga duración permanentes; no obstante, los roles proporcionan credenciales temporales. Para obtener más información, consulte Casos de uso para usuarios de IAM en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad dentro de usted Cuenta de AWS que tiene permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una persona determinada. Para asumir temporalmente un rol de IAM en el AWS Management Console, puede cambiar de un rol de usuario

a uno de IAM (consola). Puedes asumir un rol llamando a una operación de AWS API AWS CLI o usando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulta Métodos para asumir un rol en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles de federación, consulte Crear un rol para un proveedor de identidad de terceros (federación) en la Guía de usuario de IAM. Si utiliza el IAM Identity Center, debe configurar un conjunto de permisos. IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulta Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center .
- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunas Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulta Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.
- Acceso entre servicios: algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realizas una llamada en un servicio, es habitual que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.
  - Sesiones de acceso directo (FAS): cuando utilizas un usuario o un rol de IAM para realizar acciones en AWS ellas, se te considera principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. El FAS utiliza los permisos del principal que llama Servicio de AWS y los solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para

obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.

- Rol de servicio: un rol de servicio es un rol de IAM que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
- Función vinculada al servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

- Aplicaciones que se ejecutan en Amazon EC2: puedes usar un rol de IAM para administrar las credenciales temporales de las aplicaciones que se ejecutan en una EC2 instancia y realizan AWS CLI solicitudes a la AWS API. Esto es preferible a almacenar las claves de acceso en la EC2 instancia. Para asignar un AWS rol a una EC2 instancia y ponerlo a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite que los programas que se ejecutan en la EC2 instancia obtengan credenciales temporales. Para obtener más información, consulte Usar un rol de IAM para conceder permisos a las aplicaciones que se ejecutan en EC2 instancias de Amazon en la Guía del usuario de IAM.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte Información general de políticas JSON en la Guía del usuario de IAM.

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utiliza para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de la API AWS Management Console AWS CLI, la o la AWS API.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS empresa. Las políticas administradas incluyen políticas AWS administradas y políticas administradas por el cliente. Para obtener más información sobre cómo elegir una política administrada o una política insertada, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Los ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Listas de control de acceso () ACLs

Las listas de control de acceso (ACLs) controlan qué responsables (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLs son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios compatibles. AWS WAF ACLs Para obtener más información ACLs, consulte la descripción general de la lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puedes establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCPs): SCPs son políticas de JSON que especifican los permisos máximos para una organización o unidad organizativa (OU). AWS Organizations AWS Organizations es un servicio para agrupar y administrar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilitas todas las funciones de una organización, puedes aplicar políticas de control de servicios (SCPs) a una o a todas tus cuentas. El SCP limita los permisos de las entidades en las cuentas de los miembros, incluidas las de cada una Usuario raíz de la cuenta de AWS. Para obtener más información sobre Organizations SCPs, consulte las políticas de control de servicios en la Guía del AWS Organizations usuario.
- Políticas de control de recursos (RCPs): RCPs son políticas de JSON que puedes usar para establecer los permisos máximos disponibles para los recursos de tus cuentas sin actualizar las políticas de IAM asociadas a cada recurso que poseas. El RCP limita los permisos de los recursos en las cuentas de los miembros y puede afectar a los permisos efectivos de las identidades, incluidos los permisos Usuario raíz de la cuenta de AWS, independientemente de si pertenecen a su organización. Para obtener más información sobre Organizations e RCPs incluir una lista de Servicios de AWS ese apoyo RCPs, consulte Políticas de control de recursos (RCPs) en la Guía del AWS Organizations usuario.
- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades

del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo se AWS determina si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la lógica de evaluación de políticas en la Guía del usuario de IAM.

Información general sobre la administración del acceso en Amazon SQS

Cada AWS recurso es propiedad de un Cuenta de AWS, y los permisos para crear o acceder a un recurso se rigen por las políticas de permisos. Un administrador de la cuenta puede asociar políticas de permisos a identidades de IAM (usuarios, grupos y roles) y algunos servicios (como Amazon SQS) también permiten asociar políticas de permisos a recursos.

> **Note**
>
> Un administrador de la cuenta (o usuario administrador) es un usuario con privilegios administrativos. Para obtener más información, consulte Prácticas recomendadas de IAM en la Guía del usuario de IAM.

Cuando concede permisos, especifica qué usuarios obtienen los permisos, para qué recurso obtienen los permisos y qué acciones específicas desea permitir en el recurso.

Recursos y operaciones de Amazon Simple Queue Service

En Amazon SQS, el único recurso es la cola. En las políticas se emplean nombres de recurso de Amazon (ARN) para identificar los recursos a los que se aplican las políticas. Los siguientes recursos tiene asociado un ARN único:

Tipo de recurso	Formato de ARN
Cola	`arn:aws:sqs:` *`region`*`:`*`account_id`*`:`*`queue_name`

A continuación, se muestran ejemplos del formato de ARN para las colas:

- Un ARN de una cola denominada `my_queue` en la región EE.UU. Este (Ohio), que pertenece a AWS la cuenta 123456789012:

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- Un ARN para una cola denominada `my_queue` en cada una de las diferentes regiones que Amazon SQS admite:

```
arn:aws:sqs:*:123456789012:my_queue
```

- Un ARN que utiliza `*` o `?` como carácter comodín para el nombre de la cola. En los siguientes ejemplos, el ARN coincide con todas las colas que tienen el prefijo `my_prefix_`:

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

Puede obtener el valor de ARN para una cola existente si llama a la acción `GetQueueAttributes`. El valor del atributo `QueueArn` es el ARN de la cola. Para obtener más información ARNs, consulte IAM en la Guía del usuario de IAM ARNs.

Amazon SQS proporciona un conjunto de acciones que funcionan con el recurso de cola. Para obtener más información, consulte Permisos de la API de Amazon SQS: referencia de acciones y recursos.

Titularidad de los recursos

Cuenta de AWS Es propietario de los recursos que se crean en la cuenta, independientemente de quién los haya creado. En concreto, el propietario de los recursos es la Cuenta de AWS de la entidad principal (es decir, la cuenta raíz, un usuario o un rol de IAM) que autentica la solicitud de creación de recursos. Los siguientes ejemplos ilustran cómo funciona:

- Si utiliza las credenciales de su cuenta raíz Cuenta de AWS para crear una cola de Amazon SQS, será el propietario del recurso (en Amazon SQS, el recurso Cuenta de AWS es la cola de Amazon SQS).
- Si crea un usuario en su cuenta Cuenta de AWS y le concede permisos para crear una cola, el usuario podrá crearla. Sin embargo, la propietaria del recurso de cola será su Cuenta de AWS (a la que pertenece el usuario).

- Si crea un rol de IAM en su cuenta Cuenta de AWS con permisos para crear una cola de Amazon SQS, cualquier persona que pueda asumir el rol podrá crear una cola. Usted Cuenta de AWS (al que pertenece el rol) es propietario del recurso de cola.

Administración del acceso a los recursos

Una política de permisos describe los permisos concedidos a las cuentas. En la siguiente sección, se explican las opciones disponibles para crear políticas de permisos.

> **Note**
>
> En esta sección, se explica el uso de IAM en el contexto de Amazon SQS. No se proporciona información detallada sobre el servicio de IAM. Para ver la documentación completa de IAM, consulte What is IAM? (¿Qué es IAM?) en la Guía del usuario de IAM. Para obtener más información acerca de la sintaxis y las descripciones de las políticas de IAM, consulte Referencia de políticas de IAM de AWS en la Guía del usuario de IAM.

Las políticas asociadas a una identidad de IAM se denominan políticas basadas en identidad (políticas de IAM) y las políticas asociadas a un recurso se denominan políticas basadas en recursos.

Políticas basadas en identidades

Hay dos formas de proporcionar a los usuarios permisos para las colas de Amazon SQS: mediante el sistema de políticas de Amazon SQS y mediante el sistema de políticas de IAM. Puede utilizar cualquiera de estos sistemas, o ambos, para asociar políticas a usuarios o roles. En la mayoría de los casos, puede conseguir el mismo resultado mediante cualquiera de estos dos sistemas. Por ejemplo, puede hacer lo siguiente:

- Adjuntar una política de permisos a un usuario o un grupo de su cuenta: para conceder a un usuario permisos para crear una cola de Amazon SQS, adjunte una política de permisos a un usuario o a un grupo al que este pertenezca.
- Adjuntar una política de permisos a un usuario de otro Cuenta de AWS: puede adjuntar una política de permisos a un usuario de otro Cuenta de AWS para que pueda interactuar con una cola de Amazon SQS. Sin embargo, los permisos entre cuentas no se aplican a las siguientes acciones:

  Los permisos entre cuentas no se aplican a las siguientes acciones:

  - `AddPermission`

- `CancelMessageMoveTask`
- `CreateQueue`
- `DeleteQueue`
- `ListMessageMoveTask`
- `ListQueues`
- `ListQueueTags`
- `RemovePermission`
- `SetQueueAttributes`
- `StartMessageMoveTask`
- `TagQueue`
- `UntagQueue`

Para conceder acceso a estas acciones, el usuario debe pertenecer al mismo propietario de Cuenta de AWS la cola de Amazon SQS.

- Adjuntar una política de permisos a un rol (conceder permisos entre cuentas): para conceder permisos entre cuentas a una cola de SQS, debe combinar las políticas de IAM y las basadas en recursos:

  1. En la cuenta A (propietaria de la cola):
     - Adjunte una política basada en recursos a la cola de SQS. Esta política debe conceder de forma explícita los permisos necesarios (por ejemplo `SendMessage`, `ReceiveMessage`) al director de la cuenta B (como un rol de IAM).
  2. En la cuenta A, cree una función de IAM:
     - Una política de confianza que permita a la cuenta B o AN Servicio de AWS asumir la función.

       > **Note**
       >
       > Si desea que un Servicio de AWS (como Lambda oEventBridge) asuma la función, especifique el principal del servicio (por ejemplo,lambda.amazonaws.com) en la política de confianza.

     - Una política basada en la identidad que concede al rol asumido permisos para interactuar con la cola.
  3. En la cuenta B, concede permiso para asumir el rol en la cuenta A.

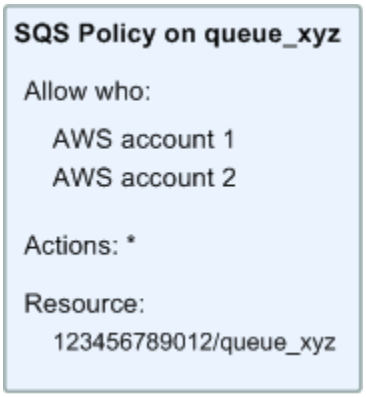
Debe configurar la política de acceso a la cola para permitir el acceso al principal de varias cuentas. Las políticas de IAM basadas en la identidad por sí solas no son suficientes para el acceso entre cuentas a las colas de SQS.

Para obtener más información sobre el uso de IAM para delegar permisos, consulte Administración de accesos en la Guía del usuario de IAM.

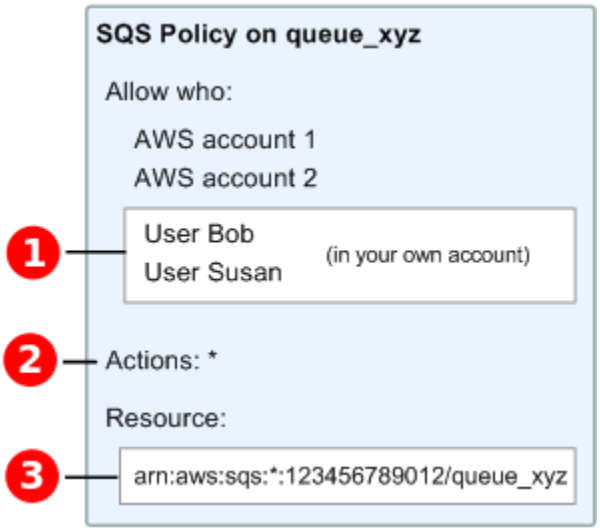
Aunque Amazon SQS funciona con las políticas de IAM tiene su propia infraestructura de políticas. Puede utilizar una política de Amazon SQS con una cola para especificar qué AWS cuentas tienen acceso a la cola. Puede especificar el tipo de acceso y las condiciones (por ejemplo, una condición que concede permisos para utilizar `SendMessage`, `ReceiveMessage` si la solicitud se realiza antes del 31 de diciembre de 2010). Las acciones específicas para las que puede conceder permisos son un subconjunto de la lista general de acciones de Amazon SQS. Cuando se escribe una política de Amazon SQS y se especifica `*` para “permitir todas las acciones de Amazon SQS”, significa que un usuario puede realizar todas las acciones de este subconjunto.

En el siguiente diagrama, se ilustra el concepto de una de estas políticas básicas de Amazon SQS, que abarca el subconjunto de acciones. La política es válida y otorga permisos a la AWS cuenta 1 y a la AWS cuenta 2 para usar cualquiera de las acciones permitidas con la cola especificada. `queue_xyz`

> **Note**
>
> El recurso de la política se especifica como`123456789012/queue_xyz`, donde `123456789012` está el ID de AWS cuenta de la cuenta propietaria de la cola.



Con la introducción de IAM y los conceptos de usuarios y nombres de recursos de Amazon (ARNs), han cambiado algunas cosas en las políticas de SQS. En el diagrama y la tabla siguientes se describen estos cambios.



1 Para obtener información sobre cómo conceder permisos a los usuarios de distintas cuentas, consulte el tutorial: Delegar el acceso entre AWS cuentas mediante funciones de IAM en la Guía del usuario de IAM.

2 El subconjunto de acciones incluidas en `*` se ha ampliado. Para obtener una lista de las acciones permitidas, consulte Permisos de la API de Amazon SQS: referencia de acciones y recursos.

3 Puede especificar el recurso mediante el nombre de recurso de Amazon (ARN), que es la forma estándar de especificar recursos en las políticas de IAM. Para obtener información acerca del formato de ARN para las colas de Amazon SQS, consulte Recursos y operaciones de Amazon Simple Queue Service.

Por ejemplo, de acuerdo con la política de Amazon SQS del diagrama anterior, cualquier persona que posea las credenciales de seguridad de la AWS cuenta 1 o la AWS cuenta 2 puede acceder. `queue_xyz` Además, los usuarios Bob y Susan de su propia cuenta de AWS (con el ID `123456789012`) pueden obtener acceso a la cola.

Antes de la introducción de IAM, Amazon SQS concedía automáticamente al creador de una cola control total sobre esa cola (es decir, acceso a todas las posibles acciones de Amazon SQS en

dicha cola). Esto ha cambiado, a menos que el creador utilice credenciales de seguridad de AWS . Cualquier usuario que tenga permisos para crear una cola debe contar también con permisos para utilizar otras acciones de Amazon SQS para poder realizar alguna acción con las colas creadas.

A continuación, se muestra una política de ejemplo que permite a un usuario utilizar todas las acciones de Amazon SQS, pero solo con las colas cuyos nombres tengan como prefijo la cadena literal `bob_queue_`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
   }]
}
```

Para obtener más información, consulte Uso de políticas con Amazon SQS e Identidades (usuarios, grupos y roles) en la Guía del usuario de IAM.

Especificación de elementos de política: acciones, efectos, recursos y entidades principales

Para cada recurso de Amazon Simple Queue Service, el servicio define un conjunto de acciones. Para conceder permisos a estas acciones, Amazon SQS define un conjunto de acciones que se pueden especificar en una política.

> **Note**
>
> Para realizar una acción de la , pueden ser necesarios permisos para más de una acción. Cuando se conceden permisos para acciones específicas, también debe identificar el recurso para el que las acciones se autorizan o deniegan.

A continuación se indican los elementos más básicos de la política:

- Recurso: en una política, se usa un nombre de recurso de Amazon (ARN) para identificar el recurso al que se aplica la política.
- Acción: utilice palabras de clave de acción para identificar las acciones de recursos que desea permitir o denegar. Por ejemplo, cuando se concede el permiso `sqs:CreateQueue`, el usuario puede realizar la acción `CreateQueue` de Amazon Simple Queue Service.
- Efecto: especifique el efecto que se producirá cuando el usuario solicite la acción específica; puede ser permitir o denegar. Si no concede acceso de forma explícita a un recurso, el acceso se deniega implícitamente. También puede denegar explícitamente el acceso a un recurso para asegurarse de que un usuario no pueda obtener acceso a él, aunque otra política le conceda acceso.
- Entidad principal: en las políticas basadas en identidades (políticas de IAM), el usuario al que se asocia esta política es la entidad principal implícita. Para las políticas basadas en recursos, debe especificar el usuario, la cuenta, el servicio u otra entidad que desee que reciba permisos (se aplica solo a las políticas basadas en recursos).

Para obtener más información sobre la sintaxis y descripciones de las políticas de Amazon SQS, consulte Referencia de la política de AWS IAM en la Guía del usuario de IAM.

Para ver una tabla con todas las acciones de Amazon Simple Queue Service y los recursos a los que se aplican, consulte Permisos de la API de Amazon SQS: referencia de acciones y recursos.

Cómo funciona Amazon Simple Queue Service con IAM

Antes de utilizar IAM para administrar el acceso a Amazon SQS, obtenga información sobre qué características de IAM se encuentran disponibles con Amazon SQS.

Características de IAM que puede utilizar con Amazon Simple Queue Service

Característica de IAM	Compatibilidad con Amazon SQS
Políticas basadas en identidades	Sí
Políticas basadas en recursos	Sí
Acciones de políticas	Sí
Recursos de políticas	Sí

Característica de IAM	Compatibilidad con Amazon SQS
Claves de condición de política (específicas del servicio)	Sí
ACLs	No
ABAC (etiquetas en políticas)	Parcial
Credenciales temporales	Sí
Sesiones de acceso directo (FAS)	Sí
Roles de servicio	Sí
Roles vinculados al servicio	No

Para obtener una visión general de cómo funcionan Amazon SQS y otros AWS servicios con la mayoría de las funciones de IAM, consulte los AWS servicios que funcionan con IAM en la Guía del usuario de IAM.

Control de acceso

Las listas de control de acceso (ACLs) controlan qué directores (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLs son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios compatibles. AWS WAF ACLs Para obtener más información ACLs, consulte la descripción general de la lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

> **Note**
>
> Es importante entender que todos Cuentas de AWS pueden delegar sus permisos a los usuarios de sus cuentas. El acceso multicuenta te permite compartir el acceso a tus AWS recursos sin tener que gestionar usuarios adicionales. Para obtener información sobre el uso del acceso entre cuentas, consulte Habilitación del acceso entre cuentas en la Guía del usuario de IAM.

Consulte Limitaciones de las políticas personalizadas de Amazon SQS para obtener más detalles sobre los permisos de contenido cruzado y las claves de condición en las políticas personalizadas de Amazon SQS.

Políticas basadas en identidad para Amazon SQS

Compatibilidad con las políticas basadas en identidad: sí

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Con las políticas basadas en identidades de IAM, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. No es posible especificar la entidad principal en una política basada en identidad porque se aplica al usuario o rol al que está asociada. Para obtener más información sobre los elementos que puede utilizar en una política de JSON, consulte Referencia de los elementos de las políticas de JSON de IAM en la Guía del usuario de IAM.

Ejemplos de políticas basadas en identidad para Amazon SQS

Para ver ejemplos de políticas basadas en identidad de Amazon SQS, consulte Prácticas recomendadas sobre las políticas.

Políticas basadas en recursos de Amazon SQS

Compatibilidad con las políticas basadas en recursos: sí

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Los ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Para habilitar el acceso entre cuentas, puede especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Añadir a una política en función de recursos una entidad principal entre cuentas es solo una parte del establecimiento de una relación de confianza. Cuando el principal y el recurso son diferentes Cuentas de AWS, el administrador de IAM de la cuenta de confianza también debe conceder a la entidad principal (usuario o rol) permiso para acceder al recurso. Para conceder el permiso, adjunte la entidad a una política basada en identidad. Sin embargo, si la política basada en recursos concede acceso a una entidad principal de la misma cuenta, no es necesaria una política basada en identidad adicional. Para obtener más información, consulte Cross account resource access in IAM en la Guía del usuario de IAM.

Acciones de políticas para Amazon SQS

Compatibilidad con las acciones de políticas: sí

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Las acciones políticas suelen tener el mismo nombre que la operación de AWS API asociada. Hay algunas excepciones, como acciones de solo permiso que no tienen una operación de API coincidente. También hay algunas operaciones que requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de las acciones de Amazon SQS, consulte Recursos definidos por Amazon Simple Queue Service en la Referencia de autorizaciones de servicio.

Las acciones de políticas de Amazon SQS utilizan el siguiente prefijo antes de la acción:

```
sqs
```

Para especificar varias acciones en una única instrucción, sepárelas con comas.

```
"Action": [
      "sqs:action1",
      "sqs:action2"
         ]
```

Para ver ejemplos de políticas basadas en identidad de Amazon SQS, consulte Prácticas recomendadas sobre las políticas.

Recursos de políticas para Amazon SQS

Compatibilidad con los recursos de políticas: sí

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). Puedes hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utiliza un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

```
"Resource": "*"
```

Para ver una lista de los tipos de recursos de Amazon SQS y sus tipos ARNs, consulte Acciones definidas por Amazon Simple Queue Service en la Referencia de autorización de servicios. Para obtener información sobre las acciones con las que puede especificar el ARN de cada recurso, consulte Recursos definidos por Amazon Simple Queue Service.

Para ver ejemplos de políticas basadas en identidad de Amazon SQS, consulte Prácticas recomendadas sobre las políticas.

Claves de condición de políticas para Amazon SQS

Compatibilidad con claves de condición de políticas específicas del servicio: sí

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puedes crear expresiones

condicionales que utilizan operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios elementos de `Condition` en una instrucción o varias claves en un único elemento de `Condition`, AWS las evalúa mediante una operación `AND` lógica. Si especifica varios valores para una única clave de condición, AWS evalúa la condición mediante una `OR` operación lógica. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puedes utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puedes conceder un permiso de usuario de IAM para acceder a un recurso solo si está etiquetado con su nombre de usuario de IAM. Para más información, consulta Elementos de la política de IAM: variables y etiquetas en la Guía del usuario de IAM.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas las claves de condición AWS globales, consulte las claves de contexto de condición AWS globales en la Guía del usuario de IAM.

Para obtener una lista de las claves de condición de Amazon SQS, consulte Claves de condición para Amazon Simple Queue Service en la Referencia de autorizaciones de servicio. Para obtener más información acerca de las acciones y los recursos con los que puede utilizar una clave de condición, consulte Recursos definidos por Amazon Simple Queue Service.

Para ver ejemplos de políticas basadas en identidad de Amazon SQS, consulte Prácticas recomendadas sobre las políticas.

ACLs en Amazon SQS

Soporta ACLs: No

Las listas de control de acceso (ACLs) controlan qué directores (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLs son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

ABAC con Amazon SQS

Compatibilidad con ABAC (etiquetas en las políticas): parcial

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define permisos en función de atributos. En AWS, estos atributos se denominan etiquetas. Puede adjuntar

etiquetas a las entidades de IAM (usuarios o roles) y a muchos AWS recursos. El etiquetado de entidades y recursos es el primer paso de ABAC. A continuación, designa las políticas de ABAC para permitir operaciones cuando la etiqueta de la entidad principal coincida con la etiqueta del recurso al que se intenta acceder.

ABAC es útil en entornos que crecen con rapidez y ayuda en situaciones en las que la administración de las políticas resulta engorrosa.

Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Si un servicio admite las tres claves de condición para cada tipo de recurso, el valor es Sí para el servicio. Si un servicio admite las tres claves de condición solo para algunos tipos de recursos, el valor es Parcial.

Para obtener más información sobre ABAC, consulte Definición de permisos con la autorización de ABAC en la Guía del usuario de IAM. Para ver un tutorial con los pasos para configurar ABAC, consulte Uso del control de acceso basado en atributos (ABAC) en la Guía del usuario de IAM.

Uso de credenciales temporales con Amazon SQS

Compatibilidad con credenciales temporales: sí

Algunas Servicios de AWS no funcionan cuando inicias sesión con credenciales temporales. Para obtener información adicional, incluida la información sobre cuáles Servicios de AWS funcionan con credenciales temporales, consulta Cómo Servicios de AWS funcionan con IAM en la Guía del usuario de IAM.

Utiliza credenciales temporales si inicia sesión en ellas AWS Management Console mediante cualquier método excepto un nombre de usuario y una contraseña. Por ejemplo, cuando accedes AWS mediante el enlace de inicio de sesión único (SSO) de tu empresa, ese proceso crea automáticamente credenciales temporales. También crea credenciales temporales de forma automática cuando inicia sesión en la consola como usuario y luego cambia de rol. Para obtener más información sobre el cambio de roles, consulte Cambio de un usuario a un rol de IAM (consola) en la Guía del usuario de IAM.

Puedes crear credenciales temporales manualmente mediante la AWS CLI API o. AWS A continuación, puede utilizar esas credenciales temporales para acceder AWS. AWS recomienda

generar credenciales temporales de forma dinámica en lugar de utilizar claves de acceso a largo plazo. Para obtener más información, consulte Credenciales de seguridad temporales en IAM.

Sesiones de acceso directo para Amazon SQS

Admite sesiones de acceso directo (FAS): sí

Cuando utilizas un usuario o un rol de IAM para realizar acciones en AWSél, se te considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos del principal que llama y los que solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Servicio de AWS Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.

Roles de servicio para Amazon SQS

Compatibilidad con roles de servicio: sí

Un rol de servicio es un rol de IAM que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.

> **Warning**
>
> Cambiar los permisos para un rol de servicio podría interrumpir la funcionalidad de Amazon SQS. Edite los roles de servicio solo cuando Amazon SQS proporcione orientación para hacerlo.

Roles vinculados a servicios para Amazon SQS

Compatibilidad con roles vinculados al servicio: no

Un rol vinculado a un servicio es un tipo de rol de servicio que está vinculado a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puedes ver, pero no editar, los permisos de los roles vinculados a servicios.

Para más información sobre cómo crear o administrar roles vinculados a servicios, consulta Servicios de AWS que funcionan con IAM. Busque un servicio en la tabla que incluya `Yes` en la columna Rol vinculado a un servicio. Seleccione el vínculo Sí para ver la documentación acerca del rol vinculado a servicios para ese servicio.

Amazon SQS actualiza las políticas gestionadas AWS

Para agregar permisos a usuarios, grupos y roles, es más fácil utilizar las políticas administradas de AWS que escribirlas uno mismo. Se necesita tiempo y experiencia para crear políticas administradas por el cliente de IAM que le brinden a su equipo solo los permisos necesarios. Para comenzar a hacerlo con rapidez, puede utilizar nuestras políticas administradas de AWS . Estas políticas cubren casos de uso comunes y están disponibles en su cuenta de AWS . Para obtener más información sobre las políticas AWS gestionadas, consulte las políticas AWS gestionadas en la Guía del usuario de IAM.

AWS los servicios mantienen y AWS actualizan las políticas gestionadas. No puede cambiar los permisos en las políticas AWS gestionadas. En ocasiones, los servicios añaden permisos adicionales a una política AWS gestionada para admitir nuevas funciones. Este tipo de actualización afecta a todas las identidades (usuarios, grupos y roles) donde se asocia la política. Lo más probable es que los servicios actualicen una política AWS administrada cuando se lanza una nueva función o cuando hay nuevas operaciones disponibles. Los servicios no eliminan los permisos de una política AWS administrada, por lo que las actualizaciones de la política no afectarán a los permisos existentes.

Además, AWS admite políticas administradas para funciones laborales que abarcan varios servicios. Por ejemplo, la política ReadOnlyAccess AWS gestionada proporciona acceso de solo lectura a todos los AWS servicios y recursos. Cuando un servicio lanza una nueva función, AWS agrega permisos de solo lectura para nuevas operaciones y recursos. Para obtener una lista y descripciones de las políticas de funciones de trabajo, consulte Políticas administradas de AWS para funciones de trabajo en la Guía del usuario de IAM.

AWS política gestionada: Amazon SQSFull Access

Puede adjuntar la política `AmazonSQSFullAccess` a las identidades de Amazon SQS. Esta política concede permisos que brindan acceso completo a Amazon SQS.

Para ver los permisos de esta política, consulta Amazon SQSFull Access en la Referencia de políticas AWS gestionadas.

AWS política gestionada: Amazon SQSRead OnlyAccess

Puede adjuntar la política `AmazonSQSReadOnlyAccess` a las identidades de Amazon SQS. Esta política concede permisos que brindan acceso de solo lectura a Amazon SQS.

Para ver los permisos de esta política, consulta Amazon SQSRead OnlyAccess en la Referencia de políticas AWS gestionadas.

AWS política gestionada: SQSUnlock QueuePolicy

Si configuró incorrectamente la política de colas de una cuenta de miembro para denegar a todos los usuarios el acceso a su cola de Amazon SQS, puede utilizar `SQSUnlockQueuePolicy` AWS la política gestionada para desbloquear la cola.

Para obtener más información sobre cómo eliminar una política de colas mal configurada que impide a todos los principales acceder a una cola de Amazon SQS, consulte Realizar una tarea privilegiada en una cuenta de AWS Organizations miembro en la Guía del usuario de IAM.

Amazon SQS actualiza las políticas gestionadas AWS

Consulte los detalles sobre las actualizaciones de las políticas AWS gestionadas de Amazon SQS desde que este servicio comenzó a realizar el seguimiento de estos cambios. Para obtener alertas automáticas sobre cambios en esta página, suscríbase a la fuente RSS en la página de historial de documentos de la API de Amazon SQS.

Cambio	Descripción	Fecha
SQSUnlockQueuePolicy	Amazon SQS ha añadido una nueva política AWS gestionad a destinada `SQSUnlock QueuePolicy` a desbloque ar una cola y eliminar una política de colas mal configura da que impide a todos los directores acceder a una cola de Amazon SQS.	15 de noviembre de 2024
		20 de junio de 2024

Cambio	Descripción	Fecha
AmazonSQSReadOnlyAccess	Amazon SQS ha añadido la acción `ListQueueTags`, que recupera todas las etiquetas asociadas a una cola de Amazon SQS específica. Le permite ver los pares clave-valor que se han asignado a la cola con fines organizativos o de metadatos. Esta acción está asociada a la operación de la API `ListQueueTags`.	
AmazonSQSReadOnlyAccess	Amazon SQS ha agregado una nueva acción que permite enumerar las tareas de movimiento de mensajes más recientes (hasta diez) en una cola de origen específica. Esta acción está asociada a la operación de la API `ListMessageMoveTasks`.	9 de junio de 2023

Solución de problemas de identidad y acceso de Amazon Simple Queue Service

Utilice la siguiente información para diagnosticar y solucionar los problemas habituales que pueden surgir cuando se trabaja con Amazon SQS e IAM.

No tengo autorización para realizar una acción en Amazon SQS

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando el usuario `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *`my-example-widget`*, pero no tiene los permisos ficticios `sqs:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 sqs:GetWidget on resource: my-example-widget
```

En este caso, la política de Mateo se debe actualizar para permitirle acceder al recurso *`my-example-widget`* mediante la acción `sqs:`*`GetWidget`*.

Si necesita ayuda, póngase en contacto con su AWS administrador. El gestionador es la persona que le proporcionó las credenciales de inicio de sesión.

No estoy autorizado a realizar tareas como: PassRole

Si recibe un error que indica que no tiene autorización para llevar a cabo la acción `iam:PassRole`, las políticas se deben actualizar para permitirle pasar un rol a Amazon SQS.

Algunas Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en Amazon SQS. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador. AWS El gestionador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis recursos de Amazon SQS

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para

que asuma el rol. En el caso de los servicios que admiten políticas basadas en recursos o listas de control de acceso (ACLs), puede usar esas políticas para permitir que las personas accedan a sus recursos.

Para obtener más información, consulte lo siguiente:

- Para saber si Amazon SQS admite estas características, consulte Cómo funciona Amazon Simple Queue Service con IAM.
- Para obtener información sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte Proporcionar acceso a un usuario de IAM en otro de su propiedad en la Cuenta de AWS Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta Cómo proporcionar acceso a recursos que Cuentas de AWS son propiedad de terceros en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulta Proporcionar acceso a usuarios autenticados externamente (identidad federada) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre las políticas basadas en roles y en recursos para el acceso entre cuentas, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Quiero desbloquear mi cola

Si Cuenta de AWS pertenece a una organización, AWS Organizations las políticas pueden impedirle acceder a los recursos de Amazon SQS. De forma predeterminada, AWS Organizations las políticas no bloquean ninguna solicitud a Amazon SQS. Sin embargo, asegúrese de que sus AWS Organizations políticas no se hayan configurado para bloquear el acceso a las colas de Amazon SQS. Para obtener instrucciones sobre cómo comprobar sus AWS Organizations políticas, consulte Enumerar todas las políticas en la Guía del AWS Organizations usuario.

Además, si configuró incorrectamente su política de colas para una cuenta de miembro para denegar a todos los usuarios el acceso a su cola de Amazon SQS, puede desbloquear la cola iniciando una sesión privilegiada para la cuenta de miembro en IAM. Una vez que inicie una sesión privilegiada, podrá eliminar la política de colas mal configurada para recuperar el acceso a la cola. Para obtener más información, consulte Realizar una tarea privilegiada en una cuenta de AWS Organizations miembro en la Guía del usuario de IAM.

Uso de políticas con Amazon SQS

Este tema ofrece ejemplos de políticas basadas en identidad en las que un administrador de la cuenta puede adjuntar políticas de permisos a identidades de IAM (usuarios, grupos y roles).

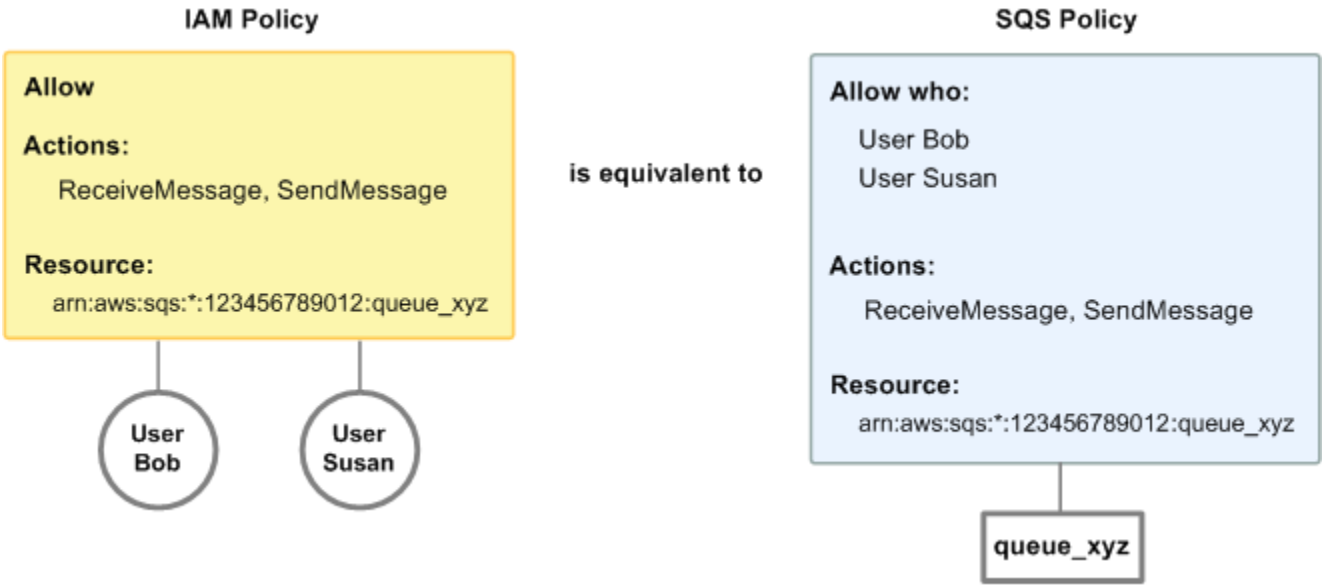
> **Important**
>
> Le recomendamos que consulte primero los temas de introducción en los que se explican los conceptos básicos y las opciones disponibles para administrar el acceso a sus recursos de Amazon Simple Queue Service. Para obtener más información, consulte Información general sobre la administración del acceso en Amazon SQS.
> Con la excepción de `ListQueues`, todas las acciones de Amazon SQS admiten permisos de nivel de recurso. Para obtener más información, consulte Permisos de la API de Amazon SQS: referencia de acciones y recursos.

Uso de políticas de Amazon SQS e IAM

Hay dos formas de proporcionar a los usuarios permisos para los recursos de Amazon SQS: mediante el sistema de políticas de Amazon SQS (políticas basadas en recursos) y mediante el sistema de políticas de IAM (políticas basadas en identidades). Puede utilizar uno o ambos métodos, con la excepción de la acción `ListQueues`, que es un permiso regional que solo se puede configurar en una política de IAM.

Por ejemplo, en el siguiente diagrama se muestra una política de IAM y una política de Amazon SQS equivalente. La política de IAM otorga los derechos a Amazon `ReceiveMessage` SQS `SendMessage` y las acciones para la cola `queue_xyz` llamada en AWS su cuenta, y la política se adjunta a los usuarios llamados Bob y Susan (Bob y Susan tienen los permisos establecidos en la política). Esta política de Amazon SQS también ofrece a Bob y Susan derechos para las acciones `ReceiveMessage` y `SendMessage` de la misma cola.

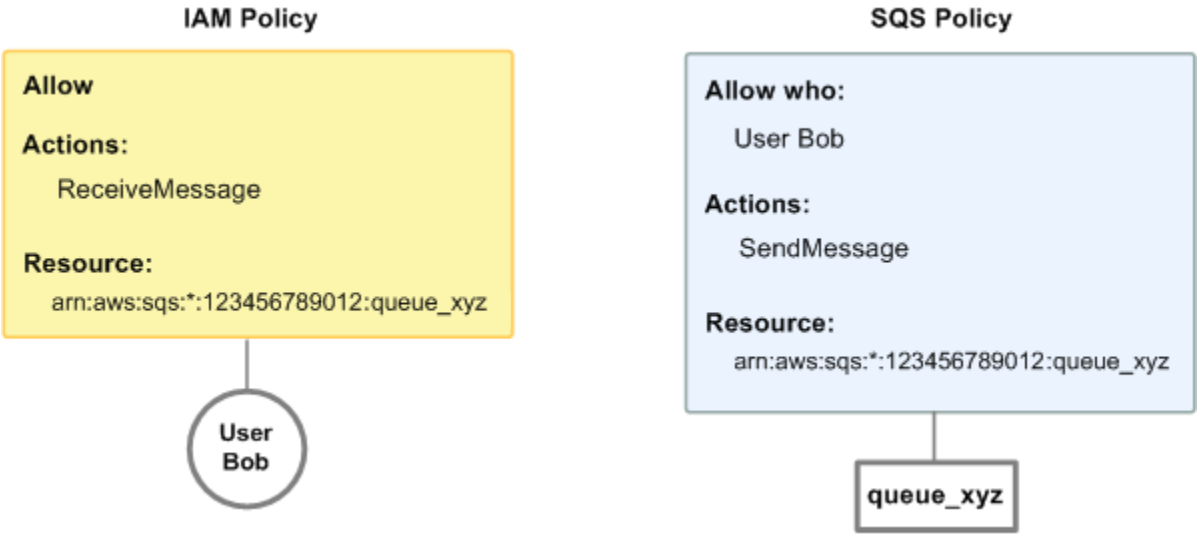
> **Note**
>
> El siguiente ejemplo muestra políticas sencillas sin condiciones. Puede especificar una condición determinada en cualquiera de las dos políticas y obtendrá el mismo resultado.



Hay una diferencia importante entre las políticas de IAM y Amazon SQS: el sistema de políticas de Amazon SQS te permite conceder permisos a AWS otras cuentas, mientras que IAM no.

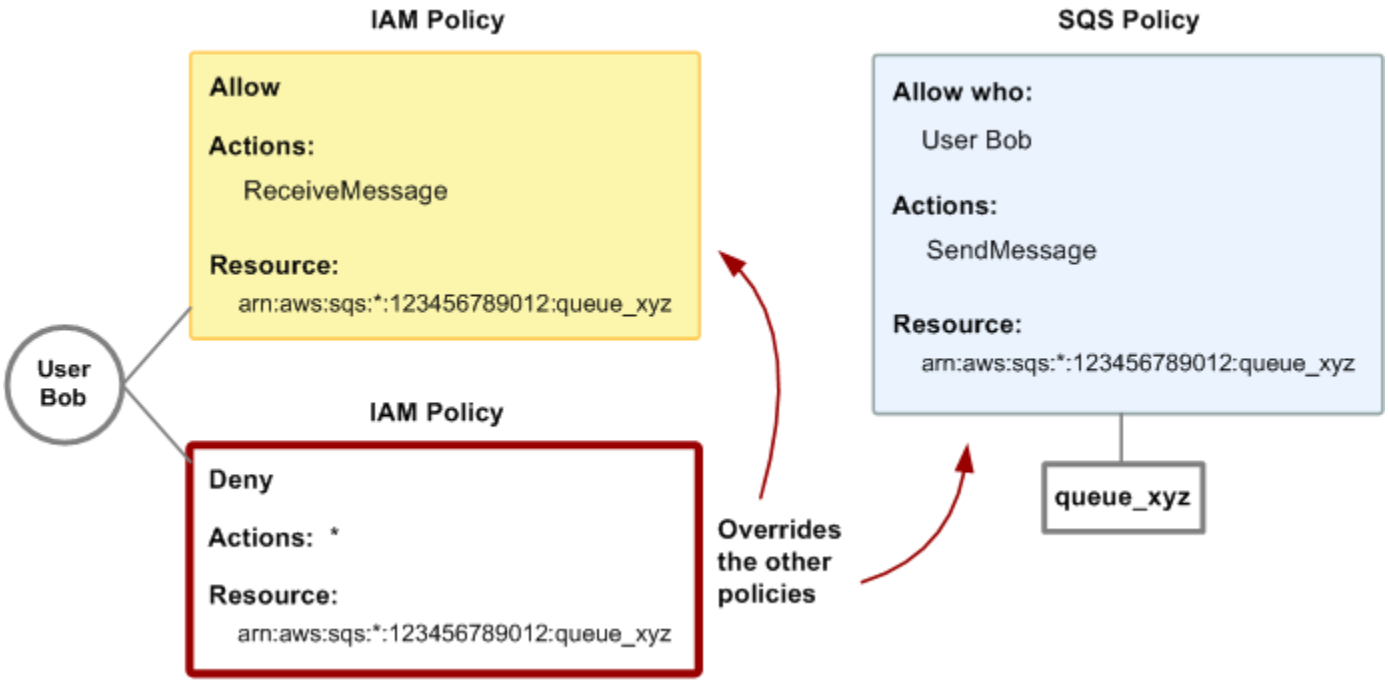
Usted decide el uso que quiere hacer de ambos sistemas para administrar los permisos. Los siguientes ejemplos muestran cómo funcionan conjuntamente los dos sistemas de política.

- En el primer ejemplo, Bob tiene una política de IAM y una política de Amazon SQS que se aplican a su cuenta. La política de IAM concede a su cuenta permiso para realizar la acción `ReceiveMessage` en `queue_xyz`, mientras que la política de Amazon SQS concede a su cuenta permiso para realizar la acción `SendMessage` en la misma cola. El siguiente diagrama ilustra este concepto.



Si Bob envía una solicitud `ReceiveMessage` a `queue_xyz`, la política de IAM permite la acción. Si Bob envía una solicitud `SendMessage` a `queue_xyz`, la política de Amazon SQS permite la acción.

- En el segundo ejemplo, Bob abusa de su acceso a `queue_xyz`, por lo que es necesario quitar todo su acceso a la cola. Para ello, lo más fácil es añadir una política que le deniegue acceso a todas las acciones de la cola. Esta política anula las otras dos porque un permiso `deny` explícito siempre anula un permiso `allow`. Para obtener más información acerca de la lógica de evaluación de las políticas, consulte Uso de políticas personalizadas con el lenguaje de la política de acceso de Amazon SQS. El siguiente diagrama ilustra este concepto.



También puede agregar a la política de Amazon SQS una instrucción adicional que deniegue a Bob cualquier tipo de acceso a la cola. Tiene el mismo efecto que agregar una política de IAM que deniegue a Bob el acceso a la cola. Para ver ejemplos de políticas que abarcan acciones y recursos de Amazon SQS, consulte Ejemplos básicos de políticas de Amazon SQS. Para obtener más información sobre la escritura de políticas de Amazon SQS, consulte Uso de políticas personalizadas con el lenguaje de la política de acceso de Amazon SQS.

Permisos necesarios para usar la consola de Amazon SQS

Un usuario que quiera trabajar con la consola de Amazon SQS debe tener el conjunto mínimo de permisos que le permita trabajar con las colas de Amazon SQS en la Cuenta de AWS del usuario.

Por ejemplo, el usuario debe tener el permiso para llamar a la acción `ListQueues` para poder enumerar colas, o el permiso para llamar a la acción `CreateQueue` para poder crear colas. Además de los permisos de Amazon SQS, para suscribir una cola de Amazon SQS a un tema de Amazon SNS, la consola también requiere permisos para las acciones de Amazon SNS.

Si crea una política de IAM que sea más restrictiva que los permisos mínimos necesarios, es posible que la consola no funcione del modo esperado para los usuarios que tienen esa política de IAM.

No es necesario permitir permisos mínimos de consola a los usuarios que solo realicen llamadas a las acciones AWS CLI o a las de Amazon SQS.

Ejemplos de políticas basadas en identidad para Amazon SQS

De forma predeterminada, los usuarios y roles no tienen permiso para crear ni modificar los recursos de Amazon SQS. Tampoco pueden realizar tareas mediante la AWS Management Console, AWS Command Line Interface (AWS CLI) o la AWS API. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Para obtener información acerca de cómo crear una política basada en identidades de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte Creación de políticas de IAM (consola) en la Guía del usuario de IAM.

Para obtener más información sobre las acciones y los tipos de recursos definidos por Amazon SQS, incluido el formato de cada uno de los tipos de recursos, consulte Acciones, recursos y claves de condición de Amazon Simple Queue Service en la Referencia de autorización de servicios. ARNs

> **Note**
>
> Al configurar los enlaces del ciclo de vida para Amazon EC2 Auto Scaling, no necesita escribir una política para enviar mensajes a una cola de Amazon SQS. Para obtener más información, consulte Amazon EC2 Auto Scaling Lifecycle Hooks en la Guía del EC2 usuario de Amazon.

Prácticas recomendadas sobre las políticas

Las políticas basadas en identidad determinan si alguien puede crear, eliminar o acceder a los recursos de Amazon SQS de la cuenta. Estas acciones pueden generar costos adicionales para su

Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas AWS administradas y avance hacia los permisos con privilegios mínimos: para empezar a conceder permisos a sus usuarios y cargas de trabajo, utilice las políticas AWS administradas que otorgan permisos para muchos casos de uso comunes. Están disponibles en su. Cuenta de AWS Le recomendamos que reduzca aún más los permisos definiendo políticas administradas por el AWS cliente que sean específicas para sus casos de uso. Con el fin de obtener más información, consulta las políticas administradas por AWS o las políticas administradas por AWS para funciones de tarea en la Guía de usuario de IAM.
- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulta Políticas y permisos en IAM en la Guía del usuario de IAM.
- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puedes usar condiciones para conceder el acceso a las acciones del servicio si se utilizan a través de una acción específica Servicio de AWS, por ejemplo AWS CloudFormation. Para obtener más información, consulta Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.
- Utiliza el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte Validación de políticas con el Analizador de acceso de IAM en la Guía del usuario de IAM.
- Requerir autenticación multifactor (MFA): si tiene un escenario que requiere usuarios de IAM o un usuario raíz en Cuenta de AWS su cuenta, active la MFA para mayor seguridad. Para exigir la MFA cuando se invoquen las operaciones de la API, añada condiciones de MFA a sus políticas. Para más información, consulte Acceso seguro a la API con MFA en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.

Uso de la consola de Amazon SQS

Para acceder a la consola de Amazon Simple Queue Service, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle enumerar y ver detalles sobre los recursos de Amazon SQS que tiene en su. Cuenta de AWS Si crea una política basada en identidades que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

No necesita conceder permisos mínimos de consola a los usuarios que solo realizan llamadas a la API AWS CLI o a la AWS API. En su lugar, permite el acceso únicamente a las acciones que coincidan con la operación de API que intentan realizar.

Para garantizar que los usuarios y los roles puedan seguir utilizando la consola de Amazon SQS, adjunte también la política gestionada de Amazon `AmazonSQSReadOnlyAccess` AWS SQS a las entidades. Para obtener más información, consulte Adición de permisos a un usuario en la Guía del usuario de IAM:

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas gestionadas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para completar esta acción en la consola o mediante programación mediante la API o. AWS CLI 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": "*"
        }
    ]
}
```

Permitir a un usuario crear colas

En el siguiente ejemplo, creamos una política para Bob que le permite obtener acceso a todas las acciones de Amazon SQS, pero solo con las colas cuyos nombres tengan como prefijo la cadena literal `alice_queue_`.

Amazon SQS no concede automáticamente al creador de una cola permisos para utilizar dicha cola. Por tanto, en la política de IAM debemos conceder de forma explícita a Bob los permisos para utilizar todas las acciones de Amazon SQS, además de para la acción `CreateQueue`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
   }]
}
```

Permitir que los desarrolladores escriban mensajes en una cola compartida

En el siguiente ejemplo, creamos un grupo para desarrolladores y adjuntamos una política que permite al grupo usar la `SendMessage` acción Amazon SQS, pero solo con la cola que pertenece a la especificada Cuenta de AWS y que lleva el nombre. `MyCompanyQueue`

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
   }]
}
```

Puede utilizar `*` en lugar de `SendMessage` para conceder las acciones siguientes de una cola compartida a una entidad principal: `ChangeMessageVisibility`, `DeleteMessage`, `GetQueueAttributes`, `GetQueueUrl`, `ReceiveMessage` y `SendMessage`.

> **Note**
>
> Aunque `*` incluye el acceso proporcionado por otros tipos de permisos, Amazon SQS considera los permisos por separado. Por ejemplo, es posible conceder los permisos `*` y `SendMessage` a un usuario, aunque `*` incluye el acceso que ofrece `SendMessage`. Este concepto también se aplica al quitar un permiso. Si una entidad principal solo tiene el permiso `*` y se solicita la eliminación del permiso `SendMessage`, la entidad principal no se queda con los permisos restantes, sino que la solicitud no tiene ningún efecto, porque la entidad principal no tenía explícitamente el permiso `SendMessage`. Si desea dejar a la entidad principal únicamente con el permiso `ReceiveMessage`, añada primero el permiso `ReceiveMessage` y, a continuación, elimine el permiso `*`.

Permitir que los administradores obtengan el tamaño general de las colas

En el siguiente ejemplo, creamos un grupo para administradores y adjuntamos una política que permite al grupo usar la `GetQueueAttributes` acción Amazon SQS con todas las colas que pertenecen a la cuenta especificada. AWS

JSON

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

Permitir que un socio envíe mensajes a una cola específica

Puede realizar esta tarea mediante una política de Amazon SQS o una política de IAM. Si su socio tiene una Cuenta de AWS, podría ser más fácil usar una política de Amazon SQS. Sin embargo, cualquier usuario de la empresa del socio que posea las credenciales AWS de seguridad puede enviar mensajes a la cola. Si desea limitar el acceso a un determinado usuario o aplicación, debe tratar al socio como a un usuario de su propia empresa y utilizar una política de IAM en lugar de una política de Amazon SQS.

En este ejemplo se realizan las siguientes acciones:

1. Cree un grupo llamado WidgetCo para representar a la empresa asociada.
2. Crear un usuario para la aplicación o el usuario específico de la compañía del socio que necesita acceso.
3. Agregue el usuario al grupo .
4. Adjuntar una política que conceda al grupo acceso únicamente a la acción `SendMessage` solo para la cola denominada `WidgetPartnerQueue`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "sqs:SendMessage",
         "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
   }]
}
```

Ejemplos básicos de políticas de Amazon SQS

En esta sección se muestran políticas de ejemplo para casos de uso Amazon SQS comunes.

Puede utilizar la consola para comprobar los efectos de cada política a medida que asocia la política al usuario. En un primer momento, como el usuario no tiene permisos, no podrá hacer nada más en la consola. Al asignar políticas al usuario, podrá verificar que este pueda realizar diversas acciones en la consola.

> **Note**
>
> Le recomendamos que utilice dos ventanas del navegador: una para conceder los permisos y otra para iniciar sesión AWS Management Console con las credenciales del usuario a fin de verificar los permisos a medida que se los concede al usuario.

Ejemplo 1: conceder un permiso a otro Cuenta de AWS

El siguiente ejemplo de política concede a Cuenta de AWS number `111122223333` el `SendMessage` permiso para la cola nombrada `444455556666/queue1` en la región EE.UU. Este (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"
   }]
}
```

Ejemplo 2: conceder dos permisos a uno Cuenta de AWS

El siguiente ejemplo de política concede `111122223333` al Cuenta de AWS número `SendMessage` y el `ReceiveMessage` permiso para la cola denominada`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"
   }]
}
```

Ejemplo 3: conceder todos los permisos a dos Cuentas de AWS

El siguiente ejemplo de política concede dos Cuentas de AWS números diferentes (`111122223333y444455556666`) permiso para utilizar todas las acciones a las que Amazon SQS permite el acceso compartido para la cola nombrada `123456789012/queue1` en la región EE.UU. Este (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"
   }]
}
```

Ejemplo 4: conceder permisos entre cuentas a un rol y un nombre de usuario

El siguiente ejemplo de política concede `role1` un permiso `111122223333` multicuenta `username1` bajo Cuenta de AWS número para utilizar todas las acciones a las que Amazon SQS permite el acceso compartido para la cola `123456789012/queue1` nombrada en la región EE.UU. Este (Ohio).

Los permisos entre cuentas no se aplican a las siguientes acciones:

- `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"
   }]
}
```

Ejemplo 5: conceder un permiso a todos los usuarios

La siguiente política de ejemplo concede a todos los usuarios (usuarios anónimos) el permiso `ReceiveMessage` para la cola denominada `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"
   }]
}
```

Ejemplo 6: conceder un permiso con restricción temporal a todos los usuarios

La siguiente política de ejemplo concede a todos los usuarios (usuarios anónimos) el permiso `ReceiveMessage` para la cola denominada `111122223333/queue1`, pero solo desde las 12:00 h (mediodía) hasta las 15:00 h el 31 de enero de 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"
         }
      }
   }]
}
```

Ejemplo 7: conceder todos los permisos a todos los usuarios de un rango de CIDR

La siguiente política de ejemplo concede a todos los usuarios (usuarios anónimos) permiso para utilizar todas las acciones posibles de Amazon SQS que se pueden compartir para la cola denominada `111122223333/queue1`, pero solo si la solicitud procede del intervalo de CIDR `192.0.2.0/24`.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_AllActions_AllowlistIP",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.0.2.0/24"
         }
      }
   }]
}
```

Ejemplo 8: permisos Allowlist y blocklist para los usuarios de diferentes rangos de CIDR

La siguiente política de ejemplo contiene dos instrucciones:

- La primera instrucción concede a todos los usuarios (usuarios anónimos) del rango de CIDR `192.0.2.0/24` (excepto `192.0.2.188`) permiso para utilizar la acción `SendMessage` para la cola denominada 111122223333/queue1.
- La segunda instrucción bloquea a todos los usuarios (usuarios anónimos) del rango de CIDR `12.148.72.0/23` y les impide utilizar la cola.

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"
         }
      }
   }]
}
```

Uso de políticas personalizadas con el lenguaje de la política de acceso de Amazon SQS

Para conceder permisos básicos (por ejemplo, `SendMessage`o `ReceiveMessage`) basándose únicamente en un Cuenta de AWS identificador, no es necesario redactar una política personalizada. En su lugar, utilice la acción Amazon SQS. `AddPermission`

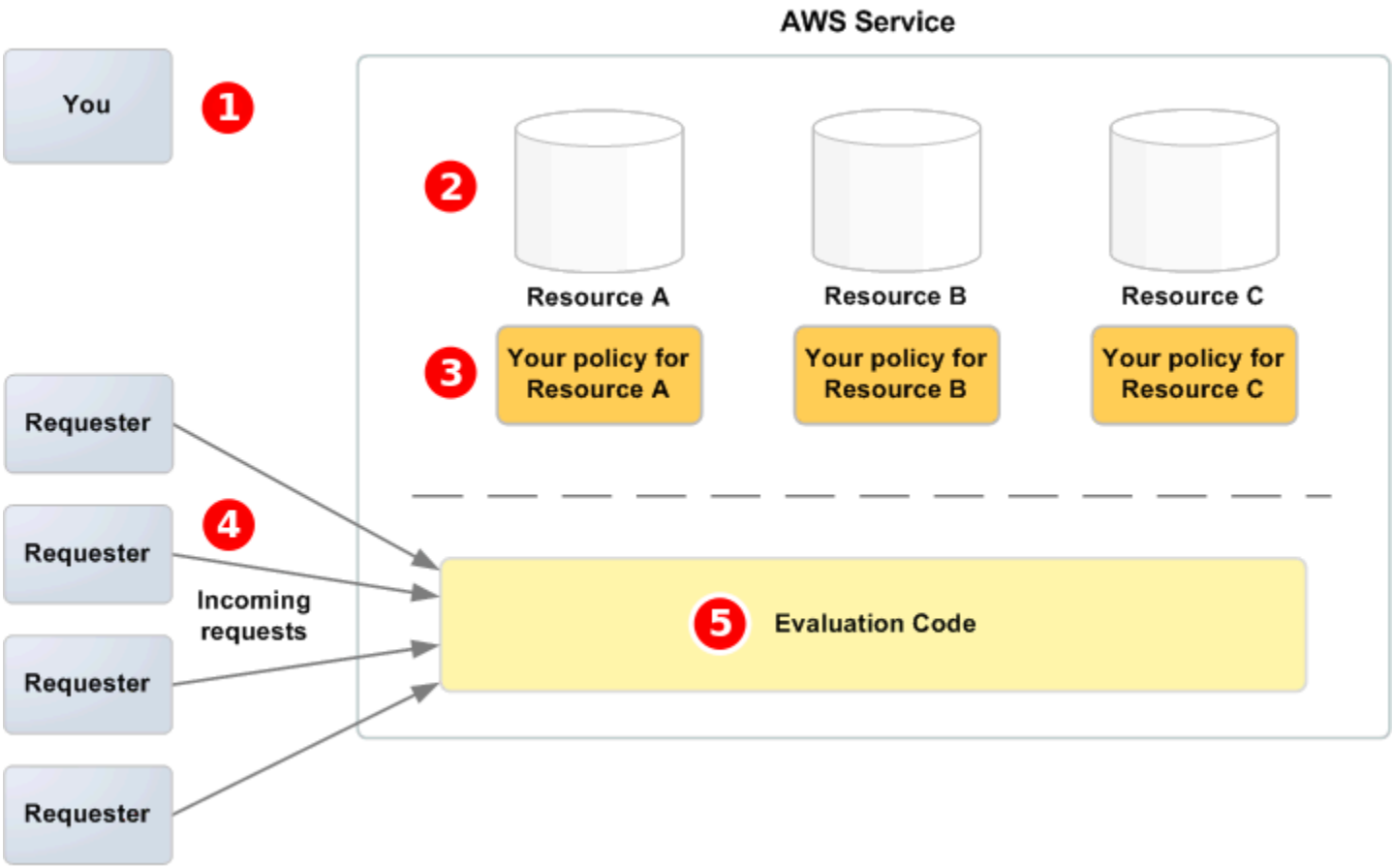
Para permitir o denegar el acceso en función de condiciones específicas, como la hora de la solicitud o la dirección IP del solicitante, debe crear una política de Amazon SQS personalizada y cargarla mediante SetQueueAttributesla acción.

Temas

- Arquitectura de control de acceso de Amazon SQS
- Flujo de trabajo del proceso de control de acceso de Amazon SQS
- Conceptos clave del lenguaje de la política de acceso de Amazon SQS
- Lógica de evaluación del lenguaje de la política de acceso de Amazon SQS
- Relaciones entre denegaciones explícitas y predeterminadas en el lenguaje de la política de acceso de Amazon SQS
- Limitaciones de las políticas personalizadas de Amazon SQS
- Ejemplos de lenguaje de la política de acceso de Amazon SQS personalizada

Arquitectura de control de acceso de Amazon SQS

En el diagrama siguiente, se describe el control de acceso para los recursos de Amazon SQS.



1

Usted, el propietario del recurso.

2

Sus recursos contenidos en el AWS servicio (por ejemplo, colas de Amazon SQS).

3

Sus políticas. Es recomendable tener una política por recurso. El AWS servicio proporciona una API que puede utilizar para cargar y gestionar sus políticas.

4

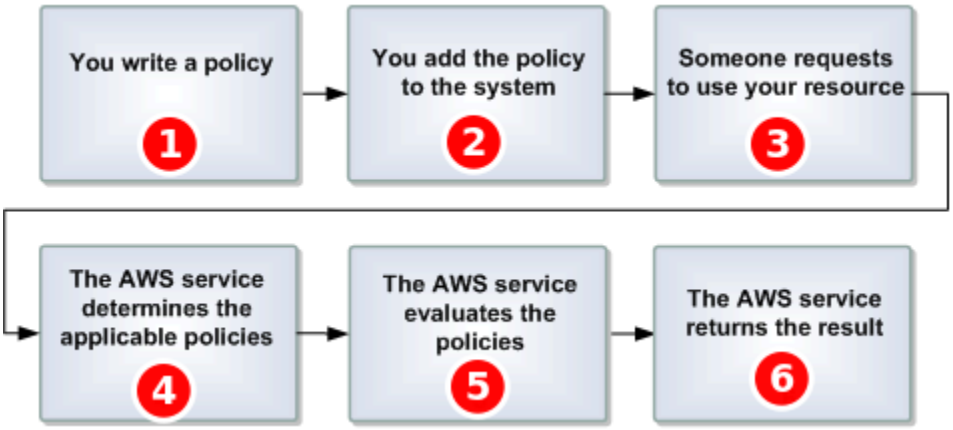
Los solicitantes y las solicitudes de entrada dirigidas al servicio de AWS .

5

El código de evaluación del lenguaje de la política de acceso. Se trata del conjunto de códigos del AWS servicio que evalúa las solicitudes entrantes comparándolas con las políticas aplicables y determina si el solicitante puede acceder al recurso.

Flujo de trabajo del proceso de control de acceso de Amazon SQS

En el diagrama siguiente, se describe el flujo de trabajo general del control de acceso con el lenguaje de la política de acceso de Amazon SQS.



1

Escribe una política de Amazon SQS para la cola.

2

Debe cargar su política en. AWS El AWS servicio proporciona una API que puedes usar para cargar tus políticas. Por ejemplo, utilice la acción `SetQueueAttributes` de Amazon SQS a fin de cargar una política para una cola de Amazon SQS determinada.

3

Alguien envía una solicitud para utilizar su cola de Amazon SQS.

4

Amazon SQS examina todas las políticas de Amazon SQS disponibles y determina cuáles son aplicables.

5

Amazon SQS evalúa las políticas y determina si el solicitante tiene permiso para utilizar su cola.

6

En función del resultado de evaluar las políticas, Amazon SQS devuelve un error `Access denied` al solicitante o continúa procesando la solicitud.

Conceptos clave del lenguaje de la política de acceso de Amazon SQS

Para escribir sus propias políticas, debe estar familiarizado con JSON y varios conceptos clave.

Permitir

El resultado de una Instrucción cuyo Effect se ha establecido en `allow`.

Action

La actividad que la Entidad principal tiene permiso para realizar, por lo general una solicitud para AWS.

Default-deny

Resultado de una Instrucción que no tiene la opción Permitir o Explicit-deny.

Condición

Cualquier restricción o detalle sobre un Permiso. Las condiciones típicas están relacionados con la fecha y la hora y con las direcciones IP.

Effect

El resultado que desea que la Instrucción de una Política devuelva cuando se evalúe. Al escribir la instrucción de la política, debe especificar el valor `deny` o `allow`. Puede haber tres resultados posibles cuando se evalúe la política: Default-deny, Permitir y Explicit-deny.

Explicit-deny

El resultado de una Instrucción cuyo Effect se ha establecido en `deny`.

Evaluación

El proceso que Amazon SQS utiliza para determinar si una solicitud de entrada se debe denegar o permitir en función de una Política.

Emisor

El usuario que escribe una Política para conceder permisos para un recurso. El emisor, por definición, es siempre el propietario del recurso. AWS no permite a los usuarios de Amazon SQS crear políticas para recursos que no son de su propiedad.

Clave

La característica específica que es la base para la restricción del acceso.

Permiso

El concepto de permitir o impedir el acceso a un recurso mediante una Condición y una Clave.

Política

El documento que actúa como contenedor de una o varias instrucciones.

Amazon SQS utiliza la política para determinar si desea conceder acceso a un usuario para un recurso.

Entidad principal

El usuario que recibe el Permiso en la Política.

Resource

El objeto para el que la Entidad principal solicita acceso.

Instrucción

Descripción formal de un único permiso, escrita en el lenguaje de la política de acceso como parte de un documento más amplio de la Política.

Solicitante

El usuario que envía una solicitud de acceso a un Resource.

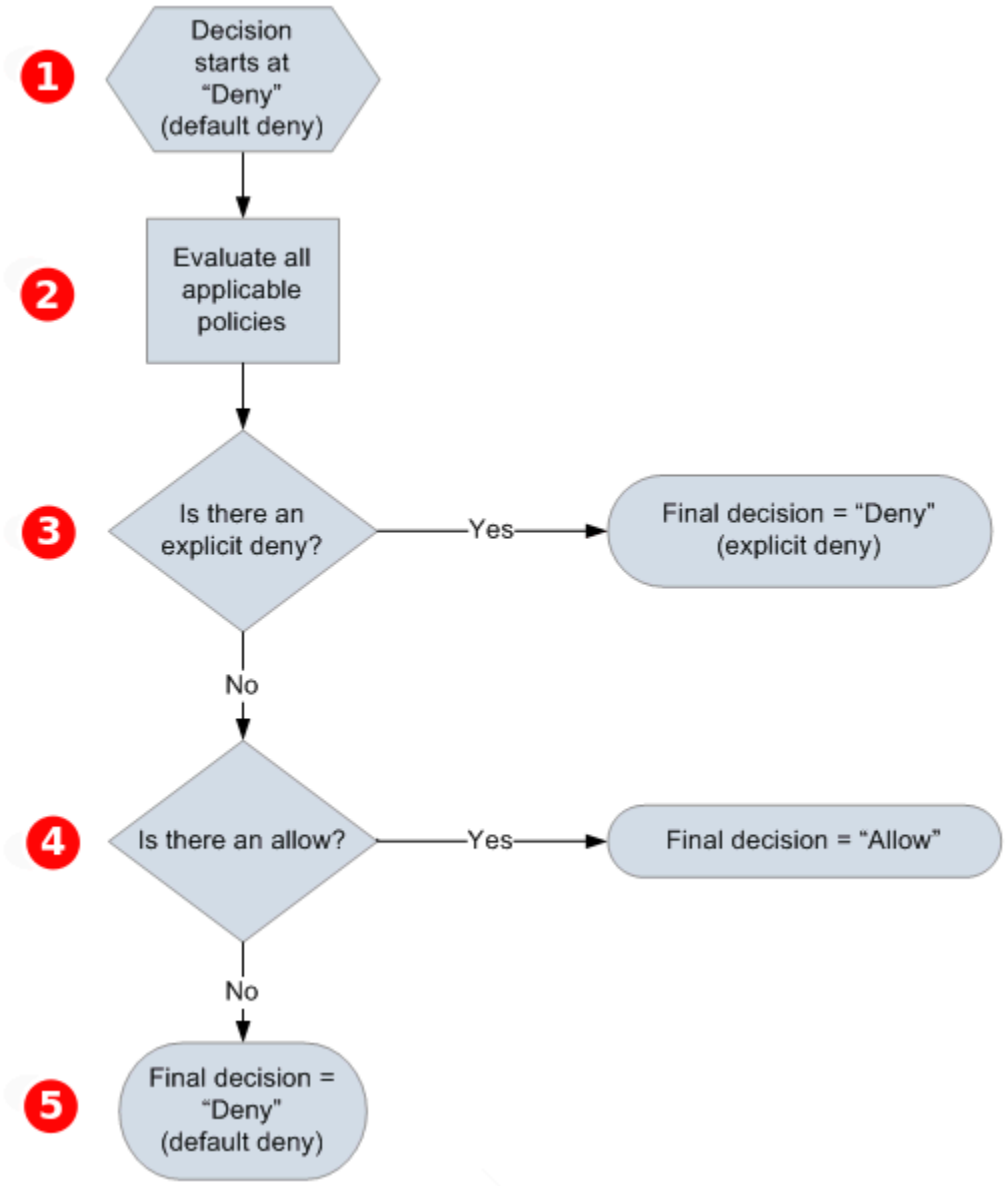
Lógica de evaluación del lenguaje de la política de acceso de Amazon SQS

En el momento de la evaluación, Amazon SQS determina si las solicitudes procedentes de una persona que no es la propietaria del recurso deben permitirse o denegarse. La lógica de evaluación sigue varias reglas básicas:

- De forma predeterminada, todas las solicitudes para utilizar su recurso procedentes de cualquier persona que no sea usted se deniegan.

- Un valor Permitir anula cualquier valor Default-deny.
- Una instrucción Explicit-deny anula cualquier instrucción allow.
- El orden en que se evalúan las políticas no es importante.

En el diagrama siguiente, se describe con detalle cómo Amazon SQS evalúa las decisiones sobre los permisos de acceso.



1

La decisión comienza con una instrucción default-deny.

2

El código de aplicación evalúa todas las políticas aplicables a la solicitud (en función del recurso, la entidad principal, la acción y las condiciones). El orden en que el código de aplicación evalúa las políticas no es importante.

3

El código de aplicación busca una instrucción explicit-deny que se puede aplicar a la solicitud. Si encuentra alguna, devuelve una decisión de denegar y el proceso termina.

4

Si no se encuentra ninguna instrucción explicit-deny, el código de cumplimiento buscará una instrucción allow que se pueda aplicar a la solicitud. Si encuentra alguna, el código de aplicación devuelve una decisión de permitir y el proceso continúa (el servicio sigue procesando la solicitud).

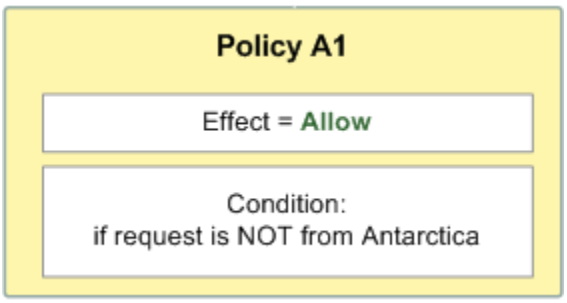
5

Si no se encuentra ninguna instrucción allow, la decisión final será deny (como no hay ninguna instrucción explicit-deny o allow, se considera una instrucción default-deny).

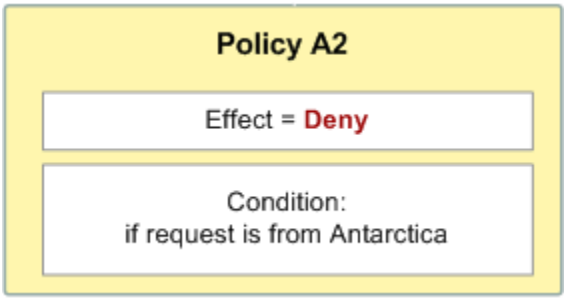
Relaciones entre denegaciones explícitas y predeterminadas en el lenguaje de la política de acceso de Amazon SQS

Si una política de Amazon SQS no se aplica directamente a una solicitud, esta da como resultado Default-deny. Por ejemplo, si un usuario solicita permiso para utilizar Amazon SQS pero la única política que se aplica al usuario puede utilizar DynamoDB, las solicitudes dan como resultado default-deny.

Si no se cumple una condición de una instrucción, la solicitud da como resultado default-deny. Si se cumplen todas las condiciones de una instrucción, la solicitud da como resultado Permitir o Explicit-deny, en función del valor del elemento Effect de la política. Las políticas no especifican qué es lo que se debe hacer si no se cumple una condición, por lo que el resultado predeterminado en este caso es default-deny. Por ejemplo, suponga que desea evitar las solicitudes que proceden de la Antártida. Puede escribir una política denominada A1 que permite una solicitud solo si no procede de la Antártida. El siguiente diagrama ilustra la política de Amazon SQS.

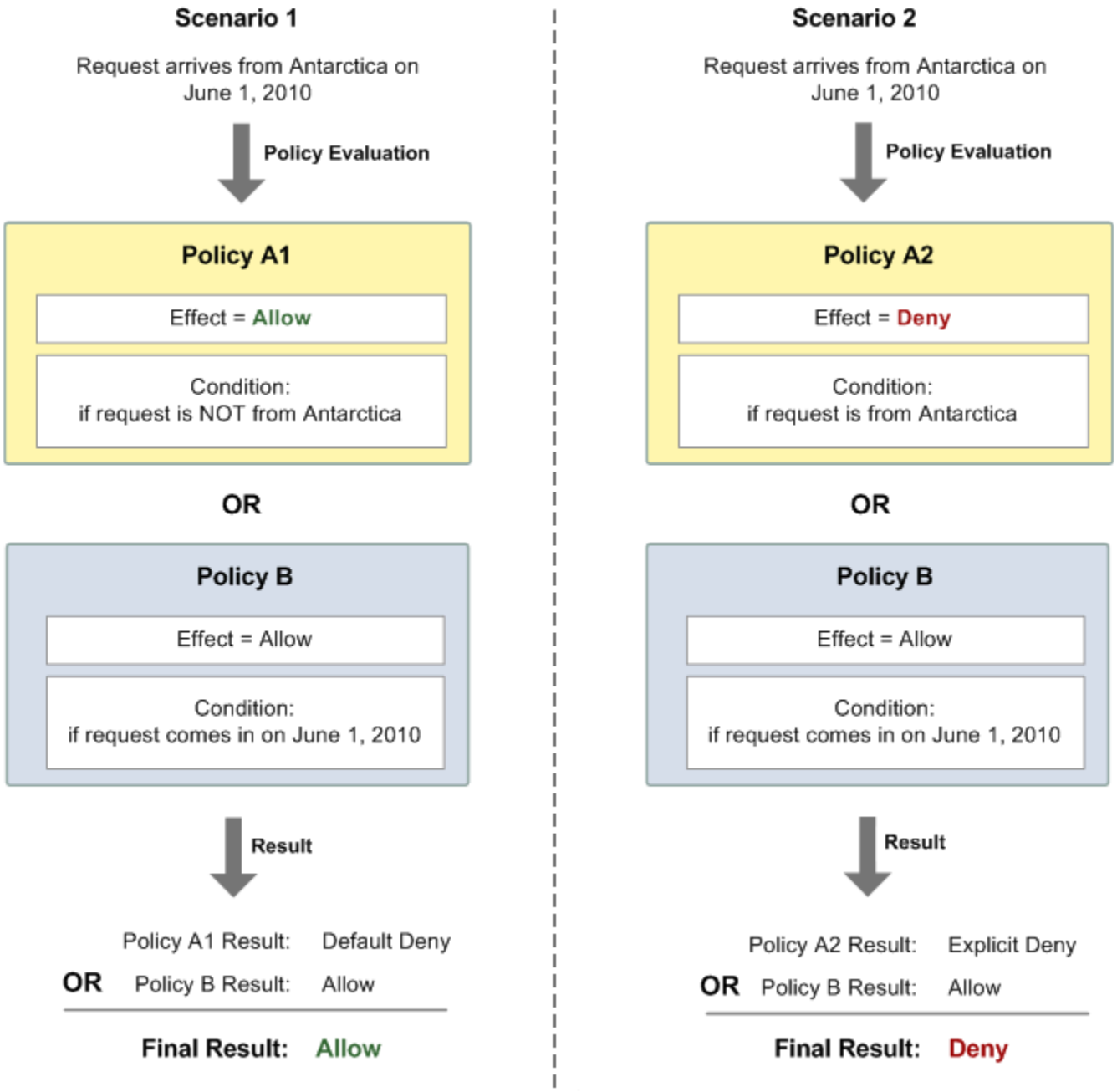


Si un usuario envía una solicitud desde Estados Unidos, la condición se cumple (la solicitud no procede de la Antártida) y la solicitud da como resultado allow. Sin embargo, si un usuario envía una solicitud desde la Antártida, la condición no se cumple y la solicitud da como resultado de forma predeterminada default-deny. Puede cambiar el resultado a explicit-deny si escribe una política A2 que deniegue explícitamente una solicitud si procede de la Antártida. El siguiente diagrama ilustra esta política.



Si un usuario envía una solicitud desde la Antártida, la condición se cumple y la solicitud da como resultado explicit-deny.

La diferencia entre default-deny y explicit-deny es importante, ya que allow puede sobrescribir la primera, pero no la segunda. Por ejemplo, la política B permite solicitudes si llegan el 1 de junio de 2010. En el siguiente diagrama se compara la combinación de esta política con las políticas A1 y A2.



En el escenario 1, la política A1 da lugar a default-deny y la política B da lugar a allow, ya que la política permite las solicitudes que entran el 1 de junio de 2010. La instrucción allow de la política B anula la instrucción default-deny de la política A1 y la solicitud se permite.

En el escenario 2, la política B2 da lugar a explicit-deny y la política B da lugar a allow. La instrucción explicit-deny de la política A2 anula la instrucción allow de la política B y la solicitud se deniega.

Limitaciones de las políticas personalizadas de Amazon SQS

Acceso entre cuentas

Los permisos entre cuentas no se aplican a las siguientes acciones:

- `AddPermission`
- `CancelMessageMoveTask`
- `CreateQueue`
- `DeleteQueue`
- `ListMessageMoveTask`
- `ListQueues`
- `ListQueueTags`
- `RemovePermission`
- `SetQueueAttributes`
- `StartMessageMoveTask`
- `TagQueue`
- `UntagQueue`

Claves de condición

En la actualidad, Amazon SQS únicamente admite un subconjunto limitado de claves de condición disponibles en IAM. Para obtener más información, consulte Permisos de la API de Amazon SQS: referencia de acciones y recursos.

Ejemplos de lenguaje de la política de acceso de Amazon SQS personalizada

A continuación, se muestran algunos ejemplos de políticas de acceso de Amazon SQS típicas.

Ejemplo 1: conceder permiso a una cuenta

En el ejemplo siguiente, la política de Amazon SQS proporciona a la `queue2` 111122223333 permiso para enviar y recibir información de la cola Cuenta de AWS , que es propiedad de la Cuenta de AWS 444455556666.

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase1",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2"
   }]
}
```

Ejemplo 2: conceder permiso a una o varias cuentas

El siguiente ejemplo de política de Amazon SQS otorga uno o más Cuentas de AWS accesos a las colas propiedad de su cuenta durante un período de tiempo específico. Es necesario escribir esta política y cargarla en Amazon SQS mediante la acción `SetQueueAttributes`, ya que la acción `AddPermission` no permite especificar una restricción de tiempo al conceder acceso a una cola.

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"
            }
         }
      }]
}
```

Ejemplo 3: conceder permiso a las solicitudes de las EC2 instancias de Amazon

El siguiente ejemplo de política de Amazon SQS proporciona acceso a las solicitudes que provienen de instancias de Amazon EC2 . Este ejemplo se basa en el ejemplo "Ejemplo 2: conceder permiso a una o varias cuentas": restringe el acceso antes del 30 de junio de 2009 a las 12:00 h (UTC) y al rango de direcciones IP `203.0.113.0/24`. Es necesario escribir esta política y cargarla en Amazon SQS mediante la acción `SetQueueAttributes`, porque la acción `AddPermission` no permite especificar una restricción de dirección IP al conceder acceso a una cola.

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"
         }
      }
   }]
}
```

Ejemplo 4: denegar acceso a una cuenta específica

El siguiente ejemplo de política de Amazon SQS deniega un Cuenta de AWS acceso específico a su cola. Este ejemplo se basa en el ejemplo «Ejemplo 1: conceder permiso a una cuenta»: deniega el acceso a lo especificado. Cuenta de AWS Es necesario escribir esta política y cargarla en Amazon SQS mediante la acción `SetQueueAttributes`, ya que la acción `AddPermission` no permite denegar el acceso a una cola (solo permite conceder acceso a una cola).

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"
   }]
}
```

Ejemplo 5: denegar el acceso si no es desde un punto de enlace de la VPC

La siguiente política de ejemplo de Amazon SQS restringe el acceso a `queue1`: 111122223333 puede realizar las acciones `SendMessage` y `ReceiveMessage` solo desde el ID `vpce-1a2b3c4d` del punto de conexión de VPC (especificado mediante la condición `aws:sourceVpce`). Para obtener más información, consulte Puntos de conexión de Amazon Virtual Private Cloud para Amazon SQS.

> **Note**
>
> - La condición `aws:sourceVpce` no requiere un ARN para el recurso de punto de enlace de la VPC, solo el ID de la VPC.
> - Puede modificar el siguiente ejemplo para restringir todas las acciones para un punto de conexión de VPC concreto mediante la denegación de todas las acciones de Amazon SQS (`sqs:*`) en la segunda instrucción. Sin embargo, una declaración de política de este tipo estipularía que todas las acciones (incluidas las acciones administrativas necesarias para modificar los permisos de cola) deben realizarse a través del punto de enlace de la VPC específico definido en la política, lo que podría impedir al usuario de la cola modificar los permisos en el futuro.

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"
            }
         }
      }
   ]
}
```

Uso de credenciales de seguridad temporales con Amazon SQS

Además de crear usuarios con sus propias credenciales de seguridad, IAM también le permite conceder credenciales de seguridad temporales a cualquier usuario, lo que le permite acceder a sus AWS servicios y recursos. Puede administrar usuarios que tengan Cuentas de AWS. También puede administrar los usuarios de su sistema que no los tengan Cuentas de AWS (usuarios federados). Además, las aplicaciones que cree para acceder a sus AWS recursos también se pueden considerar «usuarios».

Puede utilizar estas credenciales de seguridad temporales para realizar solicitudes a Amazon SQS. Las bibliotecas de la API computan el valor de firma necesario con esas credenciales para autenticar su solicitud. Si envía las solicitudes con las credenciales caducadas, Amazon SQS deniega la solicitud.

> **Note**
>
> No se puede establecer una política basada en credenciales temporales.

Requisitos previos

1. Utilice IAM para crear credenciales de seguridad temporales:

- Token de seguridad
- ID de clave de acceso
- Clave de acceso secreta

2. Prepare la cadena para firmar con el ID de clave de acceso temporal y el token de seguridad.
3. Utilice la clave de acceso secreta temporal en lugar de su propia clave de acceso secreta para firmar la solicitud de la API de consultas.

> **Note**
>
> Al enviar la solicitud de la API de consultas firmada, utilice el ID de clave de acceso temporal en lugar de su propio ID de clave de acceso e incluya el token de seguridad. Para obtener más información sobre la compatibilidad de IAM con las credenciales de seguridad temporales, consulte Cómo conceder acceso temporal a sus AWS recursos en la Guía del usuario de IAM.

Para llamar a una acción de la API de consultas de Amazon SQS mediante credenciales de seguridad temporales

1. Solicita un token de seguridad temporal utilizando. AWS Identity and Access Management Para obtener más información, consulte Creación de credenciales de seguridad temporales para permitir el acceso a los usuarios de IAM en la Guía del usuario de IAM.

   IAM devuelve un token de seguridad, un ID de clave de acceso y una clave de acceso secreta.
2. Prepare la consulta con el ID de clave de acceso temporal en lugar de su propio ID de clave de acceso e incluya el token de seguridad. Firme la solicitud con la clave de acceso secreta temporal en lugar de utilizar su propia clave.
3. Envíe la cadena de consulta firmada con el ID de clave de acceso temporal y el token de seguridad.

   En el siguiente ejemplo, se muestra cómo se utilizan las credenciales de seguridad temporales para autenticar una solicitud de Amazon SQS. La estructura de *AUTHPARAMS* depende de la firma de la solicitud de API. Para obtener más información, consulte Firmar solicitudes de AWS API en la Referencia general de 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
```

El siguiente ejemplo utiliza credenciales de seguridad temporales para enviar dos mensajes con la acción `SendMessageBatch`.

```
https://sqs.us-east-2.amazonaws.com/
?Action=SendMessageBatch
&SendMessageBatchRequestEntry.1.Id=test_msg_001
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201
&SendMessageBatchRequestEntry.2.Id=test_msg_002
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202
&SendMessageBatchRequestEntry.2.DelaySeconds=60
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
&AWSAccessKeyId=AKIAI44QH8DHBEXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

Administración de acceso para colas cifradas de Amazon SQS con políticas de privilegio mínimo

Puede utilizar Amazon SQS para intercambiar datos confidenciales entre aplicaciones mediante el cifrado del servidor (SSE) integrado con AWS Key Management Service (KMS). Con la integración de Amazon SQS AWS KMS, puede gestionar de forma centralizada las claves que protegen Amazon SQS, así como las claves que protegen sus demás recursos. AWS

Varios AWS servicios pueden actuar como fuentes de eventos que envían eventos a Amazon SQS. Para permitir que una fuente de eventos acceda a la cola cifrada de Amazon SQS, debe configurar la cola con una clave gestionada por el cliente. AWS KMS A continuación, utilice la política de claves para permitir que el servicio utilice los métodos de API necesarios. AWS KMS El servicio también requiere permisos de autenticación de acceso para que la cola pueda enviar eventos. Puede

conseguirlo mediante una política de Amazon SQS, que es una política basada en recursos que puede utilizar para controlar el acceso a la cola de Amazon SQS y a sus datos.

En las siguientes secciones se proporciona información sobre cómo controlar el acceso a la cola cifrada de Amazon SQS mediante la política de Amazon SQS y la política de claves. AWS KMS Las políticas de esta guía le ayudarán a conseguir el privilegio mínimo.

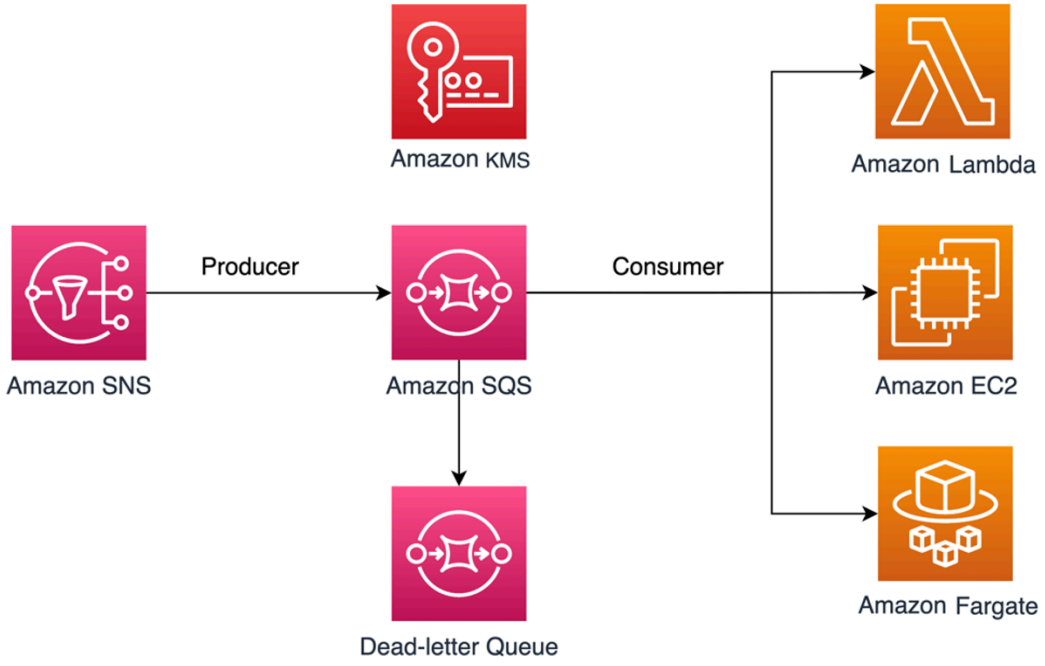
En esta guía también se describe cómo las políticas basadas en recursos abordan el problema del suplente confuso mediante la utilización de las claves de contexto de condición de IAM globales `aws:SourceArn`, `aws:SourceAccount` y `aws:PrincipalOrgID`.

Temas

- Descripción general
- Política de claves de privilegio mínimo para Amazon SQS
- Instrucciones de política de Amazon SQS para la cola de mensajes fallidos
- Prevención del problema del suplente confuso entre servicios
- Uso de Analizador de acceso de IAM para revisar el acceso entre cuentas

Descripción general

En este tema, le guiaremos a través de un caso de uso común para ilustrar cómo puede crear la política de claves y la política de colas de Amazon SQS. Este caso de uso se muestra en la siguiente imagen.



En este ejemplo, el productor de mensajes es un tema de Amazon Simple Notification Service (SNS) que está configurado para la distribución ramificada de mensajes a su cola cifrada de Amazon SQS. El consumidor de mensajes es un servicio informático, como una AWS Lambdafunción, una instancia de Amazon Elastic Compute Cloud (EC2) o un AWS Fargatecontenedor. A continuación, la cola de Amazon SQS se configura para enviar los mensajes fallidos a una cola de mensajes fallidos (DLQ). Esto resulta útil para depurar la aplicación o el sistema de mensajería, ya que DLQs permite aislar los mensajes no consumidos y determinar por qué no se procesaron correctamente. En la solución definida en este tema, se utiliza un servicio de computación como una función de Lambda para procesar los mensajes almacenados en la cola de Amazon SQS. Si el consumidor de mensajes se encuentra en una nube privada virtual (VPC), la instrucción de política `DenyReceivingIfNotThroughVPCE` incluida en esta guía le permite restringir la recepción de mensajes a esa VPC específica.

> **Note**
>
> Esta guía contiene solo los permisos de IAM necesarios en forma de instrucciones de política. Para crear la política, debe añadir las declaraciones a su política de Amazon SQS o a su política AWS KMS clave. Esta guía no proporciona instrucciones sobre cómo crear la cola o la clave de Amazon SQS. AWS KMS Para obtener instrucciones sobre cómo crear estos recursos, consulte Creación de una cola de Amazon SQS y Creación de claves.

La política de Amazon SQS definida en esta guía no admite el redireccionamiento de mensajes directamente a la misma cola de Amazon SQS o a una cola diferente.

Política de claves de privilegio mínimo para Amazon SQS

En esta sección, describimos los permisos con privilegios mínimos necesarios AWS KMS para la clave administrada por el cliente que utiliza para cifrar la cola de Amazon SQS. Con estos permisos, puede limitar el acceso solo a las entidades previstas a la vez que implementa el privilegio mínimo. La política de claves debe constar de las siguientes instrucciones de política, que describimos detalladamente a continuación:

- Otorgue permisos de administrador a la clave AWS KMS
- Concesión de acceso de solo lectura a los metadatos de clave
- Concesión de permisos de KMS de Amazon SNS a Amazon SNS para publicar mensajes en la cola
- Permiso a los consumidores para descifrar mensajes de la cola

Otorgue permisos de administrador a la clave AWS KMS

Para crear una AWS KMS clave, debe proporcionar permisos de AWS KMS administrador a la función de IAM que utilice para implementar la AWS KMS clave. Estos permisos de administrador se definen en la siguiente instrucción de política de `AllowKeyAdminPermissions`. Cuando añadas esta declaración a tu política de AWS KMS claves, asegúrate de *`<admin-role ARN>`* sustituirla por el nombre de recurso de Amazon (ARN) de la función de IAM utilizada para implementar la AWS KMS clave, gestionar la AWS KMS clave o ambas. Puede ser el rol de IAM de su canalización de implementación o el rol de administrador de su organización en AWS Organizations.

```
{
  "Sid": "AllowKeyAdminPermissions",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<admin-role ARN>"
    ]
  },
  "Action": [
    "kms:Create*",
```

```
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:TagResource",
    "kms:UntagResource",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

> **Note**
>
> En una política AWS KMS clave, el valor del `Resource` elemento debe ser`*`, lo que significa «esta AWS KMS clave». El asterisco (`*`) identifica la AWS KMS clave a la que se adjunta la política clave.

Concesión de acceso de solo lectura a los metadatos de clave

Para conceder a otros roles de IAM acceso de solo lectura a los metadatos de clave, agregue la instrucción `AllowReadAccessToKeyMetaData` a su política de claves. Por ejemplo, la siguiente declaración le permite enumerar todas las AWS KMS claves de su cuenta con fines de auditoría. Esta declaración concede al usuario AWS raíz acceso de solo lectura a los metadatos clave. Por lo tanto, cualquier entidad principal de IAM en la cuenta puede tener acceso a los metadatos de clave cuando sus políticas basadas en identidad tengan los permisos enumerados en la siguiente instrucción: `kms:Describe*`, `kms:Get*` y `kms:List*`. Asegúrese de reemplazarlos *`<account-ID>`* con su propia información.

```
{
  "Sid": "AllowReadAcesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
```

```
    ]
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}
```

Concesión de permisos de KMS de Amazon SNS a Amazon SNS para publicar mensajes en la cola

Para permitir que su tema de Amazon SNS publique mensajes en su cola cifrada de Amazon SQS, agregue la instrucción de política `AllowSNSToSendToSQS` a su política de claves. Esta declaración otorga a Amazon SNS permisos para usar la AWS KMS clave para publicar en su cola de Amazon SQS. Asegúrese de sustituirla por su *<account-ID>* propia información.

> **Note**
>
> Lo `Condition` indicado en la declaración limita el acceso únicamente al servicio Amazon SNS en la misma AWS cuenta.

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
```

```
}
```

Permiso a los consumidores para descifrar mensajes de la cola

La siguiente instrucción `AllowConsumersToReceiveFromTheQueue` concede al consumidor de mensajes de Amazon SQS los permisos necesarios para descifrar los mensajes recibidos de la cola de Amazon SQS cifrada. Cuando adjunte la declaración de política, *<consumer's runtime role ARN>* sustitúyala por el ARN del rol de ejecución de IAM del consumidor del mensaje.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

Política de Amazon SQS de privilegio mínimo

Esta sección le guía a través de las políticas de colas de Amazon SQS de privilegio mínimo para el caso de uso que se trata en esta guía (por ejemplo, de Amazon SNS a Amazon SQS). La política definida está diseñada para evitar el acceso no intencionado mediante la utilización de una combinación de las instrucciones `Deny` y `Allow`. Las instrucciones `Allow` conceden acceso a la entidad o entidades previstas. Las instrucciones `Deny` impiden que otras entidades no previstas accedan a la cola de Amazon SQS, al mismo tiempo que excluyen a la entidad prevista en la condición de política.

La política de Amazon SQS incluye las siguientes instrucciones, que describimos detalladamente a continuación:

- Restricción de los permisos de administración de Amazon SQS
- Restricción de las acciones de la cola de Amazon SQS de la organización especificada
- Concesión de permisos de Amazon SQS a los consumidores

- Aplicación del cifrado de los datos en tránsito
- Restricción de la transmisión de mensajes a un tema específico de Amazon SNS
- (Opcional) Restricción de la recepción de mensajes a un punto de conexión de VPC específico

Restricción de los permisos de administración de Amazon SQS

La siguiente instrucción de política `RestrictAdminQueueActions` restringe los permisos de administración de Amazon SQS solo al rol o a los roles de IAM que utilice para implementar la cola, administrarla o ambas cosas. Asegúrese de sustituir *`<placeholder values>`* con su propia información. Especifique el ARN del rol de IAM utilizado para implementar la cola de Amazon SQS, así como el de cualquier rol de administrador que deba tener permisos ARNs de administración de 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>"
      ]
    }
  }
}
```

Restricción de las acciones de la cola de Amazon SQS de la organización especificada

Como ayuda para proteger sus recursos de Amazon SQS del acceso externo (acceso por parte de una entidad ajena a su organización de AWS), utilice la siguiente instrucción. Esta instrucción limita

el acceso a la cola de Amazon SQS a la organización que especifique en la `Condition`. Asegúrese de *<SQS queue ARN>* sustituirlo por el ARN de la función de IAM utilizada para implementar la cola de Amazon SQS y, por el *<org-id>* ID de su organización.

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

Concesión de permisos de Amazon SQS a los consumidores

Para recibir mensajes de la cola de Amazon SQS, debe proporcionar al consumidor de mensajes los permisos necesarios. La siguiente instrucción de política concede al consumidor que especifique los permisos necesarios para consumir mensajes de la cola de Amazon SQS. Al añadir la declaración a su política de Amazon SQS, asegúrese de *<consumer's IAM runtime role ARN>* sustituirla por el ARN de la función de ejecución de IAM utilizada por el consumidor y por el ARN de la función de IAM utilizada para implementar la cola de Amazon SQS. *<SQS queue ARN>*

```
{
  "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>"
}
```

Para impedir que otras entidades reciban mensajes de la cola de Amazon SQS, agregue la instrucción `DenyOtherConsumersFromReceiving` a la política de la cola de Amazon SQS. Esta instrucción restringe el consumo de mensajes al consumidor que usted especifique y no permite que otros consumidores tengan acceso, aunque sus permisos de identidad se lo concedan. Asegúrese de sustituir y por su propia información. *<SQS queue ARN>* *<consumer's runtime role ARN>*

```
{
  "Sid": "DenyOtherConsumersFromReceiving",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": "<consumer's execution role ARN>"
    }
  }
}
```

Aplicación del cifrado de los datos en tránsito

La siguiente instrucción de política `DenyUnsecureTransport` obliga a los consumidores y productores a utilizar canales seguros (conexiones TLS) para enviar y recibir mensajes de la cola

de Amazon SQS. Asegúrese de sustituirlo por *<SQS queue ARN>* el ARN de la función de IAM utilizada para implementar la cola de Amazon SQS.

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}
```

Restricción de la transmisión de mensajes a un tema específico de Amazon SNS

La siguiente instrucción de política `AllowSNSToSendToTheQueue` permite que el tema de Amazon SNS especificado envíe mensajes a la cola de Amazon SQS. Asegúrese de *<SQS queue ARN>* sustituirlo por el ARN de la función de IAM utilizada para implementar la cola de Amazon SQS y por el ARN *<SNS topic ARN>* del tema Amazon SNS.

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
```

```
  }
}
```

La siguiente instrucción de política `DenyAllProducersExceptSNSFromSending` impide que otros productores envíen mensajes a la cola. Sustituya *<SQS queue ARN>* y *<SNS topic ARN>* con su propia información.

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(Opcional) Restricción de la recepción de mensajes a un punto de conexión de VPC específico

Para restringir la recepción de mensajes a solo un punto de conexión de VPC específico, agregue la siguiente instrucción de política a su política de colas de Amazon SQS. Esta instrucción impide que un consumidor de mensajes reciba mensajes de la cola a menos que los mensajes procedan del punto de conexión de VPC deseado. *<SQS queue ARN>*Sustitúyalo por el ARN de la función de IAM utilizada para implementar la cola de Amazon SQS y por el ID *<vpce_id>* del punto de enlace de la VPC.

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
```

```
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

Instrucciones de política de Amazon SQS para la cola de mensajes fallidos

Agregue las siguientes instrucciones de política, identificadas por su ID de instrucción, a su política de acceso de DLQ:

- `RestrictAdminQueueActions`
- `DenyQueueActionsOutsideOrg`
- `AllowConsumersToReceiveFromTheQueue`
- `DenyOtherConsumersFromReceiving`
- `DenyUnsecureTransport`

Además de agregar las instrucciones de política anteriores a su política de acceso de DLQ, también debe agregar una instrucción para restringir la transmisión de mensajes a las colas de Amazon SQS, como se describe en la sección siguiente.

Restricción de la transmisión de mensajes a las colas de Amazon SQS

Para restringir el acceso solo a las colas de Amazon SQS de la misma cuenta, agregue la siguiente instrucción de política `DenyAnyProducersExceptSQS` a la política de colas de DLQ. Esta instrucción no limita la transmisión de mensajes a una cola específica porque necesita implementar la DLQ antes de crear la cola principal, por lo que no conocerá el ARN de Amazon SQS cuando cree la DLQ. Si necesita limitar el acceso solo a una cola de Amazon SQS, modifique `aws:SourceArn` en la `Condition` con el ARN de su cola de origen de Amazon SQS cuando lo conozca.

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
```

```
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

> **Important**
>
> Las políticas de colas de Amazon SQS definidas en esta guía no restringen la acción `sqs:PurgeQueue` a un rol o roles de IAM determinados. La acción `sqs:PurgeQueue` le permite eliminar todos los mensajes de la cola de Amazon SQS. También puede utilizar esta acción para realizar cambios en el formato de los mensajes sin reemplazar la cola de Amazon SQS. Al depurar una aplicación, puede borrar la cola de Amazon SQS para eliminar mensajes potencialmente erróneos. Cuando pruebe la aplicación, puede dirigir un elevado volumen de mensajes a través de la cola de Amazon SQS y, a continuación, purgarla para empezar de cero antes de pasar a producción. La razón de no restringir esta acción a un rol determinado se debe a que es posible que este rol no se conozca al implementar la cola de Amazon SQS. Deberá agregar este permiso a la política basada en identidades del rol para poder purgar la cola.

Prevención del problema del suplente confuso entre servicios

El problema del suplente confuso es una cuestión de seguridad en la que una entidad que no tiene permiso para realizar una acción puede obligar a una entidad con más privilegios a realizar la acción. Para evitarlo, AWS proporciona herramientas que te ayudan a proteger tu cuenta si permites que terceros (lo que se conoce como cuentas cruzadas) u otros AWS servicios (conocidos como servicios cruzados) accedan a los recursos de tu cuenta. Las instrucciones de política de esta sección pueden ayudarlo a prevenir el problema del suplente confuso entre servicios.

La suplantación entre servicios puedes producirse cuando un servicio (el servicio que lleva a cabo las llamadas) llama a otro servicio (el servicio al que se llama). El servicio que lleva a cabo las llamadas se puede manipular para utilizar sus permisos a fin de actuar en función de los recursos de otro cliente de una manera en la que no debe tener permiso para acceder. Como ayuda para protegerse contra este problema, las políticas basadas en recursos definidas en esta publicación utilizan las claves de contexto de condición IAM globales `aws:SourceArn`, `aws:SourceAccount` y `aws:PrincipalOrgID`. Esto limita los permisos que tiene un servicio a un recurso específico, una cuenta específica o una organización específica en AWS Organizations.

Uso de Analizador de acceso de IAM para revisar el acceso entre cuentas

Puede utilizar AWS IAM Access Analyzer para revisar sus políticas de colas AWS KMS y políticas clave de Amazon SQS y avisarle cuando una cola o clave de Amazon SQS conceda acceso a AWS KMS una entidad externa. Analizador de acceso de IAM le ayuda a identificar los recursos de su organización y sus cuentas que se comparten con una entidad externa. Esta zona de confianza puede ser una AWS cuenta o la organización dentro de AWS Organizations que especifique al activar IAM Access Analyzer.

IAM Access Analyzer identifica los recursos compartidos con entidades externas mediante el uso de un razonamiento basado en la lógica para analizar las políticas basadas en los recursos de su entorno. AWS Para cada instancia de un recurso compartido fuera de su zona de confianza, Analizador de acceso genera un resultado. Los resultados incluyen información sobre el acceso y la entidad principal externa a la que se le concede. Revise los resultados para determinar si el acceso es intencionado y seguro, o si el acceso es no intencionado y supone un riesgo para la seguridad. En caso de accesos no intencionados, revise la política afectada y corríjala. Consulte esta entrada del blog para obtener más información sobre cómo AWS IAM Access Analyzer identifica el acceso no deseado a sus recursos. AWS

Para obtener más información sobre AWS IAM Access Analyzer, consulte la documentación de IAM Access Analyzer.AWS

Permisos de la API de Amazon SQS: referencia de acciones y recursos

A la hora de configurar Control de acceso y escribir políticas de permisos que puede asociar a una identidad de IAM, puede utilizar la siguiente tabla como referencia. La incluye cada acción de Amazon Simple Queue Service, las acciones correspondientes para las que puede conceder permisos para realizar la acción y el AWS recurso para el que puede conceder los permisos.

Especifique las acciones en el campo `Action` de la política y el valor del recurso en el campo `Resource` de la política. Para especificar una acción, use el prefijo `sqs:` seguido del nombre de acción (por ejemplo, `sqs:CreateQueue`).

En la actualidad, Amazon SQS admite las claves de contexto de condición global disponibles en IAM.

API de Amazon Simple Queue Service y permisos requeridos para las acciones

AddPermission

Acciones: `sqs:AddPermission`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

ChangeMessageVisibility

Acciones: `sqs:ChangeMessageVisibility`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

ChangeMessageVisibilityBatch

Acciones: `sqs:ChangeMessageVisibilityBatch`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

CreateQueue

Acciones: `sqs:CreateQueue`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessage

Acciones: `sqs:DeleteMessage`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessageBatch

Acciones: `sqs:DeleteMessageBatch`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

DeleteQueue

Acciones: `sqs:DeleteQueue`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

GetQueueAttributes

Acciones: `sqs:GetQueueAttributes`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

GetQueueUrl

Acciones: `sqs:GetQueueUrl`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

ListDeadLetterSourceQueues

Acciones: `sqs:ListDeadLetterSourceQueues`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

ListQueues

Acciones: `sqs:ListQueues`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

ListQueueTags

Acciones: `sqs:ListQueueTags`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

PurgeQueue

Acciones: `sqs:PurgeQueue`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

ReceiveMessage

Acciones: `sqs:ReceiveMessage`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

RemovePermission

Acciones: `sqs:RemovePermission`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

SendMessage y SendMessageBatch

Acciones: `sqs:SendMessage`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

SetQueueAttributes

Acciones: `sqs:SetQueueAttributes`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

TagQueue

Acciones: `sqs:TagQueue`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

UntagQueue

Acciones: `sqs:UntagQueue`

Recurso: `arn:aws:sqs:region:account_id:queue_name`

Registro y monitoreo en Amazon SQS

Amazon Simple Queue Service está integrado con AWS CloudTrailun servicio que proporciona un registro de las acciones realizadas por un usuario, rol o un Servicio de AWS. CloudTrail captura todas las llamadas a la API de Amazon SQS como eventos. Las llamadas capturadas incluyen llamadas desde la consola de Amazon SQS y llamadas de código a las operaciones de la API de Amazon SQS. Con la información recopilada por CloudTrail, puede determinar la solicitud que se realizó a Amazon SQS, la dirección IP desde la que se realizó la solicitud, cuándo se realizó y detalles adicionales.

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario le ayuda a determinar lo siguiente:

- Si la solicitud se realizó con las credenciales del usuario raíz o del usuario.
- Si la solicitud se realizó en nombre de un usuario de IAM Identity Center.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro Servicio de AWS.

CloudTrail está activa en tu cuenta Cuenta de AWS cuando creas la cuenta y tienes acceso automáticamente al historial de CloudTrail eventos. El historial de CloudTrail eventos proporciona un registro visible, consultable, descargable e inmutable de los últimos 90 días de eventos de gestión registrados en un. Región de AWSPara obtener más información, consulte Uso del historial de CloudTrail eventos en la Guía del usuario.AWS CloudTrail La visualización del historial de eventos no conlleva ningún CloudTrail cargo.

Para tener un registro continuo de los eventos de Cuenta de AWS los últimos 90 días, crea un almacén de datos de eventos de senderos o CloudTraillagos.

CloudWatch Alarmas Amazon

Supervise una única métrica durante el período de tiempo que especifique y lleve a cabo una o más acciones en función del valor de la métrica en relación con un umbral definido durante varios períodos. Por ejemplo, puede configurar una CloudWatch alarma para enviar una notificación a un tema de Amazon SNS o activar una acción para enviar un mensaje a una cola de Amazon SQS. CloudWatch las alarmas no realizan acciones simplemente porque estén en un estado específico; el estado debe cambiar y permanecer en ese estado durante un número definido de períodos.

Para obtener más información, consulte Creación de CloudWatch alarmas para las métricas de Amazon SQS y Creación de alarmas para colas de cartas sin salida con Amazon CloudWatch.

Amazon CloudWatch Logs

Supervise, almacene y acceda a los archivos de registro relacionados con Amazon SQS configurando sus aplicaciones o funciones de Lambda que procesan los mensajes para enviar los registros a Logs. CloudWatch Puede utilizar estos registros para analizar el procesamiento de mensajes, depurar problemas y supervisar el rendimiento de los flujos de trabajo de Amazon SQS.

Para obtener más información, consulte Registro de llamadas a la API de Amazon Simple Queue Service mediante AWS CloudTrail.

CloudWatch Eventos de Amazon

Utilice Amazon CloudWatch Events para detectar cambios o eventos específicos en su AWS entorno y enrutarlos a una cola de Amazon SQS. Esto le permite capturar datos de eventos, activar flujos de trabajo o almacenar eventos para procesarlos más adelante.

Para obtener más información, consulte Automatización de las notificaciones de AWS los servicios a Amazon SQS mediante Amazon EventBridge en esta guía la evolución de Amazon CloudWatch Events en la Guía del EventBridge usuario de Amazon. EventBridge

AWS CloudTrail Registros

CloudTrail captura un registro detallado de las acciones realizadas en Amazon SQS por los usuarios, roles o. Servicios de AWS Estos registros le permiten realizar un seguimiento de las llamadas a la API `SendMessageReceiveMessage`, por ejemplo `DeleteQueue`, o, y

proporcionan detalles clave, como quién realizó la solicitud, cuándo se produjo y la dirección IP de origen.

Para obtener más información, consulte Registro de llamadas a la API de Amazon Simple Queue Service mediante AWS CloudTrail.

AWS Trusted Advisor

Trusted Advisor utiliza las mejores prácticas desarrolladas a partir del servicio a AWS los clientes para ayudar a optimizar el uso de Amazon SQS. Revisa las colas de Amazon SQS y proporciona recomendaciones prácticas para mejorar la seguridad, mejorar la fiabilidad del procesamiento de mensajes y reducir los costes. Por ejemplo, puede sugerirle que habilite las colas con letra muerta o que mejore sus políticas de acceso a las colas para garantizar la seguridad de las operaciones.

Para obtener más información, consulte AWS Trusted Advisor en la Guía del usuario de Soporte .

CloudTrail senderos

Un rastro permite CloudTrail entregar archivos de registro a un bucket de Amazon S3. Todos los senderos creados con él AWS Management Console son multirregionales. Puede crear un registro de seguimiento de una sola región o multirregionales mediante la AWS CLI. Se recomienda crear un sendero multirregional, ya que puedes capturar toda la actividad de tu Regiones de AWS cuenta. Si crea un registro de seguimiento de una sola región, solo podrá ver los eventos registrados en la Región de AWS del registro de seguimiento. Para obtener más información acerca de los registros de seguimiento, consulte Creación de un registro de seguimiento para su Cuenta de AWS y Creación de un registro de seguimiento para una organización en la Guía del usuario de AWS CloudTrail .

Puede enviar una copia de sus eventos de administración en curso a su bucket de Amazon S3 sin coste alguno CloudTrail mediante la creación de una ruta; sin embargo, hay cargos por almacenamiento en Amazon S3. Para obtener más información sobre CloudTrail los precios, consulte AWS CloudTrail Precios. Para obtener información acerca de los precios de Amazon S3, consulte Precios de Amazon S3.

CloudTrail Almacenes de datos de eventos en Lake

CloudTrail Lake le permite ejecutar consultas basadas en SQL en sus eventos. CloudTrail Lake convierte los eventos existentes en formato JSON basado en filas al formato Apache ORC. ORC es un formato de almacenamiento en columnas optimizado para una recuperación rápida de datos. Los eventos se agregan en almacenes de datos de eventos, que son recopilaciones inmutables de eventos en función de criterios que se seleccionan aplicando selectores de eventos avanzados. Los selectores que se aplican a un almacén de datos de eventos controlan los

eventos que perduran y están disponibles para la consulta. Para obtener más información sobre CloudTrail Lake, consulte Cómo trabajar con AWS CloudTrail Lake en la Guía del AWS CloudTrail usuario.

CloudTrail Los almacenes de datos y las consultas sobre eventos de Lake conllevan costes. Cuando crea un almacén de datos de eventos, debe elegir la opción de precios que desee utilizar para él. La opción de precios determina el costo de la incorporación y el almacenamiento de los eventos, así como el período de retención predeterminado y máximo del almacén de datos de eventos. Para obtener más información sobre CloudTrail los precios, consulte AWS CloudTrail Precios.

Registro de llamadas a la API de Amazon Simple Queue Service mediante AWS CloudTrail

CloudTrail le permite registrar y supervisar las operaciones de Amazon SQS mediante dos tipos de eventos: eventos de datos y eventos de administración. Esto facilita el seguimiento y la auditoría de la actividad de Amazon SQS en su cuenta.

Eventos de datos de Amazon SQS en CloudTrail

Los eventos de datos proporcionan información sobre las operaciones de recursos realizadas en o dentro de un recurso (por ejemplo, el envío de mensajes a un objeto de Amazon SQS). Se denominan también operaciones del plano de datos. Los eventos de datos suelen ser actividades de gran volumen. De forma predeterminada, CloudTrail no registra los eventos de datos. El historial de CloudTrail eventos no registra los eventos de datos.

Se aplican cargos adicionales a los eventos de datos. Para obtener más información sobre CloudTrail los precios, consulta AWS CloudTrail Precios.

Puede registrar eventos de datos para los tipos de recursos de Amazon SQS mediante la CloudTrail consola o las operaciones AWS CLI de la CloudTrail API. Para obtener más información sobre cómo registrar los eventos de datos, consulte Registro de eventos de datos con la AWS Management Console y Registro de eventos de datos con la AWS Command Line Interface en la Guía del usuario de AWS CloudTrail .

Para registrar los eventos de datos de Amazon SQS CloudTrail, debe utilizar selectores de eventos avanzados para configurar los recursos o acciones específicos de Amazon SQS que desee registrar. Incluya el tipo de recurso `AWS::SQS::Queue`para capturar las acciones relacionadas

con las colas. Puede refinar aún más sus preferencias de registro mediante el uso de filtros como `eventName` `SendMessage`(por ejemplo, los de eventos). Para obtener más información, consulta AdvancedEventSelector en la CloudTrail Referencia de la API de .

Tipo de evento de datos (consola)	resources.type value	Datos APIs registrados en CloudTrail
Cola de Amazon SQS	`AWS::SQS::Queue`	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch • GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

Utilice selectores de eventos avanzados para filtrar los campos y registrar solo los eventos importantes. Para obtener más información acerca de estos campos, consulte AdvancedFieldSelector en la Referencia de la API de AWS CloudTrail .

Eventos de administración de Amazon SQS en CloudTrail

Los eventos de administración proporcionan información sobre las operaciones de administración que se llevan a cabo en los recursos de su Cuenta de AWS empresa. Se denominan también operaciones del plano de control. De forma predeterminada, CloudTrail registra los eventos de administración.

Amazon SQS registra las siguientes operaciones del plano de control CloudTrail como eventos de administración.

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTasks
- PurgeQueue
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Ejemplo de evento de Amazon SQS

Un evento representa una solicitud única de cualquier fuente e incluye información sobre la operación de API solicitada, la fecha y la hora de la operación, los parámetros de la solicitud, etc. CloudTrail Los archivos de registro no son un registro ordenado de las llamadas a la API pública, por lo que los eventos no aparecen en ningún orden específico.

En el siguiente ejemplo, se muestra un CloudTrail evento que demuestra la `SendMessage` operación.

```
{
  "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**
>
> La `ListQueues` operación es un caso único porque no actúa sobre un recurso específico. Como resultado, el campo ARN no incluye un nombre de cola y, en su lugar, utiliza un comodín (*).

Para obtener información sobre el contenido de los CloudTrail registros, consulte el contenido de los CloudTrailregistros en la Guía del usuario AWS CloudTrail .

Supervisión de las colas de Amazon SQS mediante CloudWatch

Amazon SQS y Amazon CloudWatch están integrados para que pueda utilizarlos CloudWatch para ver y analizar las métricas de sus colas de Amazon SQS. Puede ver y analizar las métricas de sus colas desde la consola de Amazon SQS, CloudWatch la consola o mediante AWS CLIla API. CloudWatch También puede configurar CloudWatch alarmas para las métricas de Amazon SQS.

CloudWatch Las métricas de sus colas de Amazon SQS se recopilan automáticamente y se envían a CloudWatch intervalos de un minuto. Estas métricas se recopilan en todas las colas que cumplen con las CloudWatch pautas de actividad. CloudWatch considera que una cola está activa durante un máximo de seis horas si contiene algún mensaje o si alguna acción permite acceder a ella.

Cuando una cola de Amazon SQS permanece inactiva durante más de seis horas, el servicio Amazon SQS se considera inactivo y deja de enviar métricas al servicio. CloudWatch Los datos faltantes o los datos que representan cero no se pueden visualizar en las CloudWatch métricas de Amazon SQS durante el período en que la cola de Amazon SQS estuvo inactiva.

> **Note**
>
> - Se puede activar una cola de Amazon SQS cuando el usuario que llama a una API en relación con la cola no está autorizado y la solicitud falla.
> - La consola de Amazon SQS realiza una llamada a la API `GetQueueAttributes` cuando se abre la página de la cola. La solicitud la de API `GetQueueAttributes` activa la cola.
> - Cuando una cola se activa desde un estado inactivo, se produce un retraso de hasta 15 minutos en CloudWatch las métricas.

- No se cobran cargos por las métricas de Amazon SQS publicadas en. CloudWatch Se ofrecen como parte del servicio de Amazon SQS.
- CloudWatch las métricas se admiten tanto para las colas estándar como para las FIFO.

Acceso a CloudWatch las métricas de Amazon SQS

Amazon SQS y Amazon CloudWatch están integrados para que pueda utilizarlos CloudWatch para ver y analizar las métricas de sus colas de Amazon SQS. Puede ver y analizar las métricas de sus colas desde la consola de Amazon SQS, CloudWatch la consola o mediante AWS CLIla API. CloudWatch También puede configurar CloudWatch alarmas para las métricas de Amazon SQS.

Uso de la consola de Amazon SQS

Utilice la consola de Amazon SQS para acceder a las métricas de hasta 10 colas de Amazon SQS y analizarlas.

1. Inicie sesión en la consola de Amazon SQS.
2. En la lista de colas, elija (marque) las casillas correspondientes a las colas a cuyas métricas desea obtener acceso. Puede mostrar métricas para un máximo de 10 colas.
3. Elija la pestaña Monitorización.

   En la sección SQS metrics se muestran varios gráficos.
4. Para comprender lo que un determinado gráfico representa, pase el ratón por encima del icono

   situado junto al gráfico correspondiente o consulte CloudWatch Métricas disponibles para Amazon SQS.
5. Para cambiar el intervalo de tiempo para todos los gráficos al mismo tiempo, en Time Range, elija el intervalo de tiempo deseado (por ejemplo, Last Hour).
6. Para ver estadísticas adicionales para un gráfico individual, elija el gráfico.
7. En el cuadro de diálogo CloudWatch Monitoring Details (Detalles de monitoreo de), seleccione una estadística en la lista Statistic (Estadística), (por ejemplo, Sum). Para ver una lista de las estadísticas admitidas, consulte CloudWatch Métricas disponibles para Amazon SQS.
8. Para cambiar el intervalo de tiempo durante el cual se muestra un gráfico individual (por ejemplo, para mostrar un intervalo de tiempo de las últimas 24 horas en lugar de los últimos cinco minutos, o para mostrar un periodo de tiempo de cada hora en lugar de cada 5 minutos), con

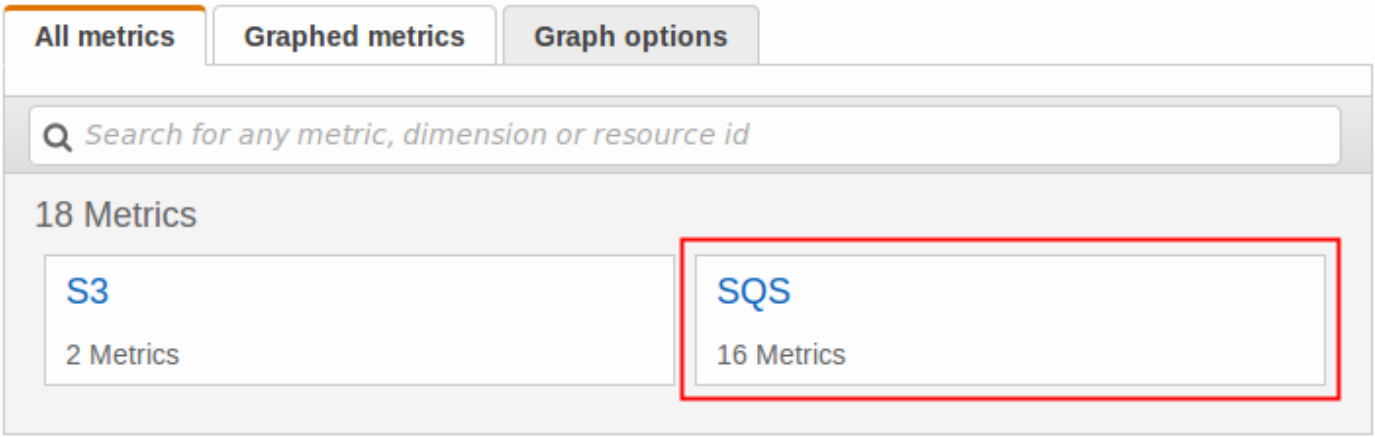
el cuadro de diálogo del gráfico todavía abierto, en Time Range, elija el intervalo de tiempo deseado (por ejemplo, Last 24 Hours). En Period, elija el periodo de tiempo deseado dentro del intervalo de tiempo especificado (por ejemplo, 1 Hour). Cuando termine de examinar el gráfico, elija Close.

9. (Opcional) Para trabajar con CloudWatch funciones adicionales, en la pestaña Supervisión, seleccione Ver todas las CloudWatch métricas y, a continuación, siga las instrucciones del procedimiento. Uso de la CloudWatch consola de Amazon

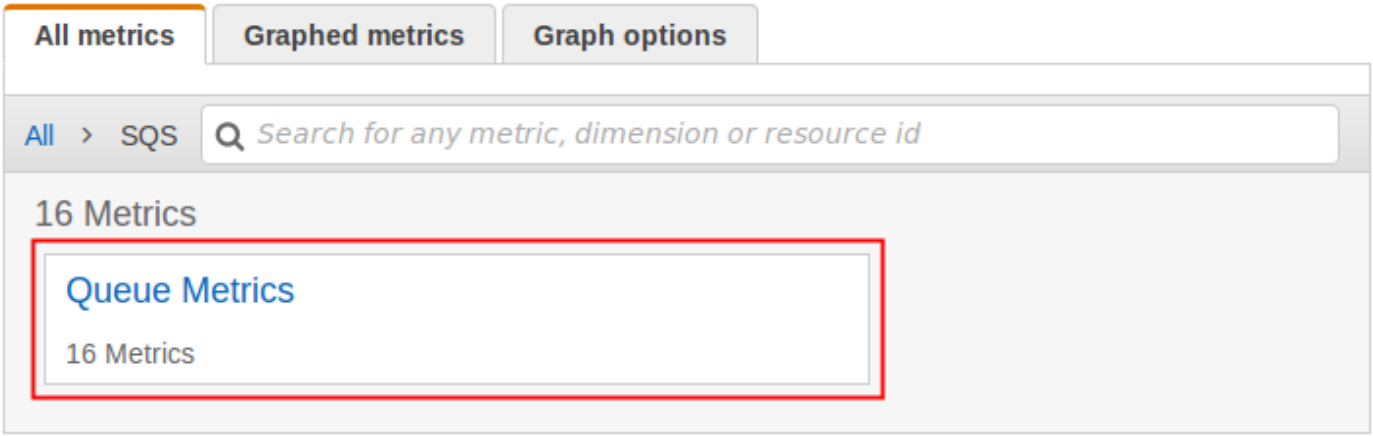
Uso de la CloudWatch consola de Amazon

Utilice la CloudWatch consola para acceder a las métricas de Amazon SQS y analizarlas.

1. Inicie sesión en la consola de CloudWatch.
2. En el panel de navegación, elija Metrics.
3. Seleccione el espacio de nombres de métricas SQS.

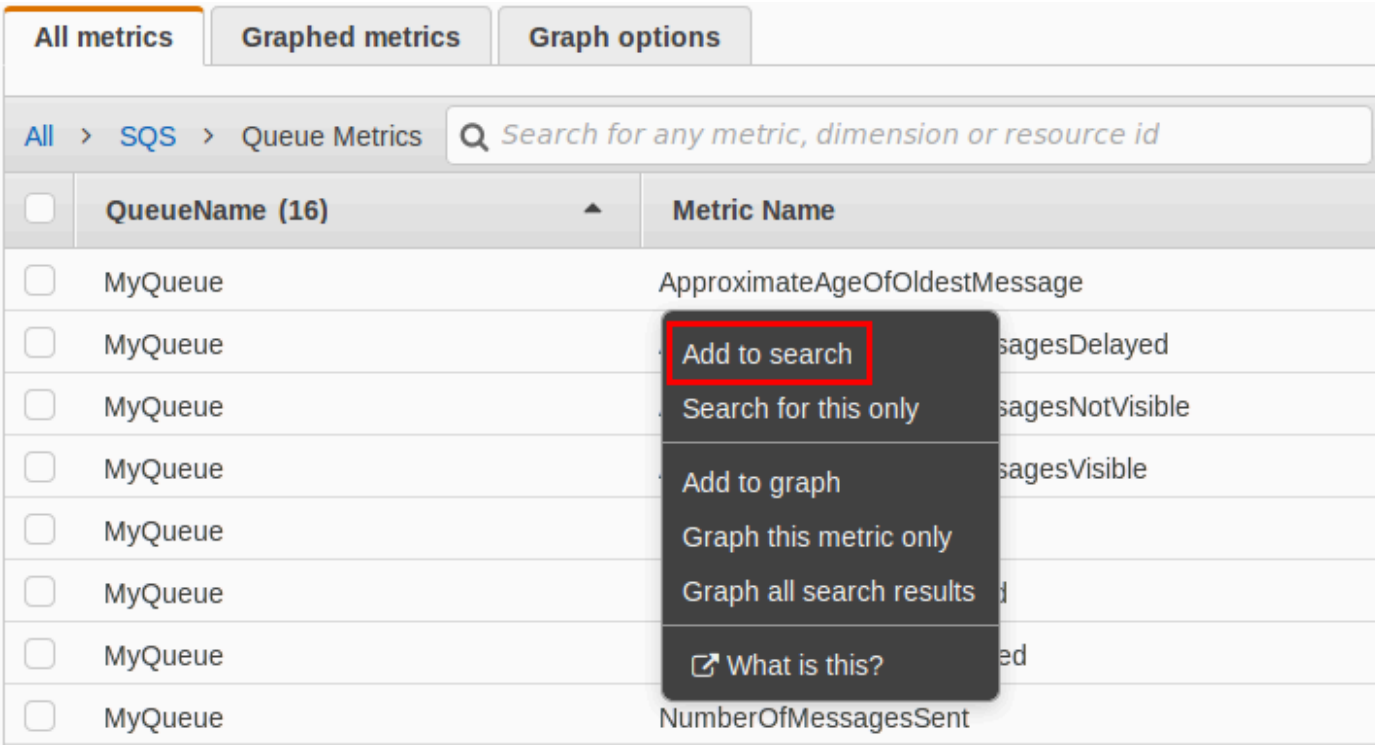


4. Seleccione la dimensión de métricas Queue Metrics.



5. Ahora puede examinar las métricas de Amazon SQS:

- Para ordenar las métricas, utilice el encabezado de columna.
- Para representar gráficamente una métrica, active la casilla de verificación situada junto a ella.
- Para filtrar por métrica, elija el nombre de la métrica y, a continuación, seleccione Add to search (Añadir a búsqueda).



Para obtener más información y opciones adicionales, consulte Graph Metrics and Using Amazon CloudWatch Dashboards en la Guía del CloudWatch usuario de Amazon.

Utilización del AWS Command Line Interface

Para acceder a las métricas de Amazon SQS mediante AWS CLI, ejecute el `get-metric-statistics` comando.

Para obtener más información, consulta Obtener estadísticas de una métrica en la Guía del CloudWatch usuario de Amazon.

Uso de la CloudWatch API

Para acceder a las métricas de Amazon SQS mediante la CloudWatch API, utilice la `GetMetricStatistics` acción.

Para obtener más información, consulta Obtener estadísticas de una métrica en la Guía del CloudWatch usuario de Amazon.

Creación de CloudWatch alarmas para las métricas de Amazon SQS

CloudWatch permite activar alarmas cuando una métrica alcanza un umbral específico. Por ejemplo, puede crear una alarma para la métrica `NumberOfMessagesSent`. Por ejemplo, si se envían más de 100 mensajes a la cola `MyQueue` en una hora, se envía una notificación por correo electrónico. Para obtener más información, consulte Creación de CloudWatch alarmas de Amazon en la Guía del CloudWatch usuario de Amazon.

1. Inicie sesión en AWS Management Console y abra la CloudWatch consola en https://console.aws.amazon.com/cloudwatch/.
2. Elija Alarms y, a continuación, seleccione Create Alarm.
3. En la sección Select Metric (Seleccionar métrica) del cuadro de diálogo Create Alarm (Crear alarma), elija Browse Metrics (Examinar métricas), SQS.
4. En SQS > Queue Metrics, elija el nombre QueueNamey el nombre de la métrica para los que desee configurar una alarma y, a continuación, elija Siguiente. Para ver una lista de las métricas disponibles, consulte CloudWatch Métricas disponibles para Amazon SQS.

   En el siguiente ejemplo, la selección que se realiza corresponde a una alarma de la métrica `NumberOfMessagesSent` para la cola `MyQueue`. La alarma se activa cuando el número de mensajes enviados supera los 100.
5. En la sección Define Alarm (Definir alarma) del cuadro de diálogo Create Alarm (Crear alarma), haga lo siguiente:

   a. En Alarm Threshold (Umbral de alarma), escriba el nombre de la alarma en Name (Nombre) y la descripción en Description (Descripción).

   b. Establezca is en > 100.

   c. Establezca for (para) en 1 out of 1 datapoints (1 de 1 punto de datos).

   d. En Alarm preview (Vista previa de alarma), establezca el valor de Period (Período) en 1 Hour (1 hora).

   e. Establezca Statistic (Estadística) en Standard (Estándar), Sum (Suma).

   f. En Actions (Acciones), establezca Whenever this alarm (Siempre que esta alarma) en State is ALARM (El estado es ALARM).

Si desea CloudWatch enviar una notificación cuando se active la alarma, seleccione un tema existente de Amazon SNS o elija Nueva lista e introduzca las direcciones de correo electrónico separadas por comas.

> **Note**
>
> Si crea un nuevo tema de Amazon SNS, debe verificar las direcciones de correo electrónico antes de que puedan recibir notificaciones. Si el estado de la alarma cambia antes de que se verifiquen las direcciones de correo electrónico, las notificaciones no se envían.

6. Elija Crear alarma.

   Se crea la alarma.

CloudWatch Métricas disponibles para Amazon SQS

Amazon SQS envía las siguientes métricas a. CloudWatch

> **Note**
>
> En el caso de algunas métricas, el resultado es aproximado debido a la arquitectura distribuida de Amazon SQS. En la mayoría de los casos, el recuento debe aproximarse al número real de mensajes de la cola.

Métricas de Amazon SQS

Amazon SQS publica automáticamente las métricas operativas en Amazon en el `AWS/SQS` espacio de CloudWatch nombres. Estas métricas le ayudan a supervisar el estado y el rendimiento de las colas. Debido a la naturaleza distribuida de SQS, muchos valores son aproximados, pero lo suficientemente precisos para la mayoría de las decisiones operativas.

> **Note**
>
> - Todas las métricas emiten valores no negativos solo cuando la cola está activa.

- Algunas métricas (por ejemplo`SentMessageSize`) no se emiten hasta que se envía al menos un mensaje.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateAgeOfOldestMessage`	La antigüedad del mensaje sin procesar más antiguo de la lista.	Segundos	Se informa si la cola contiene al menos un mensaje activo.	• En las colas estándar, si un mensaje se recibe tres o más veces y no se elimina, SQS lo mueve al final de la cola. A continuación, la métrica refleja la antigüedad del siguiente mensaje que no ha superado el umbral de recepción. Este reordenamiento se produce incluso cuando existe una política de retransmisión. • Los mensajes envenenados (los que se reciben repetidamente pero nunca se eliminan) se excluyen de esta métrica hasta que se procesen correctamente. • Cuando un mensaje se traslada a un DLQ después de superar el valor`maxReceiveCount` , la antigüedad se restablece. En ese caso, la métrica del DLQ refleja la hora a la que se trasladó el mensaje, no

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
				cuándo se envió originalmente. • Las colas FIFO no reordenan los mensajes para mantener el orden. Un mensaje fallido bloquea su grupo de mensajes hasta que se elimine o caduque. Si se configura un DLQ, el mensaje se envía allí cuando se alcanza el umbral de recepción.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateNumberOfGroupsWithInflightMessages`	Solo para FIFO. El número de grupos de mensajes con uno o más mensajes en movimiento.	Recuento	Se informa si la cola FIFO está activa.	• Se considera que un mensaje está en curso cuando un consumidor lo ha recibido de la cola, pero que aún no ha sido eliminado o ha caducado. • Esta métrica te ayuda a solucionar problemas y a optimizar el rendimiento de las colas FIFO. Los valores altos suelen indicar una gran simultaneidad. • Si la cola tiene una gran cantidad de pedidos pendientes y este valor sigue siendo bajo, considere la posibilidad de ampliar el número de consumidores o aumentar el número de grupos de mensajes activos. • Para conocer los límites de rendimiento y durante el vuelo, consulte. Cuotas de Amazon SQS

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateNumberOfMessagesDelayed`	El número de mensajes de la cola que están retrasados y que no están disponibles de forma inmediata para su recuperación.	Recuento	Se informa si hay mensajes retrasados en la cola.	• Se aplica a las colas configuradas con un retraso predeterminado y a los mensajes individualmente enviados con un `DelaySeconds` parámetro. • Los mensajes retrasados permanecen ocultos para los consumidores hasta que vence su período de retraso, lo que puede afectar a la acumulación de colas percibidas o al rendimiento.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateNumberOfMessagesNotVisible`	El número de mensajes durante el vuelo que se han recibido pero que aún no se han eliminado o que han caducado.	Recuento	Se informa si existen mensajes durante el vuelo.	• Los mensajes entran en estado de vuelo tras enviarse a un consumidor a través de la `ReceiveMessage` API. • Estos mensajes se ocultan temporalmente a otros consumidores durante el tiempo de espera de visibilidad. • Usa esta métrica para hacer un seguimiento de los retrasos en el procesamiento de los mensajes o de los consumidores atrapados.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateNumberOfMessagesVisible`	El número de mensajes actualmente disponibles para su recuperación y procesamiento.	Recuento	Se informa si la cola está activa.	• Refleja la acumulación de procesamiento actual en la cola. • No hay un límite estricto en cuanto al número de mensajes que se pueden acumular, pero están sujetos al período de retención configurado en la cola. • Un valor alto y constante puede indicar que los consumidores están insuficientemente aprovisionados o que la lógica de procesamiento está atascada.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`NumberOfEmptyReceives` [1]	El número de llamadas a la ReceiveMessageAPI que no devolvieron ningún mensaje.	Recuento	Se notificó durante las operaciones de recepción.	• Esta métrica puede ayudar a identificar las ineficiencias en el comportamiento de las encuestas o los casos de consumo infrautilizados. • Los valores altos se pueden producir cuando la cola está vacía, el consumidor utiliza sondeos cortos o los mensajes se procesan más rápido de lo que se producen. • Este no es un indicador preciso del estado de la cola. Refleja el comportamiento del servicio y puede incluir reintentos.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`NumberOfDeduplicatedSentMessages`	Solo para FIFO. El número de mensajes enviados que se deduplicaron y no se agregaron a la cola.	Recuento	Se informa si se detectan `MessageDeduplicationId` valores o contenido duplicados.	• SQS deduplica los mensajes en función del hash `MessageDeduplicationId` o del contenido (si está activado). • Un valor alto puede indicar que un productor envía repetidamente el mismo mensaje dentro del período de deduplicación de 5 minutos. • Utilice esta métrica para solucionar problemas de lógica redundante del productor o para confirmar que la deduplicación funciona según lo previsto.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave

`NumberOfMessagesDeleted` [1]	El número de mensajes borrados correctamente de la cola.	Recuento	Se informa de cada solicitud de eliminación con un identificador de recibo válido.	• Esta métrica cuenta todas las operaciones de eliminación que se han realizado correctamente, incluso si el mismo mensaje se ha eliminado más de una vez.
				• Los motivos más comunes de los higher-than-expected valores son los siguientes:
				  • Se eliminan varias veces el mismo mensaje con distintos identificadores de recepción, una vez transcurrido el tiempo de espera de visibilidad y el mensaje se vuelve a recibir.
				  • Las eliminaciones se duplican con el mismo identificador de recibo, lo que, aun así, devuelve el estado correcto y aumenta la métrica.
				• Usa esta métrica para realizar un seguimiento del éxito del procesamiento de los mensajes, pero no la trates como un recuento exacto de mensajes eliminados únicos.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`NumberOfMessagesReceived` [1]	El número de mensajes devueltos por la ReceiveMessageAPI.	Recuento	Se informa durante las operaciones de recepción.	• Esto incluye todos los mensajes devueltos a los consumidores, incluidos los que posteriormente se devuelven a la cola debido a la caducidad del tiempo de espera de visibilidad. • Un solo mensaje se puede recibir varias veces si no se elimina, lo que puede provocar que esta métrica supere el número de mensajes enviados. • Utilízala para realizar un seguimiento de la actividad de los consumidores, pero no la trates como un recuento de mensajes únicos procesados.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`NumberOfMessagesSent` [1]	El número de mensajes que se han añadido correctamente a una cola.	Recuento	Se notifica por cada envío manual realizado correctamente.	• Las llamadas manuales realizadas `SendMessage` o `SendMessageBatch` contabilizadas, incluidas las que se dirigen directamente a un DLQ. • Los mensajes que se transfieren automáticamente a un DLQ después de superar el límite no `maxReceiveCount` se incluyen en esta métrica. • Como resultado, `NumberOfMessagesSent` puede ser inferior a, `NumberOfMessagesReceived` especialmente si las políticas de redrive están trasladando muchos mensajes a un segundo plano DLQs .

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`SentMessageSize` [1]	El tamaño de los mensajes enviados correctamente a la cola.	Bytes	No se emite hasta que se envíe al menos un mensaje.	• Esta métrica no aparecerá en la CloudWatch consola hasta que la cola reciba su primer mensaje. • Utilice esta métrica para realizar un seguimiento del tamaño de cada mensaje en bytes. Esto resulta útil para analizar las tendencias de la carga útil o estimar el costo del rendimiento. • El tamaño máximo de mensaje para SQS es de 1 MiB.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateNumberOfNoisyGroups`	El número de grupos de mensajes que se consideran ruidosos en una cola regular. Un grupo de mensajes ruidoso representa a un inquilino vecino ruidoso de una cola de varios usuarios.	Recuento	Si la cola está activa, se informa de un valor no negativo.	• Ayuda a identificar posibles problemas de vecinos ruidosos en entornos con varios usuarios mediante el seguimiento de los grupos de mensajes que consumen recursos desproporcionados. • Utilice esta métrica para configurar alarmas que se activen cuando el número de grupos ruidosos supere el umbral aceptable, lo que indica posibles problemas de equilibrio en las colas.
`ApproximateNumberOfMessagesVisibleInQuietGroups`	El número de mensajes visibles, excluyendo los mensajes de grupos de mensajes ruidosos.	Recuento	Si la cola está activa, se informa de un valor no negativo.	• Proporciona visibilidad de la acumulación de colas para los grupos de mensajes con velocidad estándar, excluyendo los mensajes de vecinos ruidosos. • Ayuda a identificar el verdadero retraso en el procesamiento de los grupos de mensajes típicos al filtrar el impacto de los vecinos ruidosos.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateNumberOfMessagesNotVisibleInQuietGroups`	El número de mensajes en movimiento, excluidos los mensajes de grupos de mensajes ruidosos.	Recuento	Si la cola está activa, se indica un valor no negativo.	• Realiza un seguimiento de los mensajes en movimiento (que se están procesando pero aún no se han eliminado) de grupos de mensajes que se comportan bien. • Utilice esta métrica para supervisar el rendimiento del procesamiento de los grupos de mensajes normales y detectar los cuellos de botella en el procesamiento que no se deben a vecinos ruidosos.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateNumberOfMessagesDelayedInQuietGroups`	El número de mensajes, excepto los mensajes de grupos de mensajes ruidosos que se retrasan y no están disponibles para su lectura inmediata. Los mensajes retrasados se producen cuando la cola está configurada como cola de retraso o cuando se envía un mensaje con un parámetro de retraso.	Recuento	Si la cola está activa, se informa de un valor no negativo.	• Ayuda a controlar el retraso en la acumulación de mensajes de los grupos de mensajes con patrones de rendimiento normales o esperados (a diferencia de los grupos ruidosos o de gran volumen) • Útil para comprender los requisitos de procesamiento futuros y la planificación de la capacidad para cargas de trabajo típicas.

Métrica	Descripción	Unidades	Informar sobre el comportamiento	Notas clave
`ApproximateAgeOfOldestMessageInQuietGroups`	La antigüedad del mensaje no eliminado más antiguo de la cola, excluidos los mensajes de grupos de mensajes ruidosos.	Segundos	Si la cola está activa, se indica un valor no negativo.	• Se utiliza para supervisar el cumplimiento del SLA y detectar cuellos de botella en el procesamiento en grupos de mensajes con patrones de rendimiento normales o esperados (a diferencia de los grupos de mensajes ruidosos o de gran volumen que, de otro modo, podrían sesgar la métrica). • Utilice esta métrica para configurar alarmas cuando se agoten los tiempos de espera del procesamiento de los mensajes e ignorar los mensajes envejecidos artificialmente procedentes de vecinos ruidosos.

[1] Estas métricas reflejan la actividad a nivel del sistema y pueden incluir reintentos, duplicados o mensajes retrasados. No utilices recuentos brutos para estimar el estado de las colas en tiempo real sin tener en cuenta el comportamiento del ciclo de vida de los mensajes.

Colas de cartas muertas () y métricas DLQs CloudWatch

Al trabajar con ellas DLQs, es importante entender cómo se comportan las métricas de Amazon SQS:

- **`NumberOfMessagesSent`**— Esta métrica se comporta de forma diferente para: DLQs
  - Envío manual: esta métrica captura los mensajes que se envían manualmente a un DLQ.

- Retransmisión automática: esta métrica no captura los mensajes que se mueven automáticamente a un DLQ debido a errores de procesamiento. Como resultado, es posible que las `NumberOfMessagesReceived` métricas `NumberOfMessagesSent` y muestren discrepancias para. DLQs
- Métrica recomendada para DLQs: para monitorear el estado de una DLQ, usa la `ApproximateNumberOfMessagesVisible` métrica. Esta métrica indica el número de mensajes actualmente disponibles para su procesamiento en el DLQ.

Colas y métricas justas CloudWatch

Cuando utiliza colas justas, Amazon SQS emite las siguientes métricas adicionales:

- `ApproximateNumberOfNoisyGroups`
- `ApproximateNumberOfMessagesVisibleInQuietGroups`
- `ApproximateNumberOfMessagesNotVisibleInQuietGroups`
- `ApproximateNumberOfMessagesDelayedInQuietGroups`
- `ApproximateAgeOfOldestMessageInQuietGroups`

> **Note**
>
> Cada `QuietGroup` métrica es un subconjunto de la métrica estándar equivalente a nivel de cola, pero excluye los mensajes de grupos `Approximate` vecinos ruidosos.

Grupos ruidosos

Un grupo de mensajes ruidoso representa a un inquilino vecino ruidoso de una cola de varios inquilinos.

Grupos silenciosos

Grupos de mensajes, excluidos los grupos ruidosos.

Observar el comportamiento de SQS en las colas justas

Para monitorizar el efecto de las colas de espera en las ferias de Amazon SQS, puede comparar las `Approximate..InQuietGroups` métricas con las métricas estándar a nivel de cola. Durante

los picos de tráfico de un usuario específico, las métricas generales a nivel de cola pueden revelar el aumento de los atrasos o la antigüedad de los mensajes. Sin embargo, si analizas los grupos silenciosos de forma aislada, puedes identificar que la mayoría de los grupos o inquilinos de mensajes que no son ruidosos no se ven afectados y proporcionar una estimación del número total de grupos de mensajes afectados.

Si bien estas nuevas métricas ofrecen una buena visión general del comportamiento de Amazon SQS en las colas justas, puede resultar útil saber qué inquilino específico está causando la carga. Amazon CloudWatch Contributor Insights te permite ver métricas sobre los N principales colaboradores, el número total de colaboradores únicos y su uso. Esto resulta especialmente útil en situaciones en las que se trata de miles de inquilinos que, de otro modo, generarían datos de gran cardinalidad (y costes) al emitir las métricas tradicionales.

Para ver un ejemplo de la configuración de supervisión de las colas de espera en ferias, consulte el ejemplo siguiente. GitHub

Dimensiones para las métricas de Amazon SQS

Las métricas de Amazon SQS se CloudWatch utilizan en una sola dimensión:. **`QueueName`** Todos los datos de las métricas se agrupan y filtran por el nombre de la cola.

Consejos de supervisión

Supervise el SQS de forma eficaz mediante métricas y CloudWatch alarmas clave para detectar las colas acumuladas, optimizar el rendimiento y mantenerse dentro de los límites del servicio.

- Establezca CloudWatch alarmas en función del crecimiento `ApproximateNumberOfMessagesVisible` de los atrasos para atrapar.
- `NumberOfEmptyReceives`Supervise para ajustar la frecuencia de las encuestas y reducir el costo de la API.
- Úselo `ApproximateNumberOfGroupsWithInflightMessages` en las colas FIFO para diagnosticar los límites de rendimiento.
- Revise las cuotas de SQS para comprender los umbrales métricos y los límites de servicio.

Validación de la conformidad de Amazon SQS

Para saber si una Servicio de AWS está dentro del ámbito de aplicación de programas de cumplimiento específicos, consulte Servicios de AWS Alcance por programa de cumplimiento

Servicios de AWS de cumplimiento y elija el programa de cumplimiento que le interese. Para obtener información general, consulte Programas de AWS cumplimiento > Programas AWS .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact .

Su responsabilidad de cumplimiento al Servicios de AWS utilizarlos viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- Cumplimiento de seguridad y gobernanza: en estas guías se explican las consideraciones de arquitectura y se proporcionan pasos para implementar las características de seguridad y cumplimiento.
- Referencia de servicios válidos de HIPAA: muestra una lista con los servicios válidos de HIPAA. No todos Servicios de AWS cumplen con los requisitos de la HIPAA.
- AWS Recursos de de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su industria y ubicación.
- AWS Guías de cumplimiento para clientes: comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. Las guías resumen las mejores prácticas para garantizar la seguridad Servicios de AWS y orientan los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST), el Consejo de Normas de Seguridad del Sector de Tarjetas de Pago (PCI) y la Organización Internacional de Normalización (ISO)).
- Evaluación de los recursos con reglas en la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.
- AWS Security Hub— Esto Servicio de AWS proporciona una visión completa del estado de su seguridad interior AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulte la Referencia de controles de Security Hub.
- Amazon GuardDuty: Servicio de AWS detecta posibles amenazas para sus cargas de trabajo Cuentas de AWS, contenedores y datos mediante la supervisión de su entorno para detectar actividades sospechosas y maliciosas. GuardDuty puede ayudarlo a cumplir con varios requisitos de conformidad, como el PCI DSS, al cumplir con los requisitos de detección de intrusiones exigidos por ciertos marcos de cumplimiento.

- AWS Audit Manager— Esto le Servicio de AWS ayuda a auditar continuamente su AWS uso para simplificar la gestión del riesgo y el cumplimiento de las normativas y los estándares del sector.

Resiliencia en Amazon SQS

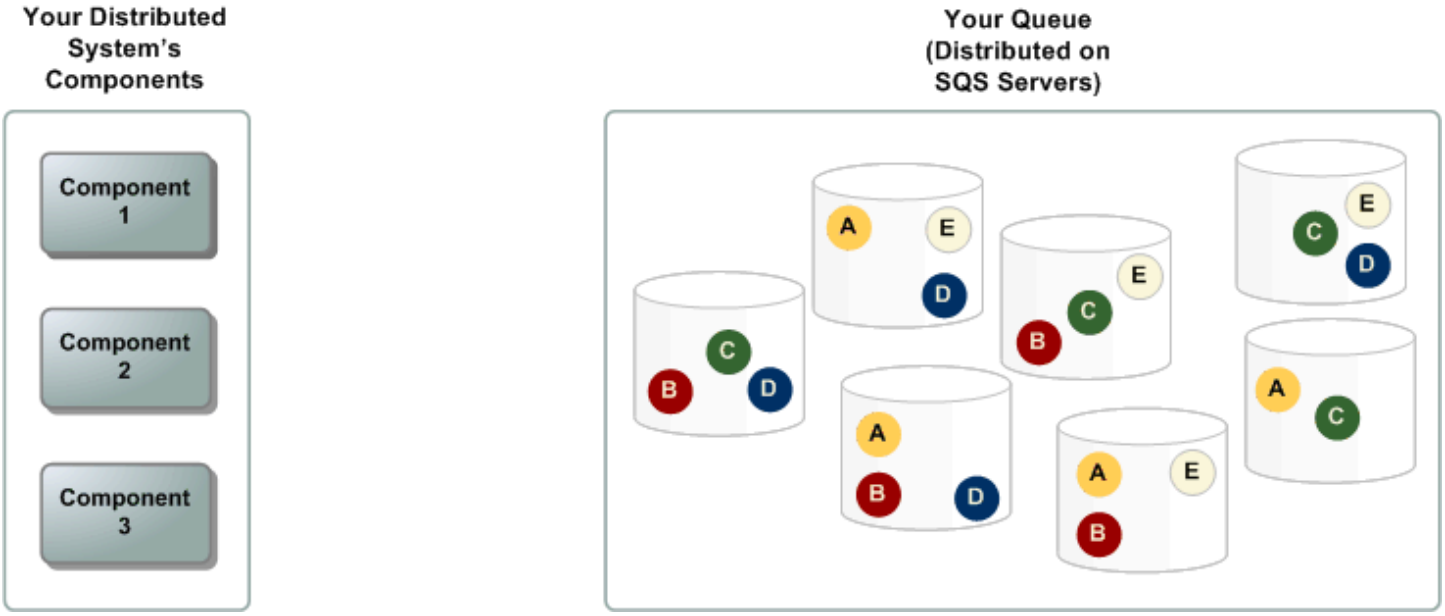
La infraestructura AWS global se basa en AWS regiones y zonas de disponibilidad. AWS Las regiones proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos. Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte AWS Infraestructura global.

Además de la infraestructura AWS global, Amazon SQS ofrece colas distribuidas.

Colas distribuidas

Un sistema de mensajería distribuida consta de tres partes principales: los componentes del sistema distribuido, la cola (distribuida en los servidores Amazon SQS) y los mensajes de la cola.

En el siguiente escenario, el sistema tiene varios productores (componentes que envían mensajes a la cola) y consumidores (componentes que reciben mensajes de la cola). La cola (que contiene los mensajes A a E) almacena de forma redundante los mensajes en varios servidores de Amazon SQS.



Seguridad de la infraestructura en Amazon SQS

Como servicio gestionado, Amazon SQS está protegido por los procedimientos de seguridad de red AWS global descritos en el documento técnico Amazon Web Services: Overview of Security Processes.

Las acciones de API AWS publicadas se utilizan para acceder a Amazon SQS a través de la red. Los clientes deben ser compatibles con Transport Layer Security (TLS) 1.2 o una versión posterior. Los clientes también deben ser compatibles con conjuntos de cifrado con confidencialidad directa total (PFS) tales como Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE).

Debe firmar las solicitudes mediante un ID de clave de acceso y una clave de acceso secreta asociada a una entidad principal de IAM. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Puede llamar a estas acciones de la API desde cualquier ubicación de red, pero Amazon SQS admite políticas de acceso basadas en recursos, que pueden incluir restricciones en función de la dirección IP del origen. También puede utilizar las políticas de Amazon SQS para controlar el acceso desde puntos de enlace de Amazon VPC específicos o específicos. VPCs Esto aísla eficazmente el acceso a la red a una cola de Amazon SQS determinada únicamente de la VPC específica de la red. AWS Para obtener más información, consulte Ejemplo 5: denegar el acceso si no es desde un punto de enlace de la VPC.

Prácticas recomendadas de seguridad para Amazon SQS

AWS proporciona numerosas funciones de seguridad para Amazon SQS, que debe revisar en el contexto de su propia política de seguridad. A continuación, se indican las prácticas recomendadas de seguridad preventiva para Amazon SQS.

> **Note**
>
> La orientación de implementación específica que se proporciona corresponde a casos de uso e implementaciones habituales. Le sugerimos que consulte estas prácticas recomendadas en el contexto de su caso de uso, arquitectura y modelo de amenaza concretos.

Comprobación de que las colas no sean accesibles de forma pública

A menos que exija explícitamente a cualquier persona de Internet que pueda leer o escribir en su cola de Amazon SQS, debe asegurarse de que su cola no sea de acceso público (a la que puedan acceder todos los habitantes del mundo o cualquier usuario autenticado). AWS

- Evite la creación de políticas con `Principal` establecido en `""`.
- Evite usar un carácter comodín (`*`). En su lugar, designe a un usuario o usuarios específicos.

Implementación del acceso a los privilegios mínimos

Cuando concede permisos, decide quién los recibe, para qué colas son los permisos y las acciones específicas de la API que desea permitir en estas colas. La implementación de los privilegios mínimos es importante para reducir los riesgos de seguridad y disminuir el efecto de errores o intenciones maliciosas.

Siga el consejo de seguridad estándar de concesión del privilegio mínimo. Es decir, conceda solo los permisos necesarios para realizar una tarea específica. Puede efectuar esta impresora con una combinación de políticas de seguridad.

Amazon SQS utiliza el modelo productor-consumidor, que requiere tres tipos de acceso a la cuenta de usuario:

- Administradores: acceso a la creación, modificación y eliminación de colas. Los administradores también controlan las políticas de cola.
- Productores: acceso al envío de mensajes a las colas.
- Consumidores: acceso a la recepción y eliminación de mensajes de las colas.

Para obtener más información, consulte las siguientes secciones:

- Identity and Access Management en Amazon SQS
- Permisos de la API de Amazon SQS: referencia de acciones y recursos
- Uso de políticas personalizadas con el lenguaje de la política de acceso de Amazon SQS

Utilice funciones de IAM para aplicaciones y AWS servicios que requieren acceso a Amazon SQS

Para que aplicaciones o AWS servicios como Amazon puedan acceder EC2 a las colas de Amazon SQS, deben utilizar AWS credenciales válidas en sus AWS solicitudes de API. Como estas credenciales no se rotan automáticamente, no debe almacenar AWS las credenciales directamente en la aplicación o la instancia. EC2

Debe utilizar un rol de IAM para administrar de manera temporal credenciales para las aplicaciones o los servicios que necesiten acceder a Amazon SQS. Cuando usas un rol, no tienes que distribuir credenciales de larga duración (como un nombre de usuario, contraseña y claves de acceso) a una EC2 instancia o AWS servicio como AWS Lambda. En su lugar, el rol proporciona permisos temporales que las aplicaciones pueden usar cuando realizan llamadas a otros AWS recursos.

Para obtener más información, consulte Roles de IAM y Situaciones habituales con los roles: usuarios, aplicaciones y servicios en la Guía del usuario de IAM.

Implementación del cifrado en el servidor

Para mitigar los problemas de fuga de datos, utilice el cifrado en reposo para cifrar sus mensajes mediante una clave almacenada en una ubicación distinta de la ubicación en la que se almacenan los mensajes. Con el cifrado del lado del servidor (SSE), se proporciona cifrado de datos en reposo. Amazon SQS cifra sus datos en el nivel de mensaje cuando los almacena y descifra los mensajes para usted cuando accede a ellos. El SSE utiliza claves administradas en AWS Key Management Service. Siempre que autentique su solicitud y tenga permiso de acceso, no existe diferencia alguna entre obtener acceso a las colas cifradas y sin cifrar.

Para obtener más información, consulte Cifrado en reposo en Amazon SQS y Administración de claves de Amazon SQS.

Aplicación del cifrado de los datos en tránsito

Sin HTTPS (TLS), un atacante basado en la red puede espiar el tráfico de la red o manipularlo mediante un ataque como. man-in-the-middle Permitir solo las conexiones cifradas a través de HTTPS (TLS) mediante la condición `aws:SecureTransport` en la política de colas para forzar que las solicitudes utilicen SSL.

Consideración del uso de puntos de conexión de VPC para obtener acceso a Amazon SQS

Si tiene colas con las que debe poder interactuar pero que no deben estar expuestas a Internet, utilice los puntos de enlace de la VPC para poner en la cola el acceso solo a los hosts dentro de una VPC concreta. Puede utilizar las políticas de colas para controlar el acceso a las colas desde puntos de enlace de Amazon VPC específicos o desde puntos específicos. VPCs

Los puntos de conexión de VPC de Amazon SQS brindan dos maneras de controlar el acceso a los mensajes:

- Puede controlar qué solicitudes, usuarios o grupos obtienen acceso a través de un punto de conexión de la VPC específico.
- Puedes controlar qué puntos de VPCs conexión o de VPC tienen acceso a tu cola mediante una política de colas.

Para obtener más información, consulte Puntos de conexión de Amazon Virtual Private Cloud para Amazon SQS y Creación de una política de punto de conexión de VPC para Amazon SQS.

Recursos de Amazon SQS relacionados

En la tabla siguiente se enumeran todos los recursos relacionados que podrían resultarle útiles cuando trabaje con este servicio.

Recurso	Descripción
Referencia de la API de Amazon Simple Queue Service	Descripciones de las acciones, parámetros y tipos de datos de la , así como una lista de errores que el servicio devuelve.
Amazon SQS en la Referencia de los comandos de la AWS CLI	Descripciones de los AWS CLI comandos que puede utilizar para trabajar con colas.
Regiones y puntos de enlace	Información acerca de las regiones y los puntos de conexión de Amazon SQS
Página del producto	Página web principal con información acerca de Amazon SQS.
Foro de debate	Foro de la comunidad donde los desarrolladores tratan aspectos técnicos relacionados con Amazon SQS.
AWS Información de soporte premium	La página web principal para obtener información sobre AWS Premium Supportone-on-one, un canal de soporte de respuesta rápida que le ayuda a crear y ejecutar aplicaciones en servicios de AWS infraestructura.

Historial de documentación

En la siguiente tabla se describen los cambios importantes realizados en la Guía para desarrolladores de Amazon Simple Queue Service desde enero de 2019. Para recibir notificaciones sobre las actualizaciones de esta documentación, suscríbase a la fuente RSS.

A veces, las funciones del servicio se implementan de forma gradual en AWS las regiones en las que el servicio está disponible. Actualizamos esta documentación solo para la primera versión. No proporcionamos información sobre la disponibilidad en regiones ni anunciamos lanzamientos posteriores en regiones. Para obtener información sobre la disponibilidad regional de las funciones del servicio y suscribirse a las notificaciones sobre actualizaciones, consulte ¿Qué hay de nuevo en el mercado? AWS .

Cambio	Descripción	Fecha
Soporte para Fair Queues	Se ha añadido la compatibilidad con colas justas para las colas estándar.	21 de julio de 2025
Se agregó soporte para terminales (IPv4 y IPv6) de doble pila	Amazon SQS ahora admite puntos de enlace de doble pila (IPv4 y IPv6), lo que permite acceder a las colas a través de ambos protocolos IP.	17 de abril de 2025
CloudTrail integración para todos los Amazon SQS APIs	CloudTrail integración agregada para todos los Amazon SQS APIs.	10 de enero de 2025
SQSUnlockQueuePolicy	Amazon SQS ha añadido una nueva política AWS gestionada destinada `SQSUnlockQueuePolicy` a desbloquear una cola y eliminar una política de colas mal configurada que impide a todos los	15 de noviembre de 2024

	directores acceder a una cola de Amazon SQS.	
AWS kms:Decrypt	Amazon SQS ya no requiere el permiso `kms:Decrypt` para la API `SendMessage`. Los clientes ahora solo necesitan el permiso `kms:GenerateDataKey` de la clave de KMS utilizada para cifrar la cola, pero siguen necesitando el permiso `kms:Decrypt` para llamar a `ReceiveMessage`.	24 de julio de 2024
Actualización de métricas FIFO	Se ha añadido soporte para `NumberOfDeduplicatedSentMessages` y `ApproximateNumberOfGroupsWithInflightMessages` a las métricas FIFO de Amazon SQS.	3 de julio de 2024
`ListQueueTags` acción admitida en la política SQSRead OnlyAccess gestionada de Amazon	La política SQSRead OnlyAccess gestionada por Amazon `ListQueueTags` permite recuperar todas las etiquetas asociadas a una cola de Amazon SQS específica.	2 de mayo de 2024
Protocolo AWS JSON	Realice solicitudes a la API mediante el protocolo AWS JSON.	27 de julio de 2023

Nueva sección en la que se describen las políticas AWS gestionadas de Amazon SQS y las actualizaciones de estas políticas	Amazon SQS ha agregado una nueva acción que permite enumerar las tareas de movimiento de mensajes más recientes (hasta diez) en una cola de origen específica. Esta acción está asociada a la operación de la API `ListMessageMoveTasks` .	7 de junio de 2023
Redrive de colas de cartas sin salida mediante APIs	Configure los redirecci onamientos de colas con letra muerta mediante Amazon SQS. APIs	7 de junio de 2023
ABAC para Amazon SQS	Control de acceso basado en atributos (ABAC) mediante etiquetas de cola para obtener permisos de acceso flexibles y escalables.	10 de noviembre de 2022
Aumentos del límite de alto rendimiento de FIFO	Aumento de las cuotas predeterminadas para el modo de alto rendimiento FIFO en regiones comerciales, además de la optimización de documentos de alto rendimien to FIFO.	20 de octubre de 2022
El cifrado del servidor (SSE) está disponible de forma predeterminada	El cifrado del servidor (SSE) utiliza el cifrado propiedad de SQS (SSE-SQS) de forma predeterminada.	26 de septiembre de 2022

Está disponible la compatibilidad con la protección de suplentes confusos de Amazon SQS	La protección del suplente confuso permite especificar nuevos encabezados en sus solicitudes, que se comprueban con las condiciones de la política de KMS cuando se utiliza el SSE administrado por Amazon SQS.	29 de diciembre de 2021
SSE administrado está disponible	SSE administrado por Amazon SQS (SSE-SQS) es un cifrado del servidor administrado que utiliza claves de cifrado propiedad de Amazon SQS para proteger los datos confidenciales enviados a través de las colas de mensajes.	23 de noviembre de 2021
Está disponible el redireccionamiento de cola de mensajes fallidos	Amazon SQS admite el redireccionamiento de cola de mensajes fallidos para colas estándar.	10 de noviembre de 2021
Está disponible el alto rendimiento para mensajes en colas FIFO	El alto rendimiento de las colas FIFO de Amazon SQS proporciona un mayor número de transacciones por segundo (TPS) para los mensajes en colas FIFO. Para obtener información sobre las cuotas de rendimiento, consulte Cuotas relacionadas con los mensajes.	27 de mayo de 2021

Está disponible el alto rendimiento para mensajes en colas FIFO en versión preliminar	El alto rendimiento de las colas FIFO de Amazon SQS se encuentra en versión preliminar y está sujeto a cambios. Esta característica proporciona un mayor número de transacciones por segundo (TPS) para los mensajes en colas FIFO. Para obtener información sobre las cuotas de rendimiento, consulte Cuotas relacionadas con los mensajes.	17 de diciembre de 2020
Nuevo diseño de la consola de Amazon SQS	Para simplificar los flujos de trabajo de desarrollo y producción, la consola de Amazon SQS cuenta con una nueva experiencia de usuario.	8 de julio de 2020
Amazon SQS admite la paginación para ListQueues y listDeadLetter SourceQueues	Puedes especificar el número máximo de resultados que se devolverán de una lista de listDeadLetterSourceQueuesListQueues o de una solicitud.	22 de junio de 2020
Amazon SQS admite métricas de CloudWatch Amazon de 1 minuto en AWS todas las regiones, excepto en las regiones (EE. AWS GovCloud UU.)	La CloudWatch métrica de un minuto para Amazon SQS está disponible en todas las regiones, excepto AWS GovCloud (US) en las regiones.	9 de enero de 2020

Amazon SQS admite métricas de 1 minuto CloudWatch	La CloudWatch métrica de un minuto para Amazon SQS solo está disponible actualmente en las siguientes regiones: EE. UU. Este (Ohio), Europa (Irlanda), Europa (Estocolmo) y Asia Pacífico (Tokio).	25 de noviembre de 2019
AWS Lambda están disponibles activadores para las colas FIFO de Amazon SQS	Puede configurar los mensajes que llegan a una cola FIFO como desencadenadores de una función de Lambda.	25 de noviembre de 2019
El cifrado del servidor (SSE) para Amazon SQS está disponible en las regiones de China	SSE para Amazon SQS está disponible en las regiones de China.	13 de noviembre de 2019
Las colas FIFO están disponibles en la región de Medio Oriente (Baréin)	Las colas FIFO están disponibles en la región de Medio Oriente (Baréin).	10 de octubre de 2019
Los puntos de enlace de Amazon Virtual Private Cloud (Amazon VPC) para Amazon SQS están disponibles en las regiones (EE. UU. Este) y AWS GovCloud (EE. UU. Oeste) AWS GovCloud	Puede enviar mensajes a sus colas de Amazon SQS desde Amazon VPC en las regiones (EE. UU. Este) y AWS GovCloud (EE. UU. Oeste). AWS GovCloud	5 de septiembre de 2019

Amazon SQS permite solucionar problemas de colas mediante el uso de atributos del sistema de AWS X-Ray mensajes	Puede solucionar los problemas de los mensajes que pasan a través de las colas de Amazon SQS utilizando X-Ray. Esta versión agrega el parámetro de solicitud `MessageSystemAttribute` (que le permite enviar encabezados de rastreo X-Ray a través de Amazon SQS) a las operaciones de API `SendMessage` y `SendMessageBatch`, el atributo `AWSTraceHeader` a la operación de API `ReceiveMessage` y el tipo de datos `MessageSystemAttributeValue`.	28 de agosto de 2019
Puede etiquetar las colas de Amazon SQS en el momento de su creación	Puede utilizar una sola llamada a la API de Amazon SQS, una función AWS del SDK o un comando AWS Command Line Interface (AWS CLI) para crear una cola y especificar sus etiquetas de forma simultánea. Además, Amazon SQS admite las claves `aws:TagKeys` y `aws:RequestTag` AWS Identity and Access Management (IAM).	22 de agosto de 2019

El cliente de colas temporales de Amazon SQS ya está disponible	Las colas temporales le ayudan a ahorrar tiempo de desarrollo y costes de implementación al utilizar patrones de mensajes comunes como, por ejemplo, solicitud-respuesta. Puede utilizar el Cliente de colas temporales para crear colas temporales de alto rendimiento, rentables y administradas por la aplicación.	25 de julio de 2019
SSE para Amazon SQS está disponible en la región AWS GovCloud (EE. UU. Este)	El cifrado del lado del servidor (SSE) para Amazon SQS está disponible en AWS GovCloud la región (EE.UU. Este).	20 de junio de 2019
Las colas FIFO están disponibles en las regiones de Asia Pacífico (Hong Kong), China (Beijing), (EE. UU. Este) y AWS GovCloud AWS GovCloud (EE. UU. Oeste)	Las colas FIFO están disponibles en las regiones de Asia Pacífico (Hong Kong), China (Beijing), (EE. UU. Este) y AWS GovCloud AWS GovCloud (EE. UU. Oeste).	15 de mayo de 2019
Las políticas de punto de conexión de Amazon VPC están disponibles para Amazon SQS	Puede crear políticas de punto de conexión de Amazon VPC para Amazon SQS.	4 de abril de 2019
Las colas FIFO están disponibles en las regiones de Europa (Estocolmo) y China (Ningxia)	Las colas FIFO están disponibles en las regiones de Europa (Estocolmo) y China (Ningxia).	14 de marzo de 2019

Las colas FIFO están disponibles en todas las regiones en las que está disponible Amazon SQS	Las colas FIFO están disponibles en las regiones de Este de EE. UU. (Ohio), Este de EE. UU. (Norte de Virginia), Oeste de EE. UU. (Norte de California), Oeste de EE. UU. (Oregón), Asia-Pacífico (Bombay), Asia-Pacífico (Seúl), Asia-Pacífico (Singapur), Asia-Pacífico (Sídney), Asia-Pacífico (Tokio), Canadá (centro), Europa (Fráncfort), Europa (Irlanda), Europa (Londres), Europa (París) y América del Sur (São Paulo).	7 de febrero de 2019

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.