Guide du développeur

Amazon Simple Queue Service

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

Amazon Simple Queue Service: Guide du développeur

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

Les marques commerciales et la présentation commerciale d’Amazon ne peuvent pas être utilisées en relation avec un produit ou un service extérieur à Amazon, d’une manière susceptible d’entraîner une confusion chez les clients, ou d’une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d’Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce qu'Amazon SQS ? ... 1
Avantages offerts par l'utilisation d'Amazon SQS ... 1
Architecture basique ... 2
Files d'attente distribuées ... 2
Cycle de vie des messages ... 2
Différences entre Amazon SQS, Amazon MQ et Amazon SNS ... 4
Premiers pas ... 6
Configuration ... 6
Étape 1 : créer un utilisateur Compte AWS et IAM ... 6
Étape 2 : Accorder un accès par programmation ... 8
Étape 3 : Préparation à l'utilisation de l'exemple de code ... 10
Étapes suivantes ... 11
Comprendre la console Amazon SQS ... 11
Types de files d'attente ... 12
Implémentation de systèmes de demande-réponse dans Amazon SQS ... 16
Création d'une file d'attente standard ... 16
Création d'une file d'attente ... 16
Envoi d'un message à l'aide d'une file d'attente standard ... 18
Création d'une file d'attente ... 19
Créer une file d'attente ... 19
Envoi d'un message à l'aide d'une file d'attente FIFO ... 22
Tâches courantes ... 23
Gestion d'une file d'attente ... 25
Modification d'une file d'attente ... 25
Réception et suppression d'un message ... 26
Confirmer qu'une file d'attente est vide ... 27
Suppression d'une file d'attente ... 29
Purge d'une file d'attente ... 29
Files d'attente standard ... 31
Livraison via Amazon SQS at-least-once ... 32
Identificateurs de files d'attente et de messages ... 32
Identifiants pour les files d'attente standard ... 32
Files d'attente FIFO ... 34
Termes clés de la file d'attente FIFO ... 35

Logique de remise FIFO ... 37
Envoi de messages ... 37
Réception de messages ... 38
Multiples nouvelles tentatives ... 39
Remarques supplémentaires sur le comportement du FIFO ... 39
Exemples pour mieux comprendre ... 40
Traitement en une seule fois ... 41
Passage d'une file d'attente standard à une file d'attente FIFO ... 41
Comportement de la file d'attente FIFO et de la simultanéité Lambda ... 43
Regroupement de messages de file d'attente FIFO ... 43
Simultanéité Lambda avec les files d'attente FIFO ... 44
Exemple de cas d'utilisation ... 44
Débit élevé pour les files d'attente FIFO ... 45
Cas d'utilisation ... 45
Partitions et distribution des données ... 46
Permettre un débit élevé pour les files d'attente FIFO ... 49
Identificateurs de files d'attente et de messages ... 50
Identifiants pour les files d'attente FIFO ... 32
Identifiants supplémentaires pour les files d'attente FIFO ... 51
Quotas ... 53
Quotas de files d'attente FIFO ... 53
Quotas Amazon SQS ... 53
Quotas de queue standard ... 55
Quotas de messages ... 57
Quotas de politiques ... 63
Fonctionnalités et capacités ... 64
Files d'attente de lettres mortes ... 64
Utilisation de politiques pour les files d'attente de lettres mortes ... 65
Comprendre les périodes de rétention des messages pour les files d'attente de lettres mortes ... 65
Configuration d'une file d'attente de lettre morte ... 66
Configuration d'une redirection de file d'attente de lettres mortes ... 67
CloudTrail exigences de mise à jour et d'autorisation ... 77
Création d'alarmes pour les files d'attente de lettres mortes à l'aide d'Amazon CloudWatch ... 82
Métadonnées des messages pour Amazon SQS ... 82

Attributs de message ... 83
Attributs de système de message ... 87
Ressources requises pour traiter les messages ... 87
Pagination des files d'attente ... 88
Balises d'allocation des coûts ... 89
Attente active de courte durée et de longue durée ... 90
Consommation des messages à l'aide de l'interrogation courte ... 90
Consommation des messages à l'aide de la recherche prolongée ... 91
Différences entre les interrogations courtes et longues ... 92
Délai de visibilité ... 92
Cas d'utilisation du délai de visibilité ... 93
Réglage et ajustement du délai de visibilité ... 94
Messages et quotas en vol ... 94
Comprendre le délai d'expiration de visibilité dans les files d'attente standard et FIFO ... 95
Gestion des défaillances ... 95
Modification et fin du délai de visibilité ... 96
Bonnes pratiques ... 96
Des files d'attente équitables ... 97
Différence avec les files d'attente FIFO ... 99
Utiliser des files d'attente équitables ... 99
Indicateurs relatifs aux files d'attente CloudWatch équitables ... 100
Files d'attente à retardement ... 100
Files d'attente temporaires ... 102
Files d'attente virtuelles ... 102
Modèle de messagerie demande-réponse (files d'attente virtuelles) ... 104
Exemple de scénario : Traitement d'une demande de connexion ... 105
Nettoyage des files d'attente ... 107
Temporisateurs de messages ... 108
Accès aux EventBridge tuyaux ... 108
Gestion de messages volumineux ... 110
Utilisation de la bibliothèque client étendue pour Java ... 110
Utilisation de la bibliothèque client étendue pour Python ... 117
Configuration d'Amazon SQS ... 121
ABAC pour Amazon SQS ... 121
Qu'est-ce que le contrôle d'accès basé sur les attributs (ABAC) ? ... 121
Pourquoi utiliser l'ABAC dans Amazon SQS ? ... 122

Identification pour le contrôle d'accès ... 123
Création d'utilisateurs IAM et de files d'attente Amazon SQS ... 123
Test du contrôle d'accès basé sur les attributs ... 127
Configuration des paramètres de file d'attente ... 128
Configuration d'une politique d'accès ... 130
Configuration de SSE-SQS pour une file d'attente ... 131
Configuration de SSE-KMS pour une file d'attente ... 133
Configuration de balises pour une file d'attente ... 134
Abonnement d'une file d'attente à une rubrique ... 135
Abonnements multi-comptes ... 136
Abonnements interrégionaux ... 137
Configuration d'un déclencheur Lambda ... 137
Prérequis ... 138
Automatiser les notifications à l'aide de EventBridge ... 139
Attributs de message ... 140
Bonnes pratiques ... 142
Gestion des erreurs et messages problématiques ... 142
Gestion des erreurs de demande dans Amazon SQS ... 142
Capture des messages problématiques dans Amazon SQS ... 143
Configuration de la conservation des files d'attente de lettres mortes dans Amazon SQS ... 143
Déduplication et regroupement des messages ... 144
Éviter le traitement incohérent des messages dans Amazon SQS ... 144
Utilisation de l'ID de déduplication du message ... 144
Utilisation de l'ID de groupe de messagerie ... 147
Utilisation de l'ID de tentative de demande de réception ... 149
Traitement des messages et chronométrage ... 150
Traitement des messages en temps opportun dans Amazon SQS ... 150
Configuration d'un long sondage dans Amazon SQS ... 151
Utilisation du mode d'interrogation approprié dans Amazon SQS ... 152
Exemples de SDK Java ... 153
Utilisation du chiffrement côté serveur ... 153
Ajout du SSE à une file d'attente existante ... 153
Désactivation du SSE pour une file d'attente ... 155
Création d'une file d'attente avec le SSE ... 155
Récupération des attributs SSE ... 156
Configuration des identifications ... 156

Établissement d'une liste de balises ... 156
Ajout ou mise à jour de balises ... 157
Suppression de balises ... 158
Envoi d'attributs de message ... 158
Définition des attributs ... 158
Envoi d'un message avec des attributs ... 160
En utilisant APIs ... 162
Effectuer des demandes d'API de requête à l'aide du protocole AWS JSON ... 163
Constitution d'un point de terminaison ... 163
Envoi de requête POST ... 164
Interprétation des réponses de l'API JSON Amazon SQS ... 165
Protocole JSON Amazon SQS AWS FAQs ... 166
Effectuer des demandes d'API de requête à l'aide du protocole de AWS requête ... 170
Constitution d'un point de terminaison ... 170
Envoi de requête GET ... 171
Envoi de requête POST ... 164
Interprétation des réponses de l'API XML Amazon SQS ... 173
Authentification des requêtes ... 174
Processus d'authentification de base avec HMAC-SHA ... 175
Partie 1 : Demande de l'utilisateur ... 176
Partie 2 : La réponse de AWS ... 177
Actions de traitement par lots ... 178
Actions de message par lots ... 179
Activation de la mise en mémoire tampon côté client et du traitement par lots des demandes avec Amazon SQS ... 180
Augmenter le débit grâce à la mise à l'échelle horizontale et au traitement par lots d'actions avec Amazon SQS ... 192
Travailler avec AWS SDKs ... 205
Utilisation de JMS ... 207
Prérequis ... 207
Utilisation de la bibliothèque de messagerie Java ... 208
Création d'une connexion JMS ... 209
Création d'une file d'attente Amazon SQS ... 210
Envoi de messages de façon synchrone ... 211
Réception des messages de façon synchrone ... 212
Réception des messages de façon asynchrone ... 214

Utilisation du mode de reconnaissance du client ... 215
Utilisation du mode de reconnaissance indépendamment de l'ordre de réception ... 216
Utilisation du client JMS avec d'autres clients Amazon SQS ... 217
Exemples Java fonctionnels pour l'utilisation de JMS avec des files d'attente standard ... 218
ExampleConfiguration.java ... 218
TextMessageSender.java ... 221
SyncMessageReceiver.java ... 223
AsyncMessageReceiver.java ... 225
SyncMessageReceiverClientAcknowledge.java ... 227
SyncMessageReceiverUnorderedAcknowledge.java ... 230
SpringExampleConfiguration.xml ... 234
SpringExample.java ... 235
ExampleCommon.java ... 238
Implémentations JMS 1.1 prises en charge ... 239
Interfaces courantes prises en charge ... 239
Types de messages pris en charge ... 240
Modes de reconnaissance des messages pris en charge ... 240
En-têtes définis par JMS et propriétés réservées ... 240
Didacticiels ... 242
Création d'une file d'attente Amazon SQS à l'aide de AWS CloudFormation ... 242
Envoi d'un message à partir d'un VPC ... 244
Étape 1 : créer une paire de EC2 clés Amazon ... 245
Étape 2 : Création de AWS ressources ... 246
Étape 3 : vérifier que votre EC2 instance n'est pas accessible au public ... 247
Étape 4 : Création du point de terminaison d'un VPC Amazon pour Amazon SQS ... 248
Étape 5 : Envoyer un message à votre file d'attente Amazon SQS ... 249
Exemples de code ... 251
Principes de base ... 265
Bonjour Amazon SQS ... 266
Actions ... 278
Scénarios ... 444
Création d'une application de messagerie ... 445
Créer une application de messagerie ... 446
Créer une application Amazon Textract Explorer ... 447
Créer et publier dans une rubrique FIFO ... 448
Détecter des personnes et des objets dans une vidéo ... 461

Gérez les messages volumineux à l'aide de S3 ... 462
Traiter les notifications d'événements S3 ... 466
Publier des messages dans des files d'attente ... 470
Envoyer et recevoir des lots de messages ... 585
Utiliser le framework de traitement des AWS messages pour .NET avec Amazon SQS ... 617
Utilisez la bibliothèque de messagerie Java Amazon SQS pour utiliser l'interface JMS ... 617
Utilisation des balises de file d'attente ... 640
Exemples sans serveur ... 643
Invoquer une fonction Lambda à partir d'un déclencheur Amazon SQS ... 643
Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS ... 652
Résolution des problèmes ... 663
Erreur d'accès refusé ... 663
Politique de file d'attente Amazon SQS et politique IAM ... 664
AWS Key Management Service (AWS KMS) autorisations ... 665
Politique de point de terminaison d'un VPC ... 666
Politique de contrôle des services de l'organisation ... 667
Erreurs d'API ... 667
QueueDoesNotExist erreur ... 667
InvalidAttributeValue erreur ... 668
ReceiptHandle erreur ... 668
Problèmes de DLQ et de redrive DLQ ... 669
Problèmes liés au DLQ ... 669
Problèmes liés au DLQ-Redrive ... 671
Problèmes d'étranglement du FIFO ... 673
Messages non renvoyés pour un appel ReceiveMessage d'API ... 674
File d'attente vide ... 674
Limite en vol atteinte ... 674
Retard du message ... 674
Le message est en cours ... 674
Méthode de sondage ... 675
Erreurs réseau ... 675
ETIMEOUT error ... 675
UnknownHostException error ... 676
Dépannage des files d'attente avec X-Ray ... 677
Sécurité ... 679

Protection des données ... 679
Chiffrement des données ... 680
Confidentialité du trafic inter-réseau ... 693
Utilisation de points de terminaison à double pile pour la connectivité ... 695
Gestion des identités et des accès ... 696
Public ciblé ... 696
Authentification par des identités ... 697
Gestion des accès à l'aide de politiques ... 701
Présentation ... 704
Fonctionnement d'Amazon Simple Queue Service avec IAM ... 711
AWS politiques gérées ... 719
Résolution des problèmes ... 722
Utilisation des stratégies ... 724
Journalisation et surveillance ... 774
Journalisation des appels d'API ... 777
Surveillance des files d'attente ... 781
Validation de conformité ... 804
Résilience ... 805
Files d'attente distribuées ... 805
Sécurité de l'infrastructure ... 806
Bonnes pratiques ... 807
S'assurer que les files d'attente ne sont pas accessibles publiquement ... 807
Implémentation d'un accès sur la base du moindre privilège ... 807
Utiliser les rôles IAM pour les applications et les AWS services qui nécessitent un accès Amazon SQS ... 808
Mise en œuvre du chiffrement côté serveur ... 808
Application du chiffrement des données en transit ... 809
Réflexion sur l'utilisation des points de terminaison de VPC pour accéder à Amazon SQS ... 809
Ressources connexes ... 810
Historique de la documentation ... 811
... dcccxx

Qu'est-ce qu'Amazon Simple Queue Service ?

Amazon Simple Queue Service (Amazon SQS) offre une file d'attente hébergée sécurisée, durable et disponible qui vous permet d'intégrer et de découpler les systèmes et les composants de logiciels distribués. Amazon SQS propose des structures communes, telles que des files d'attente de lettres mortes et des balises de répartition des coûts. Il fournit une API de services Web générique à laquelle vous pouvez accéder à l'aide de n'importe quel langage de programmation pris en charge par le AWS SDK.

Avantages offerts par l'utilisation d'Amazon SQS

- Sécurité : vous pouvez contrôler qui peut envoyer et recevoir des messages à partir d'une file d'attente Amazon SQS. Vous pouvez choisir de transmettre des données sensibles en protégeant le contenu des messages dans les files d'attente grâce au chiffrement côté serveur (SSE) géré par Amazon SQS par défaut ou grâce aux clés SSE personnalisées gérées dans AWS Key Management Service (AWS KMS).
- Durabilité : pour assurer la sécurité de vos messages, Amazon SQS les stocke sur plusieurs serveurs. Les files d'attente standard prennent en charge la livraison des at-least-once messages, tandis que les files d'attente FIFO prennent en charge le traitement des messages en une seule fois et le mode haut débit.
- Disponibilité : Amazon SQS utilise une infrastructure redondante pour fournir un accès extrêmement simultané aux messages, et une haute disponibilité pour la production et la consommation des messages.
- Capacité de mise à l'échelle : Amazon SQS peut traiter chaque demande mise en mémoire tampon de façon indépendante, en se mettant à l'échelle de manière transparente pour gérer les augmentations et les pics de charge sans aucune instruction de mise en service.
- Fiabilité : Amazon SQS verrouille vos messages pendant le traitement pour que plusieurs producteurs puissent envoyer des messages et que plusieurs consommateurs puissent recevoir des messages en même temps.
- Personnalisation : vos files d'attente n'ont pas à être parfaitement identiques. Par exemple, vous pouvez définir un retard par défaut sur une file d'attente. Vous pouvez stocker le contenu des messages supérieurs à 1 MiB à l'aide d'Amazon Simple Storage Service (Amazon S3) ou d'Amazon DynamoDB, avec Amazon SQS contenant un pointeur vers l'objet Amazon S3, ou vous pouvez diviser un message volumineux en messages plus petits.

Architecture de base Amazon SQS

Cette section décrit les composants d'un système de messagerie distribué et explique le cycle de vie d'un message Amazon SQS.

Files d'attente distribuées

Un système de messagerie distribué comporte trois parties principales : les composants de votre système distribué, votre file d'attente (distribuée sur les serveurs Amazon SQS) et les messages de la file d'attente.

Dans le scénario suivant, le système comprend plusieurs producteurs (composants qui envoient des messages à la file d'attente) et plusieurs consommateurs (composants qui reçoivent des messages de la file d'attente). La file d'attente (qui contient les messages A à E) stocke les messages de manière redondante sur plusieurs serveurs Amazon SQS.

Cycle de vie des messages

Le scénario suivant décrit le cycle de vie d'un message Amazon SQS dans une file d'attente, de sa création à sa suppression.

1 Un producteur (composant 1) envoie le message A à une file d'attente, et le message est distribué de manière redondante sur les serveurs Amazon SQS.

2 Lorsqu consommateur (composant 2) est prêt à traiter des messages, il consomme les messages de la file d'attente et le message A est renvoyé. Pendant son traitement, le message A reste dans la file d'attente et n'est pas renvoyé aux demandes de réception suivantes pendant la durée du délai de visibilité.

3 Le consommateur (composant 2) supprime le message A de la file d'attente pour empêcher que le message ne soit reçu et traité à nouveau lorsque le délai de visibilité expire.

> **Note**
>
> Amazon SQS supprime automatiquement d'une file d'attente les messages qui dépassent la période maximale de conservation des messages. La période de conservation des messages par défaut est de 4 jours. Cependant, vous pouvez configurer la période de rétention des messages sur une valeur allant de 60 secondes à 1 209 600 secondes (14 jours) avec `SetQueueAttributes`

Différences entre Amazon SQS, Amazon MQ et Amazon SNS

Amazon SQS, Amazon SNS et Amazon MQ proposent des services de messagerie gérés easy-to-use et hautement évolutifs, chacun étant conçu pour des rôles spécifiques au sein de systèmes distribués. Voici un aperçu détaillé des différences entre ces services :

Amazon SQS dissocie et fait évoluer les systèmes et composants logiciels distribués en tant que service de file d'attente. Il traite généralement les messages via un seul abonné, ce qui est idéal pour les flux de travail où la prévention des commandes et des pertes est essentielle. Pour une distribution plus large, l'intégration d'Amazon SQS à Amazon SNS permet d'utiliser un modèle de messagerie en fanout, transmettant efficacement les messages à plusieurs abonnés à la fois.

Amazon SNS permet aux éditeurs d'envoyer des messages à plusieurs abonnés par le biais de rubriques servant de canaux de communication. Les abonnés reçoivent les messages publiés via un type de point de terminaison compatible Amazon Data Firehose, tel qu'Amazon SQS, Lambda, HTTP, les e-mails, les notifications push mobiles et les SMS. Ce service est idéal pour les scénarios nécessitant des notifications immédiates, tels que l'engagement des utilisateurs en temps réel ou les systèmes d'alarme. Pour éviter la perte de messages lorsque les abonnés sont hors ligne, l'intégration d'Amazon SNS aux messages de file d'attente Amazon SQS garantit une diffusion cohérente.

Amazon MQ convient parfaitement aux entreprises qui souhaitent migrer depuis des courtiers de messages traditionnels, prenant en charge les protocoles de messagerie standard tels que AMQP et MQTT, ainsi qu'Apache ActiveMQ et RabbitMQ. Il est compatible avec les systèmes existants nécessitant une messagerie stable et fiable sans reconfiguration importante.

Le tableau suivant donne un aperçu du type de ressource de chaque service :

Type de ressource	Amazon SNS	Amazon SQS	Amazon MQ
Synchrone	Non	Non	Oui
asynchrone	Oui	Oui	Oui
Files d'attente	Non	Oui	Oui
Messagerie par publication-abonnement	Oui	Non	Oui
Agents de messages	Non	Non	Oui

Amazon SQS et Amazon SNS sont tous deux recommandés pour les nouvelles applications qui peuvent bénéficier d'une évolutivité et d'une simplicité quasiment illimitées. APIs Ils proposent généralement des solutions plus rentables pour les applications à volume élevé grâce à leurs pay-as-you-go prix. Nous recommandons Amazon MQ pour la migration d'applications provenant de courtiers de messages existants qui reposent sur la compatibilité avec JMS ou des protocoles APIs tels que le protocole AMQP (Advanced Message Queuing Protocol), le MQTT et le protocole STOMP (Simple Text Oriented Message Protocol). OpenWire

Démarrer avec Amazon SQS

Cette rubrique explique comment utiliser la console Amazon SQS pour créer et gérer des files d'attente standard et des files d'attente FIFO. Vous allez apprendre à naviguer dans la console, à afficher les attributs des files d'attente et à distinguer les types de files d'attente. Les tâches principales incluent l'envoi, la réception et la configuration des messages, l'ajustement de paramètres tels que le délai de visibilité et la conservation des messages, et la gestion de l'accès aux files d'attente par le biais de politiques.

Rubriques

- Configuration d'Amazon SQS
- Comprendre la console Amazon SQS
- Types de files d'attente Amazon SQS
- Création d'une file d'attente standard Amazon SQS et envoi d'un message
- Création d'une file d'attente FIFO Amazon SQS et envoi d'un message
- Tâches courantes pour démarrer avec Amazon SQS

Configuration d'Amazon SQS

Avant de pouvoir utiliser Amazon SQS pour la première fois, vous devez suivre les étapes suivantes :

Étape 1 : créer un utilisateur Compte AWS et IAM

Pour accéder à un AWS service, vous devez d'abord créer un Compte AWScompte Amazon.com permettant d'utiliser AWS des produits. Vous pouvez utiliser votre Compte AWS pour consulter vos rapports d'activité et d'utilisation et pour gérer l'authentification et l'accès.

Pour éviter d'utiliser votre utilisateur Compte AWS root pour les actions Amazon SQS, il est recommandé de créer un utilisateur IAM pour chaque personne ayant besoin d'un accès administratif à Amazon SQS.

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'https://portal.aws.amazon.com/billing/inscription.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique ou un SMS et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les tâches nécessitant un accès utilisateur racine.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte en accédant à https://aws.amazon.com/et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, consultez la section Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d’un utilisateur doté d’un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d’ AWS IAM Identity Center dans le Guide de l’utilisateur AWS IAM Identity Center .

2. Dans IAM Identity Center, octroyez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connexion en tant qu’utilisateur doté d’un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l’URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l’utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribution d’un accès à d’autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d’autorisations qui respecte la bonne pratique consistant à appliquer les autorisations de moindre privilège.

   Pour obtenir des instructions, consultez Création d’un ensemble d’autorisations dans le Guide de l’utilisateur AWS IAM Identity Center .

2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

   Pour obtenir des instructions, consultez Ajout de groupes dans le Guide de l’utilisateur AWS IAM Identity Center .

Étape 2 : Accorder un accès par programmation

Pour utiliser les actions Amazon SQS (par exemple, en utilisant Java ou via le AWS Command Line Interface), vous avez besoin d'un ID de clé d'accès et d'une clé d'accès secrète.

> **Note**
>
> L'ID de clé d'accès et la clé d'accès secrète sont spécifiques à AWS Identity and Access Management. Ne les confondez pas avec les identifiants d'autres AWS services, tels que les paires de EC2 clés Amazon.

Les utilisateurs ont besoin d'un accès programmatique s'ils souhaitent interagir avec AWS l'extérieur du AWS Management Console. La manière d'accorder un accès programmatique dépend du type d'utilisateur qui y accède AWS.

Pour accorder aux utilisateurs un accès programmatique, choisissez l'une des options suivantes.

Quel utilisateur a besoin d'un accès programmatique ?	Pour	Par
Identité de la main-d'œuvre (Utilisateurs gérés dans IAM Identity Center)	Utilisez des informations d'identification temporaires pour signer les demandes programmatiques adressées au AWS CLI AWS SDKs, ou AWS APIs.	Suivez les instructions de l'interface que vous souhaitez utiliser. • Pour le AWS CLI, voir Configuration du AWS CLI à utiliser AWS IAM Identity Center dans le guide de AWS Command Line Interface l'utilisateur. • Pour AWS SDKs, outils, et AWS APIs, voir Authentification IAM Identity Center dans le guide de référence AWS SDKs et Tools.
IAM	Utilisez des informations d'identification temporaires pour signer les demandes programmatiques adressées	Suivez les instructions de la section Utilisation d'informations d'identification temporaires avec AWS les ressources du Guide de l'utilisateur IAM.

Quel utilisateur a besoin d'un accès programmatique ?	Pour	Par
	au AWS CLI AWS SDKs, ou AWS APIs.	
IAM	(Non recommandé) Utilisez des informations d'identification à long terme pour signer des demandes programmatiques adressées au AWS CLI AWS SDKs, ou AWS APIs.	Suivez les instructions de l'interface que vous souhaitez utiliser. • Pour le AWS CLI, voir Authentification à l'aide des informations d'identification utilisateur IAM dans le Guide de l'AWS Command Line Interface utilisateur. • Pour les outils AWS SDKs et, voir Authentifier à l'aide d'informations d'identif ication à long terme dans le guide de référence des outils AWS SDKs et. • Pour AWS APIs, voir Gestion des clés d'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Étape 3 : Préparation à l'utilisation de l'exemple de code

Ce guide contient des exemples d'utilisation du AWS SDK for Java. Pour exécuter l'exemple de code, suivez les instructions de configuration décrites dans Démarrer avec le kit AWS SDK pour Java 2.0.

Vous pouvez développer des AWS applications dans d'autres langages de programmation, tels que GoJavaScript, Python et Ruby. Pour plus d'informations, consultez la section Outils sur lesquels vous pouvez vous appuyer AWS.

> **Note**
>
> Vous pouvez explorer Amazon SQS sans écrire de code à l'aide d'outils tels que AWS Command Line Interface (AWS CLI) ou Windows. PowerShell Vous trouverez des AWS CLI exemples dans la section Amazon SQS de la AWS CLI Command Reference. Vous trouverez des PowerShell exemples de Windows dans la section Amazon Simple Queue Service de la référence des Outils AWS pour PowerShell applets de commande.

Étapes suivantes

Vous pouvez maintenant commencer à gérer les files d'attente et les messages Amazon SQS à l'aide de l' AWS Management Console.

Comprendre la console Amazon SQS

Lorsque vous ouvrez la console Amazon SQS, choisissez Queues dans le volet de navigation. La page Files d'attente fournit des informations sur toutes vos files d'attente dans la région active.

Chaque entrée de file d'attente fournit des informations essentielles sur la file d'attente, notamment son type et ses principaux attributs. Les files d'attente standard, optimisées pour un débit maximal et un classement optimal des messages, se distinguent des files d'attente First-In-First-Out (FIFO), qui privilégient l'ordre des messages et leur caractère unique pour les applications nécessitant un séquençage strict des messages.

Éléments et actions interactifs

Sur la page Files d'attente, vous disposez de plusieurs options pour gérer vos files d'attente :

1. Actions rapides — À côté du nom de chaque file d'attente, un menu déroulant permet d'accéder rapidement aux actions courantes telles que l'envoi de messages, l'affichage ou la suppression de messages, la configuration de déclencheurs et la suppression de la file d'attente elle-même.
2. Vue détaillée et configuration : cliquez sur le nom d'une file d'attente pour ouvrir sa page de détails, dans laquelle vous pouvez approfondir les paramètres et les configurations des files d'attente. Ici, vous pouvez ajuster des paramètres tels que la période de rétention des messages, le délai de visibilité et la taille maximale des messages pour adapter la file d'attente aux exigences de votre application.

Sélection de régions et balises de ressources

Assurez-vous que vous êtes bien placé Région AWS pour accéder à vos files d'attente et les gérer efficacement. En outre, pensez à utiliser des balises de ressources pour organiser et classer vos files d'attente, afin d'améliorer la gestion des ressources, la répartition des coûts et le contrôle d'accès au sein de votre environnement AWS partagé.

En tirant parti des caractéristiques et fonctionnalités proposées dans la console Amazon SQS, vous pouvez gérer efficacement votre infrastructure de messagerie, optimiser les performances des files d'attente et garantir une distribution fiable des messages pour vos applications.

Types de files d'attente Amazon SQS

Amazon SQS prend en charge deux types de files d'attente : les files d'attente standard et les files FIFO. Utilisez le tableau suivant pour déterminer quelle file d'attente correspond le mieux à vos besoins.

Files d'attente standard	Files d'attente FIFO
Débit illimité : les files d'attente standard prennent en charge un nombre très élevé, presque illimité, d'appels d'API par seconde, par action (`SendMessage` `ReceiveMessage`, ou `DeleteMessage`). Ce débit élevé les rend idéaux pour les cas d'utilisation nécessitant le traitement rapide de gros volumes de messages, tels que le streaming de données en temps réel ou les applications à grande échelle. Bien que les files d'attente standard s'adaptent automatiquement à la demande, il est essentiel de surveiller les modèles d'utilisation pour garantir des performances optimales, en particulier dans les régions où les charges de travail sont plus élevées. At-least-once livraison — at-least-once Livraison garantie, ce qui signifie que chaque message est livré au moins une fois, mais dans certains cas, un message peut être délivré plusieurs fois en raison de nouvelles tentatives ou de retards sur le réseau. Vous devez concevoir votre application de manière à gérer les messages dupliqués potentiels en utilisant des opérations idempotentes, qui garantissent que le traitement du même message à plusieurs reprises n'affectera pas l'état du système. Commande dans les meilleurs délais : permet de passer des commandes dans les meilleurs délais, ce qui signifie qu'Amazon SQS essaie de livrer les messages dans l'ordre dans lequel ils ont été envoyés, mais cela n'est pas garanti.	Haut débit — Lorsque vous utilisez le traitement par lots, les files d'attente FIFO traitent jusqu'à 3 000 messages par seconde et par méthode d'API (`SendMessageBatch`, `ReceiveMessage` ou). `DeleteMessageBatch` Ce débit repose sur 300 appels d'API par seconde, chaque appel d'API gérant un lot de 10 messages. En activant le mode haut débit, vous pouvez augmenter jusqu'à 30 000 transactions par seconde (TPS) avec un classement simplifié au sein des groupes de messages. Sans traitement par lots, les files d'attente FIFO prennent en charge jusqu'à 300 appels d'API par seconde et par méthode d'API (`SendMessage` `ReceiveMessage`, ou). `DeleteMessage` Si vous avez besoin d'un débit supérieur, vous pouvez demander une augmentation de quota via le AWS Support Center. Pour activer le mode haut débit, voir. Permettre un débit élevé pour les files d'attente FIFO dans Amazon SQS Traitement en une seule fois : les files d'attente FIFO ne transmettent chaque message qu'une seule fois et le gardent disponible jusqu'à ce que vous le traitiez et le supprimiez. En utilisant des fonctionnalités telles que la déduplication basée sur le contenu, vous évitez les doublons de messages, même en cas de nouvelle tentative en raison de problèmes de réseau `MessageDeduplicationId` ou de délais d'attente. First-in-first-out livraison : les files d'attente FIFO garantissent que vous recevez les

Files d'attente standard	Files d'attente FIFO
Dans certains cas, les messages peuvent arriver dans le mauvais ordre, notamment en cas de débit élevé ou de reprise après échec. Pour les applications où l'ordre de traitement des messages est crucial, vous devez gérer la logique de réorganisation au sein de l'applica tion ou utiliser des files d'attente FIFO pour des garanties de commande strictes. Durabilité et redondance : les files d'attente standard garantissent une durabilité élevée en stockant plusieurs copies de chaque message dans plusieurs AWS zones de disponibilité. Cela garantit que les messages ne sont pas perdus, même en cas de défaillance de l'infrast ructure. Délai de visibilité : Amazon SQS vous permet de configurer un délai de visibilité pour contrôler la durée pendant laquelle un message reste masqué après réception, afin de garantir que les autres consommateurs ne traitent pas le message tant qu'il n'a pas été entièreme nt traité ou que le délai d'expiration n'est pas expiré.	messages dans l'ordre dans lequel ils sont envoyés au sein de chaque groupe de messages. En répartissant les messages entre plusieurs groupes, vous pouvez les traiter en parallèle tout en maintenant l'ordre au sein de chaque groupe.

Files d'attente standard	Files d'attente FIFO
Utilisez des files d'attente standard pour envoyer des données entre applications lorsque le débit est crucial, par exemple : • Dissociez les demandes des utilisateurs en direct d'un travail de fond intensif. Permettez aux utilisateurs de télécharger rapidemen t du contenu multimédia pendant que vous effectuez des tâches telles que le redimensi onnement ou l'encodage en arrière-plan, garantissant ainsi des temps de réponse rapides sans surcharger le système. • Allouez des tâches à plusieurs nœuds de travail. Répartissez un grand nombre de demandes de validation de carte de crédit sur plusieurs nœuds de travail et gérez les messages dupliqués avec des opération s idempotentes afin d'éviter les erreurs de traitement. • Messages par lots pour un traitement futur. Mettez en file d'attente plusieurs entrées pour les ajouts par lots à une base de données. L'ordre des messages n'étant pas garanti, concevez votre système pour qu'il gère out-of-order le traitement si nécessaire.	Utilisez les files d'attente FIFO pour envoyer des données entre applications lorsque l'ordre des événements est important, par exemple : • Assurez-vous que les commandes saisies par l'utilisateur sont exécutées dans le bon ordre. Il s'agit d'un cas d'utilisation clé pour les files d'attente FIFO, où l'ordre des commandes est crucial. Par exemple, si un utilisateur exécute une séquence d'actions dans une application, les files d'attente FIFO garantissent que les actions sont traitées dans l'ordre dans lequel elles ont été saisies. • Affichez le prix correct du produit en envoyant les modifications de prix dans le bon ordre. Les files d'attente FIFO garantiss ent que plusieurs mises à jour du prix d'un produit arrivent et sont traitées de manière séquentielle. Sans le FIFO, une réduction de prix peut être traitée après une augmentat ion de prix, ce qui entraîne l'affichage de données incorrectes. • Empêcher un étudiant de s'inscrire à un cours avant de créer un compte. En utilisant des files d'attente FIFO, vous vous assurez que le processus d'enregistrement se déroule dans le bon ordre. Le système traite d'abord l'enregistrement du compte, puis l'inscrip tion au cours, empêchant ainsi l'exécution prématurée de la demande d'inscription.

Implémentation de systèmes de demande-réponse dans Amazon SQS

Lors de l'implémentation d'un système de demande-réponse ou d'appel de procédure éloigné (RPC), ayez les bonnes pratiques suivantes à l'esprit :

- Créez des files de réponse au démarrage — Au lieu de créer des files de réponses par message, créez-les au démarrage, par producteur. Utilisez un attribut de message d'identification de corrélation pour mapper efficacement les réponses aux demandes.
- Évitez de partager les files de réponses entre les producteurs : assurez-vous que chaque producteur possède sa propre file de réponses. Le partage de files de réponses peut amener un producteur à recevoir des messages de réponse destinés à un autre producteur.

Pour plus d'informations sur l'implémentation du modèle demande-réponse à l'aide du client de file d'attente temporaire, consultez Modèle de messagerie demande-réponse (files d'attente virtuelles).

Création d'une file d'attente standard Amazon SQS et envoi d'un message

Vous pouvez créer une file d'attente standard et envoyer des messages à l'aide de la console Amazon SQS. Cette rubrique met également l'accent sur les meilleures pratiques, notamment l'évitement des informations sensibles dans les noms des files d'attente et l'utilisation du chiffrement géré côté serveur.

Création d'une file d'attente standard à l'aide de la console Amazon SQS

> **Important**
>
> Le 17 août 2022, le chiffrement côté serveur (SSE) par défaut a été appliqué à toutes les files d'attente Amazon SQS.
> N'ajoutez pas de données d'identification personnelle (PII) ou d'autres données confidentielles ou sensibles dans les noms de file d'attente. Les noms des files d'attente sont accessibles à de nombreux Amazon Web Services, y compris les noms de facturation et CloudWatch les journaux. Les noms de file d'attente ne sont pas destinés à être utilisés pour des données privées ou sensibles.

Pour créer une file d'attente standard Amazon SQS

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Choisissez Créez une file d'attente.
3. Pour Type, le type de file d'attente standard est défini par défaut.

> **Note**
>
> Une fois la file d'attente créée, vous ne pouvez pas modifier son type.

4. Entrez un nom pour votre file d'attente.
5. (Facultatif) La console définit les valeurs par défaut pour les paramètres de configuration de la file d'attente. Sous Configuration, vous pouvez définir de nouvelles valeurs pour les paramètres suivants :

   a. Pour le Délai de visibilité, saisissez la durée et les unités. La plage est comprise entre 0 seconde et 12 heures. La valeur par défaut est de 30 secondes.

   b. Pour Période de conservation des messages, saisissez la durée et les unités. La plage est comprise entre 1 minute et 14 jours. La valeur par défaut est de 4 jours.

   c. Pour Retard de diffusion, saisissez la durée et les unités. La plage est comprise entre 0 seconde et 15 minutes. La valeur par défaut est de 0 seconde.

   d. Pour Taille maximale du message, saisissez une valeur. La plage est comprise entre 1 KiB et 1024 KiB. La valeur par défaut est 1024 KiB.

   e. Pour le Temps d'attente du message de réception, saisissez une valeur. La plage est comprise entre 0 et 20 secondes. La valeur par défaut est 0 seconde, qui permet de définir la recherche courte. Toute valeur différente de zéro définit une recherche longue.

6. (Facultatif) Définissez une stratégie d'accès. La stratégie d'accès définit les comptes, les utilisateurs et les rôles qui peuvent accéder à la file d'attente. La stratégie d'accès définit également les actions (telles que `SendMessage`, `ReceiveMessage` ou`DeleteMessage`) auxquelles les utilisateurs peuvent accéder. La stratégie par défaut permet uniquement au propriétaire de la file d'attente d'envoyer et de recevoir des messages.

   Pour définir la stratégie d'accès, effectuez l'une des opérations suivantes :

   - Choisissez Basique pour configurer qui peut envoyer des messages à la file d'attente et qui peut recevoir des messages depuis la file d'attente. La console crée la stratégie en fonction de vos choix et affiche la stratégie d'accès qui en résulte dans le panneau JSON en lecture seule.

- Choisissez Avancé pour modifier directement la stratégie d'accès JSON. Cela vous permet de spécifier un ensemble personnalisé d'actions que chaque mandataire (compte, utilisateur ou rôle) peut effectuer.

7. Pour la Stratégie d'autorisation de redirection, choisissez Activé. Sélectionnez l'une des options suivantes : Tout autoriser (par défaut), Par file d'attente ou Refuser tout. Lorsque vous choisissez Par file d'attente, spécifiez une liste de 10 files d'attente source maximum en fonction de l'Amazon Resource Name (ARN).
8. Amazon SQS fournit un chiffrement côté serveur géré par défaut. Pour choisir un type de clé de chiffrement ou pour désactiver le chiffrement côté serveur géré par Amazon SQS, développez Chiffrement. Pour en savoir plus sur les types de clés de chiffrement, consultez Configuration du chiffrement côté serveur pour une file d'attente à l'aide de clés de chiffrement gérées par SQL et Configuration du chiffrement côté serveur pour une file d'attente à l'aide de la console Amazon SQS.

   > **Note**
   >
   > Lorsque SSE est activé, les demandes anonymes `SendMessage` et `ReceiveMessage` adressées à la file d'attente chiffrée sont rejetées. Les bonnes pratiques de sécurité d'Amazon SQS recommandent de ne pas utiliser de demandes anonymes. Si vous souhaitez envoyer des demandes anonymes à une file d'attente Amazon SQS, veillez à désactiver SSE.

9. (Facultatif) Pour configurer une file d'attente de lettres mortes pour recevoir des messages non distribuables, développez File d'attente de lettres mortes.
10. (Facultatif) Pour ajouter des balises à la file d'attente, développez Balises.
11. Choisissez Créez une file d'attente. Amazon SQS crée la file d'attente et affiche la page de Détails de la file d'attente.

Amazon SQS diffuse les informations relatives à la nouvelle file d'attente dans le système. Amazon SQS étant un système distribué, il se peut que la console affiche la file d'attente sur la page Files d'attente avec un léger retard.

Envoi d'un message à l'aide d'une file d'attente standard

Une fois votre file d'attente créée, vous pouvez lui envoyer un message.

1. Dans le volet de navigation de gauche, choisissez Files d'attente. Dans la liste des files d'attente, sélectionnez la file d'attente que vous avez créée.
2. Dans Actions, choisissez Envoyer et recevoir des messages.

   La console affiche la page Envoyer et recevoir des messages.
3. Dans le Corps du message, saisissez le texte du message.
4. Pour une file d'attente standard, vous pouvez saisir une valeur pour le Délai de livraison et choisir les unités. Par exemple, saisissez `60` et choisissez secondes. Pour de plus amples informations, veuillez consulter Temporisateurs de messages Amazon SQS.
5. Choisissez Send Message (Envoyer un message).

   Lorsque votre message est envoyé, la console affiche un message de réussite. Choisissez Afficher les détails pour afficher les informations relatives au message envoyé.

Création d'une file d'attente FIFO Amazon SQS et envoi d'un message

Vous pouvez créer une file d'attente FIFO Amazon SQS et envoyer des messages à l'aide de la console. Cette rubrique explique comment configurer les paramètres des files d'attente, notamment le délai de visibilité, la conservation des messages et la déduplication, tout en respectant les meilleures pratiques de sécurité, telles que l'évitement des informations sensibles dans les noms des files d'attente et l'activation du chiffrement côté serveur. Il couvre également la définition de politiques d'accès, la configuration de files d'attente de lettres mortes et l'envoi de messages avec des attributs spécifiques au FIFO, tels que l'ID de groupe de messages et l'ID de déduplication.

Création d'une file d'attente FIFO à l'aide de la console Amazon SQS

Vous pouvez utiliser la console Amazon SQS pour créer des files d'attente FIFO. La console fournit des valeurs par défaut pour tous les paramètres, à l'exception du nom de la file d'attente.

> **Important**
>
> Le 17 août 2022, le chiffrement côté serveur (SSE) par défaut a été appliqué à toutes les files d'attente Amazon SQS.
> N'ajoutez pas de données d'identification personnelle (PII) ou d'autres données confidentielles ou sensibles dans les noms de file d'attente. Les noms des files d'attente sont

accessibles à de nombreux Amazon Web Services, y compris les noms de facturation et CloudWatch les journaux. Les noms de file d'attente ne sont pas destinés à être utilisés pour des données privées ou sensibles.

Pour créer une file d'attente FIFO Amazon SQS

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Choisissez Créez une file d'attente.
3. Pour Type, le type de file d'attente standard est défini par défaut. Pour créer une file d'attente FIFO, choisissez FIFO.

   > **Note**
   >
   > Une fois la file d'attente créée, vous ne pouvez pas modifier son type.

4. Entrez un nom pour votre file d'attente.

   Le nom d'une file d'attente FIFO doit se terminer par le suffixe `.fifo`. Le suffixe est pris en compte dans le quota de 80 caractères pour les noms de file d'attente. Pour déterminer si une file d'attente est de type FIFO, vous pouvez vérifier si son nom se termine par le suffixe.

5. (Facultatif) La console définit les valeurs par défaut pour les paramètres de configuration de la file d'attente. Sous Configuration, vous pouvez définir de nouvelles valeurs pour les paramètres suivants :

   a. Pour le Délai de visibilité, saisissez la durée et les unités. La plage est comprise entre 0 seconde et 12 heures. La valeur par défaut est de 30 secondes.

   b. Pour Période de conservation des messages, saisissez la durée et les unités. La plage est comprise entre 1 minute et 14 jours. La valeur par défaut est de 4 jours.

   c. Pour Retard de diffusion, saisissez la durée et les unités. La plage est comprise entre 0 seconde et 15 minutes. La valeur par défaut est de 0 seconde.

   d. Pour Taille maximale du message, saisissez une valeur. La plage est comprise entre 1 KiB et 1024 KiB. La valeur par défaut est 1024 KiB.

   e. Pour le Temps d'attente du message de réception, saisissez une valeur. La plage est comprise entre 0 et 20 secondes. La valeur par défaut est 0 seconde, qui permet de définir la recherche courte. Toute valeur différente de zéro définit une recherche longue.

f. Pour une file d'attente FIFO, choisissez Déduplication basée sur le contenu pour activer cette option. Par défaut, ce paramètre est désactivé.

g. (Facultatif) Pour qu'une file d'attente FIFO permette un débit plus élevé pour l'envoi et la réception de messages dans la file d'attente, choisissez Activer le FIFO à haut débit.

   Le choix de cette option modifie les options associées (Portée de la déduplication et Limite de débit FIFO) en fonction des paramètres requis pour activer un débit élevé pour les files d'attente FIFO. Si vous modifiez l'un des paramètres requis pour utiliser le FIFO à débit élevé, le débit normal est effectif pour la file d'attente et la déduplication se produit comme indiqué. Pour plus d'informations, consultez Débit élevé pour les files d'attente FIFO dans Amazon SQS et Quotas de messages Amazon SQS.

6. (Facultatif) Définissez une stratégie d'accès. La stratégie d'accès définit les comptes, les utilisateurs et les rôles qui peuvent accéder à la file d'attente. La stratégie d'accès définit également les actions (telles que `SendMessage`, `ReceiveMessage` ou`DeleteMessage`) auxquelles les utilisateurs peuvent accéder. La stratégie par défaut permet uniquement au propriétaire de la file d'attente d'envoyer et de recevoir des messages.

   Pour définir la stratégie d'accès, effectuez l'une des opérations suivantes :

   - Choisissez Basique pour configurer qui peut envoyer des messages à la file d'attente et qui peut recevoir des messages depuis la file d'attente. La console crée la stratégie en fonction de vos choix et affiche la stratégie d'accès qui en résulte dans le panneau JSON en lecture seule.
   - Choisissez Avancé pour modifier directement la stratégie d'accès JSON. Cela vous permet de spécifier un ensemble personnalisé d'actions que chaque mandataire (compte, utilisateur ou rôle) peut effectuer.

7. Pour la Stratégie d'autorisation de redirection, choisissez Activé. Sélectionnez l'une des options suivantes : Tout autoriser (par défaut), Par file d'attente ou Refuser tout. Lorsque vous choisissez Par file d'attente, spécifiez une liste de 10 files d'attente source maximum en fonction de l'Amazon Resource Name (ARN).

8. Amazon SQS fournit un chiffrement côté serveur géré par défaut. Pour choisir un type de clé de chiffrement ou pour désactiver le chiffrement côté serveur géré par Amazon SQS, développez Chiffrement. Pour en savoir plus sur les types de clés de chiffrement, consultez Configuration du chiffrement côté serveur pour une file d'attente à l'aide de clés de chiffrement gérées par SQL et Configuration du chiffrement côté serveur pour une file d'attente à l'aide de la console Amazon SQS.

> **Note**
>
> Lorsque SSE est activé, les demandes anonymes `SendMessage` et `ReceiveMessage` adressées à la file d'attente chiffrée sont rejetées. Les bonnes pratiques de sécurité d'Amazon SQS recommandent de ne pas utiliser de demandes anonymes. Si vous souhaitez envoyer des demandes anonymes à une file d'attente Amazon SQS, veillez à désactiver SSE.

9. (Facultatif) Pour configurer une file d'attente de lettres mortes pour recevoir des messages non distribuables, développez File d'attente de lettres mortes.
10. (Facultatif) Pour ajouter des balises à la file d'attente, développez Balises.
11. Choisissez Créez une file d'attente. Amazon SQS crée la file d'attente et affiche la page de Détails de la file d'attente.

Amazon SQS diffuse les informations relatives à la nouvelle file d'attente dans le système. Amazon SQS étant un système distribué, il se peut que la console affiche la file d'attente sur la page Files d'attente avec un léger retard.

Après avoir créé une file d'attente, vous pouvez lui envoyer des messages, et recevoir et supprimer des messages. Vous pouvez également modifier tous les paramètres de configuration de la file d'attente, à l'exception du type de file d'attente.

Envoi d'un message à l'aide d'une file d'attente FIFO

Après avoir créé votre file d'attente, vous pouvez lui envoyer un message.

1. Dans le volet de navigation de gauche, choisissez Files d'attente. Dans la liste des files d'attente, sélectionnez la file d'attente que vous avez créée.
2. Dans Actions, choisissez Envoyer et recevoir des messages.

   La console affiche la page Envoyer et recevoir des messages.
3. Dans le Corps du message, saisissez le texte du message.
4. Pour une file d'attente First-In-First-Out (FIFO), entrez un ID de groupe de messages. Pour de plus amples informations, veuillez consulter Logique de distribution des files d'attente FIFO dans Amazon SQS.

5. (Facultatif) Pour une file d'attente FIFO, vous pouvez saisir un ID de déduplication des messages. Si vous avez activé la déduplication basée sur le contenu pour la file d'attente, l'ID de déduplication des messages n'est pas requis. Pour de plus amples informations, veuillez consulter Logique de distribution des files d'attente FIFO dans Amazon SQS.
6. Les files d'attente FIFO ne prennent pas en charge les temporisateurs pour les messages individuels. Pour de plus amples informations, veuillez consulter Temporisateurs de messages Amazon SQS.
7. Choisissez Send Message (Envoyer un message).

   Lorsque votre message est envoyé, la console affiche un message de réussite. Choisissez Afficher les détails pour afficher les informations relatives au message envoyé.

Tâches courantes pour démarrer avec Amazon SQS

Une fois que vous avez créé une file d'attente et appris comment envoyer, recevoir et supprimer des messages, vous pouvez essayer ce qui suit :

- Déclenchez une fonction Lambda pour traiter automatiquement les messages entrants, permettant ainsi des flux de travail axés sur les événements sans qu'il soit nécessaire de recourir à des interrogations continues.
- Configurez les files d'attente, y compris SSE et d'autres fonctionnalités.
- Envoyer un message avec des attributs.
- Envoyer un message à partir d'un VPC.
- Découvrez les fonctionnalités et l'architecture d'Amazon SQS.
- Découvrez les directives et les mises en garde qui vous aideront à tirer le meilleur parti d'Amazon SQS.
- Explorez les exemples de AWS SDK Amazon SQS, tels que le guide du AWS SDK for Java 2.x développeur.
- En savoir plus sur les commandes Amazon SQS. AWS CLI
- En savoir plus sur les actions de l'API Amazon SQS.
- Découvrez comment interagir avec Amazon SQS par programmation. Consultez la section Travailler avec le centre de AWS développement APIs et explorez-le :
  - Java
  - JavaScript

- PHP
- Python
- Ruby
- Windows & .NET

- Découvrez comment surveiller les coûts et les ressources.
- Découvrez comment protéger vos données.
- En savoir plus sur le flux de travail Amazon SQS.

Gérer une file d'attente Amazon SQS

Découvrez comment gérer les files d'attente Amazon SQS à l'aide de la console, notamment en modifiant les paramètres des files d'attente, en recevant et en supprimant des messages, en confirmant le vide des files d'attente et en supprimant ou en purgeant les files d'attente. Découvrez les meilleures pratiques pour une gestion efficace des messages, telles que l'utilisation de longs sondages, la gestion des délais de visibilité et la vérification des indicateurs via des tableaux de bord de surveillance ou le. AWS CLI Suivez les étapes pratiques pour maintenir les files d'attente et gérer les messages efficacement tout en minimisant les perturbations.

Modification d'une file d'attente Amazon SQS à l'aide de la console

Vous pouvez utiliser la console Amazon SQS pour modifier les paramètres de configuration de la file d'attente (à l'exception du type de file d'attente) et modifier ou supprimer des fonctionnalités selon vos besoins.

Pour modifier une file d'attente Amazon SQS (console)

1. Ouvrez la page Files d'attente de la console Amazon SQS.
2. Sélectionnez une file d'attente, puis choisissez Modifier.
3. (Facultatif) Sous Configuration, mettez à jour les paramètres de configuration de la file d'attente.
4. (Facultatif) Pour mettre à jour la politique d'accès, sous Politique d'accès, modifiez la politique JSON.
5. (Facultatif) Pour mettre à jour une stratégie d'autorisation de redirection de file d'attente de lettres mortes, développez Stratégie d'autorisation de redirection.
6. (Facultatif) Pour mettre à jour ou supprimer le chiffrement, développez Chiffrement.
7. (Facultatif) Pour ajouter, mettre à jour ou supprimer une file d'attente de lettres mortes (qui vous permet de recevoir des messages non distribuables), développez File d'attente de lettres mortes.
8. (Facultatif) Pour ajouter, mettre à jour ou supprimer les balises de la file d'attente, développez Balises.
9. Choisissez Enregistrer.
   - La console affiche la page Détails de la file d'attente.

Réception et suppression d'un message dans Amazon SQS

Après avoir envoyé des messages à une file d'attente Amazon SQS, vous pouvez les récupérer et les supprimer pour traiter le flux de travail de votre application. Ce processus garantit un traitement sécurisé et fiable des messages. Cette rubrique explique comment récupérer et supprimer des messages à l'aide de la console Amazon SQS et explique les principaux paramètres permettant d'optimiser cette opération. Les principaux concepts relatifs à la réception et à la suppression de messages sont les suivants :

1. Réception de messages
   - Lorsque vous récupérez des messages depuis une file d'attente Amazon SQS, vous ne pouvez pas cibler des messages spécifiques. Spécifiez plutôt le nombre maximum de messages à récupérer dans une seule demande (jusqu'à 10).
   - En raison de la nature distribuée d'Amazon SQS, la récupération d'une file d'attente contenant peu de messages peut renvoyer une réponse vide. Pour pallier ce problème :
     - Utilisez un long sondage, qui attend qu'un message soit disponible ou que le sondage expire. Cette approche réduit les coûts de sondage inutiles et améliore l'efficacité.
     - Réémettez la demande si nécessaire.
2. Visibilité et suppression des messages
   - Les messages ne sont pas supprimés automatiquement après leur extraction. Cette fonctionnalité vous permet de retraiter les messages en cas de défaillance d'une application ou d'interruption du réseau.
   - Après le traitement, vous devez envoyer explicitement une demande de suppression pour supprimer définitivement le message. Cette action confirme la réussite de la manipulation.
   - Les messages récupérés à l'aide de la console Amazon SQS restent visibles pour être récupérés à nouveau. Ajustez le paramètre de délai de visibilité pour les environnements automatisés afin de masquer temporairement les messages destinés aux autres consommateurs pendant leur traitement.
3. Délai d'expiration de visibilité
   - Ce paramètre détermine la durée pendant laquelle un message reste masqué après sa récupération. Définissez un délai d'expiration approprié pour garantir que les messages ne sont traités qu'une seule fois et pour éviter toute duplication lors du traitement distribué.

Pour recevoir et supprimer un message à l'aide de la console

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Sur la page Files d'attente, choisissez la file d'attente dont vous souhaitez recevoir des messages, puis sélectionnez Envoyer et recevoir des messages.
4. Sur la page Envoyer et recevoir des messages, sélectionnez Sondage pour les messages.

   Amazon SQS affiche une barre de progression indiquant la durée du sondage. Les messages récupérés apparaîtront dans la section Messages, avec les informations suivantes :

   - ID de message
   - Date d'envoi
   - Size
   - Nombre de personnes reçues
5. Pour supprimer des messages, choisissez ceux que vous souhaitez supprimer, puis sélectionnez Supprimer.

   Confirmez la suppression dans la boîte de dialogue Supprimer les messages en sélectionnant Supprimer.

Pour plus de détails sur les opérations avancées, notamment la récupération et la suppression de messages basées sur des API, consultez le guide de référence des API Amazon SQS.

Confirmation qu'une file d'attente Amazon SQS est vide

Dans la plupart des cas, vous pouvez utiliser une recherche prolongée pour déterminer si une file d'attente est vide. Dans de rares cas, vous pouvez recevoir des réponses vides même si une file d'attente contient encore des messages, en particulier si vous avez spécifié une faible valeur pour le paramètre Temps d'attente du message de réception lorsque vous avez créé la file d'attente. Cette section explique comment vérifier qu'une file d'attente est vide.

Confirmer qu'une file d'attente est vide (console)

1. Empêchez tous les producteurs d'envoyer des messages.
2. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
3. Dans le volet de navigation, choisissez Files d'attente.

4. Sur la page Files d'attente, choisissez une file d'attente.
5. Sélectionnez l'onglet Monitoring (Surveillance).
6. En haut à droite des tableaux de bord de surveillance, cliquez sur la flèche vers le bas à côté du symbole Actualiser. Dans le menu déroulant, choisissez Actualisation automatique. Laissez l'intervalle d'actualisation sur 1 minute.
7. Observez les tableaux de bord suivants :

   - Nombre approximatif de messages retardés
   - Nombre approximatif de messages non visibles
   - Nombre approximatif de messages visibles

   Lorsque tous affichent des valeurs de `0` pendant plusieurs minutes, cela signifie que la file d'attente est vide.

Pour confirmer qu'une file d'attente est vide (AWS CLI, AWS API)

1. Empêchez tous les producteurs d'envoyer des messages.
2. Exécutez l'une des commandes suivantes à plusieurs reprises :

   - AWS CLI: `get-queue-attributes`
   - AWS API : `GetQueueAttributes`
3. Observez les métriques pour les attributs suivants :

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

   Lorsque tous affichent la valeur `0` pendant plusieurs minutes, cela signifie que la file d'attente est vide.

Si vous vous fiez aux CloudWatch statistiques d'Amazon, assurez-vous de voir plusieurs points de données nuls consécutifs avant de considérer que cette file d'attente est vide. Pour plus d'informations sur CloudWatch les métriques, consultez CloudWatch Métriques disponibles pour Amazon SQS.

Supprimer une file d'attente Amazon SQS

Si vous n'utilisez plus de file d'attente Amazon SQS et que vous ne prévoyez pas de l'utiliser dans un futur proche, supprimez-la.

> **Tip**
>
> Si vous souhaitez vérifier qu'une file d'attente est vide avant de la supprimer, consultez Confirmation qu'une file d'attente Amazon SQS est vide.

Vous pouvez supprimer une file d'attente, même lorsqu'elle n'est pas vide. Pour supprimer les messages d'une file d'attente, mais pas la file d'attente elle-même, purgez la file d'attente.

Pour supprimer une file d'attente (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Sur la page Files d'attente, choisissez la file d'attente à supprimer.
4. Sélectionnez Delete (Supprimer).
5. Dans la boîte de dialogue Supprimer la file d'attente, confirmez la suppression en saisissant **`delete`**.
6. Sélectionnez Delete (Supprimer).

Pour supprimer une file d'attente (AWS CLI et une API)

Choisissez la méthode appropriée pour supprimer votre file d'attente en fonction de vos besoins :

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

Purger les messages d'une file d'attente à l'aide de la console Amazon SQS

Pour conserver une file d'attente Amazon SQS tout en supprimant tous ses messages, vous pouvez la purger. Cela supprimera tous les messages, y compris ceux qui sont actuellement invisibles

(en cours de vol). Le processus de purge peut prendre jusqu'à 60 secondes, alors attendez les 60 secondes complètes, quelle que soit la taille de la file d'attente.

> **Important**
>
> Lorsque vous purgez une file d'attente, vous ne pouvez récupérer aucun des messages supprimés.

Pour purger une file d'attente (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Sur la page Files d'attente, choisissez la file d'attente à purger.
4. Dans Actions, choisissez Purger.
5. Dans la boîte de dialogue Purger la file d'attente, confirmez la purge en saisissant **`purge`** et en choisissant Purger.
   - Tous les messages sont purgés de la file d'attente. La console affiche une bannière de confirmation.

Files d'attente standard Amazon SQS

Amazon SQS fournit des files d'attente standard comme type de file d'attente par défaut, prenant en charge un nombre quasiment illimité d'appels d'API par seconde pour des actions telles que `SendMessage`, et `ReceiveMessage`. `DeleteMessage` Les files d'attente standard garantissent la livraison des at-least-once messages, mais en raison de l'architecture hautement distribuée, plusieurs copies d'un message peuvent être livrées, et les messages peuvent parfois arriver en désordre. Malgré cela, les files d'attente standard font de leur mieux pour maintenir l'ordre dans lequel les messages sont envoyés.

Lorsque vous envoyez un message en utilisant`SendMessage`, Amazon SQS le stocke de manière redondante dans plusieurs zones de disponibilité (AZs) avant de l'accuser réception. Cette redondance garantit qu'aucune panne d'ordinateur, de réseau ou d'AZ ne peut rendre les messages inaccessibles.

Vous pouvez créer et configurer des files d'attente à l'aide de la console Amazon SQS. Pour obtenir des instructions complètes, veuillez consulter Création d'une file d'attente standard à l'aide de la console Amazon SQS. Pour des exemples spécifiques à Java, voir. Exemples de SDK Java Amazon SQS

Cas d'utilisation pour les files d'attente standard

Les files d'attente de messages standard conviennent à différents scénarios, à condition que votre application puisse gérer les messages qui peuvent arriver plusieurs fois ou être hors ordre. En voici quelques exemples :

- Dissocier les demandes réelles des utilisateurs d'un travail de fond intensif : les utilisateurs peuvent télécharger du contenu multimédia pendant que le système le redimensionne ou l'encode en arrière-plan.
- Allocation de tâches à plusieurs nœuds de travail : par exemple, gestion d'un volume élevé de demandes de validation de cartes de crédit.
- Traitement par lots des messages pour un traitement futur : planification de plusieurs entrées à ajouter ultérieurement à une base de données.

Pour plus d'informations sur les quotas liés aux files d'attente standard, consultezQuotas de file d'attente standard Amazon SQS.

Pour connaître les bonnes pratiques d'utilisation des files d'attente standard, consultez Bonnes pratiques relatives à Amazon SQS.

Livraison via Amazon SQS at-least-once

Amazon SQS stocke des copies de vos messages sur plusieurs serveurs à des fins de redondance et de haute disponibilité. Dans de rares occasions, l'un des serveurs qui stockent la copie d'un message peut être indisponible lors de la réception ou de la suppression d'un message.

Dans ce cas, la copie du message n'est pas supprimée sur le serveur qui n'est pas disponible et il est possible que vous obteniez à nouveau cette copie lorsque vous recevrez des messages. Concevez les applications afin qu'elles soient idempotentes (c.-à-d. qu'elles ne doivent pas être affectées si le même message est traité plus d'une fois).

Identifiants de files d'attente et de messages Amazon SQS

Cette rubrique décrit les identifiants des files d'attente standard et FIFO. Ces identifiants peuvent vous aider à trouver et à manipuler des files d'attente et des messages spécifiques.

Identifiants pour les files d'attente standard Amazon SQS

Pour plus d'informations sur les identifiants suivants, consultez la Référence d'API Amazon Simple Queue Service.

Nom et URL de la file d'attente

Lorsque vous créez une file d'attente, vous devez indiquer un nom unique pour le compte et la région AWS . Amazon SQS attribue à chaque file d'attente que vous créez un identifiant appelé URL de file d'attente qui inclut le nom de la file d'attente et les autres composants Amazon SQS. Chaque fois que vous souhaitez effectuer une action au niveau d'une file d'attente, vous devez fournir cette URL.

L'URL suivante est celle d'une file d'attente nommée `MyQueue`, qui appartient à un utilisateur dont le numéro de compte AWS est `123456789012`.

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

Vous pouvez extraire l'URL d'une file d'attente par programmation en listant vos files d'attente et en analysant la chaîne qui suit le numéro de compte. Pour de plus amples informations, veuillez consulter `ListQueues`.

ID de message

Chaque message reçoit un ID de message attribué par le système qu'Amazon SQS vous renvoie dans la réponse `SendMessage`. Cet identifiant est utile pour identifier les messages. La longueur maximale d'un ID de message est de 100 caractères.

Descripteur de réception

Chaque fois que vous recevez un message d'une file d'attente, vous recevez un descripteur de réception correspondant. Cette gestion est associée à la réception du message, et non au message lui-même. Pour supprimer le message ou pour en modifier la visibilité, vous devez fournir le descripteur de réception (et non l'ID du message). C'est pourquoi vous devez toujours recevoir un message avant de pouvoir le supprimer (vous ne pouvez pas placer un message dans la file d'attente, puis le rappeler). La longueur maximale d'un descripteur de réception est de 1 024 caractères.

> **Important**
>
> Si vous recevez un message plusieurs fois, chaque fois que vous le recevez, vous obtenez un descripteur de réception différent. Lorsque vous demandez la suppression du message, vous devez fournir le descripteur de réception le plus récent. Dans le cas contraire, la suppression peut ne pas fonctionner.

Voici un exemple de descripteur de reçu divisé en trois lignes.


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

Files d'attente FIFO Amazon SQS

Les files d'attente FIFO (premier entré, premier sorti) offrent toutes les fonctionnalités des files d'attente standard, mais sont conçues pour améliorer la messagerie entre les applications lorsque l'ordre des opérations et des événements est critique ou lorsque les doublons ne peuvent être tolérés.

Les caractéristiques les plus importantes des files d'attente FIFO sont la livraison FIFO (premier entré, premier sorti) et le traitement en une seule fois :

- L'ordre dans lequel les messages sont envoyés et reçus est strictement préservé et un message est délivré une fois et reste indisponible jusqu'à ce qu'un consommateur le traite et le supprime.
- Aucun doublon n'est ajouté à la file d'attente.

En outre, les files d'attente FIFO prennent en charge les groupes de messages qui autorisent plusieurs groupes de messages ordonnés au sein d'une même file d'attente. Il n'y a pas de quota quant au nombre de groupes de messages dans une file d'attente FIFO.

Voici des exemples de situations dans lesquelles vous pourriez utiliser des files d'attente FIFO :

1. Système de gestion des commandes pour le commerce électronique où l’ordre est essentiel.
2. Intégration à un système tiers où les événements doivent être traités dans l'ordre
3. Traitement des entrées saisies par l'utilisateur dans l'ordre saisi
4. Communications et mise en réseau : envoi et réception de données et d'informations dans le même ordre
5. Systèmes informatiques : s'assurer que les commandes saisies par l'utilisateur sont exécutées dans le bon ordre
6. Établissements d'enseignement : empêcher un étudiant de s'inscrire à un cours avant d'avoir créé un compte
7. Système de billetterie en ligne : où les billets sont distribués selon le principe du premier arrivé, premier servi

> **Note**
>
> Les files d'attente FIFO fournissent également un traitement en une seule fois, mais avec un nombre limité de transactions par seconde (TPS). Vous pouvez utiliser le mode débit élevé d'Amazon SQS avec votre file d'attente FIFO pour augmenter votre limite de transactions. Pour plus de détails sur l'utilisation du mode débit élevé, consultez Débit élevé pour les files d'attente FIFO dans Amazon SQS. Pour plus d'informations sur les quotas de débit, consultez the section called “Quotas de messages”.

Les files d'attente FIFO Amazon SQS sont disponibles dans toutes les régions où Amazon SQS est disponible.

Pour en savoir plus sur l'utilisation des files d'attente FIFO pour les commandes complexes, consultez Résoudre les problèmes de commande complexes avec les files d'attente FIFO Amazon SQS.

Pour plus d'informations sur la création et la configuration de files d'attente à l'aide de la console Amazon SQS, consultez Création d'une file d'attente standard à l'aide de la console Amazon SQS. Pour obtenir des exemples Java, consultez Exemples de SDK Java Amazon SQS.

Pour connaître les meilleures pratiques relatives à l'utilisation des files d'attente FIFO, voir. Bonnes pratiques relatives à Amazon SQS

Termes clés de la file d'attente FIFO Amazon SQS

Les termes clés suivants peuvent vous aider à mieux comprendre les fonctionnalités des files d'attente FIFO. Pour plus d'informations, consultez la Référence d'API Amazon Simple Queue Service.

Clients

Le client asynchrone en mémoire tampon Amazon SQS ne prend actuellement pas en charge les files d'attente FIFO.

ID de déduplication du message

Un jeton utilisé dans les files d'attente FIFO Amazon SQS pour identifier les messages de manière unique et éviter les doublons. Si plusieurs messages portant le même ID de déduplication sont envoyés dans un intervalle de déduplication de 5 minutes, ils sont traités comme des

doublons et une seule copie est délivrée. Si vous ne spécifiez pas d'ID de déduplication et que la déduplication basée sur le contenu est activée, Amazon SQS génère un ID de déduplication en hachant le corps du message. Ce mécanisme garantit une livraison en une seule fois en éliminant les messages dupliqués dans les délais impartis.

> **Note**
>
> Amazon SQS continue de suivre l'ID de déduplication même après réception et suppression du message.

ID de groupe de messages

Dans les files d'attente FIFO (First-In-First-Out), se `MessageGroupId` trouve un attribut qui organise les messages en groupes distincts. Les messages d'un même groupe de messages sont toujours traités un par un, dans un ordre strict, afin que deux messages du même groupe ne soient pas traités simultanément. Dans les files d'attente standard, l'utilisation `MessageGroupId` permet d'établir des files d'attente équitables. Si un ordre strict est requis, utilisez une file d'attente FIFO.

ID de tentative de demande de réception

L'ID de tentative de demande de réception est un jeton unique utilisé pour dédupliquer les `ReceiveMessage`appels dans Amazon SQS.

Numéro de séquence

Le grand numéro non consécutif qu'Amazon SQS attribue à chaque message.

Services

Si votre application utilise plusieurs AWS services, ou une combinaison de AWS services externes, il est important de savoir quelle fonctionnalité de service ne prend pas en charge les files d'attente FIFO.

Certains services AWS ou services externes qui envoient des notifications à Amazon SQS peuvent ne pas être compatibles avec les files d'attente FIFO, même si vous pouvez définir une file d'attente FIFO comme cible.

Les fonctionnalités des AWS services suivantes ne sont actuellement pas compatibles avec les files d'attente FIFO :

- Notifications d'événements Amazon S3
- Hooks de cycle de vie autoscaling
- AWS IoT Actions relatives aux règles
- AWS Lambda Files d'attente de lettres mortes

Pour plus d'informations sur la compatibilité d'autres services avec les files d'attente FIFO, consultez la documentation de votre service.

Logique de distribution des files d'attente FIFO dans Amazon SQS

Les concepts suivants expliquent comment les files d'attente FIFO Amazon SQS gèrent l'envoi et la réception de messages, en particulier lorsqu'il s'agit de l'ordre des messages et des groupes de messages. IDs

Envoi de messages

Les files d'attente FIFO Amazon SQS préservent l'ordre des messages grâce à une IDs déduplication et à un groupe de messages uniques. IDs Cette rubrique met en lumière l'importance du groupe de messages IDs pour maintenir un ordre strict au sein des groupes et met en évidence les meilleures pratiques pour garantir une diffusion fiable et ordonnée des messages entre plusieurs producteurs.

1. Conservation des commandes
   - Lorsque plusieurs messages sont envoyés successivement à une file d'attente FIFO avec déduplication unique des messages IDs, Amazon SQS les stocke et accuse réception de leur transmission. Ces messages sont ensuite reçus et traités dans l'ordre exact dans lequel ils ont été transmis.
2. ID de groupe de messages
   - Dans les files d'attente FIFO, les messages sont classés en fonction de leur ID de groupe de messages. Si plusieurs producteurs ou fils de discussion envoient des messages avec le même identifiant de groupe de messages, Amazon SQS garantit qu'ils sont stockés et traités dans l'ordre dans lequel ils arrivent.
   - Bonne pratique : pour garantir un ordre strict des messages entre plusieurs producteurs, attribuez un identifiant de groupe de messages unique à tous les messages de chaque producteur.
3. Commande par groupe

- La logique de file d'attente FIFO s'applique par ID de groupe de messages :
  - Chaque ID de groupe de messages représente un groupe de messages distinct et ordonné.
  - Au sein d'un ID de groupe de messages, tous les messages sont envoyés et reçus dans un ordre strict.
  - Les messages appartenant à des groupes de messages différents IDs peuvent arriver ou être traités dans un ordre désordonné les uns par rapport aux autres.
- Exigence - Vous devez associer un ID de groupe de messages à chaque message. Si un message est envoyé sans identifiant de groupe, l'action échoue.
- Scénario de groupe unique : si vous souhaitez que tous les messages soient traités dans un ordre strict, utilisez le même identifiant de groupe de messages pour chaque message.

Réception de messages

Les files d'attente FIFO Amazon SQS gèrent la récupération des messages, y compris le traitement par lots, les garanties des commandes FIFO et les limites relatives aux demandes de groupes de messages spécifiques. IDs Cette rubrique explique comment Amazon SQS récupère les messages au sein et entre les groupes de messages IDs tout en respectant des règles strictes de classement et de visibilité.

1. Récupération par lots
   - Lors de la réception de messages provenant d'une file d'attente FIFO comportant plusieurs groupes de messages IDs, Amazon SQS :
     - Tente de renvoyer autant de messages que possible avec le même ID de groupe de messages au cours d'un seul appel.
     - Permet aux autres consommateurs de traiter IDs simultanément les messages provenant de différents groupes de messages.
   - Clarification importante
     - Vous pouvez recevoir plusieurs messages du même identifiant de groupe de messages en un seul lot (jusqu'à 10 messages par appel en utilisant le `MaxNumberOfMessages` paramètre).
     - Cependant, vous ne pouvez pas recevoir de messages supplémentaires provenant du même identifiant de groupe de messages dans les demandes suivantes tant que :
       - Les messages actuellement reçus sont supprimés, ou
       - Ils redeviennent visibles (par exemple, une fois le délai de visibilité expiré).
2. Garantie de commande FIFO

- Les messages extraits par lots conservent leur ordre FIFO au sein du groupe.
- Si moins de 10 messages sont disponibles pour le même identifiant de groupe de messages, Amazon SQS peut inclure des messages provenant d'autres groupes de messages IDs dans le même lot, mais chaque groupe conserve l'ordre FIFO.

3. Limites applicables aux consommateurs
   - Vous ne pouvez pas demander explicitement à recevoir des messages provenant d'un identifiant de groupe de messages spécifique.

Multiples nouvelles tentatives

Les producteurs et les consommateurs peuvent réessayer en toute sécurité les actions qui ont échoué dans les files d'attente FIFO Amazon SQS sans perturber l'ordre des messages ni introduire de doublons. Cette rubrique explique comment les délais de déduplication IDs et de visibilité garantissent l'intégrité des messages lors des nouvelles tentatives.

1. Réessaie le producteur
   - Si une `SendMessage`action échoue, le producteur peut réessayer d'envoyer le message plusieurs fois avec le même identifiant de déduplication.
   - Tant que le producteur reçoit au moins un accusé de réception avant l'expiration de l'intervalle de déduplication, il réessaie :
     - N'introduisez pas de messages dupliqués.
     - Ne perturbez pas l'ordre des messages.
2. Réessaie par le consommateur
   - Si une `ReceiveMessage`action échoue, le consommateur peut réessayer autant de fois que nécessaire en utilisant le même identifiant de tentative de demande de réception.
   - Tant que le consommateur reçoit au moins un accusé de réception avant l'expiration du délai de visibilité, il réessaie :
     - Ne perturbez pas l'ordre des messages.

Remarques supplémentaires sur le comportement du FIFO

Découvrez comment gérer les délais de visibilité, activer le traitement parallèle avec plusieurs groupes de messages et garantir un traitement séquentiel strict dans les scénarios à groupe IDs unique.

1. Gestion du délai de visibilité
   - Lorsqu'un message est récupéré mais non supprimé, il reste invisible jusqu'à l'expiration du délai de visibilité.
   - Aucun message supplémentaire provenant du même identifiant de groupe de messages n'est renvoyé tant que le premier message n'est pas supprimé ou ne redevient visible.
2. Traitement simultané et parallélisme
   - Les files d'attente FIFO permettent le traitement parallèle des messages entre différents groupes de messages. IDs
   - Pour optimiser la simultanéité, concevez votre système avec plusieurs groupes de messages IDs pour des flux de travail indépendants.
3. Scénarios de groupe unique
   - Pour un traitement séquentiel strict de tous les messages d'une file d'attente FIFO, utilisez un identifiant de groupe de messages unique pour tous les messages de la file d'attente.

Exemples pour mieux comprendre

Les scénarios pratiques suivants illustrent le comportement des files d'attente FIFO dans Amazon SQS.

1. Scénario 1 : ID de groupe unique
   - Un producteur envoie cinq messages avec le même identifiant de groupe de messages (groupe A).
   - Le consommateur reçoit ces messages dans l'ordre FIFO. Jusqu'à ce que le consommateur supprime ces messages ou que le délai de visibilité expire, aucun message supplémentaire du groupe A n'est reçu.
2. Scénario 2 : groupe multiple IDs
   - Un producteur envoie cinq messages au groupe A et 5 au groupe B.
   - Le consommateur 1 traite les messages du groupe A, tandis que le consommateur 2 traite les messages du groupe B. Cela permet un traitement parallèle avec un ordre strict maintenu au sein de chaque groupe.
3. Scénario 3 : extraction par lots
   - Un producteur envoie sept messages au groupe A et trois au groupe B.
   - Un seul client récupère jusqu'à 10 messages. Si la file d'attente le permet, elle peut renvoyer :

- Sept messages du groupe A et trois du groupe B (ou moins si moins de messages sont disponibles dans un seul groupe).

Traitement effectué en une seule fois dans Amazon SQS

Contrairement aux files d'attente standard, les files d'attente FIFO n'introduisent pas de messages en double. Les files d'attente FIFO vous aident à éviter d'envoyer des doublons à une file d'attente. Si vous réessayez l'action `SendMessage` dans un délai de déduplication de 5 minutes, Amazon SQS n'ajoute pas de doublons dans la file d'attente.

Pour configurer une déduplication, vous devez effectuer l'une des actions suivantes :

- Activer la déduplication basée sur le contenu. Amazon SQS reçoit l'ordre d'utiliser un hachage SHA-256 pour générer l'ID de déduplication du message en utilisant le corps de celui-ci, mais pas ses attributs. Pour plus d'informations, consultez la documentation sur les actions `CreateQueue`, `GetQueueAttributes` et `SetQueueAttributes` dans la Référence d'API Amazon Simple Queue Service.
- Fournir explicitement l'ID de déduplication du message (ou afficher le numéro de séquence) pour le message. Pour plus d'informations, consultez la documentation sur les actions `SendMessage`, `SendMessageBatch` et `ReceiveMessage` dans la Référence d'API Amazon Simple Queue Service.

Passage d'une file d'attente standard à une file d'attente FIFO dans Amazon SQS

Si votre application existante utilise des files d'attente standard et que vous souhaitez tirer parti des fonctionnalités de commande ou de traitement en une seule fois des files d'attente FIFO, vous devez configurer correctement la file d'attente et votre application.

Considérations clés

- Création d'une file d'attente FIFO : vous ne pouvez pas convertir une file d'attente standard existante en file d'attente FIFO. Vous devez soit créer une nouvelle file d'attente FIFO pour votre application, soit supprimer la file d'attente standard existante et la recréer en tant que file d'attente FIFO.

- Paramètre de délai : les files d'attente FIFO ne prennent pas en charge les délais par message, mais uniquement les délais par file d'attente. Si votre application définit le `DelaySeconds` paramètre pour chaque message, vous devez le modifier pour le définir `DelaySeconds` sur l'ensemble de la file d'attente.
- ID de groupe de messages : fournissez un identifiant de groupe de messages pour chaque message envoyé. Cet identifiant permet le traitement parallèle des messages tout en conservant leur ordre respectif. Utilisez une dimension commerciale granulaire pour l'identifiant du groupe de messages afin de mieux l'adapter aux files d'attente FIFO. Plus les groupes de messages IDs auxquels vous distribuez des messages sont nombreux, plus le nombre de messages disponibles à la consommation augmente.
- Mode haut débit : utilisez le mode haut débit recommandé pour les files d'attente FIFO afin d'augmenter le débit. Pour plus d'informations sur les quotas de messagerie, consultezQuotas de messages Amazon SQS.

Liste de contrôle pour passer aux files d'attente FIFO

Avant d'envoyer des messages à une file d'attente FIFO, confirmez ce qui suit :

1. Configuration des paramètres de délai
   - Modifiez votre application pour supprimer les délais par message.
   - Définissez le `DelaySeconds` paramètre sur l'ensemble de la file d'attente.
2. Définir un groupe de messages IDs
   - Organisez les messages en groupes de messages en spécifiant un ID de groupe de messages basé sur une dimension commerciale.
   - Utilisez des dimensions commerciales plus granulaires pour améliorer l'évolutivité.
3. Gérer la déduplication des messages
   - Si votre application ne peut pas envoyer de messages avec des corps de message identiques, fournissez un identifiant de déduplication unique pour chaque message.
   - Si votre application envoie des messages avec des corps de message uniques, activez la déduplication basée sur le contenu.
4. Configurer le consommateur
   - En général, aucune modification de code n'est nécessaire pour le consommateur.
   - Si le traitement des messages prend du temps et que le délai de visibilité est élevé, pensez à ajouter un identifiant de tentative de demande de réception à chaque `ReceiveMessage` action.

Cela permet de réessayer les tentatives de réception en cas de défaillance du réseau et d'éviter que les files d'attente ne s'interrompent en cas d'échec des tentatives de réception.

En suivant ces étapes, vous pouvez vous assurer que votre application fonctionne correctement avec les files d'attente FIFO, en tirant pleinement parti de leurs fonctionnalités de commande et de traitement en une seule fois. Pour des informations plus détaillées, consultez le manuel Amazon Simple Queue Service API Reference.

Queue FIFO Amazon SQS et comportement de simultanéité Lambda

En utilisant une file d'attente FIFO (premier entré, premier sorti) avec Lambda, vous pouvez garantir un traitement ordonné des messages au sein de chaque groupe de messages. La fonction Lambda n'exécute pas plusieurs instances simultanément pour le même groupe de messages, ce qui permet de maintenir l'ordre. Cependant, il peut être étendu pour gérer plusieurs groupes de messages en parallèle, garantissant ainsi un traitement efficace de la charge de travail de votre file d'attente. Les points suivants décrivent le comportement des fonctions Lambda lors du traitement de messages provenant d'une file d'attente FIFO Amazon SQS par rapport au groupe de messages : IDs

- Instance unique par groupe de messages : à tout moment, une seule instance Lambda traitera les messages provenant d'un ID de groupe de messages spécifique. Cela garantit que les messages au sein d'un même groupe sont traités dans l'ordre, tout en préservant l'intégrité de la séquence FIFO.
- Traitement simultané de différents groupes : Lambda peut traiter simultanément des messages provenant de différents groupes de messages IDs en utilisant plusieurs instances. Cela signifie que lorsqu'une instance de la fonction Lambda gère les messages provenant d'un identifiant de groupe de messages, d'autres instances peuvent gérer simultanément les messages d'un autre groupe de messages IDs, en tirant parti des capacités de simultanéité de Lambda pour traiter plusieurs groupes en parallèle.

Regroupement de messages de file d'attente FIFO

Les files d'attente FIFO garantissent que les messages sont traités dans l'ordre exact dans lequel ils sont envoyés. Ils utilisent un ID de groupe de messages pour regrouper les messages qui doivent être traités de manière séquentielle.

Les messages d'un même groupe de messages sont traités dans l'ordre, et un seul message de chaque groupe est traité à la fois pour maintenir cet ordre.

Simultanéité Lambda avec les files d'attente FIFO

Après avoir créé votre file d'attente, vous pouvez lui envoyer un message.

Lorsque vous configurez une fonction Lambda pour traiter les messages provenant d'une file d'attente FIFO Amazon SQS, Lambda respecte les garanties de commande fournies par la file d'attente FIFO. Les points suivants décrivent le comportement des fonctions Lambda en termes de simultanéité et de dimensionnement lors du traitement de messages provenant d'une file d'attente FIFO Amazon SQS lors de l'utilisation d'un groupe de messages. IDs

- Concurrence au sein des groupes de messages : une seule instance Lambda traite les messages pour un ID de groupe de messages spécifique à la fois. Cela garantit que les messages au sein d'un groupe sont traités de manière séquentielle.
- Dimensionnement et groupes de messages multiples : Lambda peut évoluer pour traiter les messages simultanément, mais cette mise à l'échelle s'effectue entre différents groupes de messages. Si vous avez plusieurs groupes de messages, Lambda peut traiter plusieurs groupes en parallèle, chaque groupe étant géré par une instance Lambda distincte.

Pour plus d'informations, consultez la section Mise à l'échelle et simultanéité dans Lambda dans AWS Lambda le Guide de l'opérateur.

Exemple de cas d'utilisation

Supposons que votre file d'attente FIFO reçoive des messages avec le même ID de groupe de messages et que votre fonction Lambda dispose d'une limite de simultanéité élevée (jusqu'à 1000).

Si un message de l'ID de groupe « A » est en cours de traitement et qu'un autre message de l'ID de groupe « A » arrive, le second message ne déclenchera pas de nouvelle instance Lambda tant que le premier message ne sera pas entièrement traité.

Toutefois, si les messages des groupes IDs « A » et « B » arrivent, les deux messages peuvent être traités simultanément par des instances Lambda distinctes.

Débit élevé pour les files d'attente FIFO dans Amazon SQS

Les files d'attente FIFO à haut débit d'Amazon SQS gèrent efficacement le débit élevé de messages tout en maintenant un ordre strict des messages, garantissant ainsi la fiabilité et l'évolutivité des applications traitant de nombreux messages. Cette solution est idéale pour les scénarios exigeant à la fois un débit élevé et une livraison ordonnée des messages.

Les files d'attente FIFO à haut débit Amazon SQS ne sont pas nécessaires dans les scénarios où un ordre strict des messages n'est pas crucial et où le volume de messages entrants est relativement faible ou sporadique. Par exemple, si vous avez une application à petite échelle qui traite des messages peu fréquents ou non séquentiels, la complexité et les coûts supplémentaires associés aux files d'attente FIFO à haut débit peuvent ne pas être justifiés. En outre, si votre application n'a pas besoin des capacités de débit améliorées fournies par les files d'attente FIFO à haut débit, opter pour une file d'attente Amazon SQS standard peut s'avérer plus rentable et plus simple à gérer.

Pour améliorer la capacité de demande dans les files d'attente FIFO à haut débit, il est recommandé d'augmenter le nombre de groupes de messages. Pour plus d'informations sur les quotas de messages à débit élevé, consultez Quotas du service Amazon SQS dans le Référence générale d'Amazon Web Services

Pour plus d'informations sur les quotas par file d'attente et les stratégies de distribution des données, reportez-vous aux sections Quotas de messages Amazon SQS etPartitions et distribution de données pour un débit élevé pour les files d'attente FIFO SQS.

Cas d'utilisation du débit élevé pour les files d'attente FIFO Amazon SQS

Les cas d'utilisation suivants mettent en évidence les diverses applications des files d'attente FIFO à haut débit, démontrant leur efficacité dans tous les secteurs et dans tous les scénarios :

1. Traitement des données en temps réel : les applications traitant des flux de données en temps réel, tels que le traitement d'événements ou l'ingestion de données de télémétrie, peuvent bénéficier de files d'attente FIFO à haut débit pour gérer l'afflux continu de messages tout en préservant leur ordre pour une analyse précise.
2. Traitement des commandes de commerce électronique : sur les plateformes de commerce électronique où le maintien de l'ordre des transactions des clients est essentiel, les files d'attente FIFO à haut débit garantissent que les commandes sont traitées de manière séquentielle et sans délai, même pendant les périodes de pointe des achats.

3. Services financiers : les institutions financières traitant des transactions ou des transactions à haute fréquence s'appuient sur des files d'attente FIFO à haut débit pour traiter les données de marché et les transactions avec une latence minimale, tout en respectant des exigences réglementaires strictes en matière de commande des messages.
4. Streaming multimédia : les plateformes de streaming et les services de distribution multimédia utilisent des files d'attente FIFO à haut débit pour gérer la diffusion des fichiers multimédia et du contenu en streaming, garantissant ainsi des expériences de lecture fluides aux utilisateurs tout en maintenant le bon ordre de diffusion du contenu.

Partitions et distribution de données pour un débit élevé pour les files d'attente FIFO SQS

Amazon SQS stocke les données des files d'attente FIFO dans des partitions. Une partition est une allocation de stockage pour une file d'attente qui est automatiquement répliquée sur plusieurs zones de disponibilité au sein d'une AWS région. Vous ne gérez pas les partitions. Amazon SQS gère la gestion des partitions.

Pour les files d'attente FIFO, Amazon SQS modifie le nombre de partitions dans une file d'attente dans les situations suivantes :

- Si le taux de demandes actuel approche ou dépasse la limite de prise en charge des partitions existantes, des partitions supplémentaires sont allouées jusqu'à ce que la file d'attente atteigne le quota régional. Pour plus d'informations sur les quotas, consultez Quotas de messages Amazon SQS.
- Si les partitions actuelles sont peu utilisées, le nombre de partitions peut être réduit.

La gestion de la partition s'effectue automatiquement à l'arrière-plan et est transparente pour vos applications. Votre file d'attente et vos messages sont disponibles à tout moment.

Distribution des données par groupe de messages IDs

Pour ajouter un message à une file d'attente FIFO, Amazon SQS utilise la valeur de l'ID de groupe de messages de chaque message comme entrée dans une fonction de hachage interne. La valeur de sortie de la fonction de hachage détermine la partition dans laquelle le message sera stocké.

Le diagramme suivant illustre une file d'attente qui s'étend sur plusieurs partitions. L'ID du groupe de messages de la file d'attente est basé sur le numéro d'élément. Amazon SQS utilise sa fonction de

hachage pour déterminer où stocker un nouvel élément, en l'occurrence, en fonction de la valeur de hachage de la chaîne `item0`. Notez que les éléments sont stockés dans le même ordre dans lequel ils sont ajoutés à la file d'attente. L'emplacement de chaque élément est déterminé par la valeur de hachage de son ID de groupe de messages.

> **Note**
>
> Amazon SQS est optimisé pour une distribution uniforme des éléments entre les partitions d'une file d'attente FIFO, quel que soit le nombre de partitions. AWS recommande d'utiliser un groupe de messages IDs pouvant comporter un grand nombre de valeurs distinctes.

Optimisation de l'utilisation des partitions

Chaque partition peut prendre en charge jusqu'à 3 000 messages par seconde avec le traitement par lots, ou jusqu'à 300 messages par seconde pour les opérations d'envoi, de réception et de suppression dans les régions prises en charge. Pour plus d'informations sur les quotas de messages à débit élevé, consultez Quotas du service Amazon SQS dans le Référence générale d'Amazon Web Services

Lorsque vous utilisez le traitement par lots APIs, chaque message est acheminé selon le processus décrit dansDistribution des données par groupe de messages IDs. Les messages acheminés vers la même partition sont regroupés et traités en une seule transaction.

Pour optimiser l'utilisation des partitions pour l'`SendMessageBatch`API, il est AWS recommandé de regrouper les messages avec le même groupe de messages IDs lorsque cela est possible.

Pour optimiser l'utilisation des partitions pour `DeleteMessageBatch` et `ChangeMessageVisibilityBatch` APIs, il est AWS recommandé d'utiliser des `ReceiveMessage` requêtes dont le `MaxNumberOfMessages` paramètre est défini sur 10 et de regrouper par lots les descripteurs de réception renvoyés par une seule demande. `ReceiveMessage`

Dans l'exemple suivant, un lot de messages contenant différents groupes de messages IDs est envoyé. Le lot est divisé en trois groupes, chacun étant pris en compte dans le quota de la partition.

> **Note**
>
> Amazon SQS garantit uniquement que les messages dotés de la fonction de hachage interne du même ID de groupe de messages sont regroupés au sein d'une demande groupée.
> En fonction de la sortie de la fonction de hachage interne et du nombre de partitions, les messages contenant des groupes de messages différents IDs peuvent être regroupés.
> Comme la fonction de hachage ou le nombre de partitions peut changer à tout moment, les messages groupés à un moment donné ne le seront peut-être pas ultérieurement.

Permettre un débit élevé pour les files d'attente FIFO dans Amazon SQS

Vous pouvez activer le débit élevé pour toute file d'attente FIFO nouvelle ou existante. Cette fonctionnalité inclut trois nouvelles options lorsque vous créez et modifiez des files d'attente FIFO :

- Activer le FIFO à haut débit : met à disposition du débit plus élevé pour les messages de la file d'attente FIFO actuelle.
- Portée de la déduplication : spécifie si la déduplication a lieu au niveau de la file d'attente ou du groupe de messages.
- Limite de débit FIFO : spécifie si le quota de débit des messages de la file d'attente FIFO est défini au niveau de la file d'attente ou du groupe de messages.

Pour activer un débit élevé pour une file d'attente FIFO (console)

1. Commencez à créer ou à modifier une file d'attente FIFO.
2. Lorsque vous spécifiez les options pour la file d'attente, choisissez Activer le FIFO à haut débit.

   L'activation du débit élevé pour les files d'attente FIFO définit les options associées comme suit :

   - La Portée de la déduplication est définie sur Groupe de messages, le paramètre requis pour utiliser le débit élevé pour les files d'attente FIFO.
   - La Limite de débit FIFO est définie sur Par ID de groupe de messages, le paramètre requis pour utiliser le débit élevé pour les files d'attente FIFO.

   Si vous modifiez l'un des paramètres requis pour utiliser le débit élevé pour les files d'attente FIFO, le débit normal est effectif pour la file d'attente et la déduplication se produit comme indiqué.
3. Continuez à spécifier toutes les options pour la file d'attente. Lorsque vous avez terminé, choisissez Créer une file d'attente ou Enregistrer.

Après avoir créé ou modifié la file d'attente FIFO, vous pouvez lui envoyer des messages, en recevoir et en supprimer, le tout à un TPS plus élevé. Pour connaître les quotas de débit élevé, consultez la section Débit des messages dans Quotas de messages Amazon SQS.

Identifiants de file d'attente et de message FIFO dans Amazon SQS

Cette section décrit les identifiants des files d'attente FIFO. Ces identifiants peuvent vous aider à trouver et à manipuler des files d'attente et des messages spécifiques.

Identifiants pour les files d'attente FIFO dans Amazon SQS

Pour plus d'informations sur les identifiants suivants, consultez la Référence d'API Amazon Simple Queue Service.

Nom et URL de la file d'attente

Lorsque vous créez une file d'attente, vous devez indiquer un nom unique pour le compte et la région AWS . Amazon SQS attribue à chaque file d'attente que vous créez un identifiant appelé URL de file d'attente qui inclut le nom de la file d'attente et les autres composants Amazon SQS. Chaque fois que vous souhaitez effectuer une action au niveau d'une file d'attente, vous devez fournir cette URL.

Le nom d'une file d'attente FIFO doit se terminer par le suffixe `.fifo`. Le suffixe est pris en compte dans le quota de 80 caractères pour les noms de file d'attente. Pour déterminer si une file d'attente est de type FIFO, vous pouvez vérifier si son nom se termine par le suffixe.

Voici l'URL d'une file d'attente FIFO nommée `MyQueue` appartenant à un utilisateur possédant le numéro `123456789012` de compte AWS.

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

Vous pouvez extraire l'URL d'une file d'attente par programmation en listant vos files d'attente et en analysant la chaîne qui suit le numéro de compte. Pour de plus amples informations, veuillez consulter `ListQueues`.

ID de message

Chaque message reçoit un ID de message attribué par le système qu'Amazon SQS vous renvoie dans la réponse `SendMessage`. Cet identifiant est utile pour identifier les messages. La longueur maximale d'un ID de message est de 100 caractères.

Descripteur de réception

Chaque fois que vous recevez un message d'une file d'attente, vous recevez un descripteur de réception correspondant. Cette gestion est associée à la réception du message, et non au message lui-même. Pour supprimer le message ou pour en modifier la visibilité, vous devez fournir le descripteur de réception (et non l'ID du message). C'est pourquoi vous devez toujours recevoir un message avant de pouvoir le supprimer (vous ne pouvez pas placer un message dans la file d'attente, puis le rappeler). La longueur maximale d'un descripteur de réception est de 1 024 caractères.

> **Important**
>
> Si vous recevez un message plusieurs fois, chaque fois que vous le recevez, vous obtenez un descripteur de réception différent. Lorsque vous demandez la suppression du message, vous devez fournir le descripteur de réception le plus récent. Dans le cas contraire, la suppression peut ne pas fonctionner.

Voici un exemple de descripteur de réception (réparti sur trois lignes).


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


Identifiants supplémentaires pour les files d'attente FIFO Amazon SQS

Pour plus d'informations sur les identifiants suivants, consultez Traitement effectué en une seule fois dans Amazon SQS et la Référence d'API Amazon Simple Queue Service.

ID de déduplication du message

Un jeton utilisé dans les files d'attente FIFO Amazon SQS pour identifier les messages de manière unique et éviter les doublons. Si plusieurs messages portant le même ID de déduplication sont envoyés dans un intervalle de déduplication de 5 minutes, ils sont traités comme des doublons et une seule copie est délivrée. Si vous ne spécifiez pas d'ID de déduplication et que la déduplication basée sur le contenu est activée, Amazon SQS génère un ID de déduplication en hachant le corps du message. Ce mécanisme garantit une livraison en une seule fois en éliminant les messages dupliqués dans les délais impartis.

ID de groupe de messages

`MessageGroupId`Il s'agit d'un attribut utilisé uniquement dans les files d'attente FIFO (First-In-First-Out) Amazon SQS pour organiser les messages en groupes distincts. Les messages d'un même groupe de messages sont toujours traités un par un, dans un ordre strict, afin que deux messages du même groupe ne soient pas traités simultanément. Les files d'attente standard ne sont pas utilisées `MessageGroupId` et ne fournissent aucune garantie de commande. Si un ordre strict est requis, utilisez plutôt une file d'attente FIFO.

Numéro de séquence

Le grand numéro non consécutif qu'Amazon SQS attribue à chaque message.

Quotas Amazon SQS

Cette rubrique explique les quotas et les limites pour Amazon SQS FIFO et les files d'attente standard, en détaillant leur impact sur la création de files d'attente, la configuration et le traitement des messages. Découvrez les contraintes telles que les limites de rétention des messages, les plafonds de messages en cours de transfert et les seuils de débit, ainsi que les stratégies visant à optimiser l'efficacité grâce au traitement par lots, à l'optimisation des appels d'API et aux longs sondages. Cette rubrique couvre également les conventions de dénomination, les règles de balisage et les méthodes permettant de demander des augmentations de quotas pour répondre aux charges de travail les plus exigeantes, afin de garantir une gestion efficace des files d'attente et des performances optimales.

Quotas de file d'attente FIFO Amazon SQS

Quotas Amazon SQS

Le tableau suivant répertorie les quotas relatifs aux files d'attente FIFO.

Quota	Description
File d'attente à retardement	Le délai (minimum) par défaut pour une file d'attente est de 0 seconde. La valeur maximale est de 15 minutes.
Files d'attente répertoriées	1000 files d'attente par demande `ListQueues` .
Durée d'attente pour interrogation longue	Le temps d'attente maximal pour la recherche prolongée est de 20 secondes.
Groupes de messages	Il n'y a pas de quota quant au nombre de groupes de messages dans une file d'attente FIFO.
Messages par file d'attente (en attente)	Le nombre de messages qu'une file d'attente Amazon SQS peut stocker est illimité.
Messages par file d'attente (en transit)	Les files d'attente FIFO prennent en charge un maximum de 120 000 messages en cours de vol (messages reçus par un consommateur mais pas encore supprimés). Si

Quota	Description
	cette limite est atteinte, Amazon SQS ne renvoie pas d'erreur, mais le traitement peut en être affecté. Vous pouvez demander une augmentation au-delà de cette limite en contactant le AWS Support.
Nom de la file d'attente	Le nom d'une file d'attente FIFO doit se terminer par le suffixe `.fifo`. Le suffixe est pris en compte dans le quota de 80 caractères pour les noms de file d'attente . Pour déterminer si une file d'attente est de type FIFO, vous pouvez vérifier si son nom se termine par le suffixe.
Balise de file d'attente	Nous vous déconseillons d'ajouter plus de 50 balises à une file d'attente. Le balisage prend en charge les caractères Unicode en UTF-8.
	La balise `Key` est obligatoire, mais la balise `Value` est facultative.
	La balise `Key` et la balise `Value` sont sensibles à la casse.
	La balise `Key` et la balise `Value` peuvent inclure des caractères alphanumériques Unicode en UTF-8 et des espaces blancs. Les caractères spéciaux suivants sont acceptés : `_ . : / = + - @`
	La balise `Key` ou `Value` ne doit pas inclure le préfixe réservé `aws:` (vous ne pouvez pas supprimer les clés de balise ou les valeurs ayant ce préfixe).
	La longueur `Key` de balise maximale est de 128 caractère s Unicode en UTF-8. La balise `Key` ne doit pas être vide ou null.
	La longueur `Value` de balise maximale est de 256 caractères Unicode en UTF-8. La balise `Value` peut être vide ou null.

Quota	Description
	Les actions de balisage sont limitées à 30 TPS par action. Compte AWS Si votre application nécessite un débit plus élevé, soumettez une demande.

Quotas de file d'attente standard Amazon SQS

Le tableau suivant répertorie les quotas relatifs aux files d'attente standard.

Quota	Description
File d'attente à retardement	Le délai (minimum) par défaut pour une file d'attente est de 0 seconde. La valeur maximale est de 15 minutes.
Files d'attente répertoriées	1000 files d'attente par demande `ListQueues` .
Durée d'attente pour interrogation longue	Le temps d'attente maximal pour la recherche prolongée est de 20 secondes.
Messages par file d'attente (en attente)	Le nombre de messages qu'une file d'attente Amazon SQS peut stocker est illimité.
Messages par file d'attente (en transit)	Pour la plupart des files d'attente standard (en fonction du trafic de la file d'attente et du backlog de messages), il peut y avoir un maximum d'environ 120 000 messages en cours (reçus depuis une file d'attente par un consommat eur, mais pas encore supprimés de la file d'attente). Si vous atteignez ce quota tout en utilisant la recherche courte, Amazon SQS renvoie le message d'erreur `OverLimit` . Si vous utilisez la recherche prolongée, Amazon SQS ne renvoie aucun message d'erreur. Pour éviter d'atteindre cette limite, supprimez les messages de la file d'attente une fois qu'ils ont été traités. Vous pouvez également augmenter le nombre de files d'attente que vous utilisez pour traiter vos messages. Pour demander

Quota	Description
	une augmentation de quota, envoyez une demande de support.
Nom de la file d'attente	Le nom des files d'attente peut contenir jusqu'à 80 caractères. Les caractères suivants sont acceptés : caractères alphanumériques, tirets (`-`) et traits de soulignement (_). **Note** Les noms de file d'attente sont sensible à la casse (par exemple, `Test-queue` et `test-queue` désignent des files d'attente différentes).
Balise de file d'attente	Nous vous déconseillons d'ajouter plus de 50 balises à une file d'attente. Le balisage prend en charge les caractères Unicode en UTF-8.
	La balise `Key` est obligatoire, mais la balise `Value` est facultative.
	La balise `Key` et la balise `Value` sont sensibles à la casse.
	La balise `Key` et la balise `Value` peuvent inclure des caractères alphanumériques Unicode en UTF-8 et des espaces blancs. Les caractères spéciaux suivants sont acceptés : `_ . : / = + - @`
	La balise `Key` ou `Value` ne doit pas inclure le préfixe réservé `aws:` (vous ne pouvez pas supprimer les clés de balise ou les valeurs ayant ce préfixe).
	La longueur `Key` de balise maximale est de 128 caractère s Unicode en UTF-8. La balise `Key` ne doit pas être vide ou null.

Quota	Description
	La longueur `Value` de balise maximale est de 256 caractères Unicode en UTF-8. La balise `Value` peut être vide ou null.
	Les actions de balisage sont limitées à 30 TPS par action. Compte AWS Si votre application nécessite un débit plus élevé, soumettez une demande.

Quotas de messages Amazon SQS

Le tableau suivant répertorie les quotas relatifs aux messages.

Quota	Description
ID de message par lots	Un identifiant de message groupé peut comporter jusqu'à 80 caractères. Les caractères suivants sont acceptés : caractères alphanumériques, tirets (`-`) et traits de soulignement (`_`).
Attributs de message	Un message peut contenir jusqu'à 10 attributs de métadonnées.
Traitement par lots des messages	Une demande de traitement par lots de messages peut inclure un maximum de 10 messages. Pour plus d'informations, consultez Configuration d'Amazon SQSBuffered AsyncClient dans la section Actions groupées Amazon SQS.
Contenu des messages	Un message peut contenir uniquement du texte XML ou JSON et du texte non formaté. Les caractères Unicode suivants sont acceptés : `#x9` | `#xA` | `#xD` | `#x20` to `#xD7FF` | `#xE000` à `#xFFFD` | `#x10000` à `#x10FFFF`

Quota	Description
	Tous les caractères non inclus dans cette liste sont refusés. Pour plus d'informations, consultez la spécification W3C en matière de caractères.
ID de groupe de messages	`MessageGroupId` est obligatoire pour les files d'attente FIFO. Si vous n'en fournissez pas `MessageGroupId` lors de l'envoi d'un message à une file d'attente FIFO, l'action échoue. Dans les files d'attente standard, l'utilisation `MessageGroupId` permet d'établir des files d'attente équitables. Nous vous recommandons d'inclure un `MessageGroupId` dans tous les messages lorsque vous utilisez des files d'attente équitables. La longueur maximale de `MessageGroupId` est de 128 caractères. Valeurs valides : caractères alphanumériques et ponctuation `(!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)` .
Conservation des messages	Par défaut, un message est conservé pendant 4 jours. La durée minimale est de 60 secondes (1 minute). La durée maximale est de 1 209 600 secondes (14 jours).
Débit de message	files d'attente standard Les files d'attente standard prennent en charge un nombre très élevé, presque illimité, d'appels d'API par seconde, par action (`SendMessage` `ReceiveMessage` , ou `DeleteMessage`). Ce débit élevé les rend idéaux pour les cas d'utilisation nécessitant le traitement rapide de gros volumes de messages, tels que le streaming de données en temps réel ou les applications à grande échelle. Bien que les files d'attente standard s'adaptent automatiquement à la demande, il est essentiel de surveiller les modèles d'utilisation pour garantir des performances optimales, en particulier dans les régions où les charges de travail sont plus élevées.

Quota	Description
	files d'attente FIFO • Chaque partition d'une file d'attente FIFO est limitée à 300 transactions par seconde, par action d'API (`SendMessage` `ReceiveMessage` , et`DeleteMessage`). Cette limite s'applique spécifiquement au mode non haut débit. En passant en mode haut débit, vous pouvez dépasser cette limite par défaut. Pour activer le mode haut débit, voir. Permettre un débit élevé pour les files d'attente FIFO dans Amazon SQS • Si vous utilisez le traitement par lots, les files d'attente FIFO à faible débit prennent en charge jusqu'à 3 000 messages par seconde, par action d'API (`SendMessage` , et). `ReceiveMessage` `DeleteMessage` Les 3 000 messages par seconde représentent 300 appels d'API, chacun avec un lot de 10 messages. Débit élevé pour les files d'attente FIFO Les limites FIFO d'Amazon SQS sont basées sur le nombre de demandes d'API, et non sur les limites de messages. Pour le mode haut débit, ces limites de demandes d'API sont les suivantes : Limites de débit des transactions (appels d'API sans traitement par lots) Ces limites définissent la fréquence à laquelle chaque opération d'API (telle que SendMessageReceiveMessage, ou DeleteMessage) peut être effectuée indépendamment, garantissant ainsi des performances efficaces du système dans les limites des transactions autorisées par seconde (TPS). Les limites suivantes sont basées sur des appels d'API non groupés :

Quota	Description
	• USA Est (Virginie du Nord), USA Ouest (Oregon) et Europe (Irlande) : jusqu'à 70 000 transactions par seconde (TPS). • Est des États-Unis (Ohio) et Europe (Francfort) : jusqu'à 19 000 TPS. • Asie-Pacifique (Mumbai), Asie-Pacifique (Singapour), Asie-Pacifique (Sydney) et Asie-Pacifique (Tokyo) : jusqu'à 9 000 TPS. • Europe (Londres) et Amérique du Sud (São Paulo) : jusqu'à 4 500 TPS. • Tous les autres Régions AWS : débit par défaut de 2 400 TPS. Maximisation du débit grâce au traitement par lots Traite plusieurs messages en un seul appel d'API, ce qui augmente considérablement l'efficacité. Au lieu de traiter chaque message individuellement, le traitement par lots vous permet d'envoyer, de recevoir ou de supprimer jusqu'à 10 messages en une seule demande d'API. Cela réduit le nombre total d'appels d'API, ce qui vous permet de traiter un plus grand nombre de messages par seconde tout en respectant les limites de transaction (TPS) de la région, en maximisant le débit et les performances du système. Pour de plus amples informations, veuillez consulter Augmenter le débit grâce à la mise à l'échelle horizontale et au traitement par lots d'actions avec Amazon SQS. Les limites suivantes sont basées sur des appels d'API par lots : • USA Est (Virginie du Nord), USA Ouest (Oregon) et Europe (Irlande) : jusqu'à 700 000 messages par seconde (10 fois la limite hors lot de 70 000 TPS).

Quota	Description
	• USA Est (Ohio) et Europe (Francfort) : jusqu'à 190 000 messages par seconde. • Asie-Pacifique (Mumbai), Asie-Pacifique (Singapour), Asie-Pacifique (Sydney) et Asie-Pacifique (Tokyo) : jusqu'à 90 000 messages par seconde. • Europe (Londres) et Amérique du Sud (São Paulo) : jusqu'à 45 000 messages par seconde. • Tous les autres Régions AWS : jusqu'à 24 000 messages par seconde. Optimisation du débit au-delà du traitement par lots Bien que le traitement par lots puisse augmenter considérablement le débit, il est important d'envisager d'autres stratégies pour optimiser les performances du FIFO : • Répartir les messages entre plusieurs groupes de messages IDs : étant donné que les messages d'un même groupe sont traités de manière séquentielle, la répartition de votre charge de travail entre plusieurs groupes de messages permet un meilleur parallélisme et un débit global plus élevé. Pour de plus amples informations, veuillez consulter Partitions et distribution de données pour un débit élevé pour les files d'attente FIFO SQS. • Utilisation efficace des appels d'API — Minimisez les appels d'API inutiles, tels que les changements de visibilité fréquents ou les suppressions répétées de messages, afin d'optimiser l'utilisation de votre TPS disponible et d'améliorer l'efficacité. • Utilisez de longs sondages : utilisez de longs sondages `WaitTimeSeconds` en configurant vos demandes de réception afin de réduire le nombre de réponses vides

Quota	Description
	lorsqu'aucun message n'est disponible, de réduire les appels d'API inutiles et de mieux utiliser votre quota TPS. • Demande d'augmentation du débit : si votre applicati on nécessite un débit supérieur aux limites par défaut, demandez une augmentation à l'aide de la console Service Quotas. Cela peut être nécessaire pour les charges de travail à forte demande ou dans les régions où les limites par défaut sont plus faibles. Pour activer le mode haut débit, voir. Permettre un débit élevé pour les files d'attente FIFO dans Amazon SQS
Temporisateur de message	Le délai (minimum) par défaut pour un message est de 0 seconde. La valeur maximale est de 15 minutes.
Message size (Taille de message)	La taille minimale de message est de 1 octet (1 caractère). Le maximum est de 1 048 576 octets (1 MiB). Pour envoyer des messages supérieurs à 1 MiB, vous pouvez utiliser la bibliothèque client étendue Amazon SQS pour Java et la bibliothèque client étendue Amazon SQS pour Python. Cette bibliothèque vous permet d'envoyer un message Amazon SQS qui contient une référence à une charge utile de message dans Amazon S3. La taille de la charge utile maximale est de 2 Go. Note Cette bibliothèque étendue ne fonctionne que pour les clients synchrones.
Délai de visibilité des messages	Le délai de visibilité par défaut d'un message est de 30 secondes. La valeur minimale est 0 seconde. La valeur maximale est 12 heures.

Quota	Description
Informations sur les politiques	Le quota maximal est de 8 192 octets, 20 instructions, 50 mandataires ou 10 conditions. Pour de plus amples informations, veuillez consulter Quotas liés à la politique Amazon SQS.

Quotas liés à la politique Amazon SQS

Le tableau suivant répertorie les quotas relatifs aux stratégies.

Nom	Maximum
Octets	8 192
Conditions	10
Mandataires	50
Instructions	20
Actions par instruction	7

Fonctionnalités et capacités d'Amazon SQS

Cette rubrique présente les fonctionnalités couramment utilisées dans Amazon SQS pour gérer les files d'attente de messages, optimiser les performances, garantir une distribution fiable des messages et gérer efficacement le traitement des messages.

Utilisation de files d'attente contenant des lettres mortes dans Amazon SQS

Amazon SQS prend en charge les files d'attente contenant des lettres mortes (DLQs), que les files d'attente sources peuvent cibler pour les messages qui ne sont pas traités correctement. DLQs sont utiles pour le débogage de votre application car vous pouvez isoler les messages non consommés afin de déterminer pourquoi le traitement n'a pas abouti. Pour des performances optimales, il est recommandé de conserver la même file d'attente source et le DLQ dans la même Compte AWS région. Une fois que les messages se trouvent dans une file d'attente de lettres mortes, vous pouvez :

- Rechercher dans les journaux les exceptions qui ont pu entraîner le déplacement de messages dans une file d'attente de lettres mortes.
- Analysez le contenu des messages placés dans la file d'attente des lettres mortes pour diagnostiquer les problèmes liés à l'application.
- Déterminer si vous avez donné suffisamment de temps à votre consommateur pour traiter les messages.
- Déplacez les messages hors de la file d'attente des lettres mortes en utilisant le redrive de la file d'attente des lettres mortes.

Vous devez d'abord créer une nouvelle file d'attente avant de la configurer en tant que file d'attente contenant des lettres mortes. Pour plus d'informations sur la configuration d'une file d'attente de lettres mortes à l'aide de la console Amazon SQS, consultez Configuration d'une file d'attente contenant des lettres mortes à l'aide de la console Amazon SQS. Pour obtenir de l'aide sur les files d'attente de lettres mortes, par exemple sur la façon de configurer une alarme pour tout message déplacé vers une file d'attente de lettres mortes, consultez. Création d'alarmes pour les files d'attente de lettres mortes à l'aide d'Amazon CloudWatch

> **Note**
>
> N'utilisez pas de file d'attente de lettres mortes avec une file d'attente FIFO si vous ne souhaitez pas modifier l'ordre exact des messages ou des opérations. Par exemple, n'utilisez pas de file d'attente de lettres mortes avec des instructions dans une liste de décisions de modification (EDL) pour une suite de modifications vidéo lorsque le changement de l'ordre des modifications change le contexte des modifications ultérieures.

Utilisation de politiques pour les files d'attente de lettres mortes

Utilisez une politique de redrive pour spécifier le`maxReceiveCount`. `maxReceiveCount`C'est le nombre de fois qu'un consommateur peut recevoir un message depuis une file d'attente source avant qu'il ne soit déplacé vers une file d'attente de lettres mortes. Par exemple, si le paramètre `maxReceiveCount` est défini sur une valeur faible telle que 1, l'échec de réception d'un message entraîne son transfert dans la file d'attente des lettres mortes. Pour garantir la résilience de votre système face aux erreurs, définissez une valeur suffisamment élevée pour `maxReceiveCount` afin de permettre un nombre suffisant de tentatives.

La stratégie d'autorisation de redirection spécifie quelles files d'attente source peuvent accéder à la file d'attente de lettres mortes. Vous pouvez choisir d'autoriser toutes les files d'attente sources, d'autoriser des files d'attente sources spécifiques ou de refuser à toutes les files d'attente sources l'utilisation de la file d'attente de lettres mortes. Par défaut, toutes les files d'attente sources peuvent utiliser la file d'attente contenant des lettres mortes. Si vous choisissez d'autoriser des files d'attente spécifiques à l'aide de `byQueue` cette option, vous pouvez spécifier jusqu'à 10 files d'attente sources à l'aide de la file d'attente source Amazon Resource Name (ARN). Si vous spécifiez `denyAll`, la file d'attente ne peut pas être utilisée comme file d'attente de lettres mortes.

Comprendre les périodes de rétention des messages pour les files d'attente de lettres mortes

Pour les files d'attente standard, l'expiration d'un message est toujours basée sur son horodatage de mise en file d'attente d'origine. Lorsqu'un message est déplacé vers une file d'attente de lettres mortes, l'horodatage de la mise en file d'attente reste inchangé. La `ApproximateAgeOfOldestMessage` métrique indique à quel moment le message a été placé dans la file d'attente des lettres mortes, et non à quel moment le message a été initialement envoyé. Supposons, par exemple, qu'un message passe 1 journée dans la file d'attente d'origine avant d'être

déplacé vers une file d'attente de lettres mortes. Si la période de conservation de la file d'attente de lettres mortes est de 4 jours, le message est supprimé de la file d'attente de lettres mortes au bout de 3 jours et le paramètre `ApproximateAgeOfOldestMessage` est défini sur 3 jours. Il est donc recommandé de toujours définir la période de rétention d'une file d'attente de lettres mortes de manière à ce qu'elle soit plus longue que la période de rétention de la file d'attente d'origine.

Pour les files d'attente FIFO, l'horodatage de la mise en file d'attente est réinitialisé lorsque le message est déplacé vers une file d'attente de lettres mortes. La métrique `ApproximateAgeOfOldestMessage` indique à quel moment le message a été placé dans la file d'attente de lettres mortes. Dans le même exemple ci-dessus, le message est supprimé de la file d'attente des lettres mortes au bout de quatre jours, puis au `ApproximateAgeOfOldestMessage` bout de quatre jours.

Configuration d'une file d'attente contenant des lettres mortes à l'aide de la console Amazon SQS

Une file d'attente de lettres mortes (DLQ) est une file d'attente qui reçoit des messages qui n'ont pas été traités correctement depuis une autre file d'attente, connue sous le nom de file source. Amazon SQS ne crée pas automatiquement la file d'attente de lettres mortes. Vous devez d'abord créer la file d'attente avant de l'utiliser en tant que file d'attente de lettres mortes. Lors de la configuration d'un DLQ, le type de file d'attente doit correspondre au type de file d'attente source : une file FIFO ne peut utiliser qu'un DLQ FIFO, et une file d'attente standard ne peut utiliser qu'un DLQ standard. Vous pouvez configurer une file d'attente contenant des lettres mortes lorsque vous créez ou modifiez une file d'attente. Pour en savoir plus, consultez Utilisation de files d'attente contenant des lettres mortes dans Amazon SQS .

Pour configurer une file d'attente de lettres mortes pour une file d'attente existante (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Sélectionnez la file d'attente source (la file qui enverra les messages ayant échoué à la file d'attente contenant des lettres mortes), puis choisissez Modifier.
4. Accédez à la section File d'attente des lettres mortes et cochez Activé.
5. Dans Paramètres de la file d'attente de lettres mortes, choisissez le nom de ressource Amazon (ARN) d'une file d'attente existante que vous souhaitez utiliser comme file d'attente de lettres mortes.

6. Définissez la valeur Maximum de réception, qui définit le nombre de fois qu'un message peut être reçu avant d'être envoyé dans la file d'attente de lettres mortes (plage valide : 1 à 1 000).
7. Choisissez Enregistrer.

Découvrez comment configurer le redrive d'une file d'attente en lettres mortes dans Amazon SQS

Utilisez le redrive de la file d'attente de lettres mortes pour déplacer les messages non consommés d'une file d'attente de lettres mortes vers une autre destination à des fins de traitement. Par défaut, la redirection de la file d'attente de lettres mortes déplace les messages d'une file d'attente de lettres mortes vers une file d'attente source. Cependant, vous pouvez aussi configurer n'importe quelle autre file d'attente comme destination de redirection si les deux files d'attente sont du même type. Par exemple, si la file d'attente de lettres mortes est une file d'attente FIFO, la file d'attente de destination de redirection doit également être une file d'attente FIFO. En outre, vous pouvez configurer la vitesse de redirection pour définir la vitesse à laquelle Amazon SQS déplace les messages.

> **Note**
>
> Lorsqu'un message est déplacé d'une file d'attente FIFO vers une file d'attente de lettres mortes FIFO, l'ID de déduplication du message d'origine est remplacé par l'ID du message d'origine. Cela permet de s'assurer que la déduplication de la file d'attente de lettres mortes n'empêchera pas le stockage de deux messages indépendants partageant un identifiant de déduplication.

Les files d'attente de lettres mortes redirigent les messages dans l'ordre dans lequel ils sont reçus, en commençant par le message le plus ancien. Toutefois, la file d'attente de destination ingère les messages redirigés, de même que les nouveaux messages provenant d'autres producteurs, en fonction de l'ordre dans lequel elle les reçoit. Par exemple, si un producteur envoie des messages à une file d'attente FIFO source alors qu'il reçoit simultanément des messages redirigés provenant d'une file d'attente de lettres mortes, les messages redirigés s'entrelaceront avec les nouveaux messages du producteur.

> **Note**
>
> La tâche de redirection réinitialise la période de rétention. Tous les messages redirigés sont considérés comme de nouveaux messages avec un nouveau message `messageID` et `enqueueTime` sont affectés à des messages redirigés.

Configuration d'un redrive de file d'attente en lettres mortes pour une file d'attente standard existante à l'aide de l'API Amazon SQS

Vous pouvez configurer le redrive d'une file d'attente contenant des lettres mortes à l'aide des actions`StartMessageMoveTask`,`ListMessageMoveTasks`, et `CancelMessageMoveTask` de l'API :

Action d'API	Description
`StartMessageMoveTask`	Démarre une tâche asynchrone pour déplacer les messages d'une file d'attente source spécifiée vers une file d'attente de destination spécifiée.
`ListMessageMoveTasks`	Obtient les tâches de déplacement de messages les plus récentes (jusqu'à 10) dans une file d'attente source spécifique.
`CancelMessageMoveTask`	Annule une tâche de déplacement de message spécifiée. Un mouvement de message ne peut être annulé que lorsque le statut actuel est en cours d'exécution.

Configuration d'un redrive de file d'attente en lettres mortes pour une file d'attente standard existante à l'aide de la console Amazon SQS

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.

3. Choisissez le nom de la file d'attente que vous avez configurée en tant que file d'attente de lettres mortes.
4. Choisissez Démarrer la redirection de file d'attente de lettres mortes.
5. Dans Configuration de la redirection, pour Destination du message, effectuez l'une des opérations suivantes :
   - Pour rediriger les messages vers leur file d'attente source, choisissez Rediriger vers la ou les files d'attente source.
   - Pour rediriger les messages vers une autre file d'attente, choisissez Rediriger vers une destination personnalisée. Saisissez ensuite l'Amazon Resource Name (ARN) d'une file d'attente de destination existante.
6. Sous Paramètres de contrôle de la vitesse, choisissez l'une des options suivantes :
   - Système optimisé : redirigez les messages de la file d'attente de lettres mortes avec le nombre maximum de messages par seconde.
   - Vitesse maximale personnalisée : rediffusez les messages de la file d'attente de lettres mortes avec un débit maximal personnalisé de messages par seconde. Le débit maximal autorisé est de 500 messages par seconde.
     - Il est recommandé de commencer par une petite valeur pour la vitesse maximale personnalisée et de vérifier que la file d'attente source n'est pas submergée de messages. Ensuite, augmentez progressivement la valeur de la vitesse maximale personnalisée, en continuant à surveiller l'état de la file d'attente source.
7. Lorsque vous avez fini de configurer la redirection de la file d'attente de lettres mortes, choisissez Rediriger les messages.

   > **Important**
   >
   > Amazon SQS ne prend pas en charge le filtrage et la modification des messages lors de leur redirection à partir de la file d'attente de lettres mortes.
   > Une tâche de redirection de file d'attente de lettres mortes peut s'exécuter pendant 36 heures au maximum. Amazon SQS prend en charge un maximum de 100 tâches de redirection actives par compte.

8. Si vous souhaitez annuler la tâche de redirection des messages, sur la page Détails de votre file d'attente, choisissez Annuler la redirection de la file d'attente de lettres mortes. Lorsque vous

annulez la redirection d'un message en cours, tous les messages qui ont déjà été déplacés vers leur file d'attente de destination y resteront.

Configuration des autorisations de file d'attente pour la redirection de files d'attente de lettres mortes

Vous pouvez autoriser les utilisateurs à accéder à des actions spécifiques dans les files d'attente de lettres mortes en ajoutant des autorisations à votre stratégie. Les autorisations minimales requises pour la redirection d'une file d'attente de lettres mortes sont les suivantes :

Autorisations minimales	Méthodes d'API requises
Pour démarrer la redirection d'un message	• Ajoutez le `sqs:StartMessageMoveTask`, le `sqs:ReceiveMessage`, le `sqs:DeleteMessage` et le `sqs:GetQueueAttributes` de la file d'attente de lettres mortes. Si la file d'attente de lettres mortes ou la file d'attente source d'origine est chiffrée (également nommée file d'attente SSE), `kms:Decrypt` est également requis pour toute clé KMS utilisée pour chiffrer les messages. • Ajoutez le `sqs:SendMessage` de la file d'attente de destination. Si la file d'attente de destination est chiffrée, `kms:GenerateDataKey` et `kms:Decrypt` sont également requis.
Pour annuler un message en cours de redirection	• Ajoutez le `sqs:CancelMessageMoveTask`, le `sqs:ReceiveMessage`, le `sqs:DeleteMessage` et le `sqs:GetQueueAttributes` de la file d'attente de lettres mortes. Si la file d'attente de lettres mortes est chiffrée (également nommée file d'attente SSE), `kms:Decrypt` est également requis.
Pour afficher le statut du déplacement d'un message	• Ajoutez le `sqs:ListMessageMoveTasks` et le `sqs:GetQueueAttributes` de la file d'attente de lettres mortes.

Pour configurer les autorisations pour une paire de files d'attente chiffrées (une file d'attente source avec une file d'attente de lettres mortes)

Procédez comme suit pour configurer les autorisations minimales pour le redrive d'une file d'attente de lettres mortes (DLQ) :

1. Ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau de navigation, sélectionnez Stratégies.
3. Créez une nouvelle politique et ajoutez les autorisations suivantes. Attachez la politique à l'utilisateur ou au rôle IAM qui effectuera l'opération de redrive.
   - Autorisations pour le DLQ (file d'attente source) :
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:GetQueueAttributes`
     - `sqs:ListDeadLetterSourceQueues`
     - Spécifiez l'ARN de la ressource de la DLQ (file d'attente source) (par exemple, « arn:aws:sqs : : : »). *<DLQ_region> <DLQ_accountId> <DLQ_name>*
   - Autorisations pour la file d'attente de destination :
     - `sqs:SendMessage`
     - Spécifiez la file `Resource` ARN d'attente de destination (par exemple, « arn:aws:sqs : »). *<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*
   - Autorisations pour les clés KMS :
     - `kms:Decrypt`(Nécessaire pour déchiffrer les messages dans le DLQ.)
     - `kms:GenerateDataKey`(Nécessaire pour chiffrer les messages dans la file d'attente de destination.)
       - `Resource` ARNs:
         - L'ARN de la clé KMS utilisée pour chiffrer les messages dans le DLQ (file d'attente source) (par exemple, « arn:aws:kms : ::key/ »). *<region> <accountId> <SourceQueueKeyId>*

- L'ARN de la clé KMS utilisée pour chiffrer les messages dans la file d'attente de destination (par exemple, « arn:aws:kms : ::key/ »). *<region> <accountId> <DestinationQueueKeyId>*

Votre stratégie d'accès doit ressembler à ce qui suit :

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>"
            ]
        }
    ]
}
```

Pour configurer les autorisations pour une paire de files d'attente non chiffrées (une file d'attente source avec une file d'attente de lettres mortes)

Suivez ces étapes pour configurer les autorisations minimales requises pour gérer une file d'attente de lettres mortes (DLQ) standard non cryptée. Les autorisations minimales requises sont de recevoir, de supprimer et d'obtenir des attributs de la file d'attente de lettres mortes, et d'envoyer des attributs à la file d'attente source.

1. Ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau de navigation, sélectionnez Stratégies.
3. Créez une nouvelle politique et ajoutez les autorisations suivantes. Attachez la politique à l'utilisateur ou au rôle IAM qui effectuera l'opération de redrive.
   - Autorisations pour le DLQ (file d'attente source) :
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:ListDeadLetterSourceQueues`
     - Spécifiez l'ARN de la ressource de la DLQ (file d'attente source) (par exemple, « arn:aws:sqs : : : »). *<DLQ_region> <DLQ_accountId> <DLQ_name>*
   - Autorisations pour la file d'attente de destination :

- `sqs:SendMessage`
- Spécifiez la file `Resource` ARN d'attente de destination (par exemple, « arn:aws:sqs : »). *`<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>`*

Votre stratégie d'accès doit ressembler à ce qui suit :

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"
                }
            }
        }
```

```
    ]
}
```

Utilisation du redrive des files d'attente en lettres mortes avec le contrôle d'accès aux points de terminaison VPC

Lorsque vous limitez l'accès à la file d'attente à une `aws:sourceVpc` condition d' VPCs utilisation spécifique, vous devez créer une exception pour que les AWS services activent la fonctionnalité de redrive des files d'attente en lettres mortes (DLQ). Cela est dû au fait que le service Amazon SQS fonctionne en dehors de votre VPC lors du déplacement de messages.

Pour autoriser les opérations de redrive DLQ, ajoutez la `aws:CalledViaLast` condition à votre politique de file d'attente. Cela permet à Amazon SQS d'effectuer des appels d'API en votre nom tout en maintenant les restrictions VPC pour un accès direct.

Pour autoriser à la fois l'accès restreint par VPC et le redrive DLQ :

1. Utilisez `aws:CalledViaLast` cette condition dans votre politique de file d'attente.
2. Appliquez la politique à la fois à la file d'attente source et au DLQ
3. Maintenir les restrictions VPC pour un accès direct depuis d'autres sources

Voici un exemple de politique qui met en œuvre ces exigences :

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

- Remplacez les valeurs de l'espace réservé par vos valeurs réelles
- Cette politique utilise une déclaration « refuser » assortie de conditions, ce qui est plus sûr que l'utilisation d'instructions « autoriser »
- L'`StringNotEqualsIfExists`opérateur gère les cas où la clé de condition peut ne pas être présente dans le contexte de la demande.

Vous pouvez également utiliser la clé de `aws:ViaAWSService` condition pour autoriser l'accès basé sur le service tout en maintenant les restrictions VPC. Cette clé de condition indique si la demande provient d'un AWS service. Voici un exemple de politique qui utilise `aws:ViaAWSService` au lieu 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"
        }
      }
    }
  ]
}
```

L' BoolIfExists opérateur soumis à des `aws:ViaAWSService` conditions veille à ce que les demandes soient autorisées lorsqu'elles proviennent des services tout en maintenant les restrictions VPC pour un accès direct. Cela peut être plus simple à comprendre et à gérer, car il vérifie directement si la demande provient d'un AWS service plutôt que de vérifier quel service a effectué le dernier appel.

Pour plus d'informations sur les clés de condition utilisées dans les politiques IAM et de ressources, voir Éléments de stratégie IAM JSON : Condition.

CloudTrail exigences de mise à jour et d'autorisation pour le redrive de files d'attente de lettres mortes Amazon SQS

Le 8 juin 2023, Amazon SQS a introduit le redrive DLQ (Dead-Letter Queue) pour les SDK AWS et (CLI). AWS Command Line Interface Cette fonctionnalité vient s'ajouter au redrive DLQ déjà pris en

charge pour la AWS console. Si vous avez déjà utilisé la AWS console pour rediffuser des messages de file d'attente en lettres mortes, vous pouvez être concerné par les modifications suivantes :

CloudTrail changement de nom d'un événement

Le 15 octobre 2023, les noms des CloudTrail événements pour le redrive de la file d'attente de lettres mortes seront modifiés sur la console Amazon SQS. Si vous avez défini des alarmes pour ces CloudTrail événements, vous devez les mettre à jour dès maintenant. Les nouveaux noms d' CloudTrail événements pour DLQ redrive sont les suivants :

Nom de l'événement précédent	Nouveau nom de l'événement
`CreateMoveTask`	`StartMessageMoveTask`
`CancelMoveTask`	`CancelMessageMoveTask`

Autorisations mises à jour

Inclus dans la version du kit SDK et de la CLI, Amazon SQS a également mis à jour les autorisations de file d'attente pour la redirection de file d'attente de lettres mortes afin de respecter les bonnes pratiques de sécurité. Utilisez les types d'autorisation de file d'attente suivants pour rediriger les messages de votre DLQs.

1. Autorisations basées sur les actions (mise à jour pour les actions de l'API de file d'attente de lettres mortes)
2. Autorisations de stratégie Amazon SQS gérées
3. Stratégie d'autorisation utilisant un caractère générique sqs:*

> **Important**
>
> Pour utiliser la redirection de file d'attente de lettres mortes pour le kit SDK ou la CLI, vous devez disposer d'une stratégie d'autorisation de redirection de file d'attente de lettres mortes correspondant à l'une des options ci-dessus.

Si vos autorisations de file d'attente pour la redirection de file d'attente de lettres mortes ne correspondent pas à l'une des options ci-dessus, vous devez mettre à jour vos autorisations d'ici

le 31 août 2023. D'ici le 31 août 2023, votre compte pourra rediriger les messages en utilisant les autorisations que vous avez configurées à l'aide de la console AWS uniquement dans les régions où vous avez déjà utilisé la redirection de file d'attente de lettres mortes. Par exemple, supposons que vous ayez un « compte A » à la fois dans les régions us-east-1 et eu-west-1. Le « compte A » était utilisé pour rediriger les messages sur la AWS console dans us-east-1 avant le 8 juin 2023, mais pas dans eu-west-1. Entre le 8 juin 2023 et le 31 août 2023, si les autorisations de politique du « compte A » ne correspondent pas à l'une des options ci-dessus, elles ne peuvent être utilisées que pour rediriger des messages sur la AWS console dans us-east-1, et non dans eu-west-1.

> **Important**
>
> Si vos autorisations de redirection de file d'attente de lettres mortes ne correspondent pas à l'une de ces options après le 31 août 2023, votre compte ne sera plus en mesure de rediriger les messages de file d'attente de lettres mortes à l'aide de la console AWS .
> Toutefois, si vous avez utilisé la fonction DLQ redrive sur la AWS console en août 2023, vous disposez d'une extension jusqu'au 15 octobre 2023 pour adopter les nouvelles autorisations selon l'une de ces options.
> Pour de plus amples informations, veuillez consulter the section called “Identification des stratégies concernées”.

Vous trouverez ci-dessous des exemples d'autorisations de file d'attente pour chaque option de redirection de file d'attente de lettres mortes. Lorsque vous utilisez des files d'attente chiffrées côté serveur (SSE), l'autorisation AWS KMS clé correspondante est requise.

Basé sur l'action

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>"
        }
    ]
}
```

Stratégie gérée

Les stratégies gérées suivantes contiennent les autorisations mises à jour requises :

- Amazon SQSFull Access — Inclut les tâches de regénération de files d'attente en lettres mortes suivantes : démarrer, annuler et répertorier.
- Amazon SQSRead OnlyAccess — Fournit un accès en lecture seule et inclut la tâche de redrive de la liste des listes de lettres mortes.

Stratégie d'autorisation utilisant un caractère générique sqs*

JSON

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

Identification des stratégies concernées

Si vous utilisez des politiques gérées par le client (CMPs), vous pouvez utiliser AWS CloudTrail un IAM pour identifier les politiques affectées par la mise à jour des autorisations de file d'attente.

> **Note**
>
> Si vous utilisez `AmazonSQSFullAccess` et `AmazonSQSReadOnlyAccess`, aucune autre action n'est requise.

1. Connectez-vous à la AWS CloudTrail console.
2. Sur la page Historique des événements, sous Rechercher les attributs, utilisez le menu déroulant pour sélectionner le Nom de l'événement. Ensuite, recherchez `CreateMoveTask`.
3. Sélectionnez un événement pour ouvrir la page des Détails. Dans la section Enregistrements d'événements, récupérez le `UserName` ou le `RoleName` à partir de l'ARN `userIdentity`.
4. Connectez-vous à la console IAM.
   - Pour les utilisateurs, choisissez Utilisateurs. Sélectionnez l'utilisateur avec le `UserName` identifié à l'étape précédente.
   - Pour les rôles, choisissez Roles. Sélectionnez l'utilisateur avec le `RoleName` identifié à l'étape précédente.
5. Sur la page Détails, dans la section Autorisations, passez en revue les stratégies possédant le préfixe `sqs:` dans `Action`, ou consultez les stratégies pour lesquelles la file d'attente Amazon SQS est définie dans `Resource`.

Création d'alarmes pour les files d'attente de lettres mortes à l'aide d'Amazon CloudWatch

Configurez une CloudWatch alarme pour surveiller les messages dans une file d'attente de lettres mortes à l'aide de la `ApproximateNumberOfMessagesVisible`métrique. Pour obtenir des instructions complètes, veuillez consulter Création d' CloudWatch alarmes pour les métriques Amazon SQS. Lorsque l'alarme se déclenche, indiquant que les messages ont été placés dans la file d'attente des lettres mortes, vous pouvez interroger la file d'attente pour les consulter et les récupérer.

Métadonnées des messages pour Amazon SQS

Utilisez les attributs de message pour ajouter des métadonnées personnalisées aux messages Amazon SQS destinés à vos applications. Utilisez les attributs du système de messagerie pour stocker les métadonnées afin de les intégrer à d'autres Services AWS, tels que AWS X-Ray.

Attributs de message Amazon SQS

Amazon SQS vous permet d'inclure des métadonnées structurées (telles que les horodatages, les données géospatiales, les signatures et les identifiants) dans les messages à l'aide d'attributs de message. Chaque message peut contenir jusqu'à 10 attributs. Les attributs de message sont facultatifs et séparés du corps du message (même s'ils sont envoyés en même temps). Votre consommateur peut utiliser des attributs de message pour traiter un message d'une façon particulière sans avoir à traiter d'abord le corps du message. Pour plus d'informations sur l'envoi de messages avec des attributs à l'aide de la console Amazon SQS, consultez Envoi d'un message avec des attributs à l'aide d'Amazon SQS.

> **Note**
>
> Ne confondez pas les attributs des messages avec les attributs du système de messagerie : alors que vous pouvez utiliser les attributs de message pour joindre des métadonnées personnalisées aux messages Amazon SQS destinés à vos applications, vous pouvez utiliser les attributs du système de messagerie pour stocker les métadonnées d'autres AWS services, tels que. AWS X-Ray

Rubriques

- Composants des attributs de message
- Types de données d'attribut de message
- Calcul du résumé du MD5 message pour les attributs du message

Composants des attributs de message

> **Important**
>
> Tous les composants d'un attribut de message sont inclus dans la restriction de taille de message de 1 MiB.
> `Name`, `Type`, `Value` et le corps du message ne doivent pas être vides ni contenir la valeur null.

Chaque attribut de message est constitué des composants suivants :

- Nom : le nom de l'attribut de message peut contenir les caractères suivants : `A-Z`, `a-z`, `0-9`, trait de soulignement (`_`), tiret (`-`) et point (`.`). Les restrictions suivantes s'appliquent :
  - Il peut contenir jusqu'à 256 caractères.
  - Il ne peut pas commencer par `AWS.` ou `Amazon.` (ou toute variante de casse)
  - Il est sensible à la casse
  - Il doit être unique parmi tous les noms d'attribut pour le message
  - Il ne doit pas commencer ou se terminer par un point
  - Il ne doit pas comporter plusieurs points à la suite
- Type : type de données de l'attribut de message. Les types pris en charge incluent `String`, `Number` et `Binary`. Vous pouvez également ajouter des informations personnalisées pour tout type de données. Le type de données est soumis aux mêmes restrictions que le corps du message (pour plus d'informations, consultez `SendMessage` dans la Référence d'API Amazon Simple Queue Service). En outre, les restrictions suivantes s'appliquent :
  - Il peut contenir jusqu'à 256 caractères.
  - Il est sensible à la casse
- Valeur : valeur d'attribut du message. Pour les données de type `String`, les valeurs d'attribut sont soumises aux mêmes restrictions que le corps du message.

Types de données d'attribut de message

Les types de données d'attribut de message indiquent à Amazon SQS comment traiter les valeurs d'attribut de message correspondantes. Par exemple, si le type est `Number`, Amazon SQS valide les valeurs numériques.

Amazon SQS prend en charge les types de données logiques `String`, `Number` et `Binary`, avec la possibilité d'utiliser des étiquettes de type de données personnalisé au format *`.custom-data-type`*.

- Chaîne : les attributs `String` peuvent stocker n'importe quel texte Unicode à l'aide de caractères XML valides.
- Nombre : les attributs `Number` peuvent stocker des valeurs numériques positives ou négatives. Un nombre peut avoir jusqu'à 38 chiffres de précision et être compris entre 10^-128 et 10^+126.

> **Note**
>
> Amazon SQS supprime les zéros de début et de fin.

- Binaire : les binary attributes peuvent stocker n'importe quelle donnée binaire, telles que des données compressées, des données chiffrées ou des images.
- Personnalisé : pour créer un type de données personnalisé, ajoutez une étiquette de type personnalisé à n'importe quel type de données. Par exemple :
  - `Number.byte`, `Number.short`, `Number.int` et `Number.float` peuvent vous aider à faire la distinction entre les types de nombre.
  - `Binary.gif` et `Binary.`png peuvent vous aider à faire la distinction entre les types de fichier.

> **Note**
>
> Amazon SQS n'interprète pas, ne valide pas ou n'utilise pas les données ajoutées. L'étiquette de type personnalisé est soumise aux mêmes restrictions que le corps du message.

Calcul du résumé du MD5 message pour les attributs du message

Si vous utilisez le AWS SDK pour Java, vous pouvez ignorer cette section. La `MessageMD5ChecksumHandler` classe du SDK for Java MD5 prend en charge les résumés de messages pour les attributs de message Amazon SQS.

Si vous utilisez l'API Query ou l'une des API AWS SDKs qui ne prennent pas en charge les résumés de MD5 messages pour les attributs de message Amazon SQS, vous devez suivre les directives suivantes pour effectuer le calcul MD5 du résumé des messages.

> **Note**
>
> Incluez toujours des suffixes de type de données personnalisés dans le calcul du résumé des MD5 messages.

Présentation

Voici un aperçu de l'algorithme de calcul du résumé des MD5 messages :

1. Triez tous les attributs de message dans l'ordre croissant en fonction de leur nom.
2. Encodez les différentes parties de chaque attribut (`Name`, `Type` et `Value`) dans un tampon.
3. Calculez le résumé du message du tampon entier.

Le schéma suivant montre le codage du résumé du MD5 message pour un attribut de message unique :

Pour encoder un seul attribut de message Amazon SQS

1. Encodez le nom : la longueur (4 octets) et les octets UTF-8 du nom.
2. Encodez le type de données : la longueur (4 octets) et les octets UTF-8 du type de données.
3. Encodez le type de transport (`String` ou `Binary`) de la valeur (1 octet).

> **Note**
>
> Les types de données logiques `String` et `Number` utilisent le type de transport `String`. Le type de données logiques `Binary` utilise le type de transport `Binary`.

   a. Pour le type de transport `String`, encodez 1.
   b. Pour le type de transport `Binary`, encodez 2.
4. Encodez la valeur d'attribut.
   a. Pour un type de transport `String`, encodez la valeur d'attribut : la longueur (4 octets) et les octets UTF-8 de la valeur.
   b. Pour un type de transport `Binary`, encodez la valeur d'attribut : la longueur (4 octets) et les octets bruts de la valeur.

Attributs du système de message Amazon SQS

Vous pouvez utiliser des attributs de message pour attacher des métadonnées personnalisées aux messages Amazon SQS de vos applications, tandis que vous pouvez utiliser des attributs de système de message pour stocker des métadonnées pour d'autres services AWS , tels que AWS X-Ray. Pour plus d'informations, consultez le paramètre de demande `MessageSystemAttribute` des actions d'API `SendMessage` et `SendMessageBatch`, l'attribut `AWSTraceHeader` de l'action d'API `ReceiveMessage`, ainsi que le type de données `MessageSystemAttributeValue` dans la Référence d'API Amazon Simple Queue Service.

Les attributs de système de message sont structurés exactement comme les attributs de message, avec les exceptions suivantes :

- Actuellement, le seul attribut de système de message pris en charge est `AWSTraceHeader`. Son type `String` et sa valeur doivent être une chaîne d'en-tête de AWS X-Ray trace correctement formatée.
- La taille d'un attribut de système de message ne compte pas dans la taille totale d'un message.

Ressources requises pour traiter les messages Amazon SQS

Amazon SQS fournit des estimations du nombre approximatif de messages différés, visibles et invisibles dans une file d'attente afin de vous aider à évaluer les ressources nécessaires au traitement. Pour plus d'informations sur la visibilité, consultez la section Délai de visibilité Amazon SQS.

> **Note**
>
> Pour certaines métriques, le résultat est approximatif en raison de l'architecture distribuée d'Amazon SQS. Dans la plupart des cas, le nombre devrait être proche du nombre réel de messages dans la file d'attente.

Le tableau suivant indique le nom de l'attribut à utiliser avec l'action `GetQueueAttributes`.

Tâche	Nom d'attribut
Obtenir le nombre approximatif de messages disponibles dans la file d'attente.	`ApproximateNumberOfMessagesVisible`
Obtenir le nombre approximatif de messages dans la file d'attente qui sont retardés et qui ne peuvent pas être lus immédiatement. Cela peut se produire lorsque la file d'attente est configurée avec un délai d'attente ou que le message a été envoyé avec un paramètre de délai d'attente.	`ApproximateNumberOfMessagesDelayed`
Obtenir le nombre approximatif de messages en transit. Les messages sont considéré s comme en cours s'ils ont été expédiés à un client, mais qu'ils n'ont pas encore été supprimés ou qu'ils n'ont pas encore atteint la fin du délai de visibilité.	`ApproximateNumberOfMessagesNotVisible`

Pagination des files d'attente de listes Amazon SQS

Les méthodes d'API `listQueues` et `listDeadLetterQueues` prennent en charge les contrôles de pagination facultatifs. Par défaut, ces méthodes d'API renvoient jusqu'à 1 000 files d'attente dans le message de réponse. Vous pouvez définir le paramètre `MaxResults` de manière à ce qu'il renvoie moins de résultats à chaque réponse.

Définissez le paramètre `MaxResults` dans la demande `listDeadLetterQueues` ou `listQueues` pour spécifier le nombre maximal de résultats à renvoyer dans la réponse. Si vous ne définissez pas `MaxResults`, la réponse inclut un maximum de 1 000 résultats et la valeur `NextToken` de la réponse est nulle.

Si vous définissez `MaxResults`, la réponse inclut une valeur pour `NextToken` s'il y a des résultats supplémentaires à afficher. Utilisez `NextToken` comme paramètre dans votre prochaine demande à `listQueues` pour recevoir la page de résultats suivante. La valeur `NextToken` de la réponse est nulle s'il n'y a pas de résultats supplémentaires à afficher.

Balises de répartition des coûts Amazon SQS

Pour organiser et identifier vos files d'attente Amazon SQS pour la répartition des coûts, vous pouvez ajouter des balises de métadonnées qui identifient le but, le propriétaire ou l'environnement d'une file d'attente. Cette approche est utile lorsque vous avez un grand nombre de files d'attente. Pour configurer les balises à l'aide de la console Amazon SQS, consultez the section called “Configuration de balises pour une file d'attente”

Vous pouvez utiliser des balises de répartition des coûts pour organiser votre AWS facture afin de refléter votre propre structure de coûts. Pour ce faire, inscrivez-vous pour que votre Compte AWS facture inclue les clés et les valeurs des tags. Pour plus d'informations, consultez Configuration du rapport de répartition des coûts mensuel dans le Guide d'utilisateur AWS Billing .

Chaque balise est composée d'une paire clé-valeur que vous définissez. Par exemple, vous pouvez facilement identifier vos files d'attente de production et de test en leur attribuant des balises comme suit :

File d'attente	Clé	Valeur
`MyQueueA`	`QueueType`	`Production`
`MyQueueB`	`QueueType`	`Testing`

> **Note**
>
> Lorsque vous utilisez des balises de file d'attente, tenez compte des consignes suivantes :

- Nous vous déconseillons d'ajouter plus de 50 balises à une file d'attente. Le balisage prend en charge les caractères Unicode en UTF-8.
- Les balises n'ont aucune signification sémantique. Amazon SQS interprète les balises en tant que chaîne de caractères.
- Les balises sont sensibles à la casse.
- Une nouvelle balise dont la clé est identique à celle d'une balise existante remplace la balise existante.
- Les actions de balisage sont limitées à 30 TPS par action. Compte AWS Si votre application nécessite un débit plus élevé, soumettez une demande.

Pour afficher la liste complète des restrictions de balise, consultez Quotas de file d'attente standard Amazon SQS.

Recherches courtes et longues sur Amazon SQS

Amazon SQS propose des options d'interrogation courtes et longues pour recevoir des messages depuis une file d'attente. Tenez compte des exigences de réactivité et de rentabilité de votre application lorsque vous choisissez entre ces deux options de sondage :

- Interrogation courte (par défaut) : la `ReceiveMessage`demande interroge un sous-ensemble de serveurs (sur la base d'une distribution aléatoire pondérée) pour trouver les messages disponibles et envoie une réponse immédiate, même si aucun message n'est trouvé.
- Interrogation longue : `ReceiveMessage`interroge tous les serveurs à la recherche de messages, envoie une réponse dès qu'au moins un message est disponible, jusqu'au maximum spécifié. Une réponse vide n'est envoyée que si le temps d'attente pour le sondage expire. Cette option permet de réduire le nombre de réponses vides et potentiellement de réduire les coûts.

Les sections suivantes expliquent les détails des interrogations courtes et longues.

Consommation des messages à l'aide de l'interrogation courte

Lorsque vous consommez des messages d'une file d'attente (FIFO ou standard) à l'aide d'un court sondage, Amazon SQS échantillonne un sous-ensemble de ses serveurs (sur la base d'une distribution aléatoire pondérée) et renvoie des messages provenant uniquement de ces serveurs.

Autrement dit, une demande `ReceiveMessage` particulière peut ne pas renvoyer tous les messages. Toutefois, si votre file d'attente compte moins de 1 000 messages, une requête ultérieure renvoie vos messages. Si vous continuez à consommer les messages de vos files d'attente, Amazon SQS sonde tous ses serveurs et vous recevez tous les messages.

Le schéma diagramme suivant illustre le comportement d'attente active de courte durée des messages renvoyés par une file d'attente standard suite à une demande réception de la part de l'un des composants du système. Amazon SQS sonde plusieurs de ses serveurs (en gris) et renvoie les messages A, C, D et B à partir de ces serveurs. Le message E n'est pas renvoyé pour cette requête, mais l'est pour une requête ultérieure.

Consommation des messages à l'aide de la recherche prolongée

Lorsque le temps d'attente pour l'action de l'API `ReceiveMessage` est supérieur à 0, une recherche prolongée est activée. Le temps d'attente maximal pour la recherche prolongée est de 20 secondes. La recherche prolongée permet de réduire le coût d'utilisation d'Amazon SQS en éliminant le nombre de réponses vides (lorsqu'il n'y a aucun message disponible pour une demande `ReceiveMessage`) et de fausses réponses vides (lorsque les messages sont disponibles dans la file d'attente, mais ne sont pas inclus dans une réponse). Pour plus d'informations sur l'activation d'une recherche prolongée pour une file d'attente nouvelle ou existante à l'aide de la console Amazon SQS, consultez Configuration des paramètres de file d'attente à l'aide de la console Amazon SQS. Pour connaître les bonnes pratiques, consultez Configuration d'un long sondage dans Amazon SQS.

L'attente active de longue durée offre les avantages suivants :

- Réduisez les réponses vides en permettant à Amazon SQS d'attendre qu'un message soit disponible dans une file d'attente avant d'envoyer une réponse. A moins que la connexion expire, la réponse à la demande `ReceiveMessage` contient au moins l'un des messages disponibles et, au plus, le nombre maximum de messages spécifiés dans l'action `ReceiveMessage`. Dans de rares cas, vous pouvez recevoir des réponses vides même si une file d'attente contient encore des messages, en particulier si vous spécifiez une faible valeur pour le paramètre `ReceiveMessageWaitTimeSeconds`.
- Réduisez les fausses réponses vides en interrogeant tous les serveurs Amazon SQS, plutôt qu'un sous-ensemble de ceux-ci.
- Renvoyez des messages dès qu'ils sont disponibles.

Pour obtenir des informations sur la façon de vérifier qu'une file d'attente est vide, consultez Confirmation qu'une file d'attente Amazon SQS est vide.

Différences entre les interrogations courtes et longues

L'attente active de courte durée survient lorsque le paramètre `WaitTimeSeconds` d'une réponse `ReceiveMessage` est défini sur `0` de l'une des deux manières suivantes :

- L'appel de `ReceiveMessage` définit `WaitTimeSeconds` sur `0`.
- L'appel `ReceiveMessage` ne définit pas `WaitTimeSeconds`, mais l'attribut de file d'attente `ReceiveMessageWaitTimeSeconds` est défini sur `0`.

Délai de visibilité Amazon SQS

Lorsque vous recevez un message d'une file d'attente Amazon SQS, il reste dans la file d'attente mais devient temporairement invisible pour les autres consommateurs. Cette invisibilité est contrôlée par le délai de visibilité, qui empêche les autres consommateurs de traiter le même message pendant que vous y travaillez. Amazon SQS propose deux options pour supprimer les messages après leur traitement :

- Suppression manuelle : vous supprimez explicitement les messages à l'aide de `DeleteMessage`cette action.
- Suppression automatique — Pris en charge dans certains cas AWS SDKs, les messages sont automatiquement supprimés en cas de traitement réussi, ce qui simplifie les flux de travail.

Cas d'utilisation du délai de visibilité

Gérez les tâches de longue durée : utilisez le délai de visibilité pour gérer les tâches nécessitant des délais de traitement prolongés. Définissez un délai de visibilité approprié pour les messages qui nécessitent un délai de traitement prolongé. Cela garantit que d'autres consommateurs ne reçoivent pas le même message pendant son traitement, évitant ainsi le double travail et préservant l'efficacité du système.

Mettez en œuvre des mécanismes de nouvelle tentative : prolongez le délai de visibilité par programmation pour les tâches qui ne sont pas terminées dans le délai initial. Si une tâche ne s'exécute pas dans le délai de visibilité initial, vous pouvez prolonger ce délai par programmation. Cela permet à votre système de réessayer de traiter le message sans qu'il ne soit visible pour les autres utilisateurs, ce qui améliore la tolérance aux pannes et la fiabilité. Associez-le à Dead-Letter Queues (DLQs) pour gérer les défaillances persistantes.

Coordonner les systèmes distribués : utilisez le délai de visibilité SQS pour coordonner les tâches entre les systèmes distribués. Définissez des délais de visibilité qui correspondent à vos délais de traitement attendus pour les différents composants. Cela permet de maintenir la cohérence et d'éviter les conditions de course dans les architectures distribuées complexes.

Optimisation de l'utilisation des ressources : ajustez les délais de visibilité SQS pour optimiser l'utilisation des ressources dans votre application. En définissant des délais d'expiration appropriés, vous pouvez vous assurer que les messages sont traités efficacement sans monopoliser inutilement les ressources. Cela conduit à une meilleure performance globale du système et à une meilleure rentabilité.

Réglage et ajustement du délai de visibilité

Le délai d'expiration de visibilité commence dès qu'un message vous est délivré. Au cours de cette période, vous êtes censé traiter et supprimer le message. Si vous ne le supprimez pas avant l'expiration du délai imparti, le message redevient visible dans la file d'attente et peut être récupéré par un autre consommateur. Le délai de visibilité par défaut pour une file d'attente est de 30 secondes, mais vous pouvez l'ajuster en fonction du temps dont votre application a besoin pour traiter et supprimer un message. Vous pouvez également définir un délai de visibilité spécifique pour les messages individuels sans modifier les paramètres généraux de la file d'attente. Utilisez cette `ChangeMessageVisibility`action pour prolonger ou raccourcir le délai d'expiration par programmation selon les besoins.

Messages et quotas en vol

Dans Amazon SQS, les messages en cours de vol sont des messages qui ont été reçus par un consommateur mais qui n'ont pas encore été supprimés. Pour les files d'attente standard, la limite est d'environ 120 000 messages en cours de vol, en fonction du trafic des files d'attente et de l'arriéré de messages. Si vous atteignez cette limite, Amazon SQS renvoie un `OverLimit` message d'erreur indiquant qu'aucun message supplémentaire ne peut être reçu tant que certains messages en cours de vol ne sont pas supprimés. Pour les files d'attente FIFO, les limites dépendent des groupes de messages actifs.

- Lors de l'utilisation d'un sondage court : si cette limite est atteinte lors de l'utilisation d'un sondage court, Amazon SQS renvoie une `OverLimit` erreur indiquant qu'aucun message supplémentaire ne peut être reçu tant que certains messages en cours de vol ne sont pas supprimés.
- Lorsque vous utilisez un long sondage : si vous utilisez un long sondage, Amazon SQS ne renvoie pas d'erreur lorsque la limite de messages en cours de vol est atteinte. Au lieu de cela, il ne renverra aucun nouveau message tant que le nombre de messages en vol ne sera pas inférieur à la limite.

Pour gérer efficacement les messages en vol :

1. Suppression rapide : supprimez les messages (manuellement ou automatiquement) après leur traitement afin de réduire le nombre de messages en vol.
2. Surveillez avec CloudWatch — Réglez des alarmes en cas de nombre élevé en vol afin d'éviter d'atteindre la limite.

3. Répartissez la charge : si vous traitez un volume élevé de messages, utilisez des files d'attente ou des clients supplémentaires pour équilibrer la charge et éviter les embouteillages.
4. Demander une augmentation du quota : envoyez une demande au AWS Support si des limites plus élevées sont requises.

Comprendre le délai d'expiration de visibilité dans les files d'attente standard et FIFO

Dans les files d'attente standard et FIFO (premier entré, premier sorti), le délai de visibilité permet d'empêcher plusieurs consommateurs de traiter le même message simultanément. Cependant, en raison du modèle de at-least-once diffusion d'Amazon SQS, il n'existe aucune garantie absolue qu'un message ne sera pas diffusé plus d'une fois pendant la période d'expiration de visibilité.

- Files d'attente standard — Le délai de visibilité dans les files d'attente standard empêche plusieurs clients de traiter le même message en même temps. Cependant, en raison du modèle de at-least-once diffusion, Amazon SQS ne garantit pas qu'un message ne sera pas diffusé plus d'une fois pendant le délai de visibilité imparti.
- Files d'attente FIFO — Pour les files d'attente FIFO, les messages ayant le même ID de groupe de messages sont traités dans un ordre strict. Lorsqu'un message avec un identifiant de groupe de messages est en cours de transfert, les messages suivants de ce groupe ne sont pas disponibles tant que le message en cours de vol n'est pas supprimé ou que le délai de visibilité n'expire pas. Cependant, cela ne « verrouille » pas le groupe indéfiniment : chaque message est traité en séquence, et ce n'est que lorsque chaque message est supprimé ou redevient visible que le message suivant de ce groupe sera mis à la disposition des consommateurs. Cette approche garantit un traitement ordonné au sein du groupe sans empêcher inutilement le groupe de délivrer des messages.

Gestion des défaillances

Si vous ne traitez pas et ne supprimez pas un message avant l'expiration du délai de visibilité, en raison d'erreurs d'application, de pannes ou de problèmes de connectivité, le message redevient visible dans la file d'attente. Il peut ensuite être récupéré par le même consommateur ou par un autre pour une autre tentative de traitement. Cela garantit que les messages ne sont pas perdus même en cas d'échec du traitement initial. Toutefois, le fait de définir un délai de visibilité trop élevé peut retarder la réapparition des messages non traités, ce qui peut ralentir les tentatives. Il est essentiel

de définir un délai de visibilité approprié en fonction du temps de traitement prévu pour un traitement rapide des messages.

Modification et fin du délai de visibilité

Vous pouvez modifier ou mettre fin au délai de visibilité à l'aide de l'`ChangeMessageVisibility`action suivante :

- Modification du délai d'expiration — Ajustez le délai de visibilité de manière dynamique à l'aide de. `ChangeMessageVisibility` Cela vous permet de prolonger ou de réduire les délais d'attente pour répondre aux besoins de traitement.
- Fin du délai d'attente : si vous décidez de ne pas traiter un message reçu, mettez fin à son délai de visibilité en réglant le délai `VisibilityTimeout` à 0 seconde pendant la durée de l'action. `ChangeMessageVisibility` Cela met immédiatement le message à la disposition des autres consommateurs pour qu'ils puissent le traiter.

Bonnes pratiques

Suivez les bonnes pratiques suivantes pour gérer les délais de visibilité dans Amazon SQS, notamment en définissant, en ajustant et en prolongeant les délais, ainsi que pour gérer les messages non traités à l'aide de files d'attente de lettres mortes (). DLQs

- Réglage et ajustement du délai d'attente. Commencez par définir le délai de visibilité pour qu'il corresponde à la durée maximale dont votre application a généralement besoin pour traiter et supprimer un message. Si vous n'êtes pas sûr du délai de traitement exact, commencez par un délai plus court (par exemple, 2 minutes) et prolongez-le si nécessaire. Mettez en œuvre un mécanisme de pulsation pour prolonger périodiquement le délai de visibilité, en veillant à ce que le message reste invisible jusqu'à la fin du traitement. Cela permet de minimiser les délais de retraitement des messages non traités et d'éviter une visibilité prématurée.
- Prolongation du délai d'attente et gestion de la limite de 12 heures. Si votre délai de traitement varie ou peut dépasser le délai initialement défini, utilisez l'`ChangeMessageVisibility`action pour prolonger le délai de visibilité pendant le traitement du message. N'oubliez pas que le délai de visibilité est limité à 12 heures à compter de la première réception du message. La prolongation du délai d'attente ne rétablit pas cette limite de 12 heures. Si votre traitement prend plus de temps que cette limite, envisagez d'utiliser AWS Step Functions ou de diviser la tâche en étapes plus petites.
- Gestion des messages non traités. Pour gérer les messages qui échouent à plusieurs tentatives de traitement, configurez une file d'attente de lettres mortes (DLQ). Cela garantit que les messages

qui ne peuvent pas être traités après plusieurs tentatives sont capturés séparément pour une analyse ou un traitement plus approfondis, empêchant ainsi leur circulation répétée dans la file d'attente principale.

Files d'attente Amazon SQS Fair

Les files d'attente Amazon SQS Fair atténuent automatiquement l'impact des voisins bruyants dans les files d'attente mutualisées contenant des messages provenant de plusieurs entités logiques, telles que des clients, des applications clientes ou des types de messages. Dans ces environnements de files d'attente partagées, l'un des indicateurs de performance essentiels est le temps d'attente, qui mesure le temps total que les messages passent dans une file d'attente entre leur arrivée et leur traitement. Lorsqu'un locataire crée un arriéré dans la file d'attente en publiant plus de messages que le système ne peut en traiter, des files d'attente équitables minimisent l'impact sur le temps d'attente des autres locataires.

État stable

Le schéma suivant illustre une file d'attente à locataires multiples contenant des messages provenant de quatre locataires distincts (étiquetés A, B, C et D). La file d'attente fonctionne de manière stable et il n'y a aucun arriéré de messages car les consommateurs reçoivent des messages dès qu'ils apparaissent dans la file d'attente. Tous les locataires connaissent de faibles temps de séjour. Toutes les capacités des consommateurs ne sont pas pleinement utilisées dans cet état stable.

Impact bruyant sur un voisin

Un impact bruyant sur les voisins se produit lorsqu'un locataire d'une file d'attente à locataires multiples crée un arriéré, ce qui augmente le temps d'attente des messages pour tous les autres locataires. Un locataire peut devenir un voisin bruyant en envoyant un plus grand volume de

messages que les autres locataires, ou lorsque les consommateurs mettent plus de temps à traiter les messages de ce locataire en particulier.

Ce diagramme montre comment l'augmentation du trafic provenant du locataire A crée un arriéré dans la file d'attente. Les consommateurs sont occupés à traiter les messages provenant uniquement du locataire A, tandis que les messages des autres locataires attendent dans l'arriéré, ce qui augmente les temps d'attente pour tous les locataires.

Atténuation grâce à des files d'attente équitables

Amazon SQS détecte les voisins bruyants en surveillant la distribution des messages entre les locataires pendant le traitement (état « en vol »). Lorsqu'un locataire reçoit un nombre disproportionné de messages en cours de vol par rapport aux autres, Amazon SQS identifie ce locataire comme un voisin bruyant et donne la priorité à la distribution des messages aux autres locataires. Cette approche réduit l'impact du temps de séjour pour les autres locataires.

Ce schéma montre comment les files d'attente équitables d'Amazon SQS résolvent le problème des voisins bruyants. Lorsqu'un locataire (locataire A) devient bruyant, Amazon SQS donne la priorité aux messages renvoyés par les autres locataires (B, C et D). Cette hiérarchisation permet de maintenir de faibles temps d'attente pour les locataires calmes (locataires B, C et D), tandis que le temps d'attente pour les messages du locataire A est élevé jusqu'à ce que le backlog des files d'attente soit consommé sans impact sur les autres locataires.

> **Note**
>
> Amazon SQS ne limite pas le taux de consommation par locataire. Il permet aux consommateurs de recevoir des messages de locataires voisins bruyants lorsque la capacité de consommation est atteinte et que la file d'attente n'a aucun autre message à renvoyer. À l'instar des files d'attente standard Amazon SQS, les files d'attente équitables permettent un débit pratiquement illimité, et il n'y a aucune limite quant au nombre de locataires que vous pouvez avoir dans votre file d'attente.

Différence avec les files d'attente FIFO

Les files d'attente FIFO maintiennent un ordre strict en limitant le nombre de messages envoyés en vol par chaque locataire. Bien que cela évite le bruit des voisins, cela limite le débit pour chaque locataire. Les files d'attente équitables sont conçues pour les scénarios multi-locataires dans lesquels un débit élevé, un faible temps d'attente et une allocation équitable des ressources sont des priorités. Les files d'attente équitables permettent à plusieurs consommateurs de traiter simultanément les messages d'un même locataire tout en aidant tous les locataires à maintenir des durées de séjour cohérentes.

Utiliser des files d'attente équitables

Les producteurs de vos messages peuvent ajouter un identifiant de locataire en définissant un `MessageGroupId` sur un message sortant :

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

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

La fonctionnalité d'équité sera automatiquement appliquée à toutes les files d'attente standard Amazon SQS pour les messages contenant cette propriété. MessageGroupId Il ne nécessite aucune modification du code client, n'a aucun impact sur la latence des API et ne comporte aucune limitation de débit.

Indicateurs relatifs aux files d'attente CloudWatch équitables

Amazon SQS fournit des CloudWatch mesures supplémentaires pour vous aider à contrôler l'atténuation de l'impact du bruit causé par les voisins. Par exemple, vous pouvez comparer les `Approximate..InQuietGroups` mesures avec les mesures standard au niveau de la file d'attente. Lors d'une augmentation du trafic pour un locataire en particulier, les indicateurs généraux relatifs aux files d'attente peuvent révéler des arriérés croissants ou un vieillissement des messages. Toutefois, en examinant les groupes silencieux séparément, vous pouvez constater que la plupart des groupes de messages ou des locataires peu bruyants ne sont pas concernés.

Vous trouverez ci-dessous un exemple où la métrique standard de l'arriéré de files d'attente (ApproximateNumberOfMessagesVisible) augmente en raison d'un locataire bruyant, tandis que l'arriéré pour les locataires non bruyants (ApproximateNumberOfMessagesVisibleInQuietGroups) reste faible.

Pour obtenir la liste complète des CloudWatch métriques Amazon SQS et leurs descriptions, consultez les CloudWatch métriques d'Amazon SQS.

Files d'attente à retardement Amazon SQS

Les files d'attente à retardement vous permettent de reporter la livraison des nouveaux messages aux clients de quelques secondes, par exemple lorsque votre application client a besoin de plus de

temps pour traiter les messages. Si vous créez une file d'attente à retardement, les consommateurs ne peuvent pas voir les messages que vous envoyez à la file d'attente pendant toute la durée du retardement. Le délai (minimum) par défaut pour une file d'attente est de 0 seconde. La valeur maximale est de 15 minutes. Pour plus d'informations sur la configuration de files d'attente à retardement à l'aide de la console, consultez Configuration des paramètres de file d'attente à l'aide de la console Amazon SQS.

> **Note**
>
> Pour les files d'attente standard, le retard par file d'attente n'est pas rétroactif : la modification du paramètre n'affecte pas le retard des messages déjà présents dans la file d'attente.
> Pour les files d'attente FIFO, le retard par file d'attente est rétroactif : la modification du paramètre affecte le retard des messages déjà présents dans la file d'attente.

Les files d'attente à retardement sont similaires aux délais de visibilité, car ces deux fonctions empêchent les utilisateurs d'accéder aux messages pendant une période donnée. La différence entre les deux est que pour, les files d'attente à retardement, un message est masqué lorsqu'il est ajouté initialement à la file d'attente, tandis que pour les délais de visibilité, un message est masqué uniquement après sa consommation de la file d'attente. Le diagramme suivant illustre la relation entre les files d'attente à retardement et les délais de visibilité.

Options de planification étendues

Bien que les files d'attente et les minuteries Amazon SQS permettent de planifier la livraison des messages jusqu'à 15 minutes dans le futur, vous pourriez avoir besoin de fonctionnalités de planification plus flexibles. Dans de tels cas, pensez à utiliser le EventBridge planificateur, qui vous permet de planifier des milliards d'actions d'API ponctuelles ou récurrentes sans limite de temps.

EventBridge Le planificateur est la solution recommandée pour les cas d'utilisation avancés de la planification des messages.

Pour définir un délai de quelques secondes pour des messages individuels, plutôt que pour une file d'attente complète, utilisez des temporisateurs de messages pour permettre à Amazon SQS d'utiliser la valeur du temporisateur `DelaySeconds` de messages au lieu de la valeur de la file d'attente`DelaySeconds`. EventBridge Le planificateur prend également en charge la planification de messages individuels.

Files d'attente temporaires Amazon SQS

Les files d'attente temporaires vous permettent de gagner du temps de développement et de réduire les coûts de déploiement lorsque vous utilisez des modèles de messages courants tels que demande-réponse. Vous pouvez utiliser le Client de file d'attente temporaire pour créer des files d'attente temporaires à haut débit, économiques et gérées par l'application.

Le client mappe automatiquement plusieurs files d'attente temporaires (des files d'attente gérées par des applications créées à la demande pour un processus particulier) sur une seule file d'attente Amazon SQS. Cela permet à votre application d'effectuer moins d'appels d'API et de bénéficier d'un meilleur débit lorsque le trafic vers chaque file d'attente temporaire est faible. Lorsqu'une file d'attente temporaire n'est plus utilisée, le client la supprime automatiquement, même si certains processus qui utilisent le client ne sont pas fermés correctement.

Voici les avantages des files d'attente temporaires :

- Celles-ci servent des canaux de communication légers pour des threads ou processus spécifiques.
- Elles peuvent être créées et supprimées sans encourir de frais supplémentaires.
- Elles sont compatibles API avec des files d'attente Amazon SQS statiques (normales). Cela signifie que le code existant qui envoie et reçoit des messages peut envoyer des messages et recevoir des messages dans des files d'attente virtuelles.

Files d'attente virtuelles

Les files d'attente virtuelles sont des structures de données locales créées par le client de file d'attente temporaire. Les files d'attente virtuelles vous permettent de combiner plusieurs destinations à faible trafic en une seule file d'attente Amazon SQS. Pour connaître les meilleures pratiques, voir Éviter de réutiliser le même identifiant de groupe de messages avec des files d'attente virtuelles.

> **Note**
>
> - La création d'une file d'attente virtuelle crée uniquement des structures de données temporaires dans lesquelles les consommateurs reçoivent les messages. Comme une file d'attente virtuelle n'effectue aucun appel d'API vers Amazon SQS, les files d'attente virtuelles n'entraînent aucun coût.
> - Les quotas TPS s'appliquent à toutes les files d'attente virtuelles sur une file d'attente hôte unique. Pour de plus amples informations, veuillez consulter Quotas de messages Amazon SQS.

La classe wrapper `AmazonSQSVirtualQueuesClient` ajoute la prise en charge des attributs liés aux files d'attente virtuelles. Pour créer une file d'attente virtuelle, vous devez appeler l'action d'API `CreateQueue` à l'aide de l'attribut `HostQueueURL`. Cet attribut spécifie la file d'attente existante qui héberge les files d'attente virtuelles.

L'URL d'une file d'attente virtuelle est au format suivant.

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

Lorsqu'un producteur appelle l'action d'API `SendMessage` ou `SendMessageBatch` sur une URL de file d'attente virtuelle, le client de file d'attente temporaire effectue les opérations suivantes :

1. Il extrait le nom de file d'attente virtuelle.
2. Il attache un nom de file d'attente virtuelle en tant qu'attribut de message supplémentaire.
3. Il envoie le message dans la file d'attente d'hôte.

Pendant que le producteur envoie des messages, un thread d'arrière-plan interroge la file d'attente hôte et envoie les messages reçus vers des files d'attente virtuelles en fonction des attributs de message correspondants.

Pendant que le consommateur appelle l'action d'API `ReceiveMessage` sur une URL de file d'attente virtuelle, le client de file d'attente temporaire bloque l'appel localement jusqu'à ce que le thread d'arrière-plan envoie un message dans la file d'attente virtuelle. (Ce processus est similaire à la récupération préalable des messages dans le client asynchrone mis en tampon : une seule action d'API peut fournir jusqu'à 10 messages à des files d'attente virtuelles.) La suppression d'une file d'attente virtuelle supprime les ressources côté client sans appeler Amazon SQS proprement dit.

La classe `AmazonSQSTemporaryQueuesClient` transforme automatiquement toutes les files d'attente qu'elle crée en files d'attente temporaires. Elle crée également automatiquement des files d'attente hôte avec les mêmes attributs de file d'attente, à la demande. Ces noms de files d'attente partagent un préfixe configurable commun (par défaut, `__RequesterClientQueues__`) qui les identifie comme des files d'attente temporaires. Cela permet au client d'agir comme un remplacement optimisant le code existant qui crée et supprime les files d'attente. Le client inclut également les interfaces `AmazonSQSRequester` et `AmazonSQSResponder` qui permettent la communication bidirectionnelle entre les files d'attente.

Modèle de messagerie demande-réponse (files d'attente virtuelles)

Le cas d'utilisation le plus courant pour les files d'attente temporaires est le modèle de messagerie demande-réponse dans lequel un demandeur crée une file d'attente temporaire pour la réception de chaque message de réponse. Pour éviter la création d'une file d'attente Amazon SQS pour chaque message de réponse, le client de file d'attente temporaire vous permet de créer et de supprimer plusieurs files d'attente temporaires sans effectuer des appels d'API Amazon SQS. Pour plus d'informations, consultez la section Implémentation de systèmes de demande-réponse.

Le schéma suivant montre une configuration commune qui utilise ce modèle.

Exemple de scénario : Traitement d'une demande de connexion

L'exemple de scénario suivant montre comment utiliser les interfaces `AmazonSQSRequester` et `AmazonSQSResponder` pour traiter une demande de connexion d'un utilisateur.

Côté client

```
public class LoginClient {

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

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

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

```
    }

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

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

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

L'envoi d'une demande de connexion effectue les opérations suivantes :

1. Crée une table temporaire.
2. Attache l'URL de la file d'attente temporaire au message en tant qu'attribut.
3. Envoie le message.
4. Reçoit une réponse de la file d'attente temporaire.
5. Supprime la file d'attente temporaire.
6. Renvoie la réponse.

Côté serveur

L'exemple suivant suppose que, lors de la construction, un thread est créé pour interroger la file d'attente et appeler la méthode `handleLoginRequest()` pour chaque message. En outre, `doLogin()` est une méthode assumée.

```
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));
    }
}
```

Nettoyage des files d'attente

Pour faire en sorte qu'Amazon SQS récupère les ressources en mémoire utilisées par les files d'attente virtuelles, lorsque votre application n'a plus besoin du client de file d'attente temporaire, il doit appeler la méthode `shutdown()`. Vous pouvez également utiliser la méthode `shutdown()` de l'interface `AmazonSQSRequester`.

Le client de file d'attente temporaire fournit également un moyen d'éliminer les files d'attente hôte orphelines. Pour chaque file d'attente qui reçoit un appel d'API sur une période de temps (par défaut, cinq minutes), le client utilise l'action d'API `TagQueue` pour ajouter des balises à une file d'attente qui continue d'être utilisée.

> **Note**
>
> Toute action d'API effectuée sur une file d'attente marque celle-ci comme étant non inactive, y compris une action `ReceiveMessage` qui ne renvoie aucun message.

Le thread d'arrière-plan utilise les actions d'API `ListQueues` et `ListTags` pour vérifier toutes les files d'attente avec le préfixe configuré, en supprimant les files d'attente qui n'ont pas été balisées pendant au moins cinq minutes. Ainsi, si un client n'est pas fermé correctement, les autres clients

actifs sont nettoyés après celui-ci. Afin de réduire la duplication de travail, tous les clients avec le même préfixe communiquent via une file d'attente de travail interne partagée, nommée d'après le préfixe.

Temporisateurs de messages Amazon SQS

Les temporisateurs de messages vous permettent de définir une période d'invisibilité initiale pour un message lorsqu'il est ajouté à une file d'attente. Par exemple, si vous envoyez un message avec un délai de 45 secondes, il reste caché aux consommateurs pendant les 45 premières secondes. Le délai (minimum) par défaut pour un message est de 0 seconde. La valeur maximale est de 15 minutes. Pour plus d'informations sur l'envoi de messages avec des temporisateurs à l'aide de la console, consultez Envoi d'un message à l'aide d'une file d'attente standard.

> **Note**
>
> Les files d'attente FIFO ne prennent pas en charge les temporisateurs pour les messages individuels.

Pour définir un délai pour l'ensemble d'une file d'attente, plutôt que pour des messages individuels, utilisez des files d'attente. Les valeurs des temporisateurs définis au niveau des messages individuels prévalent sur les valeurs `DelaySeconds` d'une file d'attente à retardement Amazon SQS.

Options de planification étendues

Bien que les files d'attente et les minuteries Amazon SQS permettent de planifier la livraison des messages jusqu'à 15 minutes dans le futur, vous pourriez avoir besoin de fonctionnalités de planification plus flexibles. Dans de tels cas, pensez à utiliser le EventBridge planificateur, qui vous permet de planifier des milliards d'actions d'API ponctuelles ou récurrentes sans limite de temps. EventBridge Le planificateur est la solution recommandée pour les cas d'utilisation avancés de la planification des messages.

Accès à Amazon EventBridge Pipes via la console Amazon SQS

Amazon EventBridge Pipes connecte les sources aux cibles. Les tubes sont destinés aux point-to-point intégrations entre les sources et les cibles prises en charge, avec la prise en charge des transformations avancées et de l'enrichissement. EventBridge Les pipes constituent un moyen hautement évolutif de connecter votre file d'attente Amazon SQS à des AWS services tels que Step

Functions, Amazon SQS et API Gateway, ainsi qu'à des applications logicielles en tant que service (SaaS) tierces telles que Salesforce.

Pour configurer un canal, vous devez choisir la source, ajouter un filtrage facultatif, définir un enrichissement facultatif et choisir la cible pour les données d'événement.

Sur la page de détails d'une file d'attente Amazon SQS, vous pouvez voir les pipelines qui utilisent cette file d'attente comme source. À partir de là, vous pouvez également :

- Lancez la EventBridge console pour afficher les détails du canal.
- Lancez la EventBridge console pour créer un nouveau canal avec la file d'attente comme source.

Pour plus d'informations sur la configuration d'une file d'attente Amazon SQS en tant que source de canal, consultez la section file d'attente Amazon SQS en tant que source dans le guide de l'utilisateur Amazon EventBridge . Pour plus d'informations sur EventBridge les tuyaux en général, voir EventBridge Tuyaux.

Pour accéder aux EventBridge canaux d'une file d'attente Amazon SQS donnée

1. Ouvrez la page Files d'attente de la console Amazon SQS.
2. Sélectionnez une file d'attente.
3. Sur la page détaillée de la file d'attente, choisissez l'onglet EventBridge Canalisations.

   L'onglet EventBridge Canalisations inclut une liste de tous les canaux actuellement configurés pour utiliser la file d'attente sélectionnée comme source, notamment :

   - nom du canal
   - état actuel
   - cible du canal
   - date à laquelle le canal a été modifié pour la dernière fois
4. Affichez plus de détails sur le pipeline ou créez-en un, si vous le souhaitez :

   - Pour accéder à plus de détails sur un pipeline :

     Sélectionnez le nom du pipeline.

     Cela ouvre la page de détails de Pipe de la EventBridge console.
   - Pour créer un pipeline :

Choisissez Connecter la file d'attente Amazon SQS au pipeline.

Cela lance la page Créer un canal de la EventBridge console, avec la file d'attente Amazon SQS spécifiée comme source de canal. Pour plus d'informations, consultez la section Création d'un EventBridge canal dans le guide de EventBridge l'utilisateur Amazon.

> **Important**
>
> Un message d'une file d'attente Amazon SQS est lu par un seul pipeline, puis supprimé de la file d'attente après avoir été traité, que le message corresponde ou non au filtre que vous avez configuré pour ce pipeline. Procédez avec prudence lorsque vous configurez plusieurs pipelines pour utiliser la même file d'attente que la source.

Gestion de messages Amazon SQS volumineux avec Extended Client Library et Amazon Simple Storage Service

Utilisez la bibliothèque client étendue Amazon SQS pour Java et la bibliothèque client étendue Amazon SQS pour Python pour envoyer des messages volumineux, en particulier pour des charges utiles comprises entre 256 Ko et 2 Go. Ces bibliothèques stockent la charge utile du message dans un compartiment Amazon S3 et envoient une référence à l'objet stocké dans la file d'attente Amazon SQS.

> **Note**
>
> Les bibliothèques client étendues Amazon SQS sont compatibles avec les files d'attente standard et FIFO.

Gestion de messages Amazon SQS volumineux à l'aide de Java et Amazon S3

Utilisez la bibliothèque client étendue Amazon SQS pour Java avec Amazon S3 pour gérer des messages Amazon SQS volumineux, en particulier pour des charges utiles comprises entre 256 Ko et 2 Go. La bibliothèque stocke la charge utile du message dans un compartiment Amazon S3 et envoie un message contenant une référence à l'objet stocké dans la file d'attente Amazon SQS.

Avec la bibliothèque client étendue Amazon SQS pour Java, vous pouvez :

- Spécifier si les messages sont toujours stockés dans Amazon S3 ou seulement lorsque leur taille dépasse 256 Ko.
- Envoyer un message qui fait référence un objet de message unique stocké dans un compartiment S3.
- Récupérez l'objet du message depuis un compartiment Amazon S3
- Supprimer l'objet du message d'un compartiment Amazon S3

Prérequis

L'exemple suivant utilise le SDK AWS Java. Pour installer et configurer le SDK, consultez la section Configurer le AWS SDK pour Java dans le Guide AWS SDK pour Java du développeur.

Avant d'exécuter l'exemple de code, configurez vos AWS informations d'identification. Pour plus d'informations, consultez la section Configurer les AWS informations d'identification et la région pour le développement dans le guide du AWS SDK pour Java développeur.

Le kit SDK pour Java et la bibliothèque client étendue Amazon SQS pour Java nécessitent le kit de développement J2SE 8.0 ou version ultérieure.

> **Note**
>
> Vous pouvez utiliser la bibliothèque client étendue Amazon SQS pour Java afin de gérer les messages Amazon SQS à l'aide d'Amazon S3 uniquement avec le AWS SDK pour Java. Vous ne pouvez pas le faire avec la AWS CLI console Amazon SQS, l'API HTTP Amazon SQS ou toute autre solution. AWS SDKs

AWS SDK for Java 2.x Exemple : utilisation d'Amazon S3 pour gérer des messages Amazon SQS volumineux

L'exemple de SDK pour SDK pour Java 2.x suivant utilise la bibliothèque client étendue pour Java pour traiter des messages volumineux. Dans le constructeur, le code suivant :

- Crée un compartiment Amazon S3 avec un nom aléatoire
- Crée une file d'attente SQS qui commence par `MyQueue`

- Encapsule un client Amazon S3 du SDK Java standard dans une instance de `AmazonSQSExtendedClient`

Dans `sendAnReceiveMessage` cette méthode, l'exemple envoie un message aléatoire qui est stocké dans un compartiment Amazon S3 car sa taille est supérieure à 256 Ko (taille de message maximale standard). Enfin, la méthode récupère le message et affiche les informations le concernant sur la console.

Vous pouvez consulter l'exemple complet danshttps://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());
    }
}
```

Vous pouvez utiliser Apache Maven pour configurer et créer Amazon SQS Extended Client pour votre projet Java, ou pour créer le SDK lui-même. Spécifiez les modules individuels du SDK que vous utilisez dans votre application.

```
<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>
```

Gestion de messages Amazon SQS volumineux à l'aide de Python et Amazon S3

Utilisez la bibliothèque client étendue Amazon SQS Amazon SQS pour Python avec Amazon S3 pour gérer les messages Amazon SQS volumineux, en particulier pour les charges utiles comprises entre 256 Ko et 2 Go. La bibliothèque stocke la charge utile du message dans un compartiment Amazon S3 et envoie un message contenant une référence à l'objet stocké dans la file d'attente Amazon SQS.

Avec la bibliothèque client étendue Amazon SQS pour Python, vous pouvez :

- Spécifiez si les charges utiles sont toujours stockées dans Amazon S3 ou uniquement stockées dans Amazon S3 lorsque la taille de la charge utile dépasse 256 Ko
- Envoyer un message qui fait référence à un seul objet de message stocké dans un compartiment Amazon S3
- Récupérez l'objet de charge utile correspondant dans un compartiment Amazon S3
- Supprimer l'objet de charge utile correspondant d'un compartiment Amazon S3

Prérequis

Les conditions requises pour utiliser la bibliothèque client étendue Amazon SQS pour Python sont les suivantes :

- Un AWS compte avec les informations d'identification nécessaires. Pour créer un AWS compte, accédez à la page d'AWS accueil, puis choisissez Créer un AWS compte. Suivez les instructions à l'écran. Pour plus d'informations sur les informations d'identification, consultez la section Informations d'identification.
- Un AWS SDK : l'exemple de cette page utilise le SDK AWS Python Boto3. Pour installer et configurer le SDK, consultez la documentation du AWS SDK pour Python dans le guide du développeur du AWS SDK pour Python
- Python 3.x (ou version ultérieure) et. `pip`
- La bibliothèque client étendue Amazon SQS pour Python, disponible auprès de PyPI

> **Note**
>
> Vous pouvez utiliser la bibliothèque client étendue Amazon SQS pour Python pour gérer les messages Amazon SQS à l'aide d'Amazon S3 uniquement avec le AWS SDK pour Python. Vous ne pouvez pas le faire avec la AWS CLI, la console Amazon SQS, l'API HTTP Amazon SQS ou toute autre solution. AWS SDKs

Configurer le stockage de messages

Le client Amazon SQS Extended utilise les attributs de message suivants pour configurer les options de stockage des messages Amazon S3 :

- `large_payload_support`: nom du compartiment Amazon S3 pour stocker les messages volumineux.
- `always_through_s3`: Si`True`, alors tous les messages sont stockés dans Amazon S3. Dans `False` le cas contraire, les messages inférieurs à 256 Ko ne seront pas sérialisés dans le compartiment s3. L'argument par défaut est `False`.
- `use_legacy_attribute`: Si tous `True` les messages publiés utilisent l'attribut de message réservé Legacy (`SQSLargePayloadSize`) au lieu de l'attribut de message réservé actuel (`ExtendedPayloadSize`).

Gestion de messages Amazon SQS volumineux avec la bibliothèque client étendue pour Python

L'exemple suivant crée un compartiment Amazon S3 avec un nom aléatoire. Il crée ensuite une file d'attente Amazon SQS nommée `MyQueue` et envoie un message qui est stocké dans un compartiment S3 et dont la taille est supérieure à 256 Ko à la file d'attente. Enfin, le code récupère le message, renvoie des informations sur ce dernier et le supprime, ainsi que la file d'attente et le compartiment.

```
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
```

Configuration des files d'attente Amazon SQS à l'aide de la console Amazon SQS

Utilisez la console Amazon SQS pour configurer et gérer les files d'attente et les fonctionnalités Amazon SQS. Vous pouvez également :

- Activez le chiffrement côté serveur pour améliorer la sécurité.
- Associez une file d'attente de lettres mortes pour gérer les messages non traités.
- Configurez un déclencheur pour appeler une fonction Lambda pour un traitement piloté par les événements.

Contrôle d'accès basé sur les attributs pour Amazon SQS

Qu'est-ce que le contrôle d'accès basé sur les attributs (ABAC) ?

Le contrôle d'accès basé sur les attributs (ABAC) est un processus d'autorisation qui définit les autorisations en fonction des balises associées aux utilisateurs et aux ressources. AWS L'ABAC fournit un contrôle d'accès granulaire et flexible basé sur des attributs et des valeurs, réduit les risques de sécurité liés aux stratégies reconfigurées basées sur les rôles et centralise l'audit et la gestion des stratégies d'accès. Pour plus de détails sur l'ABAC, consultez Qu'est-ce que l'ABAC pour AWS ? dans le Guide de l'utilisateur IAM.

Amazon SQS prend en charge l'ABAC en vous permettant de contrôler l'accès à vos files d'attente Amazon SQS en fonction des balises et des alias associés à une file d'attente Amazon SQS. Les clés de condition de balise et d'alias qui activent l'ABAC dans Amazon SQS autorisent les mandataires IAM à utiliser les files d'attente Amazon SQS sans modifier les stratégies ni gérer les octrois. Pour en savoir plus sur les clés de condition ABAC, consultez la section Clés de condition pour Amazon SQS dans la référence d'autorisation de service.

Avec ABAC, vous pouvez utiliser des balises pour configurer les autorisations et les stratégies d'accès IAM pour vos files d'attente Amazon SQS, ce qui vous permet de mettre à l'échelle votre gestion des autorisations. Vous pouvez créer une stratégie d'autorisation unique dans IAM à l'aide de balises que vous ajoutez à chaque rôle commercial, sans avoir à mettre à jour la stratégie chaque fois que vous ajoutez une nouvelle ressource. Vous pouvez également associer des balises aux mandataires IAM pour créer une stratégie ABAC. Vous pouvez concevoir des stratégies ABAC pour

autoriser les opérations Amazon SQS lorsque la balise du rôle d'utilisateur IAM qui effectue l'appel correspond à la balise de file d'attente Amazon SQS. Pour en savoir plus sur le balisage AWS, consultez les sections Stratégies de AWS balisage et. Balises de répartition des coûts Amazon SQS

> **Note**
>
> ABAC pour Amazon SQS est actuellement disponible dans AWS toutes les régions commerciales où Amazon SQS est disponible, avec les exceptions suivantes :
>
> - Asie-Pacifique (Hyderabad)
> - Asie-Pacifique (Melbourne)
> - Europe (Espagne)
> - Europe (Zurich)

Pourquoi utiliser l'ABAC dans Amazon SQS ?

Voici quelques avantages liés à l'utilisation de l'ABAC dans Amazon SQS :

- L'ABAC pour Amazon SQS nécessite moins de stratégies d'autorisation. Vous n'avez pas besoin de créer une stratégie pour chaque activité professionnelle. Vous pouvez utiliser des balises de ressource et de demande qui s'appliquent à plusieurs files d'attente, ce qui réduit la charge opérationnelle.
- Utilisez l'ABAC pour mettre rapidement à l'échelle les équipes. Les autorisations d'accès aux nouvelles ressources sont automatiquement accordées en fonction des balises lorsque les ressources sont correctement balisées lors de leur création.
- Utilisez les autorisations sur le mandataire IAM pour restreindre l'accès aux ressources. Vous pouvez créer des balises pour le mandataire IAM et les utiliser pour restreindre l'accès à des actions spécifiques correspondant aux balises du mandataire IAM. Cela vous permet d'automatiser le processus d'octroi des autorisations de demande.
- Suivez les personnes qui accèdent à vos ressources. Vous pouvez déterminer l'identité d'une session en consultant les attributs utilisateur dans AWS CloudTrail.

Rubriques

- Balisage pour le contrôle d'accès dans Amazon SQS
- Création d'utilisateurs IAM et de files d'attente Amazon SQS

- Test du contrôle d'accès basé sur les attributs dans Amazon SQS

Balisage pour le contrôle d'accès dans Amazon SQS

Voici un exemple d'utilisation de balises pour le contrôle d'accès dans Amazon SQS. La stratégie IAM limite un utilisateur IAM à toutes les actions Amazon SQS pour toutes les files d'attente qui incluent une balise de ressource indiquant l'environnement clé et la production de valeur. Pour plus d'informations, voir Contrôle d'accès basé sur les attributs avec balises et AWS Organizations.

JSON

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

Création d'utilisateurs IAM et de files d'attente Amazon SQS

Les exemples suivants expliquent comment créer une politique ABAC pour contrôler l'accès à Amazon SQS à l'aide AWS Management Console du et. AWS CloudFormation

À l'aide du AWS Management Console

Créer un utilisateur IAM

1. Connectez-vous à la console IAM AWS Management Console et ouvrez-la à https://console.aws.amazon.com/iam/l'adresse.

2. Choisissez Utilisateur dans le panneau de navigation de gauche.
3. Choisissez Ajouter des utilisateurs et saisissez un nom dans la zone de texte Nom d'utilisateur.
4. Sélectionnez Clé d'accès - Accès programmatique, puis choisissez Suivant : Autorisations.
5. Choisissez Suivant : balises.
6. Ajouter la clé d'étiquette en tant que `environment` et la valeur de balise en tant que `beta`.
7. Choisissez Suivant : Autorisations, puis Créer un utilisateur.
8. Conservez votre ID de clé d'accès et votre clé d'accès secrète dans un emplacement sécurisé.

Ajouter des autorisations d'utilisateur IAM

1. Sélectionnez l'utilisateur IAM que vous avez créé.
2. Sélectionnez Ajouter une politique en ligne.
3. Dans l'onglet JSON, collez la stratégie suivante :

   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. Choisissez Examiner une politique.
5. Choisissez Create Policy (Créer une politique).

En utilisant AWS CloudFormation

Utilisez l'exemple de AWS CloudFormation modèle suivant pour créer un utilisateur IAM associé à une politique intégrée et à une file d'attente Amazon SQS :

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

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

            Users:
              - "testUser"
            PolicyName: tagQueuePolicy

    IAMUser:
        Type: "AWS::IAM::User"
```

```
        Properties:
            Path: "/"
            UserName: "testUser"
            Tags:
              -
                Key: "environment"
                Value: "beta"
```

Test du contrôle d'accès basé sur les attributs dans Amazon SQS

Les exemples suivants vous montrent comment tester le contrôle d'accès basé sur les attributs dans Amazon SQS.

Créer une file d'attente avec la clé de balise définie sur environment et la valeur de balise définie sur prod

Exécutez cette commande AWS CLI pour tester la création de la file d'attente avec la clé de balise définie sur environnement et la valeur de balise définie sur prod. Si vous n'avez pas de AWS CLI, vous pouvez la télécharger et la configurer pour votre machine.

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

Vous recevez une erreur `AccessDenied` de la part du point de terminaison Amazon SQS :

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

Cela est dû au fait que la valeur de balise de l'utilisateur IAM ne correspond pas à la balise transmise lors de l'appel d'API `CreateQueue`. N'oubliez pas que nous avons appliqué une balise à l'utilisateur IAM avec la clé définie sur `environment` et la valeur définie sur `beta`.

Créer une file d'attente avec la clé de balise définie sur environment et la valeur de balise définie sur beta

Exécutez cette commande CLI pour tester la création d'une file d'attente avec la clé de balise définie sur `environment` et la valeur de balise définie sur `beta`.

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

Vous recevez un message confirmant la création réussie de la file d'attente, similaire à celui ci-dessous.

```
{
"QueueUrl": "<queueUrl>“
}
```

Envoi d'un message à une file d'attente

Exécutez cette commande CLI pour tester l'envoi d'un message à une file d'attente.

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

La réponse indique que le message a été correctement envoyé à la file d'attente Amazon SQS. L'autorisation d'utilisateur IAM vous permet d'envoyer un message à une file d'attente comportant une balise `beta`. La réponse inclut `MD5OfMessageBody` et `MessageId` contenant le message.

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

Configuration des paramètres de file d'attente à l'aide de la console Amazon SQS

Lorsque vous créez ou modifiez une file d'attente, vous pouvez configurer les paramètres suivants :

- Délai de visibilité : durée pendant laquelle un message reçu d'une file d'attente (par un consommateur) ne sera pas visible pour les autres consommateurs de messages. Pour plus d'informations, consultez Délai de visibilité.

> **Note**
>
> L'utilisation de la console pour configurer le délai de visibilité permet de configurer la valeur du délai pour tous les messages de la file d'attente. Pour configurer le délai d'expiration pour un ou plusieurs messages, vous devez utiliser l'un des AWS SDKs.

- Période de conservation des messages : durée pendant laquelle Amazon SQS conserve les messages qui restent dans la file d'attente. Par défaut, la file d'attente conserve les messages pendant quatre jours. Vous pouvez configurer une file d'attente pour conserver les messages jusqu'à 14 jours. Pour plus d'informations, consultez Période de conservation des messages.
- Retard de livraison : durée pendant laquelle Amazon SQS retardera la livraison d'un message ajouté à la file d'attente. Pour plus d'informations, consultez Retard de livraison.
- Taille maximale des messages : taille maximale des messages pour cette file d'attente. Pour plus d'informations, consultez Taille maximale des messages.
- Temps d'attente des messages de réception : durée maximale pendant laquelle Amazon SQS attend que les messages soient disponibles une fois que la file d'attente a reçu une demande de réception. Pour de plus amples informations, veuillez consulter Recherches courtes et longues sur Amazon SQS.
- Activez la déduplication basée sur le contenu : Amazon SQS peut créer automatiquement une déduplication en IDs fonction du corps du message. Pour de plus amples informations, veuillez consulter Files d'attente FIFO Amazon SQS.
- Activer le FIFO à haut débit : à utiliser pour activer le débit élevé pour les messages de la file d'attente. Le choix de cette option modifie les options associées (Portée de la déduplication et Limite de débit FIFO) en fonction des paramètres requis pour activer un débit élevé pour les files d'attente FIFO. Pour plus d'informations, consultez Débit élevé pour les files d'attente FIFO dans Amazon SQS et Quotas de messages Amazon SQS.
- Stratégie d'autorisation de redirection : définit les files d'attente source pouvant utiliser cette file d'attente comme file d'attente de lettres mortes. Pour de plus amples informations, veuillez consulter Utilisation de files d'attente contenant des lettres mortes dans Amazon SQS .

Pour configurer des paramètres de file d'attente pour une file d'attente existante (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente. Choisissez une file d'attente, puis sélectionnez Modifier.
3. Accédez à la section Configuration.
4. Pour le Délai de visibilité, saisissez la durée et les unités. La plage est comprise entre 0 seconde et 12 heures. La valeur par défaut est de 30 secondes.
5. Pour Période de conservation des messages, saisissez la durée et les unités. La plage est comprise entre 1 minute et 14 jours. La valeur par défaut est de 4 jours.

6. Pour une file d'attente standard, saisissez une valeur pour Temps d'attente du message de réception. La plage est comprise entre 0 et 20 secondes. La valeur par défaut est 0 seconde, qui permet de définir la recherche courte. Toute valeur différente de zéro définit une recherche longue.
7. Pour Retard de diffusion, saisissez la durée et les unités. La plage est comprise entre 0 seconde et 15 minutes. La valeur par défaut est de 0 seconde.
8. Pour Taille maximale du message, saisissez une valeur. La plage est comprise entre 1 KiB et 1024 KiB. La valeur par défaut est 1024 KiB.
9. Pour une file d'attente FIFO, choisissez Activer la déduplication basée sur le contenu pour activer cette option. Par défaut, ce paramètre est désactivé.
10. (Facultatif) Pour qu'une file d'attente FIFO permette un débit plus élevé pour l'envoi et la réception de messages dans la file d'attente, choisissez Activer le FIFO à haut débit.

    Le choix de cette option modifie les options associées (Portée de la déduplication et Limite de débit FIFO) en fonction des paramètres requis pour activer un débit élevé pour les files d'attente FIFO. Si vous modifiez l'un des paramètres requis pour utiliser le FIFO à débit élevé, le débit normal est effectif pour la file d'attente et la déduplication se produit comme indiqué. Pour plus d'informations, consultez Débit élevé pour les files d'attente FIFO dans Amazon SQS et Quotas de messages Amazon SQS.
11. Pour la Stratégie d'autorisation de redirection, choisissez Activé. Sélectionnez l'une des options suivantes : Tout autoriser (par défaut), Par file d'attente ou Refuser tout. Lorsque vous choisissez Par file d'attente, spécifiez une liste de 10 files d'attente source maximum en fonction de l'Amazon Resource Name (ARN).
12. Lorsque vous avez fini de configurer les paramètres de la file d'attente, choisissez Enregistrer.

Configuration d'une politique d'accès dans Amazon SQS

Lorsque vous modifiez une file d'attente, vous pouvez configurer sa politique d'accès pour contrôler qui peut interagir avec elle.

- La politique d'accès définit les comptes, les utilisateurs et les rôles autorisés à accéder à la file d'attente.
- Il indique les actions autorisées, telles que `SendMessageReceiveMessage`, ou `DeleteMessage`.

- Par défaut, seul le propriétaire de la file d'attente est autorisé à envoyer et à recevoir des messages.

Pour configurer la politique d'accès pour une file d'attente existante (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Choisissez une file d'attente, puis sélectionnez Modifier.
4. Accédez à la section stratégie d'accès.
5. Modifiez les déclarations de politique d'accès dans la zone de saisie. Pour en savoir plus sur les instructions de stratégie d'accès, consultez Gestion des identités et des accès dans Amazon SQS.
6. Lorsque vous avez terminé de configurer la stratégie d'accès, choisissez Enregistrer.

Configuration du chiffrement côté serveur pour une file d'attente à l'aide de clés de chiffrement gérées par SQL

Outre l'option de chiffrement côté serveur (SSE) géré par Amazon SQS par défaut, le SSE géré par Amazon SQS (SSE-SQS) vous permet de créer un chiffrement géré personnalisé côté serveur qui utilise des clés de chiffrement gérées par SQS pour protéger les données sensibles envoyées via des files d'attente de messages. Avec le SSE-SQS, vous n'avez pas besoin de créer et de gérer des clés de chiffrement, ni de modifier votre code pour chiffrer vos données. Le SSE-SQS vous permet de transmettre des données en toute sécurité et de respecter les exigences réglementaires et de conformité strictes en matière de chiffrement, sans frais supplémentaires.

Le SSE-SQS protège les données au repos à l'aide d'un chiffrement Advanced Encryption Standard 256 bits (AES-256). SSE chiffre les messages une fois reçus par Amazon SQS. Amazon SQS stocke les messages sous forme chiffrée et ne les déchiffre que lorsqu'ils sont envoyés à un consommateur autorisé.

> **Note**
>
> - L'option SSE par défaut n'est efficace que lorsque vous créez une file d'attente sans spécifier d'attributs de chiffrement.

- Amazon SQS vous permet de désactiver le chiffrement de toutes les files d'attente. Par conséquent, la désactivation du KMS-SSE n'activera pas automatiquement le SQS-SSE. Si vous souhaitez activer le SQS-SSE après avoir désactivé le KMS-SSE, vous devez ajouter un changement d'attribut dans la demande.

Pour configurer le chiffrement SSE-SQS pour une file d'attente (console)

> **Note**
>
> Toute nouvelle file d'attente créée à l'aide du point de terminaison HTTP (non-TLS) n'activera pas le chiffrement SSE-SQS par défaut. La création de files d'attente Amazon SQS à l'aide de points de terminaison HTTPS ou Signature Version 4 constitue une bonne pratique en matière de sécurité.

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Choisissez une file d'attente, puis sélectionnez Modifier.
4. Développez Chiffrement.
5. Sous Chiffrement côté serveur, choisissez Activer (par défaut).

   > **Note**
   >
   > Lorsque SSE est activé, les demandes anonymes `SendMessage` et `ReceiveMessage` adressées à la file d'attente chiffrée sont rejetées. Les bonnes pratiques de sécurité d'Amazon SQS recommandent de ne pas utiliser de demandes anonymes. Si vous souhaitez envoyer des demandes anonymes à une file d'attente Amazon SQS, veillez à désactiver SSE.

6. Sélectionnez la clé Amazon SQS (SSE-SQS). Aucuns frais supplémentaires ne vous sont facturés pour l'utilisation de cette option.
7. Choisissez Enregistrer.

Configuration du chiffrement côté serveur pour une file d'attente à l'aide de la console Amazon SQS

Pour protéger les données contenues dans les messages d'une file d'attente, Amazon SQS a activé le chiffrement côté serveur (SSE) par défaut pour toutes les files d'attente nouvellement créées. Amazon SQS s'intègre au service de gestion des clés Amazon Web Services (Amazon Web Services KMS) afin de gérer les clés KMS pour le chiffrement côté serveur (SSE). Pour plus d'informations sur l'utilisation du chiffrement SSE, consultez Chiffrement au repos dans Amazon SQS.

La clé KMS que vous attribuez à votre file d'attente doit avoir une stratégie de clé qui inclut des autorisations pour tous les mandataires autorisés à utiliser la file d'attente. Pour plus d'informations, consultez Gestion des clés.

Si vous n'êtes pas le propriétaire de la clé KMS, ou si vous vous connectez avec un compte n'ayant pas les autorisations `kms:ListAliases` et `kms:DescribeKey`, vous ne pouvez pas afficher les informations relatives à la clé KMS sur la console Amazon SQS. Demandez au propriétaire de la clé KMS de vous accorder ces autorisations. Pour plus d'informations, consultez Gestion des clés.

Lorsque vous créez ou modifiez une file d'attente, vous pouvez configurer le SSE-KMS.

Pour configurer le SSE-KMS pour une file d'attente existante (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Choisissez une file d'attente, puis sélectionnez Modifier.
4. Développez Chiffrement.
5. Sous Chiffrement côté serveur, choisissez Activer (par défaut).

> **Note**
>
> Lorsque SSE est activé, les demandes anonymes `SendMessage` et `ReceiveMessage` adressées à la file d'attente chiffrée sont rejetées. Les bonnes pratiques de sécurité d'Amazon SQS recommandent de ne pas utiliser de demandes anonymes. Si vous souhaitez envoyer des demandes anonymes à une file d'attente Amazon SQS, veillez à désactiver SSE.

6. Sélectionnez CléAWS Key Management Service (SSE-KMS).

La console affiche la Description, le Compte et l'ARN de la clé KMS.

7. Spécifiez l'ID de clé KMS pour la file d'attente. Pour de plus amples informations, veuillez consulter Termes clés.

   a. Choisissez l'option Choisir un alias de clé KMS.

   b. La clé par défaut est la clé KMS gérée par Amazon Web Services pour Amazon SQS. Pour utiliser cette clé, choisissez-la dans la liste des clés KMS.

   c. Pour utiliser une clé KMS personnalisée depuis votre compte Amazon Web Services, choisissez-la dans la liste des clés KMS. Pour obtenir des instructions sur la création de clés KMS personnalisées, consultez Création de clés dans le Guide du développeur Amazon Web Services Key Management Service.

   d. Pour utiliser une clé KMS personnalisée qui ne figure pas dans la liste, ou une clé KMS personnalisée provenant d'un autre compte Amazon Web Services, choisissez Saisir l'alias de la clé KMS, puis saisissez l'Amazon Resource Name (ARN) de la clé KMS.

8. (Facultatif) Pour la Période de réutilisation des clés de données, spécifiez une valeur comprise entre 1 minute et 24 heures. La valeur par défaut est de 5 minutes. Pour de plus amples informations, veuillez consulter Présentation de la période de réutilisation des clés de données.

9. Lorsque vous avez terminé de configurer le SSE-KMS, choisissez Enregistrer.

Configuration des balises de répartition des coûts pour une file d'attente à l'aide de la console Amazon SQS

Pour organiser et identifier vos files d'attente Amazon SQS, vous pouvez ajouter des balises de répartition des coûts. Pour de plus amples informations, veuillez consulter Balises de répartition des coûts Amazon SQS.

- L'onglet Balisage de la page Détails affiche les balises de la file d'attente.
- Vous pouvez ajouter ou modifier des balises lors de la création ou de la modification d'une file d'attente.

Pour configurer les balises d'une file d'attente existante (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.

3. Choisissez une file d'attente, puis sélectionnez Modifier.
4. Faites défiler jusqu'à la section Balises.
5. Ajouter, modifier ou supprimer des balises de file d'attente :
   a. Pour ajouter une balise, choisissez Ajouter une balise, saisissez une clé et une valeur, puis choisissez Ajouter une nouvelle balise.
   b. Pour mettre à jour une balise, modifiez sa clé et sa valeur.
   c. Pour supprimer une balise, choisissez Supprimer en regard de sa paire clé-valeur.
6. Lorsque vous avez terminé de configurer les balises, choisissez Enregistrer.

Abonnement d'une file d'attente à une rubrique Amazon SNS à l'aide de la console Amazon SQS

Vous pouvez abonner une ou plusieurs files d'attente Amazon SQS à une rubrique Amazon SNS. Lorsque vous publiez un message dans une rubrique, Amazon SNS envoie le message à chaque file d'attente abonnée. Amazon SQS gère l'abonnement et gère les autorisations requises. Pour plus d'informations sur Amazon SNS, consultez Qu'est-ce qu'Amazon SNS ? dans le Guide du développeur Amazon Simple Notification Service.

Lorsque vous abonnez une file d'attente Amazon SQS à une rubrique Amazon SNS, Amazon SNS utilise le protocole HTTPS pour transférer les messages vers Amazon SQS. Pour plus d'informations sur l'utilisation d'Amazon SNS avec des files d'attente Amazon SQS chiffrées, consultez Configuration des autorisations KMS pour les AWS services.

> **Important**
>
> Amazon SQS prend en charge un maximum de 20 instructions pour chaque politique d'accès. L'abonnement à une rubrique Amazon SNS ajoute ce type d'instruction. Le dépassement de ce montant entraînera l'échec de la livraison de l'abonnement à la rubrique.

Pour abonner une file d'attente à une rubrique Amazon SNS (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Dans la liste des files d'attente, choisissez la file d'attente à abonner à la rubrique Amazon SNS.

4. Dans Actions, choisissez Subscribe to Amazon SNS topic (Abonner à la rubrique Amazon SNS).
5. Dans le menu Spécifier une rubrique Amazon SNS disponible pour cette file d'attente, choisissez la rubrique Amazon SNS pour votre file d'attente.

   Si le sujet SNS n'est pas répertorié, choisissez Enter Amazon SNS topic ARN, puis entrez le Amazon Resource Name (ARN) du sujet.
6. Choisissez Enregistrer.
7. Pour vérifier l'abonnement, publiez un message dans le sujet et affichez-le dans la file d'attente. Pour plus d'informations, consultez Diffusion de messages Amazon SNS dans le Guide du développeur Amazon Simple Notification Service.

Abonnements multi-comptes

Si votre file d'attente Amazon SQS et votre rubrique Amazon SNS sont Comptes AWS différentes, des autorisations supplémentaires sont requises.

Propriétaire du sujet (compte A)

Modifiez la politique d'accès de la rubrique Amazon SNS pour autoriser les files d'attente Compte AWS Amazon SQS à s'abonner. Exemple de déclaration de politique :

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

Cette politique permet de s'abonner `111122223333` à un compte`MyTopic`.

Propriétaire de la file d'attente (compte B)

Modifiez la politique d'accès de la file d'attente Amazon SQS pour autoriser la rubrique Amazon SNS à envoyer des messages. Exemple de déclaration de politique :

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

Cette politique permet `MyTopic` d'envoyer des messages à`MyQueue`.

Abonnements interrégionaux

Pour vous abonner à une autre rubrique Amazon SNS Région AWS, assurez-vous que :

- La politique d'accès de la rubrique Amazon SNS autorise les abonnements entre régions.
- La politique d'accès de la file d'attente Amazon SQS permet à la rubrique Amazon SNS d'envoyer des messages entre les régions.

Pour plus d'informations, consultez la section Envoyer des messages Amazon SNS à une file d'attente Amazon SQS AWS Lambda ou à une fonction dans une autre région dans le manuel Amazon Simple Notification Service Developer Guide.

Configuration d'une file d'attente Amazon SQS pour déclencher une fonction AWS Lambda

Vous pouvez utiliser une fonction Lambda pour traiter les messages provenant d'une file d'attente Amazon SQS. Lambda interroge la file d'attente et appelle votre fonction de manière synchrone, en transmettant un lot de messages sous forme d'événement.

Configuration du délai d'expiration de visibilité

Définissez le délai de visibilité de la file d'attente à au moins six fois le délai d'expiration de la fonction. Cela garantit que Lambda dispose de suffisamment de temps pour réessayer si une fonction est limitée lors du traitement d'un lot précédent.

Utilisation d'une file d'attente de lettres mortes (DLQ)

Spécifiez une file d'attente de lettres mortes pour capturer les messages que la fonction Lambda ne parvient pas à traiter.

Gestion de plusieurs files d'attente et fonctions

Une fonction Lambda peut traiter plusieurs files d'attente en créant une source d'événements distincte pour chaque file d'attente. Vous pouvez également associer plusieurs fonctions Lambda à la même file d'attente.

Autorisations pour les files d'attente chiffrées

Si vous associez une file d'attente chiffrée à une fonction Lambda mais que Lambda n'interroge pas les messages, ajoutez l'autorisation `kms:Decrypt` à votre rôle d'exécution Lambda.

Restrictions

La file d'attente et la fonction Lambda doivent être identiques. Région AWS

Une file d'attente chiffrée qui utilise la clé par défaut (clé KMS AWS gérée pour Amazon SQS) ne peut pas appeler de fonction Lambda dans un autre. Compte AWS

Pour plus de détails sur l'implémentation, consultez la section Utilisation AWS Lambda avec Amazon SQS dans le Guide du AWS Lambda développeur.

Prérequis

Pour configurer les déclencheurs de la fonction Lambda, vous devez respecter les conditions requises suivantes :

- Si vous faites appel à un utilisateur, votre rôle Amazon SQS doit inclure les autorisations suivantes :
  - `lambda:CreateEventSourceMapping`
  - `lambda:ListEventSourceMappings`
  - `lambda:ListFunctions`
- Le rôle d'exécution Lambda doit inclure les autorisations suivantes :
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `sqs:ReceiveMessage`
- Si vous associez une file d'attente chiffrée à une fonction Lambda, ajoutez l'autorisation `kms:Decrypt` à votre rôle d'exécution Lambda.

Pour de plus amples informations, veuillez consulter Présentation de la gestion de l'accès dans Amazon SQS.

Pour configurer une file d'attente afin de déclencher une fonction Lambda (console)

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Sur la page Files d'attente, choisissez la file d'attente à configurer.
4. Sur la page de la file d'attente, choisissez l'onglet Déclencheurs de fonction Lambda.
5. Sur la page Déclencheurs de fonction Lambda, choisissez un déclencheur de fonction Lambda.

   Si la liste n'inclut pas le déclencheur de fonction Lambda dont vous avez besoin, choisissez Configurer le déclencheur de fonction Lambda. Saisissez l'Amazon Resource Name (ARN) de la fonction Lambda ou choisissez une ressource existante. Ensuite, choisissez Save (Enregistrer).
6. Choisissez Enregistrer. La console enregistre la configuration et affiche la page Détails de la file d'attente.

   Sur la page Détails, l'onglet Déclencheurs de fonction Lambda affiche la fonction Lambda et son statut. L'association de la fonction Lambda à votre file d'attente peut prendre environ 1 minute.
7. Pour vérifier le résultat de la configuration, envoyez un message à votre file d'attente et affichez la fonction Lambda déclenchée dans la console Lambda.

Automatiser les notifications envoyées par les AWS services à Amazon SQS à l'aide d'Amazon EventBridge

Amazon vous EventBridge permet d'automatiser Services AWS et de répondre à des événements, tels que des problèmes d'application ou des modifications de ressources, en temps quasi réel.

- Vous pouvez créer des règles pour filtrer des événements spécifiques et définir des actions automatisées lorsqu'une règle correspond à un événement.
- EventBridge prend en charge plusieurs cibles, notamment les files d'attente standard Amazon SQS et FIFO, qui reçoivent des événements au format JSON.

Pour plus d'informations, consultez les EventBridge cibles Amazon dans le guide de EventBridge l'utilisateur Amazon.

Envoi d'un message avec des attributs à l'aide d'Amazon SQS

Pour les files d'attente standard et FIFO, vous pouvez inclure des métadonnées structurées dans les messages, notamment des horodatages, des données géospatiales, des signatures et des identifiants. Pour de plus amples informations, veuillez consulter Attributs de message Amazon SQS.

Pour envoyer un message contenant des attributs à une file d'attente à l'aide de la console Amazon SQS

1. Ouvrez la console Amazon SQS à l'adresse. https://console.aws.amazon.com/sqs/
2. Dans le volet de navigation, choisissez Files d'attente.
3. Sur la page Files d'attente, choisissez une file d'attente.
4. Choisissez Envoyer et recevoir des messages.
5. Saisissez les paramètres d'attribut du message.
   a. Dans la zone de texte Nom, saisissez un nom unique comportant jusqu'à 256 caractères.
   b. Pour le type d'attribut, choisissez Chaîne, Nombre ou Binaire.
   c. (Facultatif) Saisissez un type de données personnalisé. Par exemple, vous pouvez ajouter **`byte`**, **`int`** ou **`float`** en tant que types de données personnalisés pour Nombre.
   d. Dans la zone de texte de la valeur, saisissez la valeur de l'attribut de message.

6. Pour ajouter un autre attribut de message, sélectionnez Ajouter un nouvel attribut.

7. Vous pouvez modifier les valeurs d'attribut à tout moment avant d'envoyer le message.

8. Pour supprimer un attribut, choisissez Supprimer. Pour supprimer le premier attribut, fermez les attributs du message.
9. Lorsque vous avez fini d'ajouter des attributs au message, choisissez Envoyer un message. Votre message est envoyé et la console affiche un message de réussite. Pour afficher les informations relatives aux attributs du message envoyé, choisissez Afficher les détails. Choisissez Terminé pour fermer la boîte de dialogue Détails du message.

Bonnes pratiques relatives à Amazon SQS

Amazon SQS gère et traite les files d'attente de messages, permettant aux différentes parties d'une application d'échanger des messages de manière fiable et à grande échelle. Cette rubrique décrit les meilleures pratiques opérationnelles clés, notamment l'utilisation de longs sondages pour réduire le nombre de réponses vides, la mise en place de files d'attente contenant des lettres mortes pour gérer les erreurs de traitement et l'optimisation des autorisations de file d'attente pour des raisons de sécurité.

Rubriques

- Gestion des erreurs et messages problématiques sur Amazon SQS
- Déduplication et regroupement des messages Amazon SQS
- Traitement et chronométrage des messages Amazon SQS

Gestion des erreurs et messages problématiques sur Amazon SQS

Cette rubrique fournit des instructions détaillées sur la gestion et l'atténuation des erreurs dans Amazon SQS, notamment les techniques de gestion des erreurs de demande, de capture des messages problématiques et de configuration de la conservation des files d'attente de lettres mortes afin de garantir la fiabilité des messages.

Rubriques

- Gestion des erreurs de demande dans Amazon SQS
- Capture des messages problématiques dans Amazon SQS
- Configuration de la conservation des files d'attente de lettres mortes dans Amazon SQS

Gestion des erreurs de demande dans Amazon SQS

Pour gérer les erreurs de demande, utilisez l'une des stratégies suivantes :

- Si vous utilisez un AWS SDK, vous disposez déjà d'une logique de réessai et d'annulation automatiques. Pour plus d'informations, consultez Error Retries and Exponential Backoff in AWS(Tentatives sur l'erreur et backoff exponentiel)) dans Référence générale d'Amazon Web Services.

- Si vous n'utilisez pas les fonctionnalités du AWS SDK pour réessayer et annuler, attendez une pause (par exemple, 200 ms) avant de réessayer l'ReceiveMessageaction après n'avoir reçu aucun message, un délai d'expiration ou un message d'erreur d'Amazon SQS. Pour permettre une utilisation ultérieure de `ReceiveMessage` qui donne les mêmes résultats, patientez plus longtemps (par exemple, 400 ms).

Capture des messages problématiques dans Amazon SQS

Pour capturer tous les messages qui ne peuvent pas être traités et pour collecter des CloudWatch statistiques précises, configurez une file d'attente de lettres mortes.

- La stratégie de redirection redirige les messages vers une file d'attente de lettres mortes lorsque la file d'attente source ne parvient pas à traiter un message après un nombre de tentatives spécifique.
- L'utilisation d'une file d'attente de lettres mortes limite le nombre de messages et réduit la possibilité d'être exposé à des messages de type « poison pill » (messages reçus mais ne pouvant pas être traités).
- L'inclusion d'un message de pilule empoisonnée dans une file d'attente peut fausser la `ApproximateAgeOfOldestMessage` CloudWatch métrique en indiquant un âge incorrect au message de pilule empoisonnée. La configuration d'une file d'attente de lettres mortes contribue à éviter les fausses alarmes lors de l'utilisation de cette métrique.

Configuration de la conservation des files d'attente de lettres mortes dans Amazon SQS

Pour les files d'attente standard, l'expiration d'un message est toujours basée sur son horodatage de mise en file d'attente d'origine. Lorsqu'un message est déplacé vers une file d'attente de lettres mortes, l'horodatage de la mise en file d'attente reste inchangé. La métrique `ApproximateAgeOfOldestMessage` indique à quel moment le message a été placé dans la file d'attente de lettres mortes, et non à quel moment le message a été initialement envoyé. Supposons, par exemple, qu'un message passe 1 journée dans la file d'attente d'origine avant d'être déplacé vers une file d'attente de lettres mortes. Si la période de conservation de la file d'attente de lettres mortes est de 4 jours, le message est supprimé de la file d'attente de lettres mortes au bout de 3 jours et le paramètre `ApproximateAgeOfOldestMessage` est défini sur 3 jours. Il est donc recommandé de toujours définir la période de rétention d'une file d'attente de lettres mortes de manière à ce qu'elle soit plus longue que la période de rétention de la file d'attente d'origine.

Pour les files d'attente FIFO, l'horodatage de la mise en file d'attente est réinitialisé lorsque le message est déplacé vers une file d'attente de lettres mortes. La métrique `ApproximateAgeOfOldestMessage` indique à quel moment le message a été placé dans la file d'attente de lettres mortes. Dans le même exemple ci-dessus, le message est supprimé de la file d'attente de lettres mortes au bout de 4 jours et le paramètre `ApproximateAgeOfOldestMessage` est défini sur 4 jours.

Déduplication et regroupement des messages Amazon SQS

Cette rubrique présente les meilleures pratiques pour garantir un traitement cohérent des messages dans Amazon SQS. Il explique comment utiliser :

- `MessageDeduplicationId`pour empêcher la duplication de messages dans les files d'attente FIFO.
- `MessageGroupId`pour gérer l'ordre des messages au sein de groupes de messages distincts.

Rubriques

- Éviter le traitement incohérent des messages dans Amazon SQS
- Utilisation de l'ID de déduplication des messages dans Amazon SQS
- Utilisation de l'ID de groupe de messages avec les files d'attente FIFO Amazon SQS
- Utilisation de l'ID de tentative de demande de réception Amazon SQS

Éviter le traitement incohérent des messages dans Amazon SQS

Amazon SQS est un système distribué. Il est donc possible qu'un consommateur ne reçoive pas de message, même lorsque Amazon SQS marque le message comme remis lors d'un retour réussi à partir d'un appel de méthode d'API `ReceiveMessage`. Dans ce cas, Amazon SQS enregistre le message tel qu'il a été remis au moins une fois, bien que le consommateur ne l'ait jamais reçu. Étant donné qu'aucune tentative supplémentaire de remise de messages n'est effectuée dans ces conditions, nous ne recommandons pas de définir le nombre maximal de réceptions sur 1 pour une file d'attente de lettres mortes.

Utilisation de l'ID de déduplication des messages dans Amazon SQS

`MessageDeduplicationId`est un jeton utilisé uniquement dans les files d'attente FIFO Amazon SQS pour empêcher la livraison de messages en double. Il garantit que dans une fenêtre de

déduplication de 5 minutes, une seule instance d'un message portant le même ID de déduplication est traitée et délivrée.

Si Amazon SQS a déjà accepté un message avec un ID de déduplication spécifique, tous les messages suivants portant le même identifiant feront l'objet d'un accusé de réception mais ne seront pas remis aux consommateurs.

> **Note**
>
> Amazon SQS continue de suivre l'ID de déduplication même après réception et suppression du message.

Rubriques

- Quand fournir un ID de déduplication des messages dans Amazon SQS
- Activation de la déduplication pour un système producteur/consommateur unique dans Amazon SQS
- Scénarios de reprise en cas de panne dans Amazon SQS
- Configuration des délais de visibilité dans Amazon SQS

Quand fournir un ID de déduplication des messages dans Amazon SQS

Le producteur doit spécifier un ID de déduplication des messages dans les scénarios suivants :

- Lors de l'envoi de corps de message identiques, ils doivent être traités comme uniques.
- Lorsque vous envoyez des messages ayant le même contenu mais des attributs différents, assurez-vous que chaque message est traité séparément.
- Lorsque vous envoyez des messages avec un contenu différent (par exemple, un compteur de nouvelles tentatives dans le corps du message) mais que vous demandez à Amazon SQS de les reconnaître comme des doublons.

Activation de la déduplication pour un système producteur/consommateur unique dans Amazon SQS

Si vous avez un seul producteur et un seul consommateur, et que les messages sont uniques car ils incluent un ID de message spécifique à l'application dans le corps du message, suivez les bonnes pratiques suivantes :

- Activez la déduplication basée sur le contenu pour la file d'attente (chacun de vos messages a un corps unique). Le producteur peut ignorer l'ID de déduplication du message.
- Lorsque la déduplication basée sur le contenu est activée pour une file d'attente Amazon SQS FIFO et qu'un message est envoyé avec un ID de déduplication, celui-ci remplace l'ID de déduplication basé sur le contenu `SendMessage`généré.
- Même si le consommateur n'est pas tenu de fournir un ID de tentative de demande de réception pour chaque demande, il vaut mieux le faire, car cela permet aux séquences échec-réessayer de s'exécuter plus rapidement.
- Vous pouvez réessayer d'envoyer ou de recevoir des demandes, car elles n'interfèrent pas avec l'ordre des messages dans les files d'attente FIFO.

Scénarios de reprise en cas de panne dans Amazon SQS

Le processus de déduplication dans les files d'attente FIFO est prioritaire. Lors de la conception de votre application, veillez à ce que le producteur et le consommateur puissent se rétablir après une panne du client ou du réseau sans créer de doublons ni d'erreurs de traitement.

Considérations relatives aux producteurs

- Amazon SQS applique une fenêtre de déduplication de 5 minutes.
- Si un producteur réessaie une `SendMessage`demande après 5 minutes, Amazon SQS la traite comme un nouveau message, ce qui peut créer des doublons.

Considérations relatives aux consommateurs

- Si un client ne traite pas un message avant l'expiration du délai de visibilité, un autre client peut le recevoir et le traiter, ce qui entraîne un double traitement.
- Ajustez le délai de visibilité en fonction du temps de traitement de votre demande.

- Utilisez l'`ChangeMessageVisibility`API pour prolonger le délai d'attente pendant qu'un message est toujours en cours de traitement.
- Si le traitement d'un message échoue à plusieurs reprises, acheminez-le vers une file d'attente de lettres mortes (DLQ) au lieu de le retraiter indéfiniment.

- Le producteur doit connaître l'intervalle de déduplication de la file d'attente. Amazon SQS a un intervalle de déduplication de 5 minutes. De nouvelles tentatives de demandes de `SendMessage` après expiration du délai de déduplication peuvent introduire des messages en double dans la file d'attente. Par exemple, un appareil mobile dans une voiture envoie des messages dont l'ordre est important. Si la voiture perd la connectivité cellulaire pendant un certain temps avant de recevoir une confirmation, une nouvelle tentative de la demande après la récupération de la connectivité cellulaire peut créer un doublon.
- Le consommateur doit disposer d'un délai de visibilité réduisant le risque de ne pas pouvoir traiter des messages avant l'expiration de ce délai. Vous pouvez étendre le délai de visibilité pendant que les messages sont en cours de traitement en appelant l'action `ChangeMessageVisibility`. Toutefois, si le délai de visibilité expire, un autre consommateur peut immédiatement commencer à traiter les messages, un message étant alors traité plusieurs fois. Pour éviter ce scénario, configurez une file d'attente de lettres mortes.

Configuration des délais de visibilité dans Amazon SQS

Pour garantir un traitement fiable des messages, définissez le délai de visibilité pour qu'il soit supérieur au délai de lecture du AWS SDK. Cela s'applique lorsque l'`ReceiveMessage`API est utilisée à la fois pour des interrogations courtes et des interrogations longues. Un délai de visibilité plus long empêche les messages d'être accessibles aux autres consommateurs avant que la demande initiale ne soit terminée, réduisant ainsi le risque de double traitement.

Utilisation de l'ID de groupe de messages avec les files d'attente FIFO Amazon SQS

Dans les files d'attente FIFO (First-In-First-Out), se `MessageGroupId`trouve un attribut qui organise les messages en groupes distincts. Les messages d'un même groupe de messages sont toujours traités un par un, dans un ordre strict, afin que deux messages du même groupe ne soient pas traités simultanément. Dans les files d'attente standard, l'utilisation `MessageGroupId` permet d'établir des files d'attente équitables. Si un ordre strict est requis, utilisez une file d'attente FIFO.

Rubriques

- Entrelacement de plusieurs groupes de messages ordonnés dans Amazon SQS
- Empêcher le traitement dupliqué dans un système multiproducteur/consommateur dans Amazon SQS
- Évitez les gros arriérés de messages avec le même ID de groupe de messages dans Amazon SQS
- Évitez de réutiliser le même ID de groupe de messages avec les files d'attente virtuelles dans Amazon SQS

Entrelacement de plusieurs groupes de messages ordonnés dans Amazon SQS

Pour entrelacer plusieurs groupes de messages ordonnés dans une seule file d'attente FIFO, attribuez un `MessageGroupId`à chaque groupe (par exemple, des données de session pour différents utilisateurs). Cela permet à plusieurs consommateurs de lire simultanément des informations dans la file d'attente tout en garantissant que les messages d'un même groupe sont traités dans l'ordre.

Lorsqu'un message contenant un message spécifique `MessageGroupId` est en cours de traitement et qu'il est invisible, aucun autre consommateur ne peut traiter les messages du même groupe tant que le délai de visibilité n'est pas expiré ou que le message n'est pas supprimé.

Empêcher le traitement dupliqué dans un système multiproducteur/consommateur dans Amazon SQS

Dans un système à haut débit et à faible latence où l'ordre des messages n'est pas une priorité, les producteurs peuvent attribuer une valeur unique `MessageGroupId`à chaque message. Cela garantit que les files d'attente FIFO Amazon SQS éliminent les doublons, même dans une configuration multi-producteurs/multi-consommateurs. Bien que cette approche évite les doublons, elle ne garantit pas l'ordre des messages puisque chaque message est traité comme un groupe indépendant.

Dans tout système comportant plusieurs producteurs et consommateurs, il existe toujours un risque de double livraison. Si un client ne traite pas un message avant l'expiration du délai de visibilité, Amazon SQS le rend à nouveau disponible, ce qui permet éventuellement à un autre consommateur de le récupérer. Pour atténuer ce problème, assurez-vous que les paramètres d'accusé de réception des messages et de délai de visibilité sont appropriés en fonction du temps de traitement.

Évitez les gros arriérés de messages avec le même ID de groupe de messages dans Amazon SQS

Les files d'attente FIFO prennent en charge un maximum de 120 000 messages en cours de vol (messages reçus par un consommateur mais pas encore supprimés). Si cette limite est atteinte, Amazon SQS ne renvoie pas d'erreur, mais le traitement peut en être affecté. Vous pouvez demander une augmentation au-delà de cette limite en contactant le AWS Support.

Les files d'attente FIFO analysent les 120 000 premiers messages pour déterminer les groupes de messages disponibles. Si un important arriéré s'accumule dans un seul groupe de messages, les messages envoyés ultérieurement par d'autres groupes resteront bloqués jusqu'à ce que le backlog soit traité.

> **Note**
>
> Un arriéré de messages peut se produire lorsqu'un client échoue à plusieurs reprises à traiter un message. Cela peut être dû à des problèmes liés au contenu des messages ou à des défaillances du côté du consommateur. Pour éviter les retards dans le traitement des messages, configurez une file d'attente de lettres mortes pour déplacer les messages non traités après plusieurs tentatives infructueuses. Cela garantit que les autres messages du même groupe de messages peuvent être traités, évitant ainsi les engorgements du système.

Évitez de réutiliser le même ID de groupe de messages avec les files d'attente virtuelles dans Amazon SQS

Lorsque vous utilisez des files d'attente virtuelles avec une file d'hôtes partagée, évitez de les réutiliser `MessageGroupId`dans différentes files d'attente virtuelles. Si plusieurs files d'attente virtuelles partagent la même file d'attente d'hôtes et contiennent des messages identiques`MessageGroupId`, ces messages peuvent se bloquer mutuellement, empêchant ainsi un traitement efficace. Pour garantir un traitement fluide des messages, attribuez des `MessageGroupId` valeurs uniques aux messages dans différentes files d'attente virtuelles.

Utilisation de l'ID de tentative de demande de réception Amazon SQS

L'ID de tentative de demande de réception est un jeton unique utilisé pour dédupliquer les `ReceiveMessage`appels dans Amazon SQS. Lors d'une panne réseau ou d'un problème de connectivité entre votre application et Amazon SQS, il est recommandé de :

- Fournissez un identifiant de tentative de demande de réception lorsque vous passez un `ReceiveMessage` appel.
- Réessayez en utilisant le même ID de tentative de demande de réception si l'opération échoue.

Traitement et chronométrage des messages Amazon SQS

Cette rubrique fournit des conseils complets sur l'optimisation de la vitesse et de l'efficacité du traitement des messages dans Amazon SQS, notamment des stratégies pour traiter les messages en temps opportun, sélectionner le meilleur mode d'interrogation et configurer des interrogations longues pour améliorer les performances.

Rubriques

- Traitement des messages en temps opportun dans Amazon SQS
- Configuration d'un long sondage dans Amazon SQS
- Utilisation du mode d'interrogation approprié dans Amazon SQS

Traitement des messages en temps opportun dans Amazon SQS

Le délai de visibilité défini dépend de la durée nécessaire à votre application pour traiter et supprimer un message. Par exemple, si votre application a besoin de 10 secondes pour traiter un message et que vous définissez un délai de visibilité de 15 minutes, vous devez attendre relativement longtemps pour tenter de retraiter le message si la dernière tentative de traitement échoue. En revanche, si votre application a besoin de 10 secondes pour traiter un message, mais que vous définissez un délai de visibilité de seulement 2 secondes, un message dupliqué est reçu par un autre consommateur alors que le consommateur d'origine est toujours en train de travailler sur le message.

Pour vérifier qu'il y a suffisamment de temps pour traiter les messages, utilisez l'une des stratégies suivantes :

- Si vous savez (ou si vous pouvez raisonnablement estimer) combien de temps est nécessaire pour traiter un message, rallongez le délai de visibilité de sorte qu'il corresponde à la durée maximale requise pour traiter et supprimer le message. Pour plus d'informations, consultez Configuration du délai de visibilité.
- Si vous ne savez pas combien de temps il faut pour traiter un message, créez une pulsation pour votre processus de consommateur : spécifiez le délai de visibilité initial (par exemple, 2 minutes)

puis, tant que votre client travaille sur le message, continuez à prolonger le délai de visibilité de 2 minutes, toutes les minutes.

> **Important**
>
> Le délai maximal de visibilité est de 12 heures à compter de l'heure où Amazon SQS reçoit la demande `ReceiveMessage`. L'extension du délai d'attente de visibilité ne réinitialise pas le maximum de 12 heures.
>
> En outre, il se peut que vous ne puissiez pas régler le délai d'expiration d'un message individuel sur 12 heures (par exemple, 43 200 secondes) puisque la demande `ReceiveMessage` déclenche le temporisateur. Par exemple, si vous recevez un message et que vous définissez immédiatement le maximum de 12 heures en envoyant un appel `ChangeMessageVisibility` avec `VisibilityTimeout` défini sur une durée égale à 43 200 secondes, il échouera probablement. En revanche, l'utilisation d'une valeur de 43 195 secondes fonctionnera, à moins qu'il n'y ait un délai important entre la demande du message via `ReceiveMessage` et la mise à jour du délai de visibilité. Si votre client a besoin de plus de 12 heures, envisagez d'utiliser Step Functions.

Configuration d'un long sondage dans Amazon SQS

Lorsque le temps d'attente pour l'action de l'API `ReceiveMessage` est supérieur à 0, une recherche prolongée est activée. Le temps d'attente maximal pour la recherche prolongée est de 20 secondes. La recherche prolongée permet de réduire le coût d'utilisation d'Amazon SQS en éliminant le nombre de réponses vides (lorsqu'il n'y a aucun message disponible pour une demande `ReceiveMessage`) et de fausses réponses vides (lorsque les messages sont disponibles dans la file d'attente, mais ne sont pas inclus dans une réponse). Pour de plus amples informations, veuillez consulter Recherches courtes et longues sur Amazon SQS.

Pour garantir un traitement optimal des messages, utilisez les stratégies suivantes :

- Dans la plupart des cas, vous pouvez définir le délai d'attente `ReceiveMessage` à 20 secondes. Si un délai de 20 secondes est trop long pour votre application, définissez un délai d'attente `ReceiveMessage` plus court (1 seconde au minimum). Si vous n'utilisez pas de AWS SDK pour accéder à Amazon SQS, ou si vous configurez AWS un SDK pour réduire le temps d'attente, il se peut que vous deviez modifier votre client Amazon SQS pour autoriser des demandes plus longues ou utiliser un temps d'attente plus court pour les longs sondages.

- Si vous implémentez l'attente active de longue durée pour plusieurs files d'attente, utilisez un thread pour chacune d'elles plutôt qu'un seul thread pour toutes les files d'attente. L'utilisation d'un seul thread pour chaque file d'attente permet à votre application de traiter les messages de chacune des files d'attente dès qu'ils sont disponibles. En revanche, en utilisant un seul thread pour l'interrogation de plusieurs files d'attente peut mettre votre application dans l'incapacité de traiter les messages disponibles dans d'autres files d'attente lorsque l'application attend (pendant une durée pouvant atteindre 20 secondes) la file d'attente qui ne comporte aucun message.

> **Important**
>
> Pour éviter les erreurs HTTP, assurez-vous que le temps d'attente de réponse HTTP pour les demandes `ReceiveMessage` est plus long que le paramètre `WaitTimeSeconds`. Pour de plus amples informations, veuillez consulter ReceiveMessage.

Utilisation du mode d'interrogation approprié dans Amazon SQS

- La recherche prolongée vous permet de consommer des messages de votre file d'attente Amazon SQS dès qu'ils sont disponibles.
  - Pour réduire le coût d'utilisation d'Amazon SQS et le nombre de réceptions vides dans une file d'attente vide (réponses à l'action `ReceiveMessage` qui ne renvoient aucun message), activez la recherche prolongée. Pour plus d'informations, consultez Recherche prolongée Amazon SQS.
  - Pour accroître l'efficacité lors de l'attente active de plusieurs threads avec plusieurs réceptions, diminuez le nombre de threads.
  - Dans la plupart des cas, l'attente active de longue durée est préférable à celle de courte durée.
- L'attente active de courte durée retourne des réponses immédiatement, même si la file d'attente Amazon SQS interrogée est vide.
  - Pour satisfaire aux exigences d'une application qui attend des réponses immédiates à la demande `ReceiveMessage`, utilisez l'attente active de courte durée.
  - L'attente active de courte durée coûte le même prix que l'attente active de longue durée.

Exemples de SDK Java Amazon SQS

Vous AWS SDK pour Java permet de créer des applications Java qui interagissent avec Amazon SQS et d'autres. Services AWS

- Pour installer et configurer le SDK, reportez-vous à la section Premiers pas du Guide du développeur AWS SDK for Java 2.x .
- Pour les opérations de base relatives aux files d'attente, telles que la création d'une file d'attente ou l'envoi d'un message, consultez la section Travailler avec les files d'attente de messages Amazon SQS dans le manuel du développeur.AWS SDK for Java 2.x
- Ce guide inclut également des exemples de fonctionnalités supplémentaires d'Amazon SQS, telles que :
  - Utilisation du chiffrement côté serveur avec les files d'attente Amazon SQS
  - Configuration des balises pour une file d'attente Amazon SQS
  - Envoi d'attributs de message à une file d'attente Amazon SQS

Utilisation du chiffrement côté serveur avec les files d'attente Amazon SQS

Utilisez le AWS SDK pour Java pour ajouter un chiffrement côté serveur (SSE) à une file d'attente Amazon SQS. Chaque file d'attente utilise une clé KMS AWS Key Management Service (AWS KMS) pour générer les clés de chiffrement des données. Cet exemple utilise la clé KMS gérée par AWS pour Amazon SQS.

Pour en savoir plus sur l'utilisation du SSE et sur le rôle de la clé KMS, consultez Chiffrement au repos dans Amazon SQS.

Ajout du SSE à une file d'attente existante

Pour activer le chiffrement côté serveur pour une file d'attente existante, utilisez la méthode `SetQueueAttributes` afin de définir l'attribut `KmsMasterKeyId`.

L'exemple de code suivant définit la clé KMS AWS KMS key comme clé KMS AWS gérée pour Amazon SQS. L'exemple définit également la période de réutilisation de la AWS KMS key sur 140 secondes.

Avant d'exécuter l'exemple de code, assurez-vous d'avoir défini vos AWS informations d'identification. Pour plus d'informations, consultez la section Configurer les AWS informations d'identification et la région pour le développement dans le guide du AWS SDK for Java 2.x développeur.

```
    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();
        }
    }
```

Désactivation du SSE pour une file d'attente

Pour désactiver le chiffrement côté serveur pour une file d'attente existante, définissez l'attribut `KmsMasterKeyId` sur une chaîne vide à l'aide de la méthode `SetQueueAttributes`.

> **Important**
>
> `null` n'est pas une valeur valide pour `KmsMasterKeyId`.

Création d'une file d'attente avec le SSE

Pour activer le SSE lorsque vous créez la file d'attente, ajoutez l'attribut `KmsMasterKeyId` à la méthode d'API `CreateQueue`.

L'exemple suivant crée une file d'attente avec le SSE activé. La file d'attente utilise la clé KMS gérée par AWS pour Amazon SQS. L'exemple définit également la période de réutilisation de la AWS KMS key sur 160 secondes.

Avant d'exécuter l'exemple de code, assurez-vous d'avoir défini vos AWS informations d'identification. Pour plus d'informations, consultez la section Configurer les AWS informations d'identification et la région pour le développement dans le guide du AWS SDK for Java 2.x développeur.

```
// 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);
```

Récupération des attributs SSE

Pour plus d'informations sur la récupération des attributs de file d'attente, consultez les Exemples de la Référence d'API Amazon Simple Queue Service.

Pour récupérer l'ID de la clé KMS ou la période de réutilisation de la clé de données pour une file d'attente particulière, exécutez la méthode `GetQueueAttributes` et récupérez les valeurs `KmsMasterKeyId` et `KmsDataKeyReusePeriodSeconds`.

Configuration des balises pour une file d'attente Amazon SQS

Utilisez des balises de répartition des coûts afin d'organiser et d'identifier vos files d'attente Amazon SQS. Les exemples suivants montrent comment configurer des balises à l'aide d' AWS SDK pour Java. Pour de plus amples informations, veuillez consulter Balises de répartition des coûts Amazon SQS.

Avant d'exécuter l'exemple de code, assurez-vous d'avoir défini vos AWS informations d'identification. Pour plus d'informations, consultez la section Configurer les AWS informations d'identification et la région pour le développement dans le guide du AWS SDK for Java 2.x développeur.

Établissement d'une liste de balises

Pour répertorier les balises d'une file d'attente, utilisez la méthode `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() ));
```

Ajout ou mise à jour de balises

Pour ajouter ou mettre à jour les valeurs de balise d'une file d'attente, utilisez la méthode `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);
```

Suppression de balises

Pour supprimer une ou plusieurs balises de la file d'attente, utilisez la méthode `UntagQueue`. L'exemple suivant supprime la balise `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);
```

Envoi d'attributs de message à une file d'attente Amazon SQS

Vous pouvez inclure des métadonnées structurées (telles que des horodatages, des données géospatiales, des signatures et des identifiants) dans des messages utilisant des attributs de message. Pour de plus amples informations, veuillez consulter Attributs de message Amazon SQS.

Avant d'exécuter l'exemple de code, assurez-vous d'avoir défini vos AWS informations d'identification. Pour plus d'informations, consultez la section Configurer les AWS informations d'identification et la région pour le développement dans le guide du AWS SDK for Java 2.x développeur.

Définition des attributs

Pour définir un attribut pour un message, ajoutez le code suivant qui utilise le type de données `MessageAttributeValue`. Pour plus d'informations, consultez Composants des attributs de message et Types de données d'attribut de message.

calcule AWS SDK pour Java automatiquement les sommes de contrôle du corps et des attributs du message et les compare aux données renvoyées par Amazon SQS. Pour plus d'informations, consultez le Guide du développeur AWS SDK for Java 2.x et Calcul du résumé du MD5 message pour les attributs du message pour les autres langages de programmation.

String

Cet exemple définit un attribut `String` nommé `Name` avec la valeur `Jane`.

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

Number

Cet exemple définit un attribut `Number` nommé `AccurateWeight` avec la valeur `230.000000000000000001`.

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

Binary

Cet exemple définit un attribut `Binary` nommé `ByteArray` avec la valeur d'une matrice de 10 octets non initialisée.

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

String (custom)

Cet exemple définit l'attribut personnalisé `String.EmployeeId` nommé `EmployeeId` avec la valeur `ABC123456`.

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

Number (custom)

Cet exemple définit l'attribut personnalisé `Number.AccountId` nommé `AccountId` avec la valeur `000123456`.

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

> **Note**
>
> Comme le type de données de base est `Number`, la méthode `ReceiveMessage` renvoie `123456`.

Binary (custom)

Cet exemple définit l'attribut personnalisé `Binary.JPEG` nommé `ApplicationIcon` avec la valeur d'une matrice de 10 octets non initialisée.

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

Envoi d'un message avec des attributs

Cet exemple ajoute les attributs à `SendMessageRequest` avant d'envoyer le message.

```
// 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 vous envoyez un message à une file d'attente First-In-First-Out (FIFO), assurez-vous que la `sendMessage` méthode s'exécute une fois que vous avez fourni l'ID du groupe de messages.
> Si vous utilisez la méthode `SendMessageBatch` au lieu de `SendMessage`, vous devez spécifier les attributs de message pour chaque message du lot.

Utilisation APIs avec Amazon SQS

Cette rubrique fournit des informations sur la création de points de terminaison Amazon SQS, l'envoi de demandes d'API de requête à l'aide des méthodes GET et POST et l'utilisation d'actions d'API par lots. Pour obtenir des informations détaillées sur les actions Amazon SQS, notamment les paramètres, les erreurs, les exemples et les types de données, consultez la Référence d'API Amazon Simple Queue Service.

Pour accéder à Amazon SQS à l'aide de différents langages de programmation, vous pouvez également utiliser AWS SDKsles fonctionnalités automatiques suivantes :

- Signature cryptographique des requêtes de service
- Nouvelles tentatives de requête
- Gestion des réponses d'erreur

Pour de plus amples informations, veuillez consulter the section called “Travailler avec AWS SDKs”.

Pour plus d'informations sur l'outil de ligne de commande, consultez les sections Amazon SQS dans la Référence de commande AWS CLI et la Référence Cmdlet Outils AWS pour PowerShell .

Amazon SQS APIs avec protocole JSON AWS

Amazon SQS utilise le protocole AWS JSON comme mécanisme de transport pour tous les Amazon APIs SQS sur les versions du AWS SDK spécifiées. AWS Le protocole JSON fournit un débit plus élevé, une latence plus faible et une application-to-application communication plus rapide. AWS Le protocole JSON est plus efficace en serialization/deserialization termes de requêtes et de réponses que le protocole de AWS requête. Si vous préférez toujours utiliser le protocole de AWS requête avec SQS APIs, consultez Quels sont les langages pris en charge pour le protocole AWS JSON utilisé dans Amazon SQS ? APIs les versions du AWS SDK qui prennent en charge le protocole de requête Amazon AWS SQS.

Amazon SQS utilise le protocole AWS JSON pour communiquer entre les clients du AWS SDK (par exemple, Java, Python, Golang JavaScript) et le serveur Amazon SQS. Une requête HTTP d'une opération d'API Amazon SQS accepte une entrée au format JSON. L'opération Amazon SQS est exécutée et la réponse d'exécution est renvoyée au client du SDK au format JSON. Comparé à la AWS requête, le AWS JSON est plus simple, plus rapide et plus efficace pour transporter les données entre le client et le serveur.

- AWS Le protocole JSON joue le rôle de médiateur entre le client et le serveur Amazon SQS.
- Le serveur ne comprend pas le langage de programmation dans lequel l'opération Amazon SQS est créée, mais il comprend le protocole AWS JSON.
- Le protocole AWS JSON utilise la sérialisation (conversion de l'objet au format JSON) et la désérialisation (conversion du format JSON en objet) entre le client et le serveur Amazon SQS.

Pour plus d'informations sur le protocole AWS JSON avec Amazon SQS, consultez. Protocole JSON Amazon SQS AWS FAQs

AWS Le protocole JSON est disponible sur la version du AWS SDK spécifiée. Pour consulter la version et les dates de sortie du SDK selon les variantes linguistiques, consultez la matrice de prise en charge des versions AWS SDKs et outils dans le guide de référence AWS SDKs and Tools

Effectuer des demandes d'API de requête à l'aide du protocole AWS JSON dans Amazon SQS

Cette rubrique explique comment créer un point de terminaison Amazon SQS, effectuer des requêtes POST et interpréter les réponses.

> **Note**
>
> AWS Le protocole JSON est pris en charge pour la plupart des variantes linguistiques. Pour accéder à la liste complète des langages pris en charge, consultez Quels sont les langages pris en charge pour le protocole AWS JSON utilisé dans Amazon SQS ? APIs.

Constitution d'un point de terminaison

Afin d'utiliser des files d'attente Amazon SQS, vous devez construire un point de terminaison. Pour plus d'informations sur les points de terminaison Amazon SQS, consultez les pages suivantes dans le Référence générale d'Amazon Web Services :

- Points de terminaison régionaux
- Points de terminaison et quotas Amazon Simple Queue Service

Chaque point de terminaison Amazon SQS est indépendant. Par exemple, si deux files d'attente sont nommées MyQueue et que l'une d'entre elles a le point de terminaison `sqs.us-east-2.amazonaws.com` tandis que l'autre a le point de terminaison `sqs.eu-west-2.amazonaws.com`, les deux files d'attente n'ont aucune donnée en commun.

L'exemple suivant correspond à un point de terminaison lançant une requête pour créer une file d'attente.

```
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**
>
> Les noms et les files d'attente URLs distinguent les majuscules et minuscules.
> La structure de *AUTHPARAMS* dépend de la signature de la demande d'API. Pour plus d'informations, consultez Signing AWS API Requests dans le manuel Amazon Web Services General Reference.

Envoi de requête POST

Une demande POST Amazon SQS envoie des paramètres de requête sous forme de formulaire dans le corps d'une demande HTTP.

Voici un exemple d'en-tête HTTP avec `X-Amz-Target` défini sur `AmazonSQS.<operationName>` et d'en-tête HTTP avec `Content-Type` défini sur `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"
}
```

Cette demande POST HTTP envoie un message à une file d'attente Amazon SQS.

> **Note**
>
> Les deux en-têtes HTTP `X-Amz-Target` et `Content-Type` sont obligatoires.
> Votre client HTTP peut ajouter d'autres éléments à la requête HTTP, en fonction de la version HTTP du client.

Interprétation des réponses de l'API JSON Amazon SQS

Lorsque vous envoyez une demande à Amazon SQS, celui-ci renvoie une réponse JSON avec les résultats. La structure de réponse dépend de l'action d'API que vous avez utilisée.

Pour comprendre le détail de ces réponses, voir :

- L'action d'API spécifique dans les actions d'API du manuel Amazon Simple Queue Service API Reference
- La Protocole JSON Amazon SQS AWS FAQs

Structure d'une réponse JSON positive

Si la demande aboutit, l'élément de réponse principal est `x-amzn-RequestId`, qui contient l'identifiant unique universel (UUID) de la demande, ainsi que d'autres champs de réponse ajoutés. Par exemple, la réponse `CreateQueue` suivante contient le champ `QueueUrl`, qui contient à son tour l'URL de la file d'attente créée.

```
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"
}
```

Structure d'une réponse d'erreur JSON

Si une demande échoue, Amazon SQS renvoie la réponse principale, y compris l'en-tête HTTP et le corps du message.

Dans l'en-tête HTTP, `x-amzn-RequestId` contient l'UUID de la demande. `x-amzn-query-error` contient deux informations : le type d'erreur et s'il s'agit d'une erreur du producteur ou du consommateur.

Dans le corps de la réponse, `"__type"` indique les autres détails de l'erreur et `Message` indique la condition d'erreur dans un format lisible.

Vous trouverez ci-dessous un exemple de réponse d'erreur au format JSON :

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

Protocole JSON Amazon SQS AWS FAQs

Cette rubrique couvre les questions fréquemment posées concernant l'utilisation du protocole AWS JSON avec Amazon SQS.

Qu'est-ce que le protocole AWS JSON et en quoi diffère-t-il des demandes et réponses d'API Amazon SQS existantes ?

JSON est l'une des méthodes de connexion les plus utilisées et acceptées pour la communication entre des systèmes hétérogènes. Amazon SQS utilise le JSON comme moyen de communication entre un client AWS SDK (par exemple, Java, Python, Golang) et le serveur JavaScript Amazon SQS. La requête HTTP d'une opération d'API Amazon SQS accepte une entrée sous forme de JSON. L'opération Amazon SQS est exécutée et la réponse d'exécution est partagée avec le client du SDK sous forme de JSON. Comparé à la requête AWS , JSON est plus efficace pour transporter les données entre le client et le serveur.

- Le protocole Amazon SQS AWS JSON joue le rôle de médiateur entre le client et le serveur Amazon SQS.
- Le serveur ne comprend pas le langage de programmation dans lequel l'opération Amazon SQS est créée, mais il comprend le protocole AWS JSON.
- Le protocole Amazon SQS AWS JSON utilise la sérialisation (conversion de l'objet au format JSON) et la désérialisation (conversion du format JSON en objet) entre le client et le serveur Amazon SQS.

Comment démarrer avec les protocoles AWS JSON pour Amazon SQS ?

Pour commencer à utiliser la dernière version du AWS SDK afin d'accélérer la messagerie pour Amazon SQS, mettez à niveau AWS votre SDK vers la version spécifiée ou vers une version ultérieure. Pour en savoir plus sur les clients SDK, consultez la colonne Guide dans le tableau ci-dessous.

Voici une liste des versions du SDK dans les variantes linguistiques du protocole AWS JSON à utiliser avec Amazon APIs SQS :

Langue	Référentiel client SDK	Version du client SDK requise	Guide
C++	frais/ aws-sdk-cpp	1,11,98	AWS SDK pour C++
Golang 1.x	frais/ aws-sdk-go	v1.47.7	AWS SDK pour Go

Langue	Référentiel client SDK	Version du client SDK requise	Guide
Golang 2.x	mousquettes/ 2 aws-sdk-go-v	v1.28.0	AWS SDK pour Go V2
Java 1.x	frais/ aws-sdk-java	1,12,585	AWS SDK pour Java
Java 2.x	mousquettes/ 2 aws-sdk-java-v	2,21,19	AWS SDK pour Java
JavaScript v2.x	frais/ aws-sdk-js	JavaScript sur AWS	
JavaScript v3.x	frais/ 3 aws-sdk-js-v	v3.447.0	JavaScript sur AWS
.NET	frais/ aws-sdk-net	3,7,681,0	AWS SDK pour .NET
PHP	frais/ aws-sdk-php	3,285,2	AWS SDK pour PHP
Python-boto3	boto/boto3	1,28,82	AWS SDK pour Python (Boto3)
Python-botocore	boto/botocore	1,31,82	AWS SDK pour Python (Boto3)
awscli	AWS CLI	1,29,82	Interface de ligne de commande AWS
Ruby	frais/ aws-sdk-ruby	1,67,0	AWS SDK pour Ruby

Quels sont les risques liés à l'activation du protocole JSON pour mes charges de travail Amazon SQS ?

Si vous utilisez une implémentation personnalisée du AWS SDK ou une combinaison de clients personnalisés et d'un AWS SDK pour interagir avec Amazon SQS qui AWS génère des réponses basées sur des requêtes (également appelées XML), cela peut être incompatible avec le protocole JSON. AWS Si vous rencontrez des problèmes, contactez le AWS Support.

Et si j'utilise déjà la dernière version du AWS SDK, mais que ma solution open source ne prend pas en charge le format JSON ?

Vous devez remplacer la version de votre kit SDK par la version antérieure à celle que vous utilisez. Voir Comment démarrer avec les protocoles AWS JSON pour Amazon SQS ? pour plus d'informations. AWS Les versions du SDK répertoriées dans Comment démarrer avec les protocoles AWS JSON pour Amazon SQS ? utilisent le protocole filaire JSON pour Amazon APIs SQS. Si vous remplacez votre AWS SDK par la version précédente, votre Amazon APIs SQS utilisera AWS la requête.

Quels sont les langages pris en charge pour le protocole AWS JSON utilisé dans Amazon SQS ? APIs

Amazon SQS prend en charge toutes les variantes linguistiques généralement AWS SDKs disponibles (GA). Actuellement, nous ne prenons pas en charge Kotlin, Rust ou Swift. Pour en savoir plus sur les autres variantes de langage, consultez la section Outils pour créer sur AWS.

Quelles sont les régions prises en charge pour le protocole AWS JSON utilisé dans Amazon SQS APIs

Amazon SQS prend en charge le protocole AWS JSON dans toutes les AWS régions où Amazon SQS est disponible.

À quelles améliorations de latence puis-je m'attendre lors de la mise à niveau vers les versions du AWS SDK spécifiées pour Amazon SQS à l'aide AWS du protocole JSON ?

AWS Le protocole JSON est plus efficace pour la sérialisation et la désérialisation des demandes et des réponses que le protocole de AWS requête. Basé sur des tests de AWS performance pour une

charge utile de messages de 5 Ko, le protocole JSON pour Amazon SQS end-to-end réduit la latence de traitement des messages jusqu'à 23 % et réduit l'utilisation du processeur et de la mémoire côté client de l'application.

Le protocole de AWS requête sera-t-il obsolète ?

AWS le protocole de requête continuera d'être pris en charge. Vous pouvez continuer à utiliser le protocole de AWS requête tant que la version de votre AWS SDK est définie sur une version précédente autre que celle répertoriée dans Comment démarrer avec les protocoles AWS JSON pour Amazon SQS.

Où puis-je trouver plus d'informations sur le protocole AWS JSON ?

Vous trouverez plus d'informations sur le protocole JSON dans Protocole AWS JSON 1.0 dans la documentation de Smithy. Pour en savoir plus sur les requêtes d'API Amazon SQS avec le protocole AWS JSON, consultez Effectuer des demandes d'API de requête à l'aide du protocole AWS JSON dans Amazon SQS.

Effectuer des demandes d'API de requête à l'aide du protocole de AWS requête dans Amazon SQS

Cette rubrique explique comment créer un point de terminaison Amazon SQS, effectuer des requêtes GET et POST et interpréter les réponses.

Constitution d'un point de terminaison

Afin d'utiliser des files d'attente Amazon SQS, vous devez construire un point de terminaison. Pour plus d'informations sur les points de terminaison Amazon SQS, consultez les pages suivantes dans la Référence générale d'Amazon Web Services :

- Points de terminaison régionaux
- Points de terminaison et quotas Amazon Simple Queue Service

Chaque point de terminaison Amazon SQS est indépendant. Par exemple, si deux files d'attente sont nommées MyQueue et que l'une d'entre elles a le point de terminaison `sqs.us-east-2.amazonaws.com` tandis que l'autre a le point de terminaison `sqs.eu-west-2.amazonaws.com`, les deux files d'attente n'ont aucune donnée en commun.

L'exemple suivant correspond à un point de terminaison lançant une requête pour créer une file d'attente.

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

> **Note**
>
> Les noms et les files d'attente URLs distinguent les majuscules et minuscules.
> La structure de *AUTHPARAMS* dépend de la signature de la demande d'API. Pour plus d'informations, consultez Signing AWS API Requests dans le manuel Amazon Web Services General Reference.

Envoi de requête GET

Une requête GET Amazon SQS est structurée comme une URL se composant des éléments suivants :

- Point de terminaison : ressource sur laquelle la requête agit (nom de la file d'attente et URL), par exemple : `https://sqs.us-east-2.amazonaws.com/`*`123456789012`*`/MyQueue`
- Action : action que vous souhaitez effectuer sur le point de terminaison. Un point d'interrogation (`?`) sépare le point de terminaison de l'action, par exemple : `?Action=SendMessage&MessageBody=Your%20Message%20Text`.
- Paramètres : tout paramètre de demande. Chaque paramètre est séparé par une esperluette (`&`), par exemple : `&Version=2012-11-05&`*`AUTHPARAMS`*.

Voici un exemple de requête GET qui envoie un message à une file d'attente Amazon SQS.

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

> **Note**
>
> Les noms et les files d'attente URLs distinguent les majuscules et minuscules.
> Dans la mesure où les requêtes GET URLs le sont, vous devez encoder toutes les valeurs de paramètres en URL. Comme les espaces ne sont pas autorisés URLs, chaque espace est codé en URL sous la forme. `%20` Pour faciliter la lecture, le reste de l'exemple n'a pas été encodé selon le format URL.

Envoi de requête POST

Une demande POST Amazon SQS envoie des paramètres de requête sous forme de formulaire dans le corps d'une demande HTTP.

L'exemple suivant illustre un en-tête HTTP avec `Content-Type` défini sur `application/x-www-form-urlencoded`.

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

L'en-tête est suivi d'une requête GET `form-urlencoded` qui envoie un message à une file d'attente Amazon SQS. Chaque paramètre est séparé par une esperluette (&).

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

> **Note**
>
> Seul l'en-tête HTTP `Content-Type` est obligatoire. L'élément *`AUTHPARAMS`* est le même que pour la requête GET.
> Votre client HTTP peut ajouter d'autres éléments à la requête HTTP, en fonction de la version HTTP du client.

Interprétation des réponses de l'API XML Amazon SQS

Lorsque vous envoyez une demande à Amazon SQS, celui-ci renvoie une réponse XML contenant les résultats de la demande. Pour comprendre la structure et les détails de ces réponses, reportez-vous aux actions d'API spécifiques dans le manuel Amazon Simple Queue Service API Reference.

Structure d'une réponse XML positive

Si la requête a abouti, l'élément de réponse principal porte le nom de l'action, mais avec `Response` ajouté (par exemple, *`ActionName`*`Response`).

Il contient les éléments enfants suivants :

- ***`ActionName`***`Result` : contient un élément spécifique à l'action. Par exemple, l'élément `CreateQueueResult` contient l'élément `QueueUrl`, qui contient à son tour l'URL de la file d'attente créée.
- **`ResponseMetadata`** : contient le `RequestId` qui contient à son tour l'UUID (Universal Unique Identifier) de la requête.

Voici un exemple de réponse ayant abouti au format XML :

```
<CreateQueueResponse
   xmlns=https://sqs.us-east-2.amazonaws.com/doc/2012-11-05/
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:type=CreateQueueResponse>
   <CreateQueueResult>
      <QueueUrl>https://sqs.us-east-2.amazonaws.com/770098461991/queue2</QueueUrl>
   </CreateQueueResult>
   <ResponseMetadata>
      <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
   </ResponseMetadata>
</CreateQueueResponse>
```

Structure d'une réponse d'erreur XML

Si une requête échoue, Amazon SQS renvoie toujours l'élément de réponse principal `ErrorResponse`. Il contient un élément `Error` et un élément `RequestId`.

L'élément `Error` contient les éléments enfants suivants :

- **`Type`** : indique si l'erreur est survenue au niveau du producteur ou du consommateur.
- **`Code`** : spécifie le type d'erreur.
- **`Message`** : spécifie la condition d'erreur dans un format lisible.
- **`Detail`** : (facultatif) spécifie des détails supplémentaires sur l'erreur.

L'élément `RequestId` contient l'UUID de la requête.

Voici un exemple de réponse d'erreur au format XML :

```
<ErrorResponse>
   <Error>
      <Type>Sender</Type>
      <Code>InvalidParameterValue</Code>
      <Message>
         Value (quename_nonalpha) for parameter QueueName is invalid.
         Must be an alphanumeric String of 1 to 80 in length.
      </Message>
   </Error>
   <RequestId>42d59b56-7407-4c4a-be0f-4c88daeea257</RequestId>
</ErrorResponse>
```

Authentification des demandes pour Amazon SQS

L'authentification est le processus d'identification et de vérification de la partie qui envoie une requête. Au cours de la première étape de l'authentification, AWS vérifie l'identité du créateur et si celui-ci est enregistré pour utiliser AWS (pour plus d'informations, consultez Étape 1 : créer un utilisateur Compte AWS et IAM). Ensuite, AWS respecte la procédure suivante :

1. Le producteur (expéditeur) obtient les autorisations requises.
2. Le créateur envoie une requête et l'autorisation au consommateur (destinataire).
3. Le consommateur utilise l'autorisation pour vérifier si le producteur a envoyé la requête.
4. L'une des situations suivantes se produit :
   - Si l'authentification réussit, le consommateur procède au traitement de la requête.
   - Si l'authentification échoue, le consommateur rejette la requête et renvoie une erreur.

Processus d'authentification de base avec HMAC-SHA

Lorsque vous accédez à Amazon SQS grâce à l'API Query, vous devez fournir les éléments suivants pour authentifier votre demande :

- L'identifiant de clé d'AWS accès qui vous identifie Compte AWS et qui est AWS utilisé pour rechercher votre clé d'accès secrète.
- La signature de demande HMAC-SHA, calculée à l'aide de votre clé d'accès secrète (un secret partagé connu de vous seul AWS ; pour plus d'informations, voir 04). RFC21 Le kit SDK AWS gère le processus de signature. Toutefois, si vous soumettez une requête de requête via HTTP ou HTTPS, vous devez inclure une signature dans chacune de ces requêtes.

  1. Dérivez une clé de signature de Signature Version 4. Pour plus d'informations, consultez Dérivation d'une clé de signature avec Java.

     > **Note**
     >
     > Amazon SQS prend en charge la version 4 de Signature, qui fournit une sécurité et des SHA256 performances améliorées par rapport aux versions précédentes. Lorsque vous créez des applications qui utilisent Amazon SQS, optez pour Signature version 4.

  2. Codez en Base64 la signature de la requête. C'est ce que fait l'exemple de code Java suivant :

```
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);
    }
}
```

- Horodatage (ou expiration) de la requête. L'horodatage que vous utilisez dans la requête doit être un objet `dateTime`, avec la date complète suivie des heures, des minutes et des secondes.

Par exemple : `2007-01-31T23:59:59Z` Bien que ce ne soit pas obligatoire, nous vous recommandons de fournir l'heure système affichée dans la zone d'heure de l'heure universelle coordonnée (Heure de Greenwich).

> **Note**
>
> Assurez-vous que votre serveur affiche l'heure correcte. Si vous spécifiez un horodatage (plutôt qu'une expiration), la demande expire automatiquement 15 minutes après l'heure spécifiée (AWS ne traite pas les demandes dont l'horodatage est daté de plus de 15 minutes avant l'heure actuelle sur les serveurs). AWS
> Si vous utilisez .NET, vous ne devez pas envoyer des heures systèmes excessivement spécifiques, en raison des diverses propositions existant sur la façon de renoncer à une précision horaire absolue. Dans ce cas, vous devez créer manuellement les objets `dateTime` avec une précision ne dépassant pas une milliseconde.

Partie 1 : Demande de l'utilisateur

Voici le processus que vous devez suivre pour authentifier les AWS demandes à l'aide d'une signature de demande HMAC-SHA.

1. Créez une demande pour AWS.
2. Calculez une signature de code d'authentification du message avec hachage à clés (HMAC-SHA) à l'aide de votre clé d'accès secrète.
3. Incluez la signature et l'identifiant de votre clé d'accès dans la demande, puis envoyez la demande à AWS.

Partie 2 : La réponse de AWS

AWS lance le processus suivant en réponse.

1. AWS utilise l'ID de clé d'accès pour rechercher votre clé d'accès secrète.
2. AWS génère une signature à partir des données de la demande et de la clé d'accès secrète, en utilisant le même algorithme que celui que vous avez utilisé pour calculer la signature que vous avez envoyée dans la demande.
3. L'une des situations suivantes se produit :
   - Si la signature AWS générée correspond à celle que vous avez envoyée dans la demande, AWS considère que la demande est authentique.
   - Si la comparaison échoue, la demande est rejetée et AWS renvoie une erreur.

Actions groupées Amazon SQS

Amazon SQS propose des actions par lots pour vous aider à réduire les coûts et à manipuler jusqu'à 10 messages en une seule action. Ces actions par lots incluent :

- `SendMessageBatch`

- `DeleteMessageBatch`
- `ChangeMessageVisibilityBatch`

Les actions par lots vous permettent d'effectuer plusieurs opérations en un seul appel d'API, ce qui permet d'optimiser les performances et de réduire les coûts. Vous pouvez tirer parti de la fonctionnalité de traitement par lots à l'aide de l'API de requête ou de n'importe quel AWS SDK prenant en charge les actions par lots Amazon SQS.

Détails importants

- Limite de taille des messages : la taille totale de tous les messages envoyés lors d'un seul `SendMessageBatch` appel ne peut pas dépasser 1 048 576 octets (1 MiB)
- Autorisations : vous ne pouvez pas définir d'autorisations explicitement pour `SendMessageBatchDeleteMessageBatch`, ou`ChangeMessageVisibilityBatch`. Définissez plutôt des autorisations pour `SendMessageDeleteMessage`, ou `ChangeMessageVisibility` définissez des autorisations pour les versions par lots correspondantes des actions.
- Support de console : la console Amazon SQS ne prend pas en charge les actions par lots. Vous devez utiliser l'API de requête ou un AWS SDK pour effectuer des opérations par lots.

Actions de message par lots

Pour optimiser davantage les coûts et l'efficacité, prenez en compte les meilleures pratiques suivantes pour les actions liées aux messages par lots :

- Actions d'API par lots : utilisez les actions d'API par lots Amazon SQS pour envoyer, recevoir et supprimer des messages, et pour modifier le délai de visibilité des messages pour plusieurs messages en une seule action. Cela réduit le nombre d'appels d'API et les coûts associés.
- Mise en mémoire tampon côté client et interrogation longue : combinez la mise en mémoire tampon côté client avec le traitement par lots de demandes en utilisant une interrogation longue avec le client asynchrone mis en mémoire tampon inclus dans le. AWS SDK pour Java Cette approche permet de minimiser le nombre de demandes et d'optimiser le traitement de gros volumes de messages.

> **Note**
>
> Le client asynchrone en mémoire tampon Amazon SQS ne prend actuellement pas en charge les files d'attente FIFO.

Activation de la mise en mémoire tampon côté client et du traitement par lots des demandes avec Amazon SQS

Le kit AWS SDK pour Java inclut `AmazonSQSBufferedAsyncClient` qui accède à Amazon SQS. Ce client permet un traitement simple par lots de demandes à l'aide de la mise en mémoire tampon côté client. Les appels passés depuis le client sont d'abord mis en mémoire tampon, puis envoyés sous forme de demande groupée à Amazon SQS.

La mise en tampon côté client autorise la mise en tampon et l'envoi en tant que demande par lots de 10 requêtes au maximum, ce qui réduit les coûts d'utilisation d'Amazon SQS et le nombre de requêtes envoyées. `AmazonSQSBufferedAsyncClient` met en tampon les appels synchrones et asynchrones. Les demandes par lots et la prise en charge de l'attente active de longue durée peuvent également permettre d'augmenter le débit. Pour de plus amples informations, veuillez consulter Augmenter le débit grâce à la mise à l'échelle horizontale et au traitement par lots d'actions avec Amazon SQS.

Dans la mesure où `AmazonSQSBufferedAsyncClient` implémente la même interface qu'`AmazonSQSAsyncClient`, la migration d'`AmazonSQSAsyncClient` vers `AmazonSQSBufferedAsyncClient` requiert généralement très peu de changements du code existant.

> **Note**
>
> Le client asynchrone en mémoire tampon Amazon SQS ne prend actuellement pas en charge les files d'attente FIFO.

Utilisation d'Amazon SQSBuffered AsyncClient

Avant de commencer, complétez les étapes détaillées dans Configuration d'Amazon SQS.

AWS SDK pour Java 1.x

Pour le AWS SDK for Java 1.x, vous pouvez en créer un `AmazonSQSBufferedAsyncClient` nouveau en vous basant sur l'exemple suivant :

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

Après avoir créé le nouveau `AmazonSQSBufferedAsyncClient`, utilisez-le pour envoyer plusieurs demandes à Amazon SQS (comme avec `AmazonSQSAsyncClient`), par exemple :

```
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);
```

Configuration d'Amazon SQSBuffered AsyncClient

`AmazonSQSBufferedAsyncClient` est préconfiguré avec des paramètres qui fonctionnent dans la plupart des cas d'utilisation. Vous pouvez configurer davantage `AmazonSQSBufferedAsyncClient`, par exemple :

1. Créez une instance de la classe `QueueBufferConfig` avec les paramètres de configuration requis.
2. Fournissez l'instance au constructeur `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 paramètres de configuration

Paramètre	Valeur par défaut	Description
`longPoll`	`true`	Lorsque `longPoll` est défini sur `true`, `AmazonSQSBufferedAsyncClient` tente d'utiliser l'attente active de longue durée lors de la consommation des messages.
`longPollWaitTimeoutSeconds`	20 s	Durée maximale, en secondes, pendant laquelle un `ReceiveMessage` reste sur le serveur en attendant l'apparition de messages dans la file d'attente avant de renvoyer un résultat de réception vide. **Note** Ce paramètre n'a pas d'impact lorsque l'attente active de longue durée est désactivée.

Paramètre	Valeur par défaut	Description
`maxBatchOpenMs`	200 ms	Durée maximale, en milliseco ndes, pendant laquelle un appel sortant attend d'autres appels du même type pour le traitement par lots. Plus le paramètre est élevé, moins il faut de lots pour effectuer la même quantité de travail (toutefois, le premier appel d'un lot doit passer plus de temps à attendre). Lorsque ce paramètre est défini sur `0`, les requêtes envoyées n'attendent pas d'autres requêtes, ce qui a pour effet de désactiver le traitement par lots.

Paramètre	Valeur par défaut	Description
`maxBatchSize`	10 requêtes par lot	Nombre maximal de messages traités dans un même lot dans le cadre d'une seule requête. Plus ce paramètre est élevé, moins le nombre de lots requis pour effectuer le même nombre de requêtes est élevé. **Note** 10 demandes par lots est la valeur maximale autorisée pour Amazon SQS.
`maxBatchSizeBytes`	1 MiB	Taille maximale d'un lot de messages, en octets, que le client essaie d'envoyer à Amazon SQS. **Note** 1 MiB est la valeur maximale autorisée pour Amazon SQS.

Paramètre	Valeur par défaut	Description
`maxDoneReceiveBatches`	10 lots	Nombre maximal de lots de réception récupérés au préalable par `AmazonSQSBufferedAsyncClient` et stockés côté client. Plus le paramètre est élevé, plus il est possible de satisfaire un grand nombre de demandes sans devoir appeler Amazon SQS (toutefois, plus le nombre de messages récupérés au préalable est important, plus ils restent longtemps dans la mémoire tampon, ce qui entraîne l'expiration de leur délai de visibilité). **Note** `0`indique que la prélecture des messages est désactivée et que les messages sont consommés uniquement à la demande.

Paramètre	Valeur par défaut	Description
`maxInflightOutboundBatches`	5 lots	Nombre maximal de lots sortants actifs pouvant être traités en même temps. Plus ce paramètre est élevé, plus les lots sortants peuvent être envoyés rapidement (sous réserve d'autres quotas, tels que l'UC ou la bande passante) et plus le nombre de threads consommés par `AmazonSQSBufferedAsyncClient` est important.

Paramètre	Valeur par défaut	Description
`maxInflightReceiveBatches`	10 lots	Nombre maximum de lots de réception actifs pouvant être traités en même temps. Plus ce paramètre est élevé, plus le nombre de messages susceptibles d'être reçus est important (sous réserve d'autres quotas, tels que l'UC ou la bande passante) et plus le nombre de threads consommés par `AmazonSQSBufferedAsyncClient` est important. **Note** `0`indique que la prélecture des messages est désactivée et que les messages sont consommés uniquement à la demande.

Paramètre	Valeur par défaut	Description
`visibilityTimeoutSeconds`	-1	Lorsqu'une valeur positive autre que zéro a été définie pour ce paramètre, le délai de visibilité définit ici prévaut sur celui de la file d'attente à partir de laquelle les messages sont consommés. **Note** `-1` indique que le paramètre par défaut est sélectionné pour la file d'attente. Vous ne pouvez pas définir de délai de visibilité sur `0`.

AWS SDK pour Java 2.x

Pour le AWS SDK for Java 2.x, vous pouvez en créer un `SqsAsyncBatchManager` nouveau en vous basant sur l'exemple suivant :

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

Après avoir créé le nouveau `SqsAsyncBatchManager`, utilisez-le pour envoyer plusieurs demandes à Amazon SQS (comme avec `SqsAsyncClient`), par exemple :

```
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);
```

Configurer SqsAsyncBatchManager

`SqsAsyncBatchManager` est préconfiguré avec des paramètres qui fonctionnent dans la plupart des cas d'utilisation. Vous pouvez configurer davantage `SqsAsyncBatchManager`, par exemple :

Création d'une configuration personnalisée via `SqsAsyncBatchManager.Builder` :

```
SqsAsyncBatchManager customizedBatchManager = SqsAsyncBatchManager.builder()
    .client(sqs)
```

```
    .scheduledExecutor(Executors.newScheduledThreadPool(5))
    .overrideConfiguration(b -> b
        .maxBatchSize(10)
        .sendRequestFrequency(Duration.ofMillis(200))
        .receiveMessageMinWaitDuration(Duration.ofSeconds(10))
        .receiveMessageVisibilityTimeout(Duration.ofSeconds(20))
        .receiveMessageAttributeNames(Collections.singletonList("*"))

 .receiveMessageSystemAttributeNames(Collections.singletonList(MessageSystemAttributeName.ALL))
    .build();
```

`BatchOverrideConfiguration` paramètres

Paramètre	Valeur par défaut	Description
`maxBatchSize`	10 requêtes par lot	Nombre maximal de messages traités dans un même lot dans le cadre d'une seule requête. Plus ce paramètre est élevé, moins le nombre de lots requis pour effectuer le même nombre de requêtes est élevé. **Note** La valeur maximale autorisée pour Amazon SQS est de 10 demandes par lot.
`sendRequestFrequency`	200 ms	Durée maximale, en milliseco ndes, pendant laquelle un appel sortant attend d'autres appels du même type pour le traitement par lots. Plus le paramètre est élevé, moins il faut de lots pour

Paramètre	Valeur par défaut	Description
		effectuer la même quantité de travail (toutefois, le premier appel d'un lot doit passer plus de temps à attendre). Lorsque ce paramètre est défini sur `0`, les requêtes envoyées n'attendent pas d'autres requêtes, ce qui a pour effet de désactiver le traitement par lots.
`receiveMessageVisibilityTimeout`	-1	Lorsqu'une valeur positive autre que zéro a été définie pour ce paramètre, le délai de visibilité définit ici prévaut sur celui de la file d'attente à partir de laquelle les messages sont consommés. **Note** 1 indique que le paramètre par défaut est sélectionné pour la file d'attente. Vous ne pouvez pas définir de délai de visibilité sur `0`.

Paramètre	Valeur par défaut	Description
`receiveMessageMinWaitDuration`	50 millisecondes	Durée minimale (en millisecondes) pendant laquelle un `receiveMessage` appel attend que les messages disponibles soient récupérés. Plus le paramètre est élevé, moins il faut de lots pour exécuter le même nombre de demandes.

Augmenter le débit grâce à la mise à l'échelle horizontale et au traitement par lots d'actions avec Amazon SQS

Amazon SQS prend en charge la messagerie haut débit. Pour plus de détails sur les limites de débit, reportez-vous àQuotas de messages Amazon SQS.

Pour optimiser le débit :

- Faites évoluer les producteurs et les consommateurs horizontalement en ajoutant plus d'instances de chacun d'eux.
- Utilisez le traitement par lots d'actions pour envoyer ou recevoir plusieurs messages en une seule demande, réduisant ainsi la charge des appels d'API.

Mise à l'échelle horizontale

Vous accédez à Amazon SQS via un protocole de demande-réponse HTTP. Par conséquent, la latence de la demande (l'intervalle de temps entre la création d'une requête et la réception de la réponse) limite le débit que vous pouvez obtenir d'un seul thread avec une connexion unique. Par exemple, si la latence entre un client EC2 basé sur Amazon et Amazon SQS dans la même région est en moyenne de 20 ms, le débit maximal d'un thread sur une seule connexion est en moyenne de 50 TPS.

La mise à l'échelle horizontale consiste à augmenter le nombre de producteurs de messages (émettant des requêtes `SendMessage`) et de consommateurs de messages (émettant des requêtes

`ReceiveMessage` et `DeleteMessage`) afin d'augmenter le débit global de votre file d'attente. Vous pouvez mettre à l'échelle horizontalement de trois manières :

- Augmenter le nombre de threads par client
- Ajouter des clients
- Augmentez le nombre de threads par client et ajouter d'autres clients

Lorsque vous ajoutez des clients, vous créez essentiellement un gain de débit linéaire pour votre file d'attente. Ainsi, si vous doublez le nombre de clients, vous obtiendrez deux fois plus de débit.

Traitement par lots des actions

Le traitement par lots effectue davantage de travail au cours de chaque aller-retour vers le service (par exemple, lorsque vous envoyez plusieurs messages avec une seule requête `SendMessageBatch`). Les actions par lots Amazon SQS sont `SendMessageBatch`, `DeleteMessageBatch` et `ChangeMessageVisibilityBatch`. Pour profiter du traitement par lots sans modifier vos producteurs ou consommateurs, vous pouvez utiliser le Client asynchrone en mémoire tampon Amazon SQS.

> **Note**
>
> Dans la mesure où `ReceiveMessage` peut traiter 10 messages simultanément, il n'y a pas d'action `ReceiveMessageBatch`.

Le traitement par lots répartit la latence de l'action sur plusieurs messages dans une demande par lots, au lieu d'accepter la totalité de la latence pour un seul message (par exemple, une demande `SendMessage`). Dans la mesure où chaque échange avec le service inclut davantage de tâches à traiter, les requêtes par lots assurent une utilisation plus efficace des threads et connexions, ce qui améliore le débit.

Vous pouvez combiner la mise à l'échelle horizontale et le traitement par lots afin de proposer un débit avec moins de threads, de connexions et de requêtes que pour les requêtes de message individuelles. Vous pouvez utiliser des actions Amazon SQS par lots pour envoyer, recevoir ou supprimer jusqu'à 10 messages à la fois. Dans la mesure où Amazon SQS facture par demande, le traitement par lots peut vous aider à réduire considérablement vos coûts.

Le traitement par lots peut ajouter un peu de complexité pour votre application (par exemple, votre application doit accumuler les messages avant de les envoyer, ou doit parfois attendre plus longtemps pour une réponse). Il reste toutefois efficace dans les cas suivants :

- Votre application génère de nombreux messages en peu de temps, si bien que le délai d'attente n'est jamais très long.
- Un consommateur de messages récupère les messages dans une file d'attente à sa discrétion, contrairement à un système où des producteurs de messages classiques ont besoin d'envoyer des messages en réponse à des événements qu'ils ne contrôlent pas.

> **Important**
>
> Une requête de traitement par lots peut aboutir même si des messages individuels inclus dans cette requête n'ont pas pu être traités. Après une requête de traitement par lot, recherchez toujours d'éventuelles erreurs concernant des messages individuels et, le cas échéant, relancez l'action.

Exemple d'utilisation de Java pour les requêtes en une seule opération et par lots

Prérequis

Ajoutez les packages `aws-java-sdk-sqs.jar`, `aws-java-sdk-ec2.jar` et `commons-logging.jar` au chemin de classe de votre version Java. L'exemple suivant illustre ces dépendances dans un fichier `pom.xml` de projet 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

L'exemple de code Java suivant implémente un schéma simple consommateur-producteur. Le thread principal génère un certain nombre de threads producteurs et consommateurs qui traitent des messages de 1 Ko pendant une durée spécifiée. Cet exemple inclut des producteurs et des consommateurs qui émettent des requêtes d'opérations simples, et d'autres qui créent des requêtes de traitement par lot.

```
/*
 * 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.
            }
        }
    }
}
```

Surveillance des métriques de volume pour l'exemple exécuté

Amazon SQS génère automatiquement des métriques de volume pour les messages envoyés, reçus et supprimés. Vous pouvez accéder à ces métriques et à d'autres via l'onglet Surveillance de votre file d'attente ou dans la console CloudWatch .

> **Note**
>
> Après le démarrage de la file d'attente, il faut patienter jusqu'à 15 minutes pour que ces métriques soient disponibles.

Utilisation d'Amazon SQS avec un SDK AWS

AWS des kits de développement logiciel (SDKs) sont disponibles pour de nombreux langages de programmation courants. Chaque SDK fournit une API, des exemples de code et de la documentation qui facilitent la création d'applications par les développeurs dans leur langage préféré.

Documentation SDK	Exemples de code
AWS SDK pour C++	AWS SDK pour C++ exemples de code
AWS CLI	AWS CLI exemples de code
AWS SDK pour Go	AWS SDK pour Go exemples de code
AWS SDK pour Java	AWS SDK pour Java exemples de code
AWS SDK pour JavaScript	AWS SDK pour JavaScript exemples de code
AWS SDK pour Kotlin	AWS SDK pour Kotlin exemples de code
AWS SDK pour .NET	AWS SDK pour .NET exemples de code
AWS SDK pour PHP	AWS SDK pour PHP exemples de code
Outils AWS pour PowerShell	Outils AWS pour PowerShell exemples de code
AWS SDK pour Python (Boto3)	AWS SDK pour Python (Boto3) exemples de code
AWS SDK pour Ruby	AWS SDK pour Ruby exemples de code
Kit AWS SDK pour Rust	Kit AWS SDK pour Rust exemples de code
AWS SDK pour SAP ABAP	AWS SDK pour SAP ABAP exemples de code

Documentation SDK	Exemples de code
Kit AWS SDK pour Swift	Kit AWS SDK pour Swift exemples de code

Exemple de disponibilité

Vous n’avez pas trouvé ce dont vous avez besoin ? Demandez un exemple de code en utilisant le lien Provide feedback (Fournir un commentaire) en bas de cette page.

Utilisation de JMS avec Amazon SQS

La bibliothèque de messagerie Java Amazon SQS est une interface de service de message Java (JMS) pour Amazon SQS qui vous permet de tirer parti d'Amazon SQS dans les applications qui utilisent déjà JMS. L'interface vous permet d'utiliser Amazon SQS en tant que fournisseur JMS avec peu de changements de code. Avec le kit AWS SDK pour Java, la bibliothèque de messagerie Java Amazon SQS vous permet de créer des connexions et des sessions JMS, ainsi que des producteurs et des consommateurs qui envoient et reçoivent des messages de files d'attente Amazon SQS.

La bibliothèque prend en charge l'envoi et la réception de messages vers une file d'attente (le point-to-point modèle JMS) conformément à la spécification JMS 1.1. Elle prend également en charge l'envoi de messages de texte, d'octets ou d'objets de façon synchrone aux files d'attente Amazon SQS. ainsi que la réception d'objets de façon synchrone ou asynchrone.

Pour plus d'informations sur les fonctionnalités de la bibliothèque de messagerie Java Amazon SQS qui prennent en charge la spécification JMS 1.1, consultez Implémentations de JMS 1.1 prises en charge par Amazon SQS et Amazon SQS. FAQs

Conditions requises pour travailler avec JMS et Amazon SQS

Avant de commencer, les prérequis suivants doivent être remplis :

- SDK pour Java

  Il existe deux façons d'inclure le kit SDK pour Java dans votre projet :

  - Téléchargez et installez le kit SDK pour Java.
  - Utilisez Maven pour obtenir la bibliothèque de messagerie Java Amazon SQS.

  > **Note**
  >
  > Le kit SDK pour Java est fourni en tant que dépendance.
  > Le kit SDK pour Java et la bibliothèque client étendue Amazon SQS pour Java nécessitent le kit de développement J2SE 8.0 ou version ultérieure.

  Pour plus d'informations sur le téléchargement du kit SDK pour Java, consultez SDK pour Java.

- Bibliothèque de messagerie Java Amazon SQS

Si vous n'utilisez pas Maven, vous devez ajouter le package `amazon-sqs-java-messaging-lib.jar` au chemin de classe Java. Pour plus d'informations sur le téléchargement de la bibliothèque, consultez Bibliothèque de messagerie Java Amazon SQS.

> **Note**
>
> La bibliothèque de messagerie Java Amazon SQS inclut la prise en charge de Maven et du framework Spring.
> Pour obtenir des exemples de code qui utilisent Maven, le framework Spring et la bibliothèque de messagerie Java Amazon SQS, consultez la section Exemples Java fonctionnels pour l'utilisation de JMS avec les files d'attente standard Amazon SQS.
>
> ```
> <dependency>
>  <groupId>com.amazonaws</groupId>
>  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
>  <version>1.0.4</version>
>  <type>jar</type>
> </dependency>
> ```

- File d'attente Amazon SQS

  Créez une file d'attente à l'aide de l'`CreateQueue`API AWS Management Console pour Amazon SQS, ou du client Amazon SQS encapsulé inclus dans la bibliothèque de messagerie Java Amazon SQS.

  - Pour plus d'informations sur la création d'une file d'attente avec Amazon SQS à l'aide de la AWS Management Console ou de l'API `CreateQueue`, consultez Création d'une file d'attente.
  - Pour plus d'informations sur l'utilisation de la bibliothèque de messagerie Java Amazon SQS, consultez Utilisation de la bibliothèque de messagerie Java Amazon SQS.

Utilisation de la bibliothèque de messagerie Java Amazon SQS

Pour commencer à utiliser le service de messagerie Java (JMS) avec Amazon SQS, utilisez les exemples de code de cette section. Les sections suivantes montrent comment créer une connexion et une session JMS, et comment envoyer et recevoir un message.

L'objet client Amazon SQS encapsulé inclus dans la bibliothèque de messagerie Java Amazon SQS vérifie l'existence d'une file d'attente Amazon SQS. Si elle n'existe pas, le client la crée.

Création d'une connexion JMS

Avant de commencer, consultez les prérequis dansConditions requises pour travailler avec JMS et Amazon SQS.

1. Créez une fabrique de connexions et appelez la méthode `createConnection` par rapport à cette fabrique.

```
// Create a new connection factory with all defaults (credentials and region) set
 automatically
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.defaultClient()
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

   La classe `SQSConnection` étend `javax.jms.Connection`. Avec les méthodes de connexion standard JMS, `SQSConnection` propose des méthodes supplémentaires, comme `getAmazonSQSClient` et `getWrappedAmazonSQSClient`. Ces deux méthodes vous permettent d'effectuer des opérations administratives non comprises dans la spécification JMS, telles que la création de files d'attente. Toutefois, la méthode `getWrappedAmazonSQSClient` fournit également une version encapsulée du client Amazon SQS utilisé par la connexion en cours. Le wrapper convertit toutes les exceptions en `JMSException` depuis le client, afin d'en faciliter l'utilisation par le code existant qui attend des occurrences `JMSException`.

2. Vous pouvez utiliser les objets client renvoyés par `getAmazonSQSClient` et `getWrappedAmazonSQSClient` pour effectuer des opérations administratives non comprises dans la spécification JMS (par exemple, vous pouvez créer une file d'attente Amazon SQS).

   Si votre code existant attend des exceptions JMS, vous devez alors utiliser `getWrappedAmazonSQSClient` :

   - Si vous utilisez `getWrappedAmazonSQSClient`, l'objet client renvoyé transforme toutes les exceptions en exceptions JMS.
   - Si vous utilisez `getAmazonSQSClient`, les exceptions sont toutes des exceptions Amazon SQS.

Création d'une file d'attente Amazon SQS

L'objet client encapsulé vérifie s'il existe une file d'attente Amazon SQS.

Si elle n'existe pas, le client la crée. Si la file d'attente existe, la fonction ne renvoie rien. Pour plus d'informations, consultez la section « Création de la file d'attente selon les besoins » section dans l'exemple TextMessageSender.java.

Pour créer une file d'attente standard

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an SQS queue named MyQueue, if it doesn't already exist
if (!client.queueExists("MyQueue")) {
    client.createQueue("MyQueue");
}
```

Pour créer une file d'attente 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**
>
> Le nom d'une file d'attente FIFO doit se terminer par le suffixe `.fifo`.
> Pour plus d'informations sur l'attribut `ContentBasedDeduplication`, consultez Traitement effectué en une seule fois dans Amazon SQS.

Envoi de messages de façon synchrone

1. Lorsque la connexion et la file d'attente Amazon SQS sous-jacente sont prêtes, créez une session JMS non traitée avec le mode `AUTO_ACKNOWLEDGE`.

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. Pour envoyer un SMS à la file d'attente, créez une identité de file d'attente JMS et un producteur de message.

```
// 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. Créez un SMS et envoyez-le à la file d'attente.

   - Pour envoyer un message à une file d'attente standard, vous n'avez pas besoin de définir de paramètres supplémentaires.

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

   - Pour envoyer un message à une file d'attente FIFO, vous devez définir l'ID de groupe de messages. Vous pouvez aussi définir un ID de déduplication de message. Pour de plus amples informations, veuillez consulter Termes clés de la file d'attente FIFO 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"));
```

Réception des messages de façon synchrone

1. Pour recevoir les messages, créez un consommateur correspondant à la même file d'attente et appelez la méthode `start`.

   Vous pouvez appeler à tout moment la méthode `start` avec cette connexion. Toutefois, le consommateur ne reçoit aucun message tant que vous ne l'appelez pas.

   ```
   // Create a consumer for the 'MyQueue'
   MessageConsumer consumer = session.createConsumer(queue);
   // Start receiving incoming messages
   connection.start();
   ```

2. Appelez la méthode `receive` au niveau du consommateur avec un délai d'attente d'une seconde, puis imprimez le contenu du message reçu.

   - Après avoir reçu un message d'une file d'attente standard, vous pouvez accéder à son contenu.

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

   - Après avoir reçu un message d'une file d'attente FIFO, vous pouvez accéder à son contenu et à d'autres attributs de message spécifiques à FIFO, comme l'ID de groupe de messages, l'ID de déduplication du message et le numéro de séquence. Pour de plus amples informations, veuillez consulter Termes clés de la file d'attente FIFO 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. Fermez la connexion et la session.

```
// Close the connection (and the session).
connection.close();
```

La sortie ressemble à ce qui suit:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

> **Note**
>
> Vous pouvez utiliser le framework Spring pour initialiser ces objets.
> Pour plus d'informations, consultez `SpringExampleConfiguration.xml`, `SpringExample.java` et les autres classes d'assistance sous `ExampleConfiguration.java` et `ExampleCommon.java` dans la section Exemples Java fonctionnels pour l'utilisation de JMS avec les files d'attente standard Amazon SQS.

Pour obtenir des exemples complets d'envoi et de réception d'objets, consultez les sections TextMessageSender.java et SyncMessageReceiver.java.

Réception des messages de façon asynchrone

Dans l'exemple de la section Utilisation de la bibliothèque de messagerie Java Amazon SQS, un message est envoyé à `MyQueue` et reçu de façon synchrone.

L'exemple suivant montre comment recevoir les messages de façon asynchrone via un écouteur.

1. Implémentez l'interface `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();
        }
    }
}
```

   La méthode `onMessage` de l'interface `MessageListener` est appelée lorsque vous recevez un message. Dans cette implémentation d'écouteur, le texte stocké dans le message est imprimé.

2. Au lieu d'appeler explicitement la méthode `receive` au niveau du consommateur, sélectionnez une instance de l'implémentation `MyListener` pour l'écouteur de messages du consommateur. Le thread principal attend pendant une seconde.

```
// 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);
```

Les autres étapes sont identiques à celles de l'exemple Utilisation de la bibliothèque de messagerie Java Amazon SQS. Pour obtenir un exemple complet de consommateur asynchrone, consultez `AsyncMessageReceiver.java` dans Exemples Java fonctionnels pour l'utilisation de JMS avec les files d'attente standard Amazon SQS.

La sortie correspondant à cet exemple est similaire à l'exemple suivant :

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Utilisation du mode de reconnaissance du client

L'exemple de la section Utilisation de la bibliothèque de messagerie Java Amazon SQS utilise le mode `AUTO_ACKNOWLEDGE` où chaque message reçu est accepté automatiquement (et, par conséquent, supprimé de la file d'attente Amazon SQS sous-jacente).

1. Pour confirmer explicitement les messages après leur traitement, vous devez créer la session en mode `CLIENT_ACKNOWLEDGE`.

   ```
   // Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
   Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
   ```

2. Lorsque le message est reçu, affichez-le, puis reconnaissez-le explicitement.

   ```
   // 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**
   >
   > Avec ce mode, lorsqu'un message est reconnu, tous les messages reçus avant celui-ci sont implicitement reconnus aussi. Par exemple, si 10 messages sont reçus et que le 10e message est reconnu (dans l'ordre dans lequel les messages sont reçus), les 9 messages précédents sont également reconnus.

Les autres étapes sont identiques à celles de l'exemple Utilisation de la bibliothèque de messagerie Java Amazon SQS. Pour obtenir un exemple complet de consommateur synchrone avec le mode de reconnaissance du client, consultez `SyncMessageReceiverClientAcknowledge.java` dans Exemples Java fonctionnels pour l'utilisation de JMS avec les files d'attente standard Amazon SQS.

La sortie correspondant à cet exemple est similaire à l'exemple suivant :

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

Utilisation du mode de reconnaissance indépendamment de l'ordre de réception

Lorsque vous utilisez le mode `CLIENT_ACKNOWLEDGE`, tous les messages reçus avant un message explicitement reconnu sont reconnus automatiquement. Pour de plus amples informations, veuillez consulter Utilisation du mode de reconnaissance du client.

La bibliothèque de messagerie Java Amazon SQS propose un autre mode de reconnaissance. Lorsque vous utilisez le mode `UNORDERED_ACKNOWLEDGE`, tous les messages reçus doivent être individuellement et explicitement reconnus par le client, quel que soit leur ordre de réception. Pour ce faire, créez une session avec le mode `UNORDERED_ACKNOWLEDGE`.

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

Les étapes suivantes sont identiques à celles de l'exemple Utilisation du mode de reconnaissance du client. Pour obtenir un exemple complet de consommateur synchrone avec le mode `UNORDERED_ACKNOWLEDGE`, consultez `SyncMessageReceiverUnorderedAcknowledge.java`.

Dans cet exemple, la sortie est similaire à l'exemple suivant :

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Utilisation du service de message Java avec d'autres clients Amazon SQS

L'utilisation du client Amazon SQS Java Message Service (JMS) avec le AWS SDK limite la taille des messages Amazon SQS à 256 Ko. Cependant, vous pouvez créer un fournisseur JMS avec n'importe quel client Amazon SQS. Par exemple, vous pouvez utiliser le client JMS avec la bibliothèque client étendue Amazon SQS pour Java afin d'envoyer un message Amazon SQS qui contient une référence à une charge utile de message (jusqu'à 2 Go) dans Amazon S3. Pour de plus amples informations, veuillez consulter Gestion de messages Amazon SQS volumineux à l'aide de Java et Amazon S3.

L'exemple de code Java suivant crée le fournisseur JMS pour la bibliothèque client étendue.

Consultez les conditions préalables Conditions requises pour travailler avec JMS et Amazon SQS avant de tester cet exemple.

```
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);
```

L'exemple de code Java suivant crée la fabrique de connexions :

```
// 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();
```

Exemples Java fonctionnels pour l'utilisation de JMS avec les files d'attente standard Amazon SQS

Les exemples de code suivants montrent comment utiliser le service de messagerie Java (JMS) avec les files d'attente Amazon SQS standard. Pour plus d'informations sur l'utilisation des files d'attente FIFO, consultez Pour créer une file d'attente FIFO, Envoi de messages de façon synchrone et Réception des messages de façon synchrone. (La réception synchrone des messages est identique pour les files d'attente standard et FIFO. Toutefois, les messages des files d'attente FIFO contiennent davantage d'attributs.)

Consultez les conditions préalables Conditions requises pour travailler avec JMS et Amazon SQS avant de tester les exemples suivants.

ExampleConfiguration.java

L'exemple de code Java SDK v 1.x suivant définit le nom de file d'attente par défaut, la région et les informations d'identification à utiliser avec les autres exemples 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

L'exemple de code Java suivant crée un producteur de messages texte.

```
/*
 * 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

L'exemple de code Java suivant crée un consommateur de messages synchrone.

```
/*
 * 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

L'exemple de code Java suivant crée un consommateur de messages asynchrone.

```
/*
 * 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

L'exemple de code Java suivant crée un consommateur synchrone avec le mode de reconnaissance du client.

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

/**
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also
 acknowledged.
 *
```

```
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws 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

L'exemple de code Java suivant crée un consommateur synchrone avec le mode de reconnaissance sans ordre de réception.

```
/*
 * 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

L'exemple de code XML suivant est un fichier de configuration bean pour 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

L'exemple de code Java suivant utilise le fichier de configuration bean pour initialiser vos objets.

```
/*
 * 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

L'exemple de code Java suivant vérifie s'il existe une file d'attente Amazon SQS, puis en crée une si ce n'est pas le cas. Il comprend également un exemple de code de journalisation.

```
/*
 * 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() );
        }
    }
}
```

Implémentations de JMS 1.1 prises en charge par Amazon SQS

La bibliothèque de messagerie Java Amazon SQS prend en charge les implémentations JMS 1.1 suivantes. Pour plus d'informations sur les fonctionnalités prises en charge de la bibliothèque de messagerie Java Amazon SQS, consultez la Foire aux questions sur Amazon SQS.

Interfaces courantes prises en charge

- `Connection`
- `ConnectionFactory`
- `Destination`

- `Session`
- `MessageConsumer`
- `MessageProducer`

Types de messages pris en charge

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

Modes de reconnaissance des messages pris en charge

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

> **Note**
>
> Le mode `UNORDERED_ACKNOWLEDGE` ne fait pas partie de la spécification JMS 1.1. Ce mode permet à Amazon SQS d'autoriser un client JMS à accepter explicitement un message.

En-têtes définis par JMS et propriétés réservées

Pour l'envoi de messages

Lorsque vous envoyez des messages, vous pouvez définir les en-têtes et propriétés ci-après pour chaque message :

- `JMSXGroupID` (obligatoire pour les files d'attente FIFO, non autorisé pour les files d'attente standard)
- `JMS_SQS_DeduplicationId` (facultatif pour les files d'attente FIFO, non autorisé pour les files d'attente standard)

Après l'envoi de messages, Amazon SQS définit les en-têtes et propriétés ci-après pour chaque message :

- `JMSMessageID`
- `JMS_SQS_SequenceNumber` (uniquement pour les files d'attente FIFO)

Pour la réception de messages

Lorsque vous recevez des messages, Amazon SQS définit les en-têtes et propriétés ci-après pour chaque message :

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID` (uniquement pour les files d'attente FIFO)
- `JMS_SQS_DeduplicationId` (uniquement pour les files d'attente FIFO)
- `JMS_SQS_SequenceNumber` (uniquement pour les files d'attente FIFO)

Didacticiels Amazon SQS

Cette rubrique propose des didacticiels destinés à vous aider à découvrir les fonctionnalités d'Amazon SQS.

Didacticiels

- Création d'une file d'attente Amazon SQS à l'aide de AWS CloudFormation
- Didacticiel : envoi d'un message à une file d'attente Amazon SQS à partir d'Amazon Virtual Private Cloud

Création d'une file d'attente Amazon SQS à l'aide de AWS CloudFormation

Utilisez la AWS CloudFormation console avec un modèle JSON ou YAML pour créer une file d'attente Amazon SQS. Pour plus de détails, consultez les sections Utilisation des AWS CloudFormation modèles et de la `AWS::SQS::Queue`ressource dans le guide de AWS CloudFormation l'utilisateur.

À utiliser AWS CloudFormation pour créer une file d'attente Amazon SQS.

1. Copiez le code JSON suivant dans un fichier nommé `MyQueue.json`. Pour créer une file d'attente standard, omettez les propriétés `FifoQueue` et `ContentBasedDeduplication`. Pour plus d'informations sur la déduplication basée sur le contenu, consultez Traitement effectué en une seule fois dans Amazon SQS.

   > **Note**
   >
   > Le nom d'une file d'attente FIFO doit se terminer par le suffixe `.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. Connectez-vous à la console AWS CloudFormation, puis choisissez Créer une pile.
3. Dans le panneau Spécifier un modèle, choisissez Charger un fichier modèle, choisissez votre `MyQueue.json` fichier, puis cliquez sur Suivant.
4. Sur la page Spécifier les détails, tapez `MyQueue` pour Nom de la pile, puis choisissez Suivant.
5. Dans la page Options, choisissez Next (Suivant).
6. Sur la page Review (Vérification), choisissez Create (Créer).

AWS CloudFormation commence à créer la `MyQueue` pile et affiche le statut CREATE_IN_PROGRESS. Lorsque le processus est terminé, AWS CloudFormation affiche l'état 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. (Facultatif) Pour afficher le nom, l'URL et l'ARN de la file d'attente, choisissez le nom de la pile, puis, sur la page suivante, développez la section Outputs.

Didacticiel : envoi d'un message à une file d'attente Amazon SQS à partir d'Amazon Virtual Private Cloud

Ce didacticiel explique comment envoyer des messages à une file d'attente Amazon SQS via un réseau privé sécurisé. Le réseau inclut :

- Un VPC contenant une instance Amazon EC2 .
- Un point de terminaison VPC d'interface, qui permet à l' EC2 instance Amazon de se connecter à Amazon SQS sans utiliser l'Internet public.

Même dans un réseau entièrement privé, vous pouvez vous connecter à l' EC2 instance Amazon et envoyer des messages à la file d'attente Amazon SQS. Pour de plus amples informations, veuillez consulter Points de terminaison Amazon Virtual Private Cloud pour Amazon SQS.

Important

- Vous pouvez utiliser Amazon Virtual Private Cloud uniquement avec des points de terminaison HTTPS Amazon SQS.
- Lorsque vous configurez Amazon SQS pour envoyer des messages depuis Amazon VPC, vous devez activer le DNS privé et spécifier les points de terminaison au format `sqs.`*`us-east-2`*`.amazonaws.com` ou `sqs.`*`us-east-2`*`.api.aws` pour le point de terminaison à double pile.
- Amazon SQS prend également en charge les points de terminaison FIPS PrivateLink via le service de point de terminaison. `com.amazonaws.region.sqs-fips` Vous

pouvez vous connecter aux points de terminaison FIPS au format suivant. `sqs-fips.region.amazonaws.com`

- Lorsque vous utilisez le point de terminaison à double pile dans Amazon Virtual Private Cloud, les demandes seront envoyées à l'aide de IPv4 et IPv6.
- Le DNS privé ne prend pas en charge les points de terminaison existants, tels que `queue.amazonaws.com` ou *`us-east-2`*`.queue.amazonaws.com`.

Étape 1 : créer une paire de EC2 clés Amazon

Une paire de clés vous permet de vous connecter à une EC2 instance Amazon. Elle se compose d'une clé publique qui chiffre vos informations de connexion, et d'une clé privée qui déchiffre ces informations.

1. Connectez-vous à la EC2console Amazon.
2. Dans le menu de navigation, sous Réseau et sécurité, choisissez Paires de clés.
3. Choisissez Create Key Pair (Créer une paire de clés).
4. Dans la boîte de dialogue Créer une paire de clés, pour Nom de la paire de clés, entrez `SQS-VPCE-Tutorial-Key-Pair`, puis choisissez Créer.
5. Votre navigateur télécharge automatiquement le fichier de clé privée `SQS-VPCE-Tutorial-Key-Pair.pem`.

   > **Important**
   >
   > Enregistrez ce fichier dans un endroit sûr. EC2 ne génère pas de `.pem` fichier pour la même paire de clés une seconde fois.

6. Pour autoriser un client SSH à se connecter à votre EC2 instance, définissez les autorisations pour votre fichier de clé privée afin que seul votre utilisateur puisse avoir des autorisations de lecture pour celui-ci, par exemple :

   ```
   chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
   ```

Étape 2 : Création de AWS ressources

Pour configurer l'infrastructure nécessaire, vous devez utiliser un AWS CloudFormation modèle, qui est un modèle pour créer une pile composée de AWS ressources, telles que des EC2 instances Amazon et des files d'attente Amazon SQS.

La pile pour ce didacticiel inclut les ressources suivantes :

- Un VPC et les ressources de mise en réseau associées, notamment un sous-réseau, un groupe de sécurité, une passerelle Internet et une table de routage
- Une EC2 instance Amazon a été lancée dans le sous-réseau VPC
- Une file d'attente Amazon SQS

1. Téléchargez le AWS CloudFormation modèle nommé `SQS-VPCE-Tutorial-CloudFormation.yaml`deGitHub.
2. Connectez-vous à la console AWS CloudFormation.
3. Sélectionnez Créer une pile.
4. Sur la page Sélectionner un modèle, choisissez Télécharger un modèle sur Amazon S3, sélectionnez le fichier `SQS-VPCE-SQS-Tutorial-CloudFormation.yaml`, puis choisissez Suivant.
5. Sur la page Spécification de détails de base de données, procédez comme suit :
   a. Dans le champ Nom de la pile, saisissez `SQS-VPCE-Tutorial-Stack`.
   b. Pour KeyName, choisissez SQS-VPCE-Tutorial-Key-Pair.
   c. Choisissez Suivant.
6. Dans la page Options, choisissez Suivant.
7. Sur la page de révision, dans la section Fonctionnalités, choisissez Je reconnais que cela AWS CloudFormation pourrait créer des ressources IAM avec des noms personnalisés. , puis choisissez Create.

AWS CloudFormation commence à créer la pile et affiche le statut CREATE_IN_PROGRESS. Lorsque le processus est terminé, AWS CloudFormation affiche l'état CREATE_COMPLETE.

Étape 3 : vérifier que votre EC2 instance n'est pas accessible au public

Votre AWS CloudFormation modèle lance une EC2 instance nommée `SQS-VPCE-Tutorial-EC2-Instance` dans votre VPC. Cette EC2 instance n'autorise pas le trafic sortant et n'est pas en mesure d'envoyer des messages à Amazon SQS. Pour le vérifier, vous devez vous connecter à l'instance, essayer de vous connecter à un point de terminaison public, puis essayer d'envoyer un message à Amazon SQS.

1. Connectez-vous à la EC2console Amazon.
2. Dans le menu de navigation, sous Instances, choisissez Instances.
3. Sélectionnez SQS-VPCE-. Tutorial-EC2Instance
4. Copiez le nom d'hôte sous DNS public, par exemple ec2-203-0-113-0.us-west-2.compute.amazonaws.com.
5. À partir du répertoire contenant la paire de clés que vous venez de créer, connectez-vous à l'instance à l'aide de la commande suivante. Par exemple :

   ```
   ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
   ```

6. Essayez de vous connecter à n'importe quel point de terminaison public. Par exemple :

   ```
   ping amazon.com
   ```

   La tentative de connexion échoue, comme prévu.
7. Connectez-vous à la console Amazon SQS.
8. Dans la liste des files d'attente, sélectionnez la file créée par votre AWS CloudFormation modèle, par exemple, VPCE-SQS-Tutorial-Stack--1 IJK. CFQueue ABCDEFGH2
9. Dans le tableau Détails, copiez l'URL, par exemple https://sqs.us-east-2.amazonaws.com/123456789012/.
10. Depuis votre EC2 instance, essayez de publier un message dans la file d'attente à l'aide de la commande suivante, par exemple :

    ```
    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."
    ```

La tentative d'envoi échoue, comme prévu.

> **Important**
>
> Plus tard, lorsque vous créerez le point de terminaison d'un VPC pour Amazon SQS, votre tentative d'envoi réussira.

Étape 4 : Création du point de terminaison d'un VPC Amazon pour Amazon SQS

Pour connecter votre VPC à Amazon SQS, vous devez définir le point de terminaison d'un VPC d'interface. Après avoir ajouté le point de terminaison, vous pouvez utiliser l'API Amazon SQS depuis l' EC2 instance de votre VPC. Cela vous permet d'envoyer des messages à une file d'attente du AWS réseau sans passer par l'Internet public.

> **Note**
>
> L' EC2 instance n'a toujours pas accès aux autres AWS services et points de terminaison sur Internet.

1. Connectez-vous à la console Amazon VPC.
2. Dans le menu de navigation, choisissez Points de terminaison.
3. Choisissez Créer un point de terminaison.
4. Sur la page Créer un point de terminaison, pour Nom du service, choisissez le nom de service pour Amazon SQS.

   > **Note**
   >
   > Les noms des services varient en fonction de la AWS région actuelle. Par exemple, si vous vous trouvez dans l'est des États-Unis (Ohio), le nom du service est com.amazonaws. ***us-east-2***.sqs.

5. Pour VPC, choisissez SQS-VPCE-Tutorial-VPC.

6. Pour Sous-réseaux, choisissez le sous-réseau dont l'ID de sous-réseau contient SQS-VPCE-Tutorial-Subnet.
7. Pour Groupe de sécurité, choisissez Select security groups (Sélectionner des groupes de sécurité), puis choisissez le groupe de sécurité dont le Nom du groupe contient SQS VPCE Tutorial Security Group.
8. Choisissez Créer un point de terminaison.

   Le point de terminaison de VPC d'interface est créé et son ID s'affiche. Par exemple, vpce-0ab1cdef2ghi3j456k.
9. Choisissez Fermer.

   La console Amazon VPC ouvre la page Points de terminaison.

Amazon VPC commence à créer le point de terminaison et affiche le statut en suspens. Lorsque le processus est terminé, Amazon VPC affiche le statut disponible.

Étape 5 : Envoyer un message à votre file d'attente Amazon SQS

Maintenant que votre VPC inclut un point de terminaison pour Amazon SQS, vous pouvez vous connecter à EC2 votre instance et envoyer des messages à votre file d'attente.

1. Reconnectez-vous à votre EC2 instance, par exemple :

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

2. Essayez de publier à nouveau un message dans la file d'attente à l'aide de la commande suivante. Par exemple :

```
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."
```

   La tentative d'envoi réussit et le MD5 résumé du corps du message ainsi que l'ID du message sont affichés, par exemple :

```
{
 "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
 "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
```

```
}
```

Pour plus d'informations sur la réception et la suppression du message de la file d'attente créée par votre AWS CloudFormation modèle (par exemple, VPCE-SQS-Tutorial-Stack--1 CFQueue IJK), consultez. ABCDEFGH2 Réception et suppression d'un message dans Amazon SQS

Pour plus d'informations sur la suppression de vos ressources, consultez ce qui suit :

- Suppression du point de terminaison d'un VPC dans le Guide de l'utilisateur Amazon VPC
- Supprimer une file d'attente Amazon SQS
- Mettez fin à votre instance dans le guide de EC2 l'utilisateur Amazon
- Suppression de votre VPC dans le Guide de l'utilisateur Amazon VPC
- Supprimer une pile sur la AWS CloudFormation console dans le guide de AWS CloudFormation l'utilisateur
- Supprimer votre paire de clés dans le guide de EC2 l'utilisateur Amazon

Exemples de code pour Amazon SQS à l'aide de AWS SDKs

Les exemples de code suivants montrent comment utiliser Amazon SQS avec un kit de développement AWS logiciel (SDK).

Les actions sont des extraits de code de programmes plus larges et doivent être exécutées dans leur contexte. Alors que les actions vous indiquent comment appeler des fonctions de service individuelles, vous pouvez les voir en contexte dans leurs scénarios associés.

Les Scénarios sont des exemples de code qui vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'un même service ou combinés à d'autres Services AWS.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Mise en route

Bonjour Amazon SQS

Les exemples de code suivants montrent comment commencer à utiliser Amazon SQS.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Code pour le CMake fichier CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS sqs)

# Set this project's name.
project("hello_sqs")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if(WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif()

add_executable(${PROJECT_NAME}
        hello_sqs.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code du fichier source 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;
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour C++ API.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
   }
  }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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);
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code AWS.

Initialisez un client Amazon SQS et listez les files d'attente.

```
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"));
};
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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")
            }
    }
}
```

- Pour plus de détails sur l'API, consultez ListQueuesla section AWS SDK pour la référence de l'API Kotlin.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Fichier `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")

    ]
)
```

Le code source 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)
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section AWS SDK pour la référence de l'API Swift.

Exemples de code

- Exemples de base pour l'utilisation d'Amazon SQS AWS SDKs
  - Bonjour Amazon SQS
  - Actions pour Amazon SQS utilisant AWS SDKs
    - Utilisation de AddPermission avec une CLI

- Utilisation ChangeMessageVisibility avec un AWS SDK ou une CLI
- Utilisation de ChangeMessageVisibilityBatch avec une CLI
- Utilisation CreateQueue avec un AWS SDK ou une CLI
- Utilisation DeleteMessage avec un AWS SDK ou une CLI
- Utilisation DeleteMessageBatch avec un AWS SDK ou une CLI
- Utilisation DeleteQueue avec un AWS SDK ou une CLI
- Utilisation GetQueueAttributes avec un AWS SDK ou une CLI
- Utilisation GetQueueUrl avec un AWS SDK ou une CLI
- Utilisation de ListDeadLetterSourceQueues avec une CLI
- Utilisation ListQueues avec un AWS SDK ou une CLI
- Utilisation de PurgeQueue avec une CLI
- Utilisation ReceiveMessage avec un AWS SDK ou une CLI
- Utilisation de RemovePermission avec une CLI
- Utilisation SendMessage avec un AWS SDK ou une CLI
- Utilisation SendMessageBatch avec un AWS SDK ou une CLI
- Utilisation SetQueueAttributes avec un AWS SDK ou une CLI

- Scénarios pour Amazon SQS utilisant AWS SDKs
  - Créez une application Web qui envoie et récupère des messages à l'aide d'Amazon SQS
  - Créer une application de messagerie avec Step Functions
  - Créer une application Amazon Textract Explorer
  - Création et publication sur une rubrique FIFO Amazon SNS à l'aide d'un SDK AWS
  - Détectez les personnes et les objets dans une vidéo avec Amazon Rekognition à l'aide d'un SDK AWS
  - Gérez les messages Amazon SQS volumineux à l'aide d'Amazon S3 avec un SDK AWS
  - Recevez et traitez les notifications d'événements Amazon S3 à l'aide d'un AWS SDK
  - Publiez des messages Amazon SNS dans les files d'attente Amazon SQS à l'aide d'un SDK AWS
  - Envoyez et recevez des lots de messages avec Amazon SQS à l'aide d'un SDK AWS
  - Utilisez le cadre de traitement des AWS messages pour .NET pour publier et recevoir des messages Amazon SQS

- Utilisez la bibliothèque de messagerie Java Amazon SQS pour utiliser l'interface Java Message Service (JMS) pour Amazon SQS
- Utilisez les balises de file d'attente et Amazon SQS à l'aide d'un SDK AWS
- Exemples de solutions sans serveur pour Amazon SQS
  - Invoquer une fonction Lambda à partir d'un déclencheur Amazon SQS
  - Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

Exemples de base pour l'utilisation d'Amazon SQS AWS SDKs

Les exemples de code suivants montrent comment utiliser les bases d'Amazon Simple Queue Service avec AWS SDKs.

Exemples

- Bonjour Amazon SQS
- Actions pour Amazon SQS utilisant AWS SDKs
  - Utilisation de AddPermission avec une CLI
  - Utilisation ChangeMessageVisibility avec un AWS SDK ou une CLI
  - Utilisation de ChangeMessageVisibilityBatch avec une CLI
  - Utilisation CreateQueue avec un AWS SDK ou une CLI
  - Utilisation DeleteMessage avec un AWS SDK ou une CLI
  - Utilisation DeleteMessageBatch avec un AWS SDK ou une CLI
  - Utilisation DeleteQueue avec un AWS SDK ou une CLI
  - Utilisation GetQueueAttributes avec un AWS SDK ou une CLI
  - Utilisation GetQueueUrl avec un AWS SDK ou une CLI
  - Utilisation de ListDeadLetterSourceQueues avec une CLI
  - Utilisation ListQueues avec un AWS SDK ou une CLI
  - Utilisation de PurgeQueue avec une CLI
  - Utilisation ReceiveMessage avec un AWS SDK ou une CLI
  - Utilisation de RemovePermission avec une CLI
  - Utilisation SendMessage avec un AWS SDK ou une CLI
  - Utilisation SendMessageBatch avec un AWS SDK ou une CLI

- Utilisation SetQueueAttributes avec un AWS SDK ou une CLI

Bonjour Amazon SQS

Les exemples de code suivants montrent comment commencer à utiliser Amazon SQS.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Code pour le CMake fichier CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS sqs)

# Set this project's name.
project("hello_sqs")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
```

```
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if(WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif()

add_executable(${PROJECT_NAME}
        hello_sqs.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code du fichier source 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;
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour C++ API.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
  }
 }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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);
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Initialisez un client Amazon SQS et listez les files d'attente.

```
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"));
};
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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")
```

```
            }
    }
}
```

- Pour plus de détails sur l'API, consultez ListQueuesla section AWS SDK pour la référence de l'API Kotlin.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Fichier `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")

    ]
)
```

Le code source 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)
        }
    }
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Actions pour Amazon SQS utilisant AWS SDKs

Les exemples de code suivants montrent comment effectuer des actions Amazon SQS individuelles avec. AWS SDKs Chaque exemple inclut un lien vers GitHub, où vous pouvez trouver des instructions pour configurer et exécuter le code.

Ces extraits appellent l'API Amazon SQS et sont des extraits de code de programmes plus volumineux qui doivent être exécutés en contexte. Vous pouvez voir les actions dans leur contexte dans Scénarios pour Amazon SQS utilisant AWS SDKs .

Les exemples suivants incluent uniquement les actions les plus couramment utilisées. Pour une liste complète, consultez le manuel Amazon Simple Queue Service API Reference.

Exemples

- Utilisation de AddPermission avec une CLI
- Utilisation ChangeMessageVisibility avec un AWS SDK ou une CLI
- Utilisation de ChangeMessageVisibilityBatch avec une CLI
- Utilisation CreateQueue avec un AWS SDK ou une CLI
- Utilisation DeleteMessage avec un AWS SDK ou une CLI
- Utilisation DeleteMessageBatch avec un AWS SDK ou une CLI
- Utilisation DeleteQueue avec un AWS SDK ou une CLI
- Utilisation GetQueueAttributes avec un AWS SDK ou une CLI
- Utilisation GetQueueUrl avec un AWS SDK ou une CLI
- Utilisation de ListDeadLetterSourceQueues avec une CLI
- Utilisation ListQueues avec un AWS SDK ou une CLI
- Utilisation de PurgeQueue avec une CLI
- Utilisation ReceiveMessage avec un AWS SDK ou une CLI
- Utilisation de RemovePermission avec une CLI
- Utilisation SendMessage avec un AWS SDK ou une CLI
- Utilisation SendMessageBatch avec un AWS SDK ou une CLI
- Utilisation SetQueueAttributes avec un AWS SDK ou une CLI

Utilisation de **`AddPermission`** avec une CLI

Les exemples de code suivants illustrent comment utiliser `AddPermission`.

CLI

AWS CLI

Pour ajouter une autorisation à une file d'attente

Cet exemple permet au AWS compte spécifié d'envoyer des messages à la file d'attente spécifiée.

Commande :

```
aws sqs add-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue --aws-account-ids 12345EXAMPLE --actions SendMessage
```

Sortie :

```
None.
```

- Pour plus de détails sur l'API, reportez-vous AddPermissionà la section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple permet à la personne spécifiée Compte AWS d'envoyer des messages depuis la file d'attente spécifiée.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous AddPermissionà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple permet à la personne spécifiée Compte AWS d'envoyer des messages depuis la file d'attente spécifiée.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous AddPermissionà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`ChangeMessageVisibility`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `ChangeMessageVisibility`.

C++

Kit de développement logiciel (SDK) for C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour modifier la visibilité du délai d'expiration d'un message

Cet exemple fait passer le délai de visibilité du message spécifié à 10 heures (10 heures* 60 minutes* 60 secondes).

Commande :

```
aws sqs change-message-visibility --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --
visibility-timeout 36000
```

Sortie :

```
None.
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des AWS CLI commandes.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message Amazon SQS et modifiez la visibilité du délai d'expiration.

```
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;
};
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message Amazon SQS et modifiez la visibilité du délai d'expiration.

```
// 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");
    }
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des AWS SDK pour JavaScript API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple modifie le délai de visibilité du message avec l'identifiant de réception spécifié dans la file d'attente spécifiée à 10 heures (10 heures* 60 minutes* 60 secondes = 36 000 secondes).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple modifie le délai de visibilité du message avec l'identifiant de réception spécifié dans la file d'attente spécifiée à 10 heures (10 heures* 60 minutes* 60 secondes = 36 000 secondes).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityà la section Référence des AWS SDK pour Ruby API.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation de **`ChangeMessageVisibilityBatch`** avec une CLI

Les exemples de code suivants illustrent comment utiliser `ChangeMessageVisibilityBatch`.

CLI

AWS CLI

Pour modifier la visibilité du délai d'expiration de plusieurs messages sous forme de lot

Cet exemple fait passer le délai d'expiration des 2 messages spécifiés à 10 heures (10 heures* 60 minutes* 60 secondes).

Commande :

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

Fichier d'entrée (change-message-visibility-batch.json) :

```
[
  {
    "Id": "FirstMessage",
        "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
        "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBkTUH...HifSnw==",
        "VisibilityTimeout": 36000
  }
]
```

Sortie :

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityBatchà la section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple modifie le délai de visibilité pour 2 messages avec les descripteurs de réception spécifiés dans la file d'attente spécifiée. Le délai de visibilité du premier message passe à 10 heures (10 heures* 60 minutes* 60 secondes = 36 000 secondes). Le délai de visibilité du second message passe à 5 heures (5 heures* 60 minutes* 60 secondes = 18 000 secondes).

```
$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
```

Sortie :

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityBatchà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple modifie le délai de visibilité pour 2 messages avec les descripteurs de réception spécifiés dans la file d'attente spécifiée. Le délai de visibilité du premier message passe à 10 heures (10 heures* 60 minutes* 60 secondes = 36 000 secondes). Le délai de visibilité du second message passe à 5 heures (5 heures* 60 minutes* 60 secondes = 18 000 secondes).

```
$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
```

Sortie :

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- Pour plus de détails sur l'API, reportez-vous ChangeMessageVisibilityBatchà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`CreateQueue`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `CreateQueue`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Publier des messages dans des files d'attente
- Envoyer et recevoir des lots de messages
- Utilisez la bibliothèque de messagerie Java Amazon SQS pour utiliser l'interface JMS

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une file d'attente portant un nom spécifique.

```
    /// <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;
    }
```

Créez une file d'attente Amazon SQS et envoyez-lui un message.

```
    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;
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour créer une file d'attente

Cet exemple crée une file d'attente portant le nom spécifié, définit la période de rétention des messages à 3 jours (3 jours* 24 heures* 60 minutes* 60 secondes) et définit la file d'attente de lettres mortes de la file d'attente comme étant la file d'attente spécifiée avec un nombre maximum de 1 000 messages reçus.

Commande :

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

Fichier d'entrée (create-queue.json) :

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}
```

Sortie :

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des AWS CLI commandes.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
}
```

- Pour plus de détails sur l'API, voir CreateQueuela section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code 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);
        }
    }
}
```

- Pour plus de détails sur l'API, voir CreateQueuela section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une file d'attente standard Amazon SQS.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
```

```
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Créez une file d'attente Amazon SQS avec de longues interrogations.

```
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;
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.

- Pour plus de détails sur l'API, voir CreateQueuela section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une file d'attente standard Amazon SQS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Créez une file d'attente Amazon SQS qui attend l'arrivée d'un message.

```
// 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);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir CreateQueuela section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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()
    }
}
```

- Pour plus de détails sur l'API, consultez CreateQueuela section AWS SDK pour la référence de l'API Kotlin.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple crée une file d'attente portant le nom spécifié.

```
New-SQSQueue -QueueName MyQueue
```

Sortie :

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple crée une file d'attente portant le nom spécifié.

```
New-SQSQueue -QueueName MyQueue
```

Sortie :

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez CreateQueuele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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__
```

- Pour plus de détails sur l'API, voir CreateQueuela section Référence des AWS SDK pour Ruby API.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une file d'attente standard 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.
```

Créez une file d'attente Amazon SQS qui attend l'arrivée d'un message.

```
    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.
```

- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
        }
```

- Pour plus de détails sur l'API, reportez-vous CreateQueueà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`DeleteMessage`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `DeleteMessage`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Envoyer et recevoir des lots de messages

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message depuis une file d'attente Amazon SQS, puis supprimez-le.

```
        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;
        }
    }
```

- Pour plus de détails sur l'API, voir DeleteMessagela section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus de détails sur l'API, voir DeleteMessagela section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour supprimer un message

Cet exemple supprime le message spécifié.

Commande :

```
aws sqs delete-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

Sortie :

```
None.
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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);
        }
```

- Pour plus de détails sur l'API, voir DeleteMessagela section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez et supprimez des messages 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,
        })),
      }),
    );
  }
};
```

- Pour plus de détails sur l'API, voir DeleteMessagela section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez et supprimez des messages 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);
      }
```

```
    });
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir DeleteMessagela section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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!")
    }
}
```

- Pour plus de détails sur l'API, consultez DeleteMessagela section AWS SDK pour la référence de l'API Kotlin.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple supprime le message avec le descripteur de réception spécifié de la file d'attente spécifiée.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple supprime le message avec le descripteur de réception spécifié de la file d'attente spécifiée.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez DeleteMessagele AWS manuel de référence de l'API SDK for Python (Boto3).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`DeleteMessageBatch`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `DeleteMessageBatch`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Traiter les notifications d'événements S3
- Publier des messages dans des files d'attente
- Envoyer et recevoir des lots de messages

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
    }
```

- Pour plus de détails sur l'API, voir DeleteMessageBatchla section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
            }
```

- Pour plus de détails sur l'API, voir DeleteMessageBatchla section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour supprimer plusieurs messages par lots

Cet exemple supprime les messages spécifiés.

Commande :

```
aws sqs delete-message-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-
batch.json
```

Fichier d'entrée (delete-message-batch.json) :

```
[
  {
        "Id": "FirstMessage",
        "ReceiptHandle": "AQEB1mgl...Z4GuLw=="
  },
```

```
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

Sortie :

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageBatchà la section Référence des AWS CLI commandes.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
}
```

- Pour plus de détails sur l'API, voir DeleteMessageBatchla section Référence des AWS SDK pour Go API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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,
        })),
      }),
    );
  }
};
```

- Pour plus de détails sur l'API, voir DeleteMessageBatchla section Référence des AWS SDK pour JavaScript API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple supprime 2 messages avec les descripteurs de réception spécifiés de la file d'attente spécifiée.

```
$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
```

Sortie :

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageBatchà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple supprime 2 messages avec les descripteurs de réception spécifiés de la file d'attente spécifiée.

```
$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
```

Sortie :

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageBatchà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez DeleteMessageBatchle AWS manuel de référence de l'API SDK for Python (Boto3).

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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.")
        }
```

- Pour plus de détails sur l'API, reportez-vous DeleteMessageBatchà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`DeleteQueue`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `DeleteQueue`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Publier des messages dans des files d'attente
- Envoyer et recevoir des lots de messages
- Utilisez la bibliothèque de messagerie Java Amazon SQS pour utiliser l'interface JMS

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez une file d'attente à l'aide de son 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;
    }
```

- Pour plus de détails sur l'API, voir DeleteQueuela section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus de détails sur l'API, voir DeleteQueuela section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour supprimer une file d'attente

Cet exemple supprime la file d'attente spécifiée.

Commande :

```
aws sqs delete-queue --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

Sortie :

```
None.
```

- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section Référence des AWS CLI commandes.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
}
```

- Pour plus de détails sur l'API, voir DeleteQueuela section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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);
        }
    }
}
```

- Pour plus de détails sur l'API, voir DeleteQueuela section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez une file d'attente 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;
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir DeleteQueuela section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Supprimez une file d'attente 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);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir DeleteQueuela section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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!")
    }
}
```

- Pour plus de détails sur l'API, consultez DeleteQueuela section AWS SDK pour la référence de l'API Kotlin.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple supprime la file d'attente spécifiée.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple supprime la file d'attente spécifiée.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez DeleteQueuele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
```

- Pour plus de détails sur l'API, voir DeleteQueuela section Référence des AWS SDK pour Ruby API.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
        MESSAGE 'SQS queue deleted' TYPE 'I'.
    ENDTRY.
```

- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
        }
```

- Pour plus de détails sur l'API, reportez-vous DeleteQueueà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`GetQueueAttributes`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `GetQueueAttributes`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Traiter les notifications d'événements S3
- Publier des messages dans des files d'attente

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
    }
```

- Pour plus de détails sur l'API, voir GetQueueAttributesla section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
```

```
            }
```

- Pour plus de détails sur l'API, voir GetQueueAttributesla section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour obtenir les attributs d'une file d'attente

Cet exemple obtient tous les attributs de la file d'attente spécifiée.

Commande :

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

Sortie :

```
{
  "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"
  }
}
```

Cet exemple obtient uniquement la taille maximale des messages et les attributs de délai de visibilité de la file d'attente spécifiée.

Commande :

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-
names MaximumMessageSize VisibilityTimeout
```

Sortie :

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- Pour plus de détails sur l'API, reportez-vous GetQueueAttributesà la section Référence des AWS CLI commandes.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
}
```

- Pour plus de détails sur l'API, voir GetQueueAttributesla section Référence des AWS SDK pour Go API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
};
```

- Pour plus de détails sur l'API, voir GetQueueAttributesla section Référence des AWS SDK pour JavaScript API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple répertorie tous les attributs de la file d'attente spécifiée.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Sortie :

```
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]...}
```

Exemple 2 : Cet exemple répertorie séparément uniquement les attributs spécifiés pour la file d'attente spécifiée.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Sortie :

```
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]}
```

- Pour plus de détails sur l'API, reportez-vous GetQueueAttributesà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple répertorie tous les attributs de la file d'attente spécifiée.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Sortie :

```
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]...}
```

Exemple 2 : Cet exemple répertorie séparément uniquement les attributs spécifiés pour la file d'attente spécifiée.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Sortie :

```
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]}
```

- Pour plus de détails sur l'API, reportez-vous GetQueueAttributesà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
            }
        }
```

- Pour plus de détails sur l'API, reportez-vous GetQueueAttributesà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`GetQueueUrl`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `GetQueueUrl`.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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.");
            }
        }
    }
```

- Pour plus de détails sur l'API, voir GetQueueUrlla section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus de détails sur l'API, voir GetQueueUrlla section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour obtenir une URL de file d'attente

Cet exemple permet d'obtenir l'URL de la file d'attente spécifiée.

Commande :

```
aws sqs get-queue-url --queue-name MyQueue
```

Sortie :

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- Pour plus de détails sur l'API, reportez-vous GetQueueUrlà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
            GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

- Pour plus de détails sur l'API, voir GetQueueUrlla section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez l'URL d'une file d'attente 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;
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir GetQueueUrlla section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Obtenez l'URL d'une file d'attente 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);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir GetQueueUrlla section Référence des AWS SDK pour JavaScript API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple répertorie l'URL de la file d'attente portant le nom spécifié.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Sortie :

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous GetQueueUrlà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple répertorie l'URL de la file d'attente portant le nom spécifié.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Sortie :

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous GetQueueUrlà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez GetQueueUrlle AWS manuel de référence de l'API SDK for Python (Boto3).

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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.
```

- Pour plus de détails sur l'API, reportez-vous GetQueueUrlà la section de référence du AWS SDK pour l'API SAP ABAP.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation de **`ListDeadLetterSourceQueues`** avec une CLI

Les exemples de code suivants illustrent comment utiliser `ListDeadLetterSourceQueues`.

CLI

AWS CLI

Pour répertorier les files d'attente contenant des sources de lettres mortes

Cet exemple répertorie les files d'attente associées à la file source de lettres mortes spécifiée.

Commande :

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Sortie :

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- Pour plus de détails sur l'API, reportez-vous ListDeadLetterSourceQueuesà la section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple répertorie toutes les URLs files d'attente qui utilisent la file d'attente spécifiée comme file d'attente de lettres mortes.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Sortie :

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Pour plus de détails sur l'API, reportez-vous ListDeadLetterSourceQueuesà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple répertorie toutes les URLs files d'attente qui utilisent la file d'attente spécifiée comme file d'attente de lettres mortes.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Sortie :

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- Pour plus de détails sur l'API, reportez-vous ListDeadLetterSourceQueuesà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`ListQueues`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `ListQueues`.

C++

Kit de développement logiciel (SDK) for C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
}
```

- Pour plus de détails sur l'API, voir ListQueuesla section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour répertorier les files d'attente

Cet exemple répertorie toutes les files d'attente.

Commande :

```
aws sqs list-queues
```

Sortie :

```
{
  "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"
  ]
}
```

Cet exemple répertorie uniquement les files d'attente qui commencent par « Mon ».

Commande :

```
aws sqs list-queues --queue-name-prefix My
```

Sortie :

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des AWS CLI commandes.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
  }
 }
```

```
}
```

- Pour plus de détails sur l'API, voir ListQueuesla section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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);
        }
```

- Pour plus de détails sur l'API, voir ListQueuesla section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez vos files d'attente 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;
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir ListQueuesla section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Répertoriez vos files d'attente 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);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir ListQueuesla section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
        }
    }
}
```

- Pour plus de détails sur l'API, consultez ListQueuesla section AWS SDK pour la référence de l'API Kotlin.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple répertorie toutes les files d'attente.

```
Get-SQSQueue
```

Sortie :

```
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
```

Exemple 2 : Cet exemple répertorie toutes les files d'attente qui commencent par le nom spécifié.

```
Get-SQSQueue -QueueNamePrefix My
```

Sortie :

```
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
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple répertorie toutes les files d'attente.

```
Get-SQSQueue
```

Sortie :

```
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
```

Exemple 2 : Cet exemple répertorie toutes les files d'attente qui commencent par le nom spécifié.

```
Get-SQSQueue -QueueNamePrefix My
```

Sortie :

```
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
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez ListQueuesle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, voir ListQueuesla section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Récupérez la première file d'attente Amazon SQS répertoriée dans la région.

```
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())
}
```

- Pour plus de détails sur l'API, voir ListQueuesla section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

```
    TRY.
        oo_result = lo_sqs->listqueues( ).        " oo_result is returned for
 testing purposes. "
        MESSAGE 'Retrieved list of queues.' TYPE 'I'.
```

```
    ENDTRY.
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
            }
        }
```

- Pour plus de détails sur l'API, reportez-vous ListQueuesà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation de **`PurgeQueue`** avec une CLI

Les exemples de code suivants illustrent comment utiliser `PurgeQueue`.

CLI

AWS CLI

Pour purger une file d'attente

Cet exemple supprime tous les messages de la file d'attente spécifiée.

Commande :

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyNewQueue
```

Sortie :

```
None.
```

- Pour plus de détails sur l'API, reportez-vous PurgeQueueà la section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple supprime tous les messages de la file d'attente spécifiée.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous PurgeQueueà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple supprime tous les messages de la file d'attente spécifiée.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous PurgeQueueà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`ReceiveMessage`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `ReceiveMessage`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Gérez les messages volumineux à l'aide de S3
- Traiter les notifications d'événements S3
- Publier des messages dans des files d'attente
- Envoyer et recevoir des lots de messages

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez les messages d'une file d'attente à l'aide de son 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;
    }
```

Recevez un message depuis une file d'attente Amazon SQS, puis supprimez-le.

```
        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 ruque 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;
        }
    }
```

- Pour plus de détails sur l'API, voir ReceiveMessagela section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus de détails sur l'API, voir ReceiveMessagela section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour recevoir un message

Cet exemple reçoit jusqu'à 10 messages disponibles et renvoie tous les attributs disponibles.

Commande :

```
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
```

Sortie :

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

Cet exemple reçoit le prochain message disponible, renvoyant uniquement les SentTimestamp attributs SenderId et ainsi que l'attribut PostalCode message.

Commande :

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-
names SenderId SentTimestamp --message-attribute-names PostalCode
```

Sortie :

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

- Pour plus de détails sur l'API, reportez-vous ReceiveMessageà la section Référence des AWS CLI commandes.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
}
```

- Pour plus de détails sur l'API, voir ReceiveMessagela section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
```

- Pour plus de détails sur l'API, voir ReceiveMessagela section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message depuis une file d'attente 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,
        })),
      }),
    );
  }
};
```

Recevez un message depuis une file d'attente Amazon SQS grâce à la prise en charge des longs sondages.

```
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;
};
```

- Pour plus de détails sur l'API, voir ReceiveMessagela section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message depuis une file d'attente Amazon SQS grâce à la prise en charge des longs sondages.

```
// 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);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir ReceiveMessagela section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
        }
    }
}
```

- Pour plus de détails sur l'API, consultez ReceiveMessagela section AWS SDK pour la référence de l'API Kotlin.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple répertorie les informations relatives aux 10 prochains messages à recevoir pour la file d'attente spécifiée. Les informations contiendront des valeurs pour les attributs de message spécifiés, s'ils existent.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Sortie :

```
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==
```

- Pour plus de détails sur l'API, reportez-vous ReceiveMessageà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple répertorie les informations relatives aux 10 prochains messages à recevoir pour la file d'attente spécifiée. Les informations contiendront des valeurs pour les attributs de message spécifiés, s'ils existent.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Sortie :

```
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==
```

- Pour plus de détails sur l'API, reportez-vous ReceiveMessageà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez ReceiveMessagele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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__
```

- Pour plus de détails sur l'API, voir ReceiveMessagela section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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(())
}
```

- Pour plus de détails sur l'API, voir ReceiveMessagela section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Recevez un message depuis une file d'attente 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.
```

Recevez un message depuis une file d'attente Amazon SQS grâce à la prise en charge des longs sondages.

```
   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.
```

- Pour plus de détails sur l'API, reportez-vous ReceiveMessageà la section de référence du AWS SDK pour l'API SAP ABAP.

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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("---")
        }
```

- Pour plus de détails sur l'API, reportez-vous ReceiveMessageà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation de **`RemovePermission`** avec une CLI

Les exemples de code suivants illustrent comment utiliser `RemovePermission`.

CLI

AWS CLI

Pour supprimer une autorisation

Cet exemple supprime l'autorisation portant l'étiquette spécifiée de la file d'attente spécifiée.

Commande :

```
aws sqs remove-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue
```

Sortie :

```
None.
```

- Pour plus de détails sur l'API, reportez-vous RemovePermissionà la section Référence des AWS CLI commandes.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple supprime les paramètres d'autorisation portant l'étiquette spécifiée de la file d'attente spécifiée.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous RemovePermissionà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple supprime les paramètres d'autorisation portant l'étiquette spécifiée de la file d'attente spécifiée.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous RemovePermissionà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`SendMessage`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `SendMessage`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans les exemples de code suivants :

- Gérez les messages volumineux à l'aide de S3
- Envoyer et recevoir des lots de messages

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une file d'attente Amazon SQS et envoyez-lui un message.

```
    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;
        }
    }
```

- Pour plus de détails sur l'API, voir SendMessagela section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus de détails sur l'API, voir SendMessagela section Référence des AWS SDK pour C ++ API.

CLI

AWS CLI

Pour envoyer un message

Cet exemple envoie un message avec le corps du message, le délai et les attributs de message spécifiés à la file d'attente spécifiée.

Commande :

```
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
```

Fichier d'entrée (send-message.json) :

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

Sortie :

```
{
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- Pour plus de détails sur l'API, reportez-vous SendMessageà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Voici deux exemples de `SendMessage` l'opération :

- Envoyer un message avec un corps et un délai

- Envoyer un message avec un corps et des attributs de message

Envoyez un message avec un corps et un délai.

```
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);
        }
    }
}
```

Envoyez un message avec un corps et des attributs de message.

```
    /**
     * <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);
```

```
        }
    }
```

- Pour plus de détails sur l'API, voir SendMessagela section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message à une file d'attente 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;
};
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir SendMessagela section Référence des AWS SDK pour JavaScript API.

SDK pour JavaScript (v2)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Envoyez un message à une file d'attente 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);
  }
});
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK pour JavaScript.
- Pour plus de détails sur l'API, voir SendMessagela section Référence des AWS SDK pour JavaScript API.

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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.")
```

```
    }
}
```

- Pour plus de détails sur l'API, consultez SendMessagela section AWS SDK pour la référence de l'API Kotlin.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple envoie un message avec les attributs et le corps du message spécifiés à la file d'attente spécifiée avec un délai de livraison du message de 10 secondes.

```
$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
```

Sortie :

```
MD5OfMessageAttributes              MD5OfMessageBody                    MessageId

----------------------              ----------------                    ---------

1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- Pour plus de détails sur l'API, reportez-vous SendMessageà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple envoie un message avec les attributs et le corps du message spécifiés à la file d'attente spécifiée avec un délai de livraison du message de 10 secondes.

```
$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
```

Sortie :

```
MD5OfMessageAttributes              MD5OfMessageBody                   MessageId

----------------------              ----------------                   ---------

1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- Pour plus de détails sur l'API, reportez-vous SendMessageà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l’exemple complet et découvrez comment le configurer et l’exécuter dans le référentiel d’exemples de code 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
```

- Pour plus de détails sur l'API, consultez SendMessagele AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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__
```

- Pour plus de détails sur l'API, voir SendMessagela section Référence des AWS SDK pour Ruby API.

Rust

SDK pour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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(())
}
```

- Pour plus de détails sur l'API, voir SendMessagela section de référence de l'API AWS SDK for Rust.

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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.
```

- Pour plus de détails sur l'API, reportez-vous SendMessageà la section de référence du AWS SDK pour l'API SAP ABAP.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`SendMessageBatch`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `SendMessageBatch`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Envoyer et recevoir des lots de messages

CLI

AWS CLI

Pour envoyer plusieurs messages par lots

Cet exemple envoie 2 messages avec les corps de message, les délais et les attributs de message spécifiés à la file d'attente spécifiée.

Commande :

```
aws sqs send-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-batch.json
```

Fichier d'entrée (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"
      }
        }
  }
]
```

Sortie :

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

- Pour plus de détails sur l'API, reportez-vous SendMessageBatchà la section Référence des AWS CLI commandes.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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);
```

- Pour plus de détails sur l'API, voir SendMessageBatchla section Référence des AWS SDK for Java 2.x API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple envoie 2 messages avec les attributs et les corps de message spécifiés à la file d'attente spécifiée. La livraison est retardée de 15 secondes pour le premier message et de 10 secondes pour le second.

```
$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
```

Sortie :

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- Pour plus de détails sur l'API, reportez-vous SendMessageBatchà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple envoie 2 messages avec les attributs et les corps de message spécifiés à la file d'attente spécifiée. La livraison est retardée de 15 secondes pour le premier message et de 10 secondes pour le second.

```
$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
```

Sortie :

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- Pour plus de détails sur l'API, reportez-vous SendMessageBatchà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, consultez SendMessageBatchle AWS manuel de référence de l'API SDK for Python (Boto3).

Ruby

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, reportez-vous SendMessageBatchà la section Référence des AWS SDK pour Ruby API.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisation **`SetQueueAttributes`** avec un AWS SDK ou une CLI

Les exemples de code suivants illustrent comment utiliser `SetQueueAttributes`.

Les exemples d'actions sont des extraits de code de programmes de plus grande envergure et doivent être exécutés en contexte. Vous pouvez voir cette action en contexte dans l'exemple de code suivant :

- Publier des messages dans des files d'attente

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Définissez l'attribut de politique d'une file d'attente pour un sujet.

```
    /// <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;
    }
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS SDK pour .NET API.

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

Configurer une file d'attente de lettres mortes.

```
        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();
}
```

Configurez une file d'attente Amazon SQS pour utiliser des interrogations longues.

```
        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();
}
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS SDK pour C++ API.

CLI

AWS CLI

Pour définir les attributs de file d'attente

Cet exemple définit pour la file d'attente spécifiée un délai de livraison de 10 secondes, une taille de message maximale de 128 Ko (128 Ko x 1 024 octets), une période de rétention des messages de 3 jours (3 jours* 24 heures* 60 minutes* 60 secondes), un temps d'attente des messages de réception de 20 secondes et un délai de visibilité par défaut de 60 secondes. Cet exemple associe également la file d'attente de lettres mortes spécifiée à un nombre maximum de 1 000 messages reçus.

Commande :

```
aws sqs set-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-
attributes.json
```

Fichier d'entrée (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"
}
```

Sortie :

```
None.
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS CLI commandes.

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS SDK pour Go API.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Configurez un Amazon SQS pour utiliser le chiffrement côté serveur (SSE) à l'aide d'une clé KMS personnalisée.

```
   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();
```

```
        }
    }
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS SDK for Java 2.x API.

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
};
```

Configurez une file d'attente Amazon SQS pour utiliser des interrogations longues.

```
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;
};
```

Configurer une file d'attente de lettres mortes.

```
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;
};
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des AWS SDK pour JavaScript API.

PowerShell

Outils pour PowerShell V4

Exemple 1 : Cet exemple montre comment définir une politique d'abonnement à une file d'attente à une rubrique SNS. Lorsqu'un message est publié dans le sujet, un message est envoyé à la file d'attente abonnée.

```
# 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 }
```

Exemple 2 : Cet exemple définit les attributs spécifiés pour la file d'attente spécifiée.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des Outils AWS pour PowerShell applets de commande (V4).

Outils pour PowerShell V5

Exemple 1 : Cet exemple montre comment définir une politique d'abonnement à une file d'attente à une rubrique SNS. Lorsqu'un message est publié dans le sujet, un message est envoyé à la file d'attente abonnée.

```
# 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 }
```

Exemple 2 : Cet exemple définit les attributs spécifiés pour la file d'attente spécifiée.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section Référence des Outils AWS pour PowerShell applets de commande (V5).

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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.")
        }
```

- Pour plus de détails sur l'API, reportez-vous SetQueueAttributesà la section AWS SDK pour la référence de l'API Swift.

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Scénarios pour Amazon SQS utilisant AWS SDKs

Les exemples de code suivants vous montrent comment implémenter des scénarios courants dans Amazon SQS avec. AWS SDKs Ces scénarios vous montrent comment accomplir des tâches spécifiques en appelant plusieurs fonctions au sein d'Amazon SQS ou en les combinant à d'autres. Services AWS Chaque exemple inclut un lien vers le code source complet, où vous trouverez des instructions sur la configuration et l'exécution du code.

Les scénarios ciblent un niveau d'expérience intermédiaire pour vous aider à comprendre les actions de service dans leur contexte.

Exemples

- Créez une application Web qui envoie et récupère des messages à l'aide d'Amazon SQS
- Créer une application de messagerie avec Step Functions
- Créer une application Amazon Textract Explorer
- Création et publication sur une rubrique FIFO Amazon SNS à l'aide d'un SDK AWS
- Détectez les personnes et les objets dans une vidéo avec Amazon Rekognition à l'aide d'un SDK AWS

- Gérez les messages Amazon SQS volumineux à l'aide d'Amazon S3 avec un SDK AWS
- Recevez et traitez les notifications d'événements Amazon S3 à l'aide d'un AWS SDK
- Publiez des messages Amazon SNS dans les files d'attente Amazon SQS à l'aide d'un SDK AWS
- Envoyez et recevez des lots de messages avec Amazon SQS à l'aide d'un SDK AWS
- Utilisez le cadre de traitement des AWS messages pour .NET pour publier et recevoir des messages Amazon SQS
- Utilisez la bibliothèque de messagerie Java Amazon SQS pour utiliser l'interface Java Message Service (JMS) pour Amazon SQS
- Utilisez les balises de file d'attente et Amazon SQS à l'aide d'un SDK AWS

Créez une application Web qui envoie et récupère des messages à l'aide d'Amazon SQS

Les exemples de code suivants montrent comment créer une application de messagerie à l'aide d'Amazon SQS.

Java

SDK pour Java 2.x

Montre comment utiliser l'API Amazon SQS pour développer une API Spring REST qui envoie et récupère des messages.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK pour Kotlin

Montre comment utiliser l'API Amazon SQS pour développer une API Spring REST qui envoie et récupère des messages.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Comprehend
- Amazon SQS

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Créer une application de messagerie avec Step Functions

L'exemple de code suivant montre comment créer une application de AWS Step Functions messagerie qui extrait les enregistrements de messages d'une table de base de données.

Python

SDK pour Python (Boto3)

Montre comment utiliser le AWS SDK pour Python (Boto3) with AWS Step Functions pour créer une application de messagerie qui récupère les enregistrements de messages d'une table Amazon DynamoDB et les envoie via Amazon Simple Queue Service (Amazon SQS). La machine d'état intègre une AWS Lambda fonction permettant de scanner la base de données à la recherche de messages non envoyés.

- Créez une machine d’état qui extrait et met à jour des enregistrements de message d’une table Amazon DynamoDB.
- Mettez à jour la définition de la machine d’état pour envoyer des messages à Amazon Simple Queue Service (Amazon SQS).
- Démarrez et arrêtez les exécutions de la machine.
- Connectez-vous à Lambda, DynamoDB et Amazon SQS à partir d’une machine d’état à l’aide d’intégrations de services.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- DynamoDB

- Lambda
- Amazon SQS
- Step Functions

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Créer une application Amazon Textract Explorer

Les exemples de code suivants expliquent comment explorer la sortie Amazon Textract via une application interactive.

JavaScript

SDK pour JavaScript (v3)

Montre comment utiliser le AWS SDK pour JavaScript pour créer une application React qui utilise Amazon Textract pour extraire des données d'une image de document et les afficher sur une page Web interactive. Cet exemple s’exécute dans un navigateur Web et nécessite une identité Amazon Cognito authentifiée pour les informations d’identification. Il utilise Amazon Simple Storage Service (Amazon S3) pour le stockage et, pour les notifications, il interroge une file d’attente Amazon Simple Queue Service (Amazon SQS) abonnée à une rubrique Amazon Simple Notification Service (Amazon SNS).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK pour Python (Boto3)

Montre comment utiliser Amazon Textract pour détecter des éléments de texte, de formulaire et de tableau dans une image de document. AWS SDK pour Python (Boto3) L'image d'entrée et la sortie d'Amazon Textract sont affichées dans une application Tkinter qui vous permet d'explorer les éléments détectés.

- Soumettez une image de document à Amazon Textract et explorez la sortie des éléments détectés.
- Soumettez des images directement à Amazon Textract ou via un compartiment Amazon Simple Storage Service (Amazon S3).
- Utilisez le mode asynchrone APIs pour démarrer une tâche qui publie une notification dans une rubrique Amazon Simple Notification Service (Amazon SNS) une fois la tâche terminée.
- Interrogez un service Amazon Simple Queue Service (Amazon SQS) pour obtenir un message de fin de tâche et affichez les résultats.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Création et publication sur une rubrique FIFO Amazon SNS à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment créer une rubrique FIFO Amazon SNS et y publier des informations.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple

- crée une rubrique FIFO Amazon SNS, deux files d'attente FIFO Amazon SQS et une file d'attente standard.
- abonne les files d'attente à la rubrique et publie un message dans la rubrique.

Le test vérifie que le message a bien été reçu pour chaque file d'attente. L'exemple complet montre également l'ajout de stratégies d'accès et supprime les ressources à la fin.

```
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);
        }
    }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - CreateTopic
  - Publish
  - Subscribe

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une rubrique FIFO Amazon SNS, abonnez les files d'attentes standard et FIFO Amazon SQS à la rubrique et publiez un message dans la rubrique.

```
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
```

- Pour plus d’informations sur l’API, consultez les rubriques suivantes dans la Référence du kit SDK AWS pour l'API Python (Boto3).
  - CreateTopic
  - Publish
  - Subscribe

SAP ABAP

Kit SDK pour SAP ABAP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez une rubrique FIFO, abonnez une file d'attente FIFO Amazon SQS à la rubrique et publiez un message dans une rubrique 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.
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la Référence du kit SDK AWS de l'API SAP ABAP.
  - CreateTopic
  - Publish
  - Subscribe

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Détectez les personnes et les objets dans une vidéo avec Amazon Rekognition à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment détecter des personnes et des objets dans une vidéo avec Amazon Rekognition.

Java

SDK pour Java 2.x

Montre comment utiliser l'API Java Amazon Rekognition afin de créer une application qui détecte les visages et les objets dans des vidéos stockées dans un compartiment Amazon Simple Storage Service (Amazon S3). L'application envoie à l'administrateur une notification par e-mail contenant les résultats à l'aide d'Amazon Simple Email Service (Amazon SES).

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK pour Python (Boto3)

Utilisez Amazon Rekognition pour détecter des visages, des objets et des personnes dans des vidéos en démarrant des tâches de détection asynchrone. Cet exemple montre également comment configurer Amazon Rekognition pour notifier une rubrique Amazon Simple Notification Service (Amazon SNS) lorsque les tâches sont terminées et abonner une file d'attente Amazon Simple Queue Service (Amazon SQS) à la rubrique. Lorsque la file d'attente reçoit un message concernant une tâche, elle est récupérée et les résultats sont affichés.

Il est préférable de visionner cet exemple sur GitHub. Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez l'exemple complet sur GitHub.

Les services utilisés dans cet exemple

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Gérez les messages Amazon SQS volumineux à l'aide d'Amazon S3 avec un SDK AWS

L'exemple de code suivant montre comment utiliser la bibliothèque client étendue Amazon SQS pour gérer des messages Amazon SQS volumineux.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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;
        }
    }
```

- Pour de plus amples informations, consultez le Guide du développeur AWS SDK for Java 2.x.
- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - CreateBucket
  - PutBucketLifecycleConfiguration
  - ReceiveMessage
  - SendMessage

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Recevez et traitez les notifications d'événements Amazon S3 à l'aide d'un AWS SDK

L'exemple de code suivant montre comment utiliser les notifications d'événements S3 de manière orientée objet.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Cet exemple montre comment traiter un événement de notification S3 à l'aide d'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);
        }
    }
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - DeleteMessageBatch
  - GetQueueAttributes
  - PutBucketNotificationConfiguration
  - ReceiveMessage

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Publiez des messages Amazon SNS dans les files d'attente Amazon SQS à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment :

- Créer une rubrique (FIFO ou non FIFO).
- Abonner plusieurs files d'attente à la rubrique avec la possibilité d'appliquer un filtre.
- Publier des messages dans la rubrique.
- Interroger les files d'attente pour les messages reçus.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

```
/// <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;
    }
}
```

Créez une classe qui englobe les opérations 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;
    }
}
```

Créez une classe qui englobe les opérations 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;
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour .NET .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage

- SetQueueAttributes
- Subscribe
- Unsubscribe

C++

SDK pour C++

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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();
}
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK pour C++ .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe

- Unsubscribe

Go

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Exécutez un scénario interactif à une invite de commande.

```
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))
}
```

Définissez une structure qui englobe les actions Amazon SNS utilisées dans cet exemple.

```
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
}
```

Définissez une structure qui englobe les actions Amazon SQS utilisées dans cet exemple.

```
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
}
```

Nettoyez les ressources.

```
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)
  }
 }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour Go .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish

- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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 "";
    }
}
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

JavaScript

SDK pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

C'est le point d'entrée de ce scénario.

```
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();
};
```

Le code précédent fournit les dépendances nécessaires et lance le scénario. La section suivante contient l'essentiel de l'exemple.

```
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();
    }
  }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK pour JavaScript .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Kotlin

SDK pour Kotlin

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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
    }
}
```

- Pour plus d’informations sur l’API, consultez les rubriques suivantes dans AWS SDK for Kotlin API reference.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic

- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Swift

Kit SDK pour Swift

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code 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)
        }
    }
}
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans la référence de l'API du kit SDK AWS pour Swift.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Envoyez et recevez des lots de messages avec Amazon SQS à l'aide d'un SDK AWS

Les exemples de code suivants montrent comment :

- Créez une file d'attente Amazon SQS.

- Envoyez des lots de messages à la file d'attente.
- Recevez des lots de messages depuis la file d'attente.
- Supprimez des lots de messages de la file d'attente.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Comme le montrent les exemples suivants, vous pouvez gérer les opérations de messages par lots avec Amazon SQS en utilisant deux approches différentes avec : AWS SDK for Java 2.x

SendRecvBatch.java utilise des opérations par lots explicites. Vous créez manuellement des lots de messages et vous appelez `sendMessageBatch()` et `deleteMessageBatch()` directement. Vous gérez également les réponses par lots, y compris les messages ayant échoué. Cette approche vous permet de contrôler totalement le dimensionnement des lots et la gestion des erreurs. Cependant, il faut davantage de code pour gérer la logique de traitement par lots.

SimpleProducerConsumer.java utilise la `SqsAsyncBatchManager` bibliothèque de haut niveau pour le traitement automatique des requêtes par lots. Vous passez `sendMessage()` des `deleteMessage()` appels individuels avec les mêmes méthodes de signature que le client standard. Le SDK met automatiquement en mémoire tampon ces appels et les envoie sous forme d'opérations par lots. Cette approche nécessite des modifications de code minimales tout en offrant des avantages en termes de performances de traitement par lots.

Utilisez le traitement par lots explicite lorsque vous avez besoin d'un contrôle précis de la composition des lots et de la gestion des erreurs. Utilisez le traitement par lots automatique lorsque vous souhaitez optimiser les performances avec un minimum de modifications de code.

SendRecvBatch.java - Utilise des opérations par lots explicites avec les messages.

```
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 - Utilise le traitement automatique des messages par lots.

```
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.
            }
        }
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Python

SDK pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Créez des fonctions pour encapsuler les fonctions de message 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
```

Utilisez les fonctions wrapper pour envoyer et recevoir des messages par lots.

```
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)
```

- Pour plus d'informations sur l'API, consultez les rubriques suivantes dans AWS SDK for Python (Boto3) API Reference.
  - CreateQueue
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisez le cadre de traitement des AWS messages pour .NET pour publier et recevoir des messages Amazon SQS

L'exemple de code suivant montre comment créer des applications qui publient et reçoivent des messages Amazon SQS à l'aide de l' AWS infrastructure de traitement des messages pour .NET.

.NET

SDK pour .NET

Fournit un didacticiel pour le cadre de traitement des AWS messages pour .NET. Le didacticiel crée une application Web qui permet à l'utilisateur de publier un message Amazon SQS et une application de ligne de commande qui reçoit le message.

Pour obtenir le code source complet et les instructions de configuration et d'exécution, consultez le didacticiel complet dans le guide du AWS SDK pour .NET développeur et l'exemple ci-dessous GitHub.

Les services utilisés dans cet exemple

- Amazon SQS

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisez la bibliothèque de messagerie Java Amazon SQS pour utiliser l'interface Java Message Service (JMS) pour Amazon SQS

L'exemple de code suivant montre comment utiliser la bibliothèque de messagerie Java Amazon SQS pour travailler avec l'interface JMS.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

Les exemples suivants fonctionnent avec les files d'attente Amazon SQS standard et incluent :

- Envoi d'un texto.
- Réception synchrone des messages.
- Réception de messages de manière asynchrone.
- Réception de messages à l'aide du mode CLIENT_ACKNOWLEDGE.
- Réception de messages à l'aide du mode UNORDERED_ACKNOWLEDGE.
- Utiliser Spring pour injecter des dépendances.
- Classe utilitaire qui fournit les méthodes communes utilisées dans les autres exemples.

Pour plus d'informations sur l'utilisation de JMS avec Amazon SQS, consultez le manuel du développeur Amazon SQS.

Envoi d'un 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);
        }
    }
```

Réception synchrone des messages.

```
    /**
     * 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);
        }
    }
```

Réception de messages de manière asynchrone.

```
    /**
     * 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" );
    }
```

Réception de messages à l'aide du mode CLIENT_ACKNOWLEDGE.

```
    /**
     * This method demonstrates how message acknowledgment affects message
processing in a standard
     * Amazon SQS queue using the Amazon SQS Java Messaging Library. It sends
messages to the queue,
     * then shows how JMS (Java Message Service) client acknowledgment mode
handles both explicit
     * and implicit message confirmations, including how acknowledging one
message can automatically
     * acknowledge previous messages.
     *
     * @throws 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();
       }
   }
```

Réception de messages à l'aide du mode UNORDERED_ACKNOWLEDGE.

```
   /**
    * Demonstrates message acknowledgment behavior in UNORDERED_ACKNOWLEDGE mode
with Amazon SQS JMS.
    * In this mode, each message must be explicitly acknowledged regardless of
receive order.
    * Unacknowledged messages return to the queue after the visibility timeout
expires,
    * unlike CLIENT_ACKNOWLEDGE mode where acknowledging one message
acknowledges all previous messages.
    *
    * @throws 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();
        }
    }
```

Utiliser Spring pour injecter des dépendances.

```
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);
        }
    }
}
```

Définitions du haricot de printemps.

```
<?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>
```

Classe utilitaire qui fournit les méthodes communes utilisées dans les autres exemples.

```
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);
            }
        };
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - CreateQueue
  - DeleteQueue

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Utilisez les balises de file d'attente et Amazon SQS à l'aide d'un SDK AWS

L'exemple de code suivant montre comment effectuer une opération de balisage avec Amazon SQS.

Java

SDK pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples de code AWS.

L'exemple suivant crée des balises pour une file d'attente, répertorie les balises et supprime une balise.

```
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);
        }
    }
}
```

- Pour plus d'informations sur l'API consultez les rubriques suivantes dans la référence de l'API AWS SDK for Java 2.x .
  - ListQueueTags
  - TagQueue
  - UntagQueue

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Exemples de solutions sans serveur pour Amazon SQS

Les exemples de code suivants montrent comment utiliser Amazon SQS avec. AWS SDKs

Exemples

- Invoquer une fonction Lambda à partir d'un déclencheur Amazon SQS
- Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

Invoquer une fonction Lambda à partir d'un déclencheur Amazon SQS

Les exemples de code suivants montrent comment implémenter une fonction Lambda qui reçoit un événement déclenché par la réception de messages à partir d'une file d'attente SQS. La fonction extrait les messages du paramètre d'événement et consigne le contenu de chaque message.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement SQS avec Lambda en utilisant .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

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement SQS avec Lambda à l'aide de 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 pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement SQS avec Lambda à l'aide de 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 pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement SQS avec JavaScript Lambda en utilisant.

```
// 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;
  }
}
```

Consommation d'un événement SQS avec TypeScript Lambda en utilisant.

```
// 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

Kit SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement SQS avec Lambda à l'aide de 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 pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement SQS avec Lambda à l'aide de 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

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Utilisation d'un événement SQS avec Lambda à l'aide de 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 pour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Consommation d'un événement SQS avec Lambda à l'aide de 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
}
```

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Signalement des échecs d'articles par lots pour les fonctions Lambda à l'aide d'un déclencheur Amazon SQS

Les exemples de code suivants montrent comment implémenter une réponse par lots partielle pour les fonctions Lambda qui reçoivent des événements d'une file d'attente SQS. La fonction signale

les défaillances échecs d'articles par lots dans la réponse, en indiquant à Lambda de réessayer ces messages ultérieurement.

.NET

SDK pour .NET

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de .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

Kit SDK for Go V2

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de 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 pour Java 2.x

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de 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 pour JavaScript (v3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signaler les défaillances d'éléments de lot SQS avec JavaScript Lambda à l'aide de.

```
// 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}`);
}
```

Signaler les défaillances d'éléments de lot SQS avec TypeScript Lambda à l'aide de.

```
// 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

Kit SDK pour PHP

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de 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 pour Python (Boto3)

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de 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

Kit SDK pour Ruby

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de 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 pour Rust

> **Note**
>
> Il y en a plus à ce sujet GitHub. Trouvez l'exemple complet et découvrez comment le configurer et l'exécuter dans le référentiel d'exemples sans serveur.

Signalement des échecs d'articles par lots SQS avec Lambda à l'aide de 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
}
```

Pour obtenir la liste complète des guides de développement du AWS SDK et des exemples de code, consultezUtilisation d'Amazon SQS avec un SDK AWS. Cette rubrique comprend également des informations sur le démarrage et sur les versions précédentes de SDK.

Résolution des problèmes dans Amazon SQS

Cette rubrique fournit des conseils de résolution des problèmes et erreurs courants que vous pouvez rencontrer lors de l'utilisation de la console Amazon SQS, de l'API Amazon SQS ou d'autres outils avec Amazon SQS. Si vous rencontrez un problème qui n'est pas répertorié ici, vous pouvez utiliser le bouton Commentaire sur cette page pour le signaler.

Pour plus de conseils de dépannage et de réponses aux questions courantes de support, visitez le Centre de connaissances AWS.

Rubriques

- Résoudre une erreur de refus d'accès dans Amazon SQS
- Résoudre les erreurs d'API Amazon SQS
- Résoudre les problèmes liés à la file d'attente des lettres mortes Amazon SQS et au redrive DLQ
- Résoudre les problèmes de régulation FIFO dans Amazon SQS
- Résoudre les problèmes liés aux messages non renvoyés lors d'un appel d'API Amazon ReceiveMessage SQS
- Résoudre les erreurs réseau Amazon SQS
- Dépannage des files d'attente Amazon Simple Queue Service avec AWS X-Ray

Résoudre une erreur de refus d'accès dans Amazon SQS

Les rubriques suivantes présentent les causes `AccessDenied` ou les `AccessDeniedException` erreurs les plus courantes des appels d'API Amazon SQS. Pour plus d'informations sur la résolution de ces erreurs, consultez Comment résoudre les erreurs « » ou « AccessDenied » lors des AccessDeniedException appels d'API Amazon SQS ? dans le guide du centre de AWS connaissances.

Exemples de messages d'erreur :

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to
        the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

- ou -

```
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.
```

Politique de file d'attente Amazon SQS et politique IAM

Pour vérifier si le demandeur dispose des autorisations appropriées pour effectuer une opération Amazon SQS, procédez comme suit :

- Identifiez le principal IAM qui effectue l'appel d'API Amazon SQS. Si le principal IAM possède le même compte, la politique de file d'attente Amazon SQS ou AWS la politique Identity and Access Management (IAM) doivent inclure des autorisations permettant d'autoriser explicitement l'accès à l'action.
- Si le principal est une entité IAM :
  - Vous pouvez identifier votre utilisateur ou votre rôle IAM en cochant le coin supérieur droit du AWS Management Console ou en utilisant la commande. `aws sts get-caller-identity`
  - Vérifiez les politiques IAM attachées à l'utilisateur ou au rôle IAM. Vous pouvez choisir l'une des méthodes suivantes :
    - Testez les politiques IAM avec le simulateur de politiques IAM.
    - Passez en revue les différents types de politiques IAM.
  - Si nécessaire, modifiez votre politique d'utilisateur IAM.
  - Vérifiez la politique de file d'attente et modifiez-la si nécessaire.
- Si le principal est un AWS service, la politique de file d'attente Amazon SQS doit autoriser explicitement l'accès.
- Si le principal est un principal multicompte, la politique de file d'attente Amazon SQS et la politique IAM doivent autoriser explicitement l'accès.
- Si la politique utilise un élément de condition, vérifiez que la condition restreint l'accès.

> **Important**
>
> Un refus explicite dans l'une ou l'autre des politiques remplace une autorisation explicite. Voici quelques exemples de base des politiques Amazon SQS.

AWS Key Management Service autorisations

Si le chiffrement côté serveur (SSE) de votre file d'attente Amazon SQS est activé et qu'un client est géré AWS KMS key, les autorisations doivent être accordées à la fois aux producteurs et aux consommateurs. Pour vérifier si une file d'attente est chiffrée, vous pouvez utiliser l'`KmsMasterKeyId`attribut `GetQueueAttributes`API ou depuis la console de file d'attente sous Chiffrement.

- Autorisations requises pour les producteurs :

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

- Autorisations requises pour les consommateurs :

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- Autorisations requises pour l'accès entre comptes :

```
{
"Effect": "Allow",
"Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

Choisissez l'une des options suivantes pour activer le chiffrement d'une file d'attente Amazon SQS :

- SSE-Amazon SQS (clé de chiffrement créée et gérée par le service Amazon SQS.)
- AWS clé par défaut gérée (alias/aws/sqs)
- Clé gérée par le client

Toutefois, si vous utilisez une clé KMS AWS gérée, vous ne pouvez pas modifier la politique de clé par défaut. Par conséquent, pour donner accès à d'autres services et à des comptes croisés, utilisez une clé gérée par le client. Cela vous permet de modifier la politique clé.

Politique de point de terminaison d'un VPC

Si vous accédez à Amazon SQS via un point de terminaison Amazon Virtual Private Cloud (Amazon VPC), la politique de point de terminaison Amazon SQS VPC doit autoriser l'accès. Vous pouvez créer une politique pour les points de terminaison Amazon VPC pour Amazon SQS, dans laquelle vous pouvez spécifier les éléments suivants :

1. Le principal qui peut exécuter des actions.
2. Les actions qui peuvent être effectuées.
3. Les ressources sur lesquelles les actions peuvent être exécutées.

Dans l'exemple suivant, la politique de point de terminaison VPC indique que l'utilisateur IAM *MyUser* est autorisé à envoyer des messages à la file d'attente Amazon SQS. *MyQueue* L'accès aux autres actions, aux utilisateurs IAM et aux ressources Amazon SQS est refusé via le point de terminaison VPC.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

Politique de contrôle des services de l'organisation

Si vous Compte AWS appartenez à une organisation, AWS Organizations les politiques peuvent vous empêcher d'accéder à vos files d'attente Amazon SQS. Par défaut, AWS Organizations les politiques ne bloquent aucune demande adressée à Amazon SQS. Assurez-vous toutefois que vos AWS Organizations politiques n'ont pas été configurées pour bloquer l'accès aux files d'attente Amazon SQS. Pour savoir comment vérifier vos AWS Organizations politiques, consultez la section Liste de toutes les politiques dans le guide de AWS Organizations l'utilisateur.

Résoudre les erreurs d'API Amazon SQS

Les rubriques suivantes présentent les erreurs les plus courantes renvoyées lors des appels d'API Amazon SQS et expliquent comment les résoudre.

QueueDoesNotExist erreur

Cette erreur sera renvoyée lorsque le service Amazon SQS ne trouvera pas la file d'attente mentionnée pour l'action Amazon SQS.

Causes possibles et mesures d'atténuation :

- Région incorrecte : vérifiez la configuration du client Amazon SQS pour vérifier que vous avez configuré la bonne région sur le client. Lorsque vous ne configurez pas de région sur le client, le SDK AWS CLI choisit la région dans le fichier de configuration ou dans la variable d'environnement. Si le SDK ne trouve aucune région dans le fichier de configuration, il définit la région sur us-east-1 par défaut.
- La file d'attente a peut-être été supprimée récemment : si la file d'attente a été supprimée avant que l'appel d'API ne soit effectué, l'appel d'API renverra cette erreur. Vérifiez CloudTrail les DeleteQueueopérations effectuées avant le moment où l'erreur s'est produite.
- Problèmes d'autorisation : si l'utilisateur ou le rôle demandeur AWS Identity and Access Management (IAM) ne dispose pas des autorisations requises, le message d'erreur suivant peut s'afficher :

  ```
  The specified queue does not exist or you do not have access to it.
  ```

  Vérifiez les autorisations et effectuez l'appel d'API avec les autorisations correctes.

Pour plus de détails sur la résolution de l'`QueueDoesNotExist`erreur, consultez Comment résoudre l' QueueDoesNotExist erreur lorsque je passe des appels d'API à ma file d'attente Amazon SQS ? dans le guide du centre de AWS connaissances.

InvalidAttributeValue erreur

Cette erreur sera renvoyée lors de la mise à jour de la politique de ressources de file d'attente Amazon SQS ou des propriétés avec une politique ou un principal incorrect.

Causes possibles et mesures d'atténuation :

- Politique de ressources non valide : vérifiez que la politique de ressources contient tous les champs obligatoires. Pour plus d'informations, consultez les sections Référence aux éléments de stratégie IAM JSON et Validation des politiques IAM. Vous pouvez également utiliser le générateur de politiques IAM pour créer et tester une politique de ressources Amazon SQS. Assurez-vous que la politique est au format JSON.
- Principal non valide : assurez-vous que l'`Principal`élément existe dans la politique de ressources et que la valeur est valide. Si l'`Principal`élément de votre politique de ressources Amazon SQS inclut une entité IAM, assurez-vous que l'entité existe avant d'utiliser la politique. Amazon SQS valide la politique de ressources et vérifie l'entité IAM. Si l'entité IAM n'existe pas, vous recevrez un message d'erreur. Pour confirmer les entités IAM, utilisez le `GetRole`et `GetUser` APIs.

Pour plus d'informations sur la résolution d'une `InvalidAttributeValue` erreur, consultez Comment résoudre l' QueueDoesNotExist erreur lorsque je passe des appels d'API à ma file d'attente Amazon SQS ? dans le guide du centre de AWS connaissances.

ReceiptHandle erreur

Lors d'un appel d'`DeleteMessage`API, l'erreur `ReceiptHandleIsInvalid` `InvalidParameterValue` peut être renvoyée si le descripteur de réception est incorrect ou a expiré.

- ReceiptHandleIsInvalid erreur : si le descripteur du reçu est incorrect, vous recevrez un message d'erreur similaire à cet exemple :

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
 The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- InvalidParameterValue erreur : si le descripteur du reçu est expiré, vous recevrez un message d'erreur similaire à cet exemple :

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:
 Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The
 receipt handle has expired.
```

Causes possibles et mesures d'atténuation :

L'identifiant de réception est créé pour chaque message reçu et n'est valide que pendant le délai de visibilité. Lorsque le délai de visibilité expire, le message devient visible dans la file d'attente pour les consommateurs. Lorsque vous recevez à nouveau le message du consommateur, vous recevez un nouvel identifiant de réception. Pour éviter les erreurs de traitement de réception incorrectes ou expirées, utilisez le bon identifiant de réception pour supprimer le message pendant le délai de visibilité de la file d'attente Amazon SQS.

Pour plus d'informations sur la résolution `ReceiptHandle` d'une erreur, consultez Comment résoudre les erreurs « » et ReceiptHandleIsInvalid « » lorsque j'utilise l'appel d'API Amazon DeleteMessage SQS ? InvalidParameterValue dans le guide du centre de AWS connaissances.

Résoudre les problèmes liés à la file d'attente des lettres mortes Amazon SQS et au redrive DLQ

Les rubriques suivantes présentent les causes les plus courantes des problèmes liés à Amazon SQS DLQ et DLQ redrive, et expliquent comment les résoudre. Pour plus d'informations, consultez Comment résoudre les problèmes liés au redrive Amazon SQS DLQ ? dans le guide du centre de AWS connaissances.

Problèmes liés au DLQ

Découvrez les problèmes courants liés à la DLQ et découvrez comment les résoudre.

L'affichage des messages avec la console peut entraîner leur envoi dans une file d'attente de lettres mortes

Amazon SQS affiche un message dans la console selon la stratégie de redirection de la file d'attente correspondante. Par conséquent, si vous consultez un message dans la console le nombre de fois

spécifié dans la politique de redrive de la file d'attente correspondante, le message est déplacé vers la file d'attente contenant des lettres mortes de la file d'attente correspondante.

Pour régler ce comportement, vous pouvez procéder de l'une des manières suivantes :

- Augmentez le paramètre Maximum Receives pour la stratégie de redirection de la file d'attente correspondante.
- N'affichez pas les messages de la file d'attente correspondante dans la console.

Les métriques **`NumberOfMessagesSent`** et **`NumberOfMessagesReceived`** d'une file d'attente de lettres mortes ne correspondent pas

Si vous envoyez un message à une file d'attente de lettres mortes manuellement, il est capturé par la métrique `NumberOfMessagesSent`. Toutefois, si un message est envoyé à une file d'attente de lettres mortes en raison de l'échec d'une tentative de traitement, il n'est pas capturé par cette métrique. Il est donc possible que les valeurs de `NumberOfMessagesSent` et `NumberOfMessagesReceived`soient différentes.

Création et configuration d'un redrive dans une file d'attente contenant des lettres mortes

Le redrive de la file d'attente de lettres mortes nécessite que vous définissiez les autorisations appropriées pour qu'Amazon SQS puisse recevoir des messages de la file d'attente de lettres mortes et envoyer des messages à la file d'attente de destination. Si vous ne disposez pas des autorisations appropriées, la tâche de redynamisation de la file d'attente contenant des lettres mortes peut échouer. Vous pouvez consulter l'état de votre tâche de retransmission des messages pour résoudre les problèmes, puis réessayer.

Gestion des défaillances des messages de file d'attente standard et FIFO

Les files d'attente standard continuent de traiter les messages jusqu'à l'expiration de la période de conservation. Ce traitement continu réduit les risques de blocage de la file d'attente par des messages non consommés. Le fait d'avoir un grand nombre de messages que le consommateur ne parvient pas à supprimer à plusieurs reprises peut augmenter les coûts et alourdir la charge matérielle. Pour réduire les coûts, déplacez les messages ayant échoué vers la file d'attente des lettres mortes.

Les files d'attente standard autorisent également un grand nombre de messages en vol. Si la majorité de vos messages ne peuvent pas être consommés et ne sont pas envoyés dans une file d'attente de

lettres mortes, votre taux de traitement des messages peut ralentir. Pour maintenir l'efficacité de votre file d'attente, assurez-vous que votre application gère correctement le traitement des messages.

Les files d'attente FIFO garantissent un traitement unique en consommant les messages d'un groupe de messages dans l'ordre. Par conséquent, bien que le consommateur puisse continuer à récupérer les messages commandés dans un autre groupe de messages, le premier groupe de messages reste indisponible tant que le message bloquant la file d'attente n'est pas traité avec succès ou n'est pas déplacé vers une file d'attente de lettres mortes.

De plus, les files d'attente FIFO permettent de réduire le nombre de messages en vol. Pour éviter que votre file d'attente FIFO ne soit bloquée par un message, assurez-vous que votre application gère correctement le traitement des messages.

Pour plus d'informations, consultez Quotas de messages Amazon SQS et Bonnes pratiques relatives à Amazon SQS.

Problèmes liés au DLQ-Redrive

Découvrez les problèmes courants liés au DLQ-Redrive et découvrez comment les résoudre.

AccessDenied problème d'autorisation

L'`AccessDenied`erreur se produit lorsque le redrive DLQ échoue parce que l'entité AWS Identity and Access Management (IAM) ne dispose pas des autorisations requises.

Exemple de message d'erreur :

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

Les autorisations d'API suivantes sont requises pour effectuer des demandes de redrive DLQ :

Pour démarrer le redrive d'un message, procédez comme suit :

- Autorisations relatives à la file d'attente des lettres mortes :
  - `sqs:StartMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt`— Lorsque la file d'attente de lettres mortes ou la file d'attente source d'origine sont cryptées.

- Autorisations de file d'attente de destination :
  - `sqs:SendMessage`
  - `kms:GenerateDataKey`— Lorsque la file d'attente de destination est cryptée.
  - `kms:Decrypt`— Lorsque la file d'attente de destination est cryptée.

Pour annuler le renvoi d'un message en cours :

- Autorisations relatives à la file d'attente des lettres mortes :
  - `sqs:CancelMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt`— Lorsque la file d'attente de lettres mortes ou la file d'attente source d'origine sont cryptées.

Pour afficher l'état du déplacement d'un message :

- Autorisations relatives à la file d'attente des lettres mortes :
  - `sqs:ListMessageMoveTasks`
  - `sqs:GetQueueAttributes`

`NonExistentQueue`Erreur

L'`NonExistentQueue`erreur se produit lorsque la file d'attente source Amazon SQS n'existe pas ou a été supprimée. Vérifiez et reconduisez vers une file d'attente Amazon SQS présente.

Exemple de message d'erreur :

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSourceErreur

L'`CouldNotDetermineMessageSource`erreur se produit lorsque vous tentez de démarrer un redrive DLQ avec les scénarios suivants :

- Un message Amazon SQS envoyé directement au DLQ avec API. `SendMessage`

- Message issu de la rubrique AWS Lambda ou de la fonction Amazon Simple Notification Service (Amazon SNS) avec le DLQ configuré.

Pour résoudre cette erreur, choisissez Redrive vers une destination personnalisée lorsque vous démarrez le redrive. Entrez ensuite l'ARN de la file d'attente Amazon SQS pour déplacer tous les messages de la DLQ vers la file d'attente de destination.

Exemple de message d'erreur :

```
Failed: CouldNotDetermineMessageSource
```

Résoudre les problèmes de régulation FIFO dans Amazon SQS

Par défaut, les files d'attente FIFO prennent en charge 300 transactions par seconde, par action d'API pour `SendMessageReceiveMessage`, et. `DeleteMessage` Les requêtes supérieures à 300 TPS génèrent l'`ThrottlingException`erreur même si les messages de la file d'attente sont disponibles. Pour pallier ce problème, vous pouvez utiliser les méthodes suivantes :

- Permettre un débit élevé pour les files d'attente FIFO dans Amazon SQS.
- Utilisez les actions `SendMessageBatch` par lots de l'API Amazon SQS et `ChangeMessageVisibilityBatch` pour augmenter la limite TPS de 3 000 messages par seconde et par action d'API, et pour réduire les coûts. `DeleteMessageBatch` Pour l'`ReceiveMessage`API, définissez le `MaxNumberofMessages` paramètre pour recevoir jusqu'à dix messages par transaction. Pour de plus amples informations, veuillez consulter Actions groupées Amazon SQS.
- Pour les files d'attente FIFO à haut débit, suivez les recommandations pour optimiser l'utilisation des partitions. Envoyez des messages par lots avec le même groupe IDs de messages. Supprimez des messages ou modifiez les valeurs du délai d'expiration de visibilité des messages par lots avec des descripteurs de réception provenant des mêmes demandes d'`ReceiveMessage`API.
- Augmentez le nombre de `MessageGroupId`valeurs uniques. Cela permet une distribution uniforme entre les partitions de file d'attente FIFO. Pour plus d'informations, consultez Utilisation de l'ID de groupe de messages Amazon SQS.

Pour plus d'informations, consultez Pourquoi ma file d'attente FIFO Amazon SQS ne renvoie pas tous les messages ou les messages des autres groupes de messages ? dans le guide du centre de AWS connaissances.

Résoudre les problèmes liés aux messages non renvoyés lors d'un appel d'API Amazon ReceiveMessage SQS

Les rubriques suivantes décrivent les causes les plus courantes pour lesquelles un message Amazon SQS peut ne pas être renvoyé aux clients, ainsi que la manière de les résoudre. Pour plus d'informations, consultez Pourquoi ne puis-je pas recevoir de messages depuis ma file d'attente Amazon SQS ? dans le guide du centre de AWS connaissances.

File d'attente vide

Pour déterminer si une file d'attente est vide, utilisez une longue interrogation pour appeler l'`ReceiveMessage`API. Vous pouvez également utiliser les `ApproximateNumberOfMessagesDelayed` CloudWatch métriques `ApproximateNumberOfMessagesVisibleApproximateNumberOfMessagesNotVisible`, et. Si toutes les valeurs métriques sont définies sur 0 pendant plusieurs minutes, la file d'attente est considérée comme vide.

Limite en vol atteinte

Si vous utilisez un long sondage et si la limite de vol de la file d'attente (20 000 pour le FIFO, 120 000 pour le standard par défaut) est dépassée, Amazon SQS ne renverra pas de messages d'erreur dépassant les limites de quota.

Retard du message

Si la file d'attente Amazon SQS est configurée comme une file d'attente différée, ou si les messages ont été envoyés avec des temporisateurs, les messages ne sont pas visibles avant la fin du délai. Pour vérifier si une file d'attente est configurée comme file d'attente différée, utilisez l'`DelaySecondsattribut` `GetQueueAttributes`API ou depuis la console de file d'attente sous Délai de livraison. Vérifiez la `ApproximateNumberOfMessagesDelayed` CloudWatch métrique pour savoir si des messages sont retardés.

Le message est en cours

Si un autre consommateur a interrogé le message, celui-ci sera diffusé ou invisible pendant la période d'expiration du délai de visibilité. Les sondages supplémentaires peuvent renvoyer un reçu vide. Vérifiez la ApproximateNumberOfMessagesVisible CloudWatchmétrique pour connaître le nombre de messages pouvant être reçus. Dans le cas des files d'attente FIFO, si un message portant

l'ID du groupe de messages est en cours de traitement, aucun autre message ne sera renvoyé, sauf si vous le supprimez ou s'il devient visible. Cela est dû au fait que l'ordre des messages est maintenu au niveau du groupe de messages dans une file d'attente FIFO.

Méthode de sondage

Si vous utilisez un sondage court, (WaitTimeSecondsest 0) Amazon SQS échantillonne un sous-ensemble de ses serveurs et renvoie des messages provenant uniquement de ces serveurs. Par conséquent, il est possible que vous ne receviez pas les messages même s'ils sont disponibles pour être reçus. Les demandes de sondage suivantes renverront les messages.

Si vous utilisez un long sondage, Amazon SQS interroge tous les serveurs et envoie une réponse après avoir collecté au moins un message disponible, et jusqu'au nombre maximum spécifié. Si la valeur de ReceiveMessage WaitTimeSecondsest trop faible, il est possible que vous ne receviez pas tous les messages disponibles.

Résoudre les erreurs réseau Amazon SQS

Les rubriques suivantes décrivent les causes les plus courantes des problèmes de réseau dans Amazon SQS et expliquent comment les résoudre.

ETIMEOUT error

L'ETIMEOUTerreur se produit lorsque le client ne parvient pas à établir une connexion TCP avec un point de terminaison Amazon SQS.

Résolution de problèmes

- Vérifiez la connexion réseau

  Testez votre connexion réseau à Amazon SQS en exécutant des commandes telles que. `telnet`

  Example: telnet sqs.us-east-1.amazonaws.com 443
- Vérifiez les paramètres réseau
  - Assurez-vous que les règles de votre pare-feu local, les itinéraires et les listes de contrôle d'accès (ACLs) autorisent le trafic sur le port que vous utilisez.
  - Les règles de sortie (sortie) du groupe de sécurité doivent autoriser le trafic vers le port 80 ou 443.
  - Les règles de sortie (sortie) ACL du réseau doivent autoriser le trafic vers le port TCP 80 ou 443.

- Les règles d'entrée (ACL) du réseau doivent autoriser le trafic sur les ports TCP 1024 à 65535.
- Les instances Amazon Elastic Compute Cloud (Amazon EC2) qui se connectent à l'Internet public doivent disposer d'une connexion Internet.

- Points de terminaison Amazon Virtual Private Cloud (Amazon VPC)

  Si vous accédez à Amazon SQS via un point de terminaison Amazon VPC, le groupe de sécurité des points de terminaison doit autoriser le trafic entrant vers le groupe de sécurité du client sur le port 443. L'ACL réseau associée au sous-réseau du point de terminaison VPC doit avoir cette configuration :

  - Les règles de sortie (sortie) ACL du réseau doivent autoriser le trafic sur les ports TCP 1024 à 65535 (ports éphémères).
  - Les règles d'entrée (ACL) du réseau doivent autoriser le trafic sur le port 443.

En outre, la politique de point de terminaison VPC AWS Identity and Access Management (IAM) d'Amazon SQS doit autoriser l'accès. L'exemple de politique de point de terminaison VPC suivant indique que l'utilisateur IAM *MyUser* est autorisé à envoyer des messages à la file d'attente Amazon SQS. *MyQueue* L'accès aux autres actions, aux utilisateurs IAM et aux ressources Amazon SQS est refusé via le point de terminaison VPC.

```
{
    "Statement": [{
        "Action": ["sqs:SendMessage"],
        "Effect": "Allow",
        "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
        "Principal": {
            "AWS": "arn:aws:iam:123456789012:user/MyUser"
        }
    }]
}
```

UnknownHostException error

L'UnknownHostExceptionerreur se produit lorsque l'adresse IP de l'hôte n'a pas pu être déterminée.

Résolution de problèmes

Utilisez l'nslookuputilitaire pour renvoyer l'adresse IP associée au nom d'hôte :

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI ou SDK pour les anciens points de terminaison Python :

```
nslookup <region>.queue.amazonaws.com
```

Si vous recevez un résultat infructueux, suivez les instructions de la section Comment fonctionne le DNS et comment résoudre les défaillances partielles ou intermittentes du DNS ? dans le guide du centre de AWS connaissances.

Si vous avez reçu une sortie valide, il s'agit probablement d'un problème au niveau de l'application. Pour résoudre les problèmes au niveau de l'application, essayez les méthodes suivantes :

- Redémarrez votre application.
- Vérifiez que le cache DNS de votre application Java n'est pas défectueux. Si possible, configurez votre application pour qu'elle adhère au TTL DNS. Pour plus d'informations, voir Configuration du TTL de la JVM pour les recherches de noms DNS.

Pour plus d'informations sur la résolution des erreurs réseau, consultez Comment résoudre les erreurs de connexion « ETIMEOUT » et « » d'Amazon SQS ? UnknownHostException dans le guide du centre de AWS connaissances.

Dépannage des files d'attente Amazon Simple Queue Service avec AWS X-Ray

AWS X-Ray collecte des données sur les demandes traitées par votre application et vous permet de visualiser et de filtrer les données afin d'identifier les problèmes potentiels et les opportunités d'optimisation. Pour toute demande retracée envoyée à votre application, vous pouvez consulter des informations détaillées sur la demande, la réponse et les appels que votre application fait aux AWS ressources en aval, aux microservices, aux bases de données et au Web APIs HTTP.

Pour envoyer des en-têtes de AWS X-Ray trace via Amazon SQS, vous pouvez effectuer l'une des opérations suivantes :

- Utilisez l'en-tête de suivi `X-Amzn-Trace-Id`.

- Utilisez l'attribut de système de message `AWSTraceHeader`.

Pour collecter des données sur les erreurs et la latence, vous devez instrumenter le client `AmazonSQS` à l'aide du kit SDK AWS X-Ray.

Vous pouvez utiliser la AWS X-Ray console pour consulter la carte des connexions entre Amazon SQS et les autres services utilisés par votre application. Vous pouvez également utiliser la console pour afficher des mesures comme la latence moyenne et les taux de défaillance. Pour plus d'informations, consultez Amazon SQS et AWS X-Ray dans le Guide du développeur AWS X-Ray .

Sécurité dans Amazon SQS

Cette section fournit des informations sur la sécurité, l'authentification et le contrôle d'accès dans Amazon SQS, ainsi que sur le langage de la politique d'accès Amazon SQS.

Rubriques

- Protection des données dans Amazon SQS
- Gestion des identités et des accès dans Amazon SQS
- Journalisation et surveillance dans Amazon SQS
- Validation de conformité pour Amazon SQS
- Résilience dans Amazon SQS
- Sécurité de l'infrastructure dans Amazon SQS
- Bonnes pratiques de sécurité pour Amazon SQS

Protection des données dans Amazon SQS

Le modèle de responsabilité AWS partagée s'applique à la protection des données dans Amazon Simple Queue Service. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d’informations sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée AWS et RGPD (Règlement général sur la protection des données) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l’authentification multifactorielle (MFA) avec chaque compte.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.

- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section Utilisation des CloudTrail sentiers dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez Norme FIPS (Federal Information Processing Standard) 140-3.

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec Amazon SQS ou une autre entreprise à Services AWS l'aide de la console, de l'API ou. AWS CLI AWS SDKs Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Les sections suivantes présentent des informations sur la protection des données dans Amazon SQS.

Chiffrement des données dans Amazon SQS

La protection des données fait référence au fait de protéger les données pendant leur transit (lorsqu'elles sont transmises en direction ou en provenance d'Amazon SQS) et au repos (lorsqu'elles sont stockées sur des disques dans les centres de données Amazon SQS). Vous pouvez protéger les données en transit à l'aide de la technologie Secure Sockets Layer (SSL) ou du chiffrement côté client. Par défaut, Amazon SQS stocke les messages et les fichiers à l'aide du chiffrement du disque. Vous pouvez protéger les données inactives en demandant à Amazon SQS de chiffrer vos messages avant de les enregistrer dans le système de fichiers chiffré de ses centres de données. Amazon SQS recommande d'utiliser SSE pour optimiser le chiffrement des données.

Rubriques

- Chiffrement au repos dans Amazon SQS

- Gestion des clés Amazon SQS

Chiffrement au repos dans Amazon SQS

Le chiffrement côté serveur (SSE, Server-Side Encryption) vous permet de transférer des données sensibles dans des files d'attente chiffrées. SSE protège le contenu des messages dans les files d'attente à l'aide de clés de chiffrement gérées par SQS (SSE-SQS) ou de clés gérées dans le (SSE-KMS). AWS Key Management Service Pour plus d'informations sur la gestion de l'ESS à l'aide du AWS Management Console, consultez les rubriques suivantes :

- Configuration de SSE-SQS pour une file d'attente (console)
- Configuration de SSE-KMS pour une file d'attente (console)

Pour plus d'informations sur la gestion de l'ESS à l'aide des `GetQueueAttributes` actions AWS SDK pour Java (et des`CreateQueue`, `SetQueueAttributes` et), consultez les exemples suivants :

- Utilisation du chiffrement côté serveur avec les files d'attente Amazon SQS
- Configuration des autorisations KMS pour Services AWS

SSE chiffre les messages une fois reçus par Amazon SQS. Les messages sont stockés sous forme chiffrée et Amazon SQS les déchiffre uniquement lorsqu'ils sont envoyés à un consommateur autorisé.

> **Important**
>
> Toutes les demandes adressées aux files d'attente avec le chiffrement SSE activé doivent utiliser HTTPS et Signature version 4.
> Une file d'attente chiffrée qui utilise la clé par défaut (clé KMS AWS gérée pour Amazon SQS) ne peut pas appeler de fonction Lambda dans un autre. Compte AWS
> Certaines fonctionnalités des AWS services qui peuvent envoyer des notifications à Amazon SQS à l'aide de cette AWS Security Token Service `AssumeRole` action sont compatibles avec SSE mais ne fonctionnent qu'avec les files d'attente standard :
>
> - Hooks de cycle de vie autoscaling
> - AWS Lambda Files d'attente de lettres mortes

Pour plus d'informations sur la compatibilité d'autres services avec les files d'attente chiffrées, consultez Configuration des autorisations KMS pour les AWS services et la documentation de votre service.

AWS KMS combine du matériel et des logiciels sécurisés et hautement disponibles pour fournir un système de gestion des clés adapté au cloud. Lorsque vous utilisez Amazon SQS avec AWS KMS, les clés de données qui chiffrent les données de vos messages sont également chiffrées et stockées avec les données qu'elles protègent.

L'utilisation d' AWS KMS offre les avantages suivants :

- Vous pouvez créer et gérer des AWS KMS keys vous-même.
- Vous pouvez également utiliser la clé KMS AWS gérée pour Amazon SQS, qui est unique pour chaque compte et chaque région.
- Les normes AWS KMS de sécurité peuvent vous aider à respecter les exigences de conformité liées au chiffrement.

Pour plus d'informations, veuillez consulter Présentation de AWS Key Management Service dans le Manuel du développeur AWS Key Management Service .

Portée du chiffrement

Le chiffrement SSE chiffre le corps d'un message dans une file d'attente Amazon SQS.

Le chiffrement SSE ne chiffre pas les éléments suivants :

- métadonnées de la file d'attente (nom et attributs)
- métadonnées du message (ID de message, horodatage et attributs)
- métriques par file d'attente

Le chiffrement d'un message rend son contenu indisponible pour les utilisateurs non autorisés ou anonymes. Lorsque SSE est activé, les demandes anonymes `SendMessage` et `ReceiveMessage` adressées à la file d'attente chiffrée sont rejetées. Les bonnes pratiques de sécurité d'Amazon SQS déconseillent l'utilisation de demandes anonymes. Si vous souhaitez envoyer des demandes

anonymes à une file d'attente Amazon SQS, assurez-vous de désactiver SSE. Cela n'affecte pas le fonctionnement normal d'Amazon SQS :

- Un message est chiffré uniquement s'il est envoyé après l'activation du chiffrement d'une file d'attente. Amazon SQS ne chiffre pas les messages en backlog.
- Tout message chiffré reste chiffré même si le chiffrement de sa file d'attente est désactivé.

Le placement d'un message dans une file d'attente de lettres mortes n'affecte pas son chiffrement :

- Lorsqu'Amazon SQS transfère un message d'une file d'attente source chiffrée vers une file d'attente de lettres mortes non chiffrée, le message reste chiffré.
- Lorsqu'Amazon SQS transfère un message d'une file d'attente source non chiffrée vers une file d'attente de lettres mortes chiffrée, le message reste non chiffré.

Termes clés

Les termes clés suivants peuvent vous aider à mieux comprendre les fonctionnalités de chiffrement SSE. Pour des descriptions détaillées, consultez la Référence de l'API Amazon Simple Queue Service.

Clé de données

Clé (DEK) permettant de chiffrer le contenu des messages Amazon SQS.

Pour plus d'informations, consultez la section Clés de données du Guide du développeur AWS Key Management Service et du Guide du développeur AWS Encryption SDK .

Période de réutilisation des clés de données

Durée, en secondes, pendant laquelle Amazon SQS peut réutiliser une clé de données pour chiffrer ou déchiffrer des messages avant de réappeler. AWS KMS Entier en secondes, compris entre 60 secondes (1 minute) et 86 400 secondes (24 heures). La valeur par défaut est 300 (5 minutes). Pour de plus amples informations, veuillez consulter Présentation de la période de réutilisation des clés de données.

> **Note**
>
> Dans le cas peu probable où il serait impossible de l'atteindre AWS KMS, Amazon SQS continue d'utiliser la clé de données mise en cache jusqu'à ce qu'une connexion soit rétablie.

ID de clé KMS

Alias, alias ARN, ID de clé ou ARN de clé d'une clé KMS AWS gérée ou d'une clé KMS personnalisée, dans votre compte ou dans un autre compte. Bien que l'alias de la clé KMS AWS gérée pour Amazon SQS soit toujours le casa`alias/aws/sqs`, l'alias d'une clé KMS personnalisée peut, par exemple, être. `alias/MyAlias` Vous pouvez utiliser ces clés KMS pour protéger les messages des files d'attente Amazon SQS.

> **Note**
>
> Gardez à l'esprit les points suivants :
>
> - Si vous ne spécifiez pas de clé KMS personnalisée, Amazon SQS utilise la clé KMS AWS gérée pour Amazon SQS.
> - La première fois que vous utilisez le AWS Management Console pour spécifier la clé KMS AWS gérée pour Amazon SQS pour une file d'attente, la clé KMS AWS gérée est AWS KMS créée pour Amazon SQS.
> - Sinon, la première fois que vous utilisez l'`SendMessageBatch`action `SendMessage` ou sur une file d'attente avec SSE activé, vous AWS KMS créez la clé KMS AWS gérée pour Amazon SQS.

Vous pouvez créer des clés KMS, définir les politiques qui contrôlent la manière dont les clés KMS peuvent être utilisées et auditer l'utilisation des clés KMS à l'aide de la section Clés gérées par le client de la AWS KMS console ou de l'`CreateKey` AWS KMS action. Pour plus d'informations, consultez Clés KMS et Création de clés du Guide du développeur AWS Key Management Service . Pour plus d'exemples d'identifiants de clé KMS, consultez KeyIdla référence AWS Key Management Service d'API. Pour plus d'informations sur la recherche d'identifiants de clés KMS, consultez la section Recherche de l'ID et de l'ARN d'une clé du Guide du développeur AWS Key Management Service .

> **Important**
>
> L'utilisation entraîne des frais supplémentaires AWS KMS. Pour plus d'informations, veuillez consulter les sections Estimation AWS KMS des coûts et Tarification d'AWS Key Management Service.

Chiffrement d'enveloppe

La sécurité de vos données chiffrées dépend partiellement de la protection de la clé de données capable de les déchiffrer. Amazon SQS utilise la clé KMS pour chiffrer la clé de données, puis la clé de données chiffrée est stockée avec le message chiffré. La pratique qui consiste à utiliser une clé KMS pour chiffrer des clés de données s'appelle le chiffrement d'enveloppe.

Pour plus d'informations, consultez Chiffrement d'enveloppe dans le Guide du développeur AWS Encryption SDK .

Gestion des clés Amazon SQS

Amazon SQS s'intègre au AWS Key Management Service (KMS) pour gérer les clés KMS pour le chiffrement côté serveur (SSE). Consultez Chiffrement au repos dans Amazon SQS pour obtenir des informations sur le SSE et les définitions de la gestion des clés. Amazon SQS utilise des clés KMS pour valider et sécuriser les clés de données qui chiffrent et déchiffrent les messages. Les sections suivantes fournissent des informations sur l'utilisation des clés KMS et de données dans le service Amazon SQS.

Configuration des autorisations AWS KMS

Chaque clé KMS doit avoir une politique de clé. Notez que vous ne pouvez pas modifier la politique de clé d'une clé KMS AWS gérée pour Amazon SQS. La stratégie de cette clé KMS inclut des autorisations permettant à tous les mandataires du compte (autorisés à utiliser Amazon SQS) d'utiliser des files d'attente chiffrées.

Amazon SQS fait la distinction entre les appelants en fonction de leurs AWS informations d'identification, qu'ils utilisent des AWS comptes, des utilisateurs IAM ou des rôles IAM différents. En outre, le même rôle IAM avec des politiques de cadrage différentes sera traité comme des demandeurs distincts. Toutefois, lorsque vous utilisez des sessions de rôle IAM, le simple fait de les modifier `RoleSessionName` tout en conservant le même rôle IAM et les mêmes politiques de portée

ne créera pas de demandeurs distincts. Par conséquent, lorsque vous spécifiez des principes de politique AWS KMS clés, évitez de vous fier uniquement aux `RoleSessionName` différences, car ces sessions seront traitées comme le même demandeur.

Vous pouvez également spécifier les autorisations requises dans une stratégie IAM affectée aux mandataires qui produisent et consomment des messages chiffrés. Pour de plus amples informations, veuillez consulter Utilisation des stratégies IAM avec AWS KMS dans le Guide du développeur AWS Key Management Service .

> **Note**
>
> Bien que vous puissiez configurer des autorisations globales pour envoyer et recevoir depuis Amazon SQS, vous devez nommer AWS KMS explicitement l'ARN complet des clés KMS dans des régions spécifiques dans la `Resource` section d'une politique IAM.

Configuration des autorisations KMS pour les AWS services

Plusieurs AWS services agissent comme des sources d'événements qui peuvent envoyer des événements aux files d'attente Amazon SQS. Pour permettre à ces sources d'événements de fonctionner avec des files d'attente chiffrées, vous devez créer une clé KMS gérée par le client et ajouter des autorisations dans la politique des clés afin que le service utilise les méthodes d' AWS KMS API requises. Effectuez les étapes suivantes pour configurer les autorisations.

> **Warning**
>
> Lorsque vous modifiez la clé KMS pour chiffrer vos messages Amazon SQS, sachez que les messages existants chiffrés avec l'ancienne clé KMS resteront chiffrés avec cette clé. Pour déchiffrer ces messages, vous devez conserver l'ancienne clé KMS et vous assurer que sa politique en matière de clés accorde à Amazon SQS les autorisations `kms:Decrypt` pour et. `kms:GenerateDataKey` Après la mise à jour vers une nouvelle clé KMS pour chiffrer les nouveaux messages, assurez-vous que tous les messages existants chiffrés avec l'ancienne clé KMS sont traités et retirés de la file d'attente avant de supprimer ou de désactiver l'ancienne clé KMS.

1. Créez une clé KMS gérée par le client. Pour plus d’informations, consultez Création des clés dans le Guide du développeur AWS Key Management Service .

2. Pour autoriser la source de l'événement de AWS service à utiliser les méthodes `kms:Decrypt` et `kms:GenerateDataKey` API, ajoutez l'instruction suivante à la politique de clé KMS.

   JSON

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": "service.amazonaws.com"
         },
         "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
         ],
         "Resource": "*"
       }]
}
```

   Remplacez « service » dans l'exemple ci-dessus par le nom de service de la source de l'événement. Les sources d'événements incluent les services suivants.

Source de l'événement	Nom du service
CloudWatch Événements Amazon	`events.amazonaws.com`
Notifications d'événements Amazon S3	`s3.amazonaws.com`
Abonnements à des rubriques Amazon SNS	`sns.amazonaws.com`

3. Configurez une file d'attente SSE existante à l'aide de l'ARN de votre clé KMS.
4. Fournissez l'ARN de la file d'attente chiffrée à la source de l'événement.

Configurer AWS KMS les autorisations pour les producteurs

Lorsque la période de réutilisation de la clé de données expire, le prochain appel du producteur vers `SendMessage` ou `SendMessageBatch` déclenche également des appels vers `kms:Decrypt` et

`kms:GenerateDataKey`. L'appel à `kms:Decrypt` a pour but de vérifier l'intégrité de la nouvelle clé de données avant de l'utiliser. Par conséquent, le producteur doit disposer des autorisations `kms:Decrypt` et `kms:GenerateDataKey` pour la clé KMS.

Ajoutez l'instruction suivante à la stratégie IAM du producteur. N'oubliez pas d'utiliser les valeurs ARN correctes pour la ressource clé et la ressource de file d'attente.

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"
        }
    ]
}
```

Configurer AWS KMS les autorisations pour les consommateurs

Lorsque la période de réutilisation de la clé de données expire, le prochain appel du consommateur vers `ReceiveMessage` déclenche également un appel de `kms:Decrypt`, pour vérifier l'intégrité de la nouvelle clé de données avant de l'utiliser. Par conséquent, le consommateur doit disposer de l'autorisation `kms:Decrypt` pour toute clé KMS permettant de chiffrer les messages dans la file d'attente spécifiée. Si la file d'attente sert de file d'attente de lettres mortes, le consommateur doit également disposer de l'autorisation `kms:Decrypt` pour toute clé KMS permettant de chiffrer les messages dans la file d'attente source. Ajoutez l'instruction suivante à la stratégie IAM du

consommateur. N'oubliez pas d'utiliser les valeurs ARN correctes pour la ressource clé et la ressource de file d'attente.

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"
        }
    ]
}
```

Configurer AWS KMS les autorisations avec une protection adjointe confuse

Lorsque le mandataire dans une instruction de stratégie de clé est un principal de service AWS, vous pouvez utiliser les clés de condition globales `aws:SourceArn` ou `aws:SourceAccount` pour vous protéger contre le scénario de député confus. Pour utiliser ces clés de condition, définissez comme valeur l'Amazon Resource Name (ARN) de la ressource à chiffrer. Si vous ne connaissez pas l'ARN de la ressource, utilisez `aws:SourceAccount` à la place.

Dans cette stratégie de clé KMS, une ressource spécifique issue d'un service appartenant au compte `111122223333` est autorisée à appeler KMS pour les actions `Decrypt` et `GenerateDataKey`, qui se produisent lors de l'utilisation du SSE d'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"
                    ]
                }
            }
        }
    ]
}
```

Lorsque vous utilisez des files d'attente Amazon SQS compatibles SSE, les services suivants prennent en charge `aws:SourceArn` :

- Amazon SNS
- Amazon S3
- CloudWatch Événements
- AWS Lambda
- CodeBuild
- Profils des clients Amazon Connect
- AWS Auto Scaling
- Amazon Chime

Présentation de la période de réutilisation des clés de données

La période de réutilisation des clés de données définit la durée maximale de réutilisation de la même clé de données par Amazon SQS. Lorsque la période de réutilisation de la clé de données se termine, Amazon SQS génère une nouvelle clé de données. Notez les instructions suivantes concernant la période de réutilisation.

- Une période de réutilisation plus courte améliore la sécurité, mais entraîne un plus grand nombre d'appels AWS KMS, ce qui peut entraîner des frais au-delà du niveau gratuit.
- Bien que la clé de données soit mise en cache séparé pour le chiffrement et le déchiffrement, la période de réutilisation s'applique aux deux copies de cette clé.
- Lorsque la période de réutilisation des clés de données prend fin, le prochain appel `SendMessage` ou déclenche `SendMessageBatch` généralement un appel à la AWS KMS `GenerateDataKey` méthode pour obtenir une nouvelle clé de données. De plus, les prochains appels à `SendMessage` et `ReceiveMessage` déclencheront chacun un appel AWS KMS `Decrypt` pour vérifier l'intégrité de la clé de données avant de l'utiliser.
- Les principaux (Comptes AWS ou utilisateurs) ne partagent pas les clés de données (les messages envoyés par des principaux uniques reçoivent toujours des clés de données uniques). Par conséquent, le volume d'appels AWS KMS est un multiple du nombre de principaux uniques utilisés pendant la période de réutilisation des clés de données.

Estimation AWS KMS des coûts

Pour prévoir les coûts et mieux comprendre votre AWS facture, vous souhaiterez peut-être savoir à quelle fréquence Amazon SQS utilise votre clé KMS.

> **Note**
>
> Bien que la formule ci-après puisse vous donner une très bonne idée des coûts à prévoir, les coûts réels risquent d'être plus élevés en raison de la nature distribuée d'Amazon SQS.

Pour calculer le nombre de demandes d'API (R) par file d'attente, utilisez la formule suivante :

```
R = (B / D) * (2 * P + C)
```

B est la période de facturation (en secondes).

D est la période de réutilisation des clés de données (en secondes).

P est le nombre de mandataires productifs qui envoient des messages à la file d'attente Amazon SQS.

C est le nombre de mandataires consommateurs qui reçoivent des messages de la file d'attente Amazon SQS.

> **Important**
>
> En général, les mandataires productifs ont un coût deux fois plus élevé que celui des mandataires consommateurs. Pour de plus amples informations, veuillez consulter Présentation de la période de réutilisation des clés de données.
> Si le producteur et le consommateur ont des utilisateurs différents, le coût augmente.

Voici des exemples de calcul. Pour obtenir des informations précises sur la tarification, consultez Tarification AWS Key Management Service.

Exemple 1 : calcul du nombre d'appels d' AWS KMS API pour 2 principaux et 1 file d'attente

Cet exemple suppose que :

- La période de facturation va du 1er au 31 janvier (2 678 400 secondes).
- La période de réutilisation des clés de données est définie sur 5 minutes (300 secondes).
- Il y a 1 file d'attente.
- Il y a 1 mandataire productif et 1 mandataire consommateur.

```
(2,678,400 / 300) * (2 * 1 + 1) = 26,784
```

Exemple 2 : calcul du nombre d'appels d' AWS KMS API pour plusieurs producteurs et consommateurs et pour deux files d'attente

Cet exemple suppose que :

- La période de facturation va du 1er au 28 février (2 419 200 secondes).
- La période de réutilisation des clés de données est définie sur 24 heures (86 400 secondes).

- Il y a 2 files d'attente.
- La première file d'attente comporte 3 mandataires productifs et 1 mandataire consommateur.
- La seconde file d'attente comporte 5 mandataires productifs et 2 mandataires consommateurs.

```
(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532
```

AWS KMS erreurs

Lorsque vous travaillez avec Amazon SQS AWS KMS, vous pouvez rencontrer des erreurs. Les références suivantes décrivent les erreurs et les solutions de dépannage possibles.

- Erreurs AWS KMS courantes
- Erreurs de déchiffrement AWS KMS
- AWS KMS GenerateDataKey erreurs

Confidentialité du trafic interréseau dans Amazon SQS

Un point de terminaison d'Amazon Virtual Private Cloud (Amazon VPC) pour Amazon SQS est une entité logique au sein d'un VPC qui autorise la connectivité uniquement à Amazon SQS. Le VPC achemine les demandes vers Amazon SQS et les réponses en retour vers le VPC. Les sections suivantes fournissent des informations sur l'utilisation des points de terminaison de VPC et la création de politiques de point de terminaison de VPC.

Points de terminaison Amazon Virtual Private Cloud pour Amazon SQS

Si vous utilisez Amazon VPC pour héberger vos AWS ressources, vous pouvez établir une connexion entre votre VPC et Amazon SQS. Vous pouvez utiliser cette connexion pour envoyer des messages à vos files d'attente Amazon SQS sans passer par l'Internet public.

Amazon VPC vous permet de lancer AWS des ressources dans un réseau virtuel personnalisé. Vous pouvez utiliser un VPC pour contrôler vos paramètres réseau, tels que la plage d'adresses IP, les sous-réseaux, les tables de routage et les passerelles réseau. Pour plus d'informations VPCs, consultez le guide de l'utilisateur Amazon VPC.

Pour connecter votre VPC à Amazon SQS, vous devez commencer par définir le point de terminaison d'un VPC d'interface, qui vous permet de connecter votre VPC à d'autres services AWS . Le point

de terminaison assure une connectivité évolutive et fiable à Amazon SQS, sans qu'une passerelle Internet, une instance de traduction d'adresses réseau (NAT) ou une connexion VPN ne soit nécessaire. Pour plus d'informations, consultez les sections Didacticiel : envoi d'un message à une file d'attente Amazon SQS à partir d'Amazon Virtual Private Cloud et Exemple 5 : Refuser l'accès s'il n'émane pas d'un point de terminaison de VPC de ce guide et la section Points de terminaison d'un VPC d'interface (AWS PrivateLink) du Guide de l'utilisateur Amazon VPC.

> **Important**
>
> - Vous pouvez utiliser Amazon Virtual Private Cloud uniquement avec des points de terminaison HTTPS Amazon SQS.
> - Lorsque vous configurez Amazon SQS pour envoyer des messages depuis Amazon VPC, vous devez activer le DNS privé et spécifier les points de terminaison au format `sqs.`*`us-east-2`*`.amazonaws.com` ou `sqs.`*`us-east-2`*`.api.aws` pour le point de terminaison à double pile.
> - Amazon SQS prend également en charge les points de terminaison FIPS PrivateLink via le service de point de terminaison. `com.amazonaws.region.sqs-fips` Vous pouvez vous connecter aux points de terminaison FIPS au format suivant. `sqs-fips.region.amazonaws.com`
> - Lorsque vous utilisez le point de terminaison à double pile dans Amazon Virtual Private Cloud, les demandes seront envoyées à l'aide de IPv4 et IPv6.
> - Le DNS privé ne prend pas en charge les points de terminaison existants, tels que `queue.amazonaws.com` ou *`us-east-2`*`.queue.amazonaws.com`.

Création d'une stratégie de point de terminaison d'un VPC Amazon pour Amazon SQS

Vous pouvez créer une stratégie pour les points de terminaison d'un VPC Amazon pour Amazon SQS dans laquelle vous spécifiez les éléments suivants :

- Le principal qui peut exécuter des actions.
- Les actions qui peuvent être effectuées.
- Les ressources sur lesquelles les actions peuvent être exécutées.

Pour en savoir plus, consultez Contrôle de l'accès aux services avec des points de terminaison d'un VPC dans le Guide de l'utilisateur Amazon VPC.

L'exemple suivant de stratégie de point de terminaison d'un VPC spécifie que l'utilisateur `MyUser` est autorisé à envoyer des messages dans la file d'attente Amazon SQS `MyQueue`.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

Ce qui suit est refusé :

- Autres actions d'API Amazon SQS, telles que `sqs:CreateQueue` et `sqs:DeleteQueue`.
- Autres utilisateurs et règles qui tentent d'utiliser ce point de terminaison de VPC.
- Envoi de messages par `MyUser` à une autre file d'attente Amazon SQS.

> **Note**
>
> L'utilisateur peut toujours utiliser d'autres actions d'API Amazon SQS depuis l'extérieur du VPC. Pour de plus amples informations, veuillez consulter Exemple 5 : Refuser l'accès s'il n'émane pas d'un point de terminaison de VPC.

Connectez-vous à Amazon SQS à l'aide de points de terminaison à double pile (IPv4 et) IPv6

Les terminaux à double pile prennent en charge à la fois le trafic IPv4 et IPv6 le trafic. Lorsque vous envoyez une demande à un point de terminaison à double pile, l'URL du point de terminaison est une adresse IPv4 ou une IPv6 adresse. Pour plus d'informations sur les points de terminaison à double pile et FIPS, consultez le guide de référence du SDK.

Amazon SQS prend en charge les points de terminaison régionaux à double pile, ce qui signifie que vous devez spécifier la AWS région dans le nom du point de terminaison. Les noms de point de terminaison à double pile utilisent la convention de dénomination

suivante :`sqs.Region.amazonaws.com`. Par exemple, le nom du point de terminaison à double pile de la région `eu-west-1` est `sqs.eu-west-1.amazonaws.com`.

Pour obtenir la liste complète des points de terminaison Amazon SQS, consultez la AWS référence générale.

Gestion des identités et des accès dans Amazon SQS

AWS Identity and Access Management (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Des administrateurs IAM contrôlent les personnes qui peuvent être authentifiées (connectées) et autorisées (disposant d'autorisations) à utiliser des ressources Amazon SQS. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (IAM) varie en fonction du travail que vous effectuez dans Amazon SQS.

Utilisateur du service : si vous utilisez le service Amazon SQS pour accomplir votre tâche, votre administrateur vous fournira les informations d'identification et les autorisations nécessaires. Vous pourrez avoir besoin d'autorisations supplémentaires si vous utilisez davantage de fonctionnalités Amazon SQS. En comprenant bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne pouvez pas accéder à une fonctionnalité dans Amazon SQS, consultez Résolution des problèmes d'accès et d'identité Amazon Simple Queue Service.

Administrateur du service : si vous êtes le responsable des ressources Amazon SQS de votre entreprise, vous bénéficiez probablement d'un accès total à ce service. C'est à vous de déterminer les fonctions et les ressources Amazon SQS auxquelles vos utilisateurs des services pourront accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour découvrir la façon dont votre entreprise peut utiliser IAM avec Amazon SQS, veuillez consulter Fonctionnement d'Amazon Simple Queue Service avec IAM.

Administrateur IAM : si vous êtes un administrateur IAM, vous pouvez souhaiter obtenir des informations sur l'écriture de stratégies pour gérer l'accès à Amazon SQS. Pour afficher des

exemples de stratégies basées sur l'identité Amazon SQS que vous pouvez utiliser dans IAM, consultez Bonnes pratiques en matière de politiques.

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié (connecté à AWS) en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en AWS tant qu'identité fédérée en utilisant les informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS l'aide de la fédération, vous assumez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter au portail AWS Management Console ou au portail AWS d'accès. Pour plus d'informations sur la connexion à AWS, consultez la section Comment vous connecter à votre compte Compte AWS dans le guide de Connexion à AWS l'utilisateur.

Si vous y accédez AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes à l'aide de vos informations d'identification. Si vous n'utilisez pas d' AWS outils, vous devez signer vous-même les demandes. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer des demandes vous-même, consultez AWS Signature Version 4 pour les demandes d'API dans le Guide de l'utilisateur IAM.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, il vous AWS recommande d'utiliser l'authentification multifactorielle (MFA) pour renforcer la sécurité de votre compte. Pour plus d'informations, consultez Authentification multifactorielle dans le Guide de l'utilisateur AWS IAM Identity Center et Authentification multifactorielle AWS dans IAM dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une identité de connexion unique qui donne un accès complet à toutes Services AWS les ressources du compte. Cette identité est

appelée utilisateur Compte AWS root et est accessible en vous connectant avec l'adresse e-mail et le mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur racine et utilisez-les pour effectuer les tâches que seul l'utilisateur racine peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur racine, consultez Tâches nécessitant des informations d'identification d'utilisateur racine dans le Guide de l'utilisateur IAM.

Identité fédérée

La meilleure pratique consiste à obliger les utilisateurs humains, y compris ceux qui ont besoin d'un accès administrateur, à utiliser la fédération avec un fournisseur d'identité pour accéder à l'aide Services AWS d'informations d'identification temporaires.

Une identité fédérée est un utilisateur de l'annuaire des utilisateurs de votre entreprise, d'un fournisseur d'identité Web AWS Directory Service, du répertoire Identity Center ou de tout utilisateur qui y accède à l'aide des informations d'identification fournies Services AWS par le biais d'une source d'identité. Lorsque des identités fédérées y accèdent Comptes AWS, elles assument des rôles, qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Vous pouvez créer des utilisateurs et des groupes dans IAM Identity Center, ou vous pouvez vous connecter et synchroniser avec un ensemble d'utilisateurs et de groupes dans votre propre source d'identité afin de les utiliser dans toutes vos applications Comptes AWS et applications. Pour obtenir des informations sur IAM Identity Center, consultez Qu'est-ce que IAM Identity Center ? dans le Guide de l'utilisateur AWS IAM Identity Center .

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité au sein de vous Compte AWS qui possède des autorisations spécifiques pour une seule personne ou une seule application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme telles que des mots de passe et des clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons d'effectuer une rotation des clés d'accès. Pour plus d'informations, consultez Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification dans le Guide de l'utilisateur IAM.

Un groupe IAM est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez nommer un groupe IAMAdminset lui donner les autorisations nécessaires pour administrer les ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour plus d'informations, consultez Cas d'utilisation pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une identité au sein de vous Compte AWS dotée d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Pour assumer temporairement un rôle IAM dans le AWS Management Console, vous pouvez passer d'un rôle d'utilisateur à un rôle IAM (console). Vous pouvez assumer un rôle en appelant une opération d' AWS API AWS CLI ou en utilisant une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Méthodes pour endosser un rôle dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré : pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, consultez Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center .
- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, dans certains Services AWS cas,

vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

- Accès multiservices — Certains Services AWS utilisent des fonctionnalités dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, un rôle de service ou un rôle lié au service.
  - Sessions d'accès direct (FAS) : lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal appelant et Service AWS, associées Service AWS à la demande, pour adresser des demandes aux services en aval. Les demandes FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur une politique lors de la formulation de demandes FAS, consultez Transmission des sessions d'accès.
  - Rôle de service : il s'agit d'un rôle IAM attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer un rôle de service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.
  - Rôle lié à un service — Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre Compte AWS répertoire et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.
- Applications exécutées sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer les informations d'identification temporaires pour les applications qui s'exécutent sur une EC2 instance et qui envoient des demandes AWS CLI d' AWS API. Cela est préférable au stockage des clés d'accès dans l' EC2 instance. Pour attribuer un AWS rôle à une EC2 instance et le rendre disponible pour toutes ses applications, vous devez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes exécutés sur l' EC2 instance d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez Utiliser

un rôle IAM pour accorder des autorisations aux applications exécutées sur des EC2 instances Amazon dans le guide de l'utilisateur IAM.

Gestion des accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique est un objet AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit leurs autorisations. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur root ou session de rôle) fait une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations sur la structure et le contenu des documents de politique JSON, consultez Vue d'ensemble des politiques JSON dans le Guide de l'utilisateur IAM.

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d'une politique qui autorise l'action `iam:GetRole`. Un utilisateur appliquant cette politique peut obtenir des informations sur le rôle à partir de AWS Management Console AWS CLI, de ou de l' AWS API.

Politiques basées sur l'identité

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou

rôle. Les politiques gérées sont des politiques autonomes que vous pouvez associer à plusieurs utilisateurs, groupes et rôles au sein de votre Compte AWS. Les politiques gérées incluent les politiques AWS gérées et les politiques gérées par le client. Pour découvrir comment choisir entre une politique gérée et une politique en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Par exemple, les politiques de confiance de rôle IAM et les politiques de compartiment Amazon S3 sont des politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la présentation de la liste de contrôle d'accès (ACL) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge d'autres types de politiques moins courants. Ces types de politiques peuvent définir le nombre maximum d'autorisations qui vous sont accordées par des types de politiques plus courants.

- Limite d'autorisations : une limite d'autorisations est une fonctionnalité avancée dans laquelle vous définissez le nombre maximal d'autorisations qu'une politique basée sur l'identité peut accorder

à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations en résultant représentent la combinaison des politiques basées sur l'identité d'une entité et de ses limites d'autorisation. Les politiques basées sur les ressources qui spécifient l'utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez Limites d'autorisations pour des entités IAM dans le Guide de l'utilisateur IAM.

- Politiques de contrôle des services (SCPs) : SCPs politiques JSON qui spécifient les autorisations maximales pour une organisation ou une unité organisationnelle (UO) dans AWS Organizations. AWS Organizations est un service permettant de regrouper et de gérer de manière centralisée Comptes AWS les multiples propriétés de votre entreprise. Si vous activez toutes les fonctionnalités d'une organisation, vous pouvez appliquer des politiques de contrôle des services (SCPs) à l'un ou à l'ensemble de vos comptes. Le SCP limite les autorisations pour les entités figurant dans les comptes des membres, y compris chacune Utilisateur racine d'un compte AWS d'entre elles. Pour plus d'informations sur les Organizations et consultez SCPs les politiques de contrôle des services dans le Guide de AWS Organizations l'utilisateur.
- Politiques de contrôle des ressources (RCPs) : RCPs politiques JSON que vous pouvez utiliser pour définir le maximum d'autorisations disponibles pour les ressources de vos comptes sans mettre à jour les politiques IAM associées à chaque ressource que vous possédez. Le RCP limite les autorisations pour les ressources des comptes membres et peut avoir un impact sur les autorisations effectives pour les identités, y compris Utilisateur racine d'un compte AWS, qu'elles appartiennent ou non à votre organisation. Pour plus d'informations sur les Organizations RCPs, y compris une liste de ces Services AWS supports RCPs, consultez la section Resource control policies (RCPs) dans le guide de AWS Organizations l'utilisateur.
- Politiques de séance : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de séance en résultant sont une combinaison des politiques basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations, consultez Politiques de session dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser

une demande lorsque plusieurs types de politiques sont impliqués, consultez la section Logique d'évaluation des politiques dans le guide de l'utilisateur IAM.

Présentation de la gestion de l'accès dans Amazon SQS

Chaque AWS ressource appartient à un Compte AWS, et les autorisations de création ou d'accès à une ressource sont régies par des politiques d'autorisation. Un administrateur de compte peut accorder des stratégies d'autorisation à des identités IAM (utilisateurs, groupes et rôles), et certains services (tels qu'Amazon SQS) prennent également en charge l'octroi de stratégies d'autorisation à des ressources.

> **Note**
>
> Un administrateur de compte (ou utilisateur administrateur) est un utilisateur doté des privilèges d'administration. Pour plus d'informations, consultez Bonnes pratiques IAM dans le Guide de l'utilisateur IAM.

Lorsque vous accordez des autorisations, vous indiquez quels utilisateurs en bénéficient, à quelles ressources ces autorisations s'appliquent et les actions spécifiques que vous souhaitez autoriser sur la ressource.

Ressources et opérations Amazon Simple Queue Service

Dans Amazon SQS, la file d'attente est la seule ressource. Dans une stratégie, utilisez un Amazon Resource Name (ARN) pour identifier la ressource à laquelle la stratégie s'applique. La ressource suivante est dotée d'un ARN unique qui lui est associé :

Type de ressource	Format ARN
File d'attente	`arn:aws:sqs:` *`region`*`:`*`account_id`*`:`*`queue_name`*

Vous trouverez ci-dessous des exemples de format ARN pour les files d'attente :

- Un ARN pour une file d'attente nommée `my_queue` dans la région USA Est (Ohio), appartenant au AWS compte 123456789012 :

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- ARN d'une file d'attente nommée `my_queue` dans chacune des différentes régions prises en charge par Amazon SQS :

```
arn:aws:sqs:*:123456789012:my_queue
```

- ARN utilisant `*` ou `?` comme caractère générique pour le nom de la file d'attente. Dans les exemples suivants, l'ARN correspond à toutes les files d'attente dont le préfixe est `my_prefix_` :

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

Vous pouvez obtenir la valeur de l'ARN d'une file d'attente existante en appelant l'action `GetQueueAttributes`. La valeur de l'attribut `QueueArn` correspond à l'ARN de la file d'attente. Pour plus d'informations ARNs, consultez la section IAM ARNs dans le guide de l'utilisateur d'IAM.

Amazon SQS fournit un ensemble d'actions qui fonctionnent avec la ressource de file d'attente. Pour de plus amples informations, veuillez consulter Autorisations d'API Amazon SQS : référence des actions et ressources.

Présentation de la propriété des ressources

Il Compte AWS est propriétaire des ressources créées dans le compte, quelle que soit la personne qui les a créées. Plus précisément, le propriétaire des ressources est le Compte AWS de l'entité du mandataire (à savoir, le compte racine, un utilisateur ou un rôle IAM) qui authentifie la demande de création des ressources. Les exemples suivants illustrent comment cela fonctionne :

- Si vous utilisez les informations d'identification de votre compte root Compte AWS pour créer une file d'attente Amazon SQS, vous êtes le propriétaire de la ressource (dans Amazon SQS, la ressource Compte AWS est la file d'attente Amazon SQS).
- Si vous créez un utilisateur dans votre file d'attente Compte AWS et que vous lui accordez l'autorisation de créer une file d'attente, celui-ci peut créer la file d'attente. Toutefois, votre Compte AWS (auquel l'utilisateur appartient) est propriétaire de la ressource file d'attente.
- Si vous créez un rôle IAM Compte AWS avec les autorisations nécessaires pour créer une file d'attente Amazon SQS, toute personne capable d'assumer ce rôle peut créer une file d'attente. Vous Compte AWS (à qui appartient le rôle) êtes propriétaire de la ressource de file d'attente.

Gestion de l’accès aux ressources

Une stratégie d'autorisation décrit qui a accès à quoi. La section suivante explique les options disponibles pour créer des stratégies d'autorisation.

> **Note**
>
> Cette section décrit l'utilisation d'IAM dans le contexte d'Amazon SQS. Elle ne fournit pas d'informations détaillées sur le service IAM. Pour une documentation complète sur IAM, consultez la rubrique Qu'est-ce que IAM ? dans le Guide de l'utilisateur IAM. Pour plus d'informations sur la syntaxe et les descriptions des stratégies IAM, consultez Référence de stratégie AWS IAM dans le Guide de l'utilisateur IAM.

Les politiques attachées à une identité IAM sont appelées politiques basées sur une entité (politiques IAM) et les politiques attachées à une ressource sont appelées politiques basées sur une ressource.

Politiques basées sur l’identité

Deux options s'offrent à vous pour autoriser les utilisateurs à accéder à vos files d'attente Amazon SQS : le système de stratégies Amazon SQS ou le système de stratégies IAM. Vous pouvez utiliser l'un et/ou l'autre des systèmes pour associer des stratégies à des utilisateurs ou à des rôles. Dans la plupart des cas, vous obtenez le même résultat avec l'un ou l'autre système. Par exemple, vous pouvez effectuer les opérations suivantes :

- Attacher une stratégie d'autorisation à un utilisateur ou à un groupe de votre compte : pour autoriser un utilisateur à créer une file d'attente Amazon SQS, attachez une stratégie d'autorisations à cet utilisateur ou à un groupe auquel il appartient.
- Associer une politique d'autorisation à un utilisateur d'un autre Compte AWS — Vous pouvez associer une politique d'autorisation à un utilisateur d'un autre utilisateur Compte AWS pour lui permettre d'interagir avec une file d'attente Amazon SQS. Toutefois, les autorisations entre comptes ne s'appliquent pas aux actions suivantes :

  Les autorisations intercompte ne s'appliquent pas aux actions suivantes :

  - `AddPermission`
  - `CancelMessageMoveTask`
  - `CreateQueue`
  - `DeleteQueue`

- ListMessageMoveTask
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Pour accorder l'accès à ces actions, l'utilisateur doit appartenir au même Compte AWS propriétaire de la file d'attente Amazon SQS.

- Associer une politique d'autorisation à un rôle (accorder des autorisations entre comptes) : pour accorder des autorisations entre comptes à une file d'attente SQS, vous devez combiner des politiques IAM et basées sur les ressources :

  1. Dans le compte A (propriétaire de la file d'attente) :
     - Attachez une politique basée sur les ressources à la file d'attente SQS. Cette politique doit explicitement accorder les autorisations nécessaires (par exemple SendMessage, ReceiveMessage) au principal dans le compte B (comme un rôle IAM).
  2. Dans le compte A, créez un rôle IAM :
     - Une politique de confiance qui permet au compte B ou an Service AWS d'assumer le rôle.

       > **Note**
       >
       > Si vous souhaitez qu'un Service AWS (tel que Lambda ouEventBridge) assume le rôle, spécifiez le principal du service (par exemple,lambda.amazonaws.com) dans la politique de confiance.

     - Une politique basée sur l'identité qui accorde au rôle assumé les autorisations d'interagir avec la file d'attente.
  3. Dans le compte B, accordez l'autorisation d'assumer le rôle dans le compte A.

Vous devez configurer la politique d'accès de la file d'attente pour autoriser le principal multicompte. Les politiques basées sur l'identité IAM ne sont pas suffisantes à elles seules pour l'accès entre comptes aux files d'attente SQS.

Pour en savoir plus sur l'utilisation d'IAM pour déléguer des autorisations, consultez Gestion des accès dans le Guide de l'utilisateur IAM.

Si Amazon SQS utilise des stratégies IAM, il dispose aussi de sa propre infrastructure de stratégies. Vous pouvez utiliser une politique Amazon SQS avec une file d'attente pour spécifier quels AWS comptes ont accès à la file d'attente. Vous pouvez définir le type d'accès et les conditions (par exemple, une condition accordant des autorisations pour utiliser `SendMessage`, `ReceiveMessage` si la demande a lieu avant le 31 décembre 2010). Les actions spécifiques pour lesquelles vous pouvez accorder des autorisations représentent un sous-ensemble de la liste complète des actions Amazon SQS. Lorsque vous écrivez une stratégie Amazon SQS et que vous spécifiez `*` pour « autoriser toutes les actions Amazon SQS », cela signifie qu'un utilisateur peut effectuer toutes les actions de ce sous-ensemble.

Le schéma suivant illustre le concept de l'une des stratégies Amazon SQS de base, qui couvre le sous-ensemble d'actions. La politique est pour`queue_xyz`, et elle donne aux AWS comptes 1 et AWS 2 l'autorisation d'utiliser n'importe laquelle des actions autorisées avec la file d'attente spécifiée.

> **Note**
>
> La ressource spécifiée dans la politique est la suivante `123456789012/queue_xyz` : où `123456789012` est l'ID de AWS compte du compte propriétaire de la file d'attente.

Avec l'introduction de l'IAM et des concepts d'utilisateurs et d'Amazon Resource Names (ARNs), certaines choses ont changé en ce qui concerne les politiques SQS. Le graphique et le tableau suivants décrivent ces modifications.

1 Pour plus d'informations sur l'octroi d'autorisations aux utilisateurs de différents comptes, voir Tutoriel : déléguer l'accès entre AWS comptes à l'aide de rôles IAM dans le guide de l'utilisateur IAM.

2 Le sous-ensemble d'actions inclus dans `*` est plus vaste. Pour obtenir la liste des actions autorisées, consultez Autorisations d'API Amazon SQS : référence des actions et ressources.

3 Vous pouvez spécifier les ressources en indiquant leur Amazon Resource Name (ARN). Il s'agit de la méthode standard pour spécifier des ressources dans les stratégies IAM. Pour plus d'informations sur le format ARN pour les files d'attente Amazon SQS, consultez Ressources et opérations Amazon Simple Queue Service.

Par exemple, conformément à la politique Amazon SQS décrite dans le schéma précédent, toute personne possédant les informations de sécurité du AWS compte 1 ou du AWS compte 2 peut y accéder. `queue_xyz` De plus, les utilisateurs Bob et Susan, qui appartiennent à votre compte AWS (avec l'ID `123456789012`) peuvent également accéder à la file d'attente.

Avant l'introduction d'IAM, Amazon SQS accordait automatiquement au créateur d'une file d'attente un contrôle complet sur celle-ci (c'est-à-dire l'accès à toutes les actions Amazon SQS possibles sur cette file d'attente). Ce n'est plus le cas, sauf si le créateur utilise des informations d'identification de sécurité AWS . Tout utilisateur qui dispose d'autorisations pour créer une file d'attente doit également avoir les autorisations nécessaires pour utiliser les autres actions Amazon SQS afin de pouvoir exploiter les files d'attente créées.

Dans l'exemple suivant, la stratégie autorise un utilisateur à utiliser toutes les actions Amazon SQS, mais seulement avec les files d'attente dont le nom comporte la chaîne littérale `bob_queue_` en préfixe.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
   }]
}
```

Pour plus d'informations, consultez Utilisation de politiques avec Amazon SQS et Identités (utilisateurs, groupes et rôles) dans le Guide de l'utilisateur IAM.

Spécification des éléments d'une politique : actions, effets, ressources et principaux

Pour chaque ressource Amazon Simple Queue Service, le service définit un ensemble d'actions. Pour accorder des autorisations pour ces actions, Amazon SQS définit un ensemble d'actions que vous pouvez spécifier dans une stratégie.

> **Note**
>
> Une action peut exiger des autorisations pour plusieurs actions. Lors de l'octroi des autorisations pour des actions spécifiques, vous identifiez également la ressource pour laquelle les actions sont autorisées ou refusées.

Voici les éléments les plus élémentaires d'une politique :

- Ressource : dans une politique, vous utilisez un Amazon Resource Name (ARN) pour identifier la ressource à laquelle la politique s'applique.
- Action : vous utilisez des mots clés d'action pour identifier les actions de ressource que vous voulez accorder ou refuser. Par exemple, l'autorisation `sqs:CreateQueue` permet à l'utilisateur d'effectuer l'action Amazon Simple Queue Service `CreateQueue`.

- Effet : vous spécifiez l'effet produit lorsque l'utilisateur demande l'action spécifique, qui peut être une autorisation ou un refus. Si vous n'accordez pas explicitement l'accès à une ressource, il est implicitement refusé. Vous pouvez aussi explicitement refuser l'accès à une ressource, ce que vous pouvez faire afin de vous assurer qu'un utilisateur n'y a pas accès, même si une stratégie différente accorde l'accès.
- Principal : dans les politiques basées sur une identité (politiques IAM), l'utilisateur auquel la politique est attachée est le principal implicite. Pour les politiques basées sur une ressource, vous spécifiez l'utilisateur, le compte, le service ou une autre entité qui doit recevoir les autorisations (s'applique uniquement aux politiques basées sur une ressource).

Pour en savoir plus sur la syntaxe des stratégies Amazon SQS et pour obtenir des descriptions, consultez Référence de stratégie IAM AWS dans le Guide de l'utilisateur IAM.

Pour visualiser un tableau répertoriant toutes les actions Amazon Simple Queue Service et les ressources auxquelles elles s'appliquent, consultez Autorisations d'API Amazon SQS : référence des actions et ressources.

Fonctionnement d'Amazon Simple Queue Service avec IAM

Avant d'utiliser IAM pour gérer l'accès à Amazon SQS, découvrez les fonctionnalités IAM qui peuvent être utilisées avec Amazon SQS.

Fonctionnalités IAM que vous pouvez utiliser avec Amazon Simple Queue Service

Fonctionnalité IAM	Assistance Amazon SQS
Politiques basées sur l'identité	Oui
Politiques basées sur les ressources	Oui
Actions de politique	Oui
Ressources de politique	Oui
Clés de condition de politique (spécifiques au service)	Oui
ACLs	Non

Fonctionnalité IAM	Assistance Amazon SQS
ABAC (identifications dans les politiques)	Partielle
Informations d'identification temporaires	Oui
Transmission des sessions d'accès (FAS)	Oui
Rôles de service	Oui
Rôles liés à un service	Non

Pour obtenir une vue d'ensemble de la façon dont Amazon SQS et les autres AWS services fonctionnent avec la plupart des fonctionnalités IAM, consultez les AWS services compatibles avec IAM dans le guide de l'utilisateur IAM.

Contrôle d'accès

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la présentation de la liste de contrôle d'accès (ACL) dans le guide du développeur Amazon Simple Storage Service.

> **Note**
>
> Il est important de comprendre que tous Comptes AWS peuvent déléguer leurs autorisations aux utilisateurs de leurs comptes. L'accès entre comptes vous permet de partager l'accès à vos AWS ressources sans avoir à gérer d'autres utilisateurs. Pour plus d'informations sur l'accès entre comptes, consultez Activation de l'accès entre comptes dans le Guide de l'utilisateur IAM.
> Consultez Limites des politiques personnalisées d'Amazon SQS pour en savoir plus sur les autorisations et les clés de condition relatives au contenu croisé dans les stratégies personnalisées d'Amazon SQS.

Stratégies basées sur l'identité pour Amazon SQS

Prend en charge les politiques basées sur l'identité : oui

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Définition d'autorisations IAM personnalisées avec des politiques gérées par le client dans le Guide de l'utilisateur IAM.

Avec les politiques IAM basées sur l'identité, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. Vous ne pouvez pas spécifier le principal dans une politique basée sur une identité, car celle-ci s'applique à l'utilisateur ou au rôle auquel elle est attachée. Pour découvrir tous les éléments que vous utilisez dans une politique JSON, consultez Références des éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

Exemples de stratégies basées sur l'identité pour Amazon SQS

Pour voir des exemples de stratégies Amazon SQS basées sur l'identité, consultez Bonnes pratiques en matière de politiques.

Stratégies basées sur les ressources au sein d'Amazon SQS

Prend en charge les politiques basées sur les ressources : oui

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Par exemple, les politiques de confiance de rôle IAM et les politiques de compartiment Amazon S3 sont des politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Pour permettre un accès intercompte, vous pouvez spécifier un compte entier ou des entités IAM dans un autre compte en tant que principal dans une politique basée sur les ressources. L'ajout

d'un principal intercompte à une politique basée sur les ressources ne représente qu'une partie de l'instauration de la relation d'approbation. Lorsque le principal et la ressource sont différents Comptes AWS, un administrateur IAM du compte sécurisé doit également accorder à l'entité principale (utilisateur ou rôle) l'autorisation d'accéder à la ressource. Pour ce faire, il attache une politique basée sur une identité à l'entité. Toutefois, si une politique basée sur des ressources accorde l'accès à un principal dans le même compte, aucune autre politique basée sur l'identité n'est requise. Pour plus d'informations, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Actions de stratégie pour Amazon SQS

Prend en charge les actions de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Action` d'une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une politique. Les actions de stratégie portent généralement le même nom que l'opération AWS d'API associée. Il existe quelques exceptions, telles que les actions avec autorisations uniquement qui n'ont pas d'opération API correspondante. Certaines opérations nécessitent également plusieurs actions dans une politique. Ces actions supplémentaires sont nommées actions dépendantes.

Intégration d'actions dans une stratégie afin d'accorder l'autorisation d'exécuter les opérations associées.

Pour afficher la liste des actions Amazon SQS, consultez les Ressources définies par Amazon Simple Queue Service dans la Référence de l'autorisation de service.

Les actions de stratégie dans Amazon SQS utilisent le préfixe suivant avant l'action :

```
sqs
```

Pour indiquer plusieurs actions dans une seule déclaration, séparez-les par des virgules.

```
"Action": [
      "sqs:action1",
      "sqs:action2"
```

```
        ]
```

Pour voir des exemples de stratégies Amazon SQS basées sur l'identité, consultez Bonnes pratiques en matière de politiques.

Ressources de stratégie pour Amazon SQS

Prend en charge les ressources de politique : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets auxquels l'action s'applique. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de définir une ressource à l'aide de son Amazon Resource Name (ARN). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Pour consulter la liste des types de ressources Amazon SQS et leurs caractéristiques ARNs, consultez la section Actions définies par Amazon Simple Queue Service dans le Service Authorization Reference. Pour connaître les actions avec lesquelles vous pouvez spécifier l'ARN de chaque ressource, consultez les Ressources définies par Amazon Simple Queue Service.

Pour voir des exemples de stratégies Amazon SQS basées sur l'identité, consultez Bonnes pratiques en matière de politiques.

Clés de condition de stratégie pour Amazon SQS

Prend en charge les clés de condition de politique spécifiques au service : oui

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` (ou le bloc `Condition`) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément `Condition` est facultatif. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande.

Si vous spécifiez plusieurs éléments `Condition` dans une instruction, ou plusieurs clés dans un seul élément `Condition`, AWS les évalue à l'aide d'une opération `AND` logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une `OR` opération logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez accorder à un utilisateur IAM l'autorisation d'accéder à une ressource uniquement si elle est balisée avec son nom d'utilisateur IAM. Pour plus d'informations, consultez Éléments d'une politique IAM : variables et identifications dans le Guide de l'utilisateur IAM.

AWS prend en charge les clés de condition globales et les clés de condition spécifiques au service. Pour voir toutes les clés de condition AWS globales, voir les clés de contexte de condition AWS globales dans le guide de l'utilisateur IAM.

Pour consulter la liste des clés de condition Amazon SQS, consultez Clés de condition pour Amazon Simple Queue Service dans la Référence de l'autorisation de service. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez Ressources définies par Amazon Simple Queue Service.

Pour voir des exemples de stratégies Amazon SQS basées sur l'identité, consultez Bonnes pratiques en matière de politiques.

ACLs dans Amazon SQS

Supports ACLs : Non

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

ABAC avec Amazon SQS

Prend en charge ABAC (identifications dans les politiques) : partiellement

Le contrôle d'accès par attributs (ABAC) est une stratégie d'autorisation qui définit des autorisations en fonction des attributs. Dans AWS, ces attributs sont appelés balises. Vous pouvez associer des balises aux entités IAM (utilisateurs ou rôles) et à de nombreuses AWS ressources. L'étiquetage des entités et des ressources est la première étape d'ABAC. Vous concevez ensuite des politiques ABAC pour autoriser des opérations quand l'identification du principal correspond à celle de la ressource à laquelle il tente d'accéder.

L'ABAC est utile dans les environnements qui connaissent une croissance rapide et pour les cas où la gestion des politiques devient fastidieuse.

Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'élément de condition d'une politique utilisant les clés de condition `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`.

Si un service prend en charge les trois clés de condition pour tous les types de ressources, alors la valeur pour ce service est Oui. Si un service prend en charge les trois clés de condition pour certains types de ressources uniquement, la valeur est Partielle.

Pour plus d'informations sur ABAC, consultez Définition d'autorisations avec l'autorisation ABAC dans le Guide de l'utilisateur IAM. Pour accéder à un didacticiel décrivant les étapes de configuration de l'ABAC, consultez Utilisation du contrôle d'accès basé sur les attributs (ABAC) dans le Guide de l'utilisateur IAM.

Utilisation d'informations d'identification temporaires avec Amazon SQS

Prend en charge les informations d'identification temporaires : oui

Certains Services AWS ne fonctionnent pas lorsque vous vous connectez à l'aide d'informations d'identification temporaires. Pour plus d'informations, y compris celles qui Services AWS fonctionnent avec des informations d'identification temporaires, consultez Services AWS la section relative à l'utilisation d'IAM dans le guide de l'utilisateur d'IAM.

Vous utilisez des informations d'identification temporaires si vous vous connectez à l' AWS Management Console aide d'une méthode autre qu'un nom d'utilisateur et un mot de passe. Par exemple, lorsque vous accédez à AWS l'aide du lien d'authentification unique (SSO) de votre entreprise, ce processus crée automatiquement des informations d'identification temporaires. Vous

créez également automatiquement des informations d'identification temporaires lorsque vous vous connectez à la console en tant qu'utilisateur, puis changez de rôle. Pour plus d'informations sur le changement de rôle, consultez Passage d'un rôle utilisateur à un rôle IAM (console) dans le Guide de l'utilisateur IAM.

Vous pouvez créer manuellement des informations d'identification temporaires à l'aide de l' AWS API AWS CLI or. Vous pouvez ensuite utiliser ces informations d'identification temporaires pour y accéder AWS. AWS recommande de générer dynamiquement des informations d'identification temporaires au lieu d'utiliser des clés d'accès à long terme. Pour plus d'informations, consultez Informations d'identification de sécurité temporaires dans IAM.

Transférer les sessions d'accès pour Amazon SQS

Prend en charge les sessions d'accès direct (FAS) : oui

Lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal appelant et Service AWS, associées Service AWS à la demande, pour adresser des demandes aux services en aval. Les demandes FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur une politique lors de la formulation de demandes FAS, consultez Transmission des sessions d'accès.

Fonctions du service pour Amazon SQS

Prend en charge les rôles de service : oui

Un rôle de service est un rôle IAM qu'un service endosse pour accomplir des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer un rôle de service à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.

> **Warning**
>
> La modification des autorisations d'une fonction du service peut altérer la fonctionnalité d'Amazon SQS. Ne modifiez des fonctions du service que quand Amazon SQS vous le conseille.

Rôles liés à un service pour Amazon SQS

Prend en charge les rôles liés à un service : non

Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre Compte AWS répertoire et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

Pour plus d'informations sur la création ou la gestion des rôles liés à un service, consultez Services AWS qui fonctionnent avec IAM. Recherchez un service dans le tableau qui inclut un `Yes` dans la colonne Rôle lié à un service. Choisissez le lien Oui pour consulter la documentation du rôle lié à ce service.

Mises à jour des politiques gérées par Amazon SQS AWS

Pour ajouter des autorisations à des utilisateurs, des groupes et des rôles, il est plus facile d'utiliser des politiques gérées par AWS que d'écrire des politiques vous-même. Il faut du temps et de l'expertise pour créer des politiques gérées par le client IAM qui ne fournissent à votre équipe que les autorisations dont elle a besoin. Pour démarrer rapidement, vous pouvez utiliser nos politiques gérées AWS . Ces politiques couvrent des cas d'utilisation courants et sont disponibles dans votre compte AWS . Pour plus d'informations sur les politiques AWS gérées, voir les politiques AWS gérées dans le guide de l'utilisateur IAM.

AWS les services maintiennent et mettent à jour les politiques AWS gérées. Vous ne pouvez pas modifier les autorisations dans les politiques AWS gérées. Les services ajoutent parfois des autorisations supplémentaires à une politique AWS gérée pour prendre en charge de nouvelles fonctionnalités. Ce type de mise à jour affecte toutes les identités (utilisateurs, groupes et rôles) auxquelles la politique est attachée. Les services sont plus susceptibles de mettre à jour une politique AWS gérée lorsqu'une nouvelle fonctionnalité est lancée ou lorsque de nouvelles opérations sont disponibles. Les services ne suppriment pas les autorisations d'une politique AWS gérée. Les mises à jour des politiques n'endommageront donc pas vos autorisations existantes.

En outre, AWS prend en charge les politiques gérées pour les fonctions professionnelles qui couvrent plusieurs services. Par exemple, la politique ReadOnlyAccess AWS gérée fournit un accès en lecture seule à tous les AWS services et ressources. Lorsqu'un service lance une nouvelle fonctionnalité, il AWS ajoute des autorisations en lecture seule pour les nouvelles opérations et ressources. Pour

obtenir la liste des politiques de fonctions professionnelles et leurs descriptions, consultez la page politiques gérées par AWS pour les fonctions de tâche dans le Guide de l'utilisateur IAM.

AWS politique gérée : Amazon SQSFull Access

Vous pouvez attacher la stratégie `AmazonSQSFullAccess` à vos identités Amazon SQS. Cette stratégie accorde des autorisations qui permettent un accès complet à Amazon SQS.

Pour consulter les autorisations associées à cette politique, consultez Amazon SQSFull Access dans le manuel AWS Managed Policy Reference.

AWS politique gérée : Amazon SQSRead OnlyAccess

Vous pouvez attacher la stratégie `AmazonSQSReadOnlyAccess` à vos identités Amazon SQS. Cette stratégie accorde des autorisations qui permettent un accès en lecture seule à Amazon SQS.

Pour consulter les autorisations associées à cette politique, consultez Amazon SQSRead OnlyAccess dans le AWS Managed Policy Reference.

AWS politique gérée : SQSUnlock QueuePolicy

Si vous avez mal configuré votre politique de file d'attente pour qu'un compte membre refuse à tous les utilisateurs l'accès à votre file d'attente Amazon SQS, vous pouvez utiliser la politique `SQSUnlockQueuePolicy` AWS gérée pour déverrouiller la file d'attente.

Pour plus d'informations sur la façon de supprimer une politique de file d'attente mal configurée qui empêche tous les principaux d'accéder à une file d'attente Amazon SQS, consultez la section Exécuter une tâche privilégiée sur AWS Organizations un compte membre dans le guide de l'utilisateur IAM.

Mises à jour des politiques gérées par Amazon SQS AWS

Consultez les informations relatives aux mises à jour des politiques AWS gérées pour Amazon SQS depuis que ce service a commencé à suivre ces modifications. Pour recevoir des alertes automatiques sur les modifications apportées à cette page, abonnez-vous au flux RSS de la page Historique des documents d'Amazon SQS.

Modification	Description	Date
SQSUnlockQueuePolicy		15 novembre 2024

Modification	Description	Date
	Amazon SQS a ajouté une nouvelle politique AWS gérée appelée `SQSUnlockQueuePolicy` à déverrouiller une file d'attente et à supprimer une politique de file d'attente mal configurée qui empêche tous les principaux d'accéder à une file d'attente Amazon SQS.	
AmazonSQSReadOnlyAccess	Amazon SQS a ajouté l'`ListQueueTags` action, qui récupère toutes les balises associées à une file d'attente Amazon SQS spécifiée. Il vous permet de visualiser les paires clé-valeur attribuées à la file d'attente à des fins d'organisation ou de métadonnées. Cette action est associée à l'opération d'API `ListQueueTags` .	20 juin 2024
AmazonSQSReadOnlyAccess	Amazon SQS a ajouté une nouvelle action qui vous permet de répertorier les tâches de transfert de messages les plus récentes (jusqu'à 10) dans une file d'attente source spécifique. Cette action est associée à l'opération d'API `ListMessageMoveTasks` .	9 juin 2023

Résolution des problèmes d'accès et d'identité Amazon Simple Queue Service

Utilisez les informations suivantes pour identifier et résoudre les problèmes courants que vous pouvez rencontrer lorsque vous utilisez Amazon SQS et IAM.

Je ne suis pas autorisé à effectuer une action dans Amazon SQS

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à effectuer une action, vos stratégies doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *`my-example-widget`* fictive, mais ne dispose pas des autorisations `sqs:`*`GetWidget`* fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 sqs:GetWidget on resource: my-example-widget
```

Dans ce cas, la stratégie de Mateo doit être mise à jour pour l'autoriser à accéder à la ressource *`my-example-widget`* à l'aide de l'action `sqs:`*`GetWidget`*.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter l'action `iam:PassRole`, vos stratégies doivent être mises à jour pour vous permettre de transmettre un rôle à Amazon SQS.

Certains vous Services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, un utilisateur doit disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour effectuer une action dans Amazon SQS. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je souhaite autoriser des personnes extérieures à moi Compte AWS à accéder à mes ressources Amazon SQS

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si Amazon SQS est compatible avec ces fonctionnalités, consultez Fonctionnement d'Amazon Simple Queue Service avec IAM.
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section Fournir un accès à des ressources Comptes AWS détenues par des tiers dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez Fournir un accès à des utilisateurs authentifiés en externe (fédération d'identité) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez Accès intercompte aux ressources dans IAM dans le Guide de l'utilisateur IAM.

Je souhaite déverrouiller ma file d'attente

Si vous Compte AWS appartenez à une organisation, AWS Organizations les politiques peuvent vous empêcher d'accéder aux ressources Amazon SQS. Par défaut, AWS Organizations les politiques ne bloquent aucune demande adressée à Amazon SQS. Assurez-vous toutefois que vos AWS Organizations politiques n'ont pas été configurées pour bloquer l'accès aux files d'attente Amazon SQS. Pour savoir comment vérifier vos AWS Organizations politiques, consultez la section Liste de toutes les politiques dans le guide de AWS Organizations l'utilisateur.

En outre, si vous avez mal configuré votre politique de file d'attente pour un compte membre afin de refuser à tous les utilisateurs l'accès à votre file d'attente Amazon SQS, vous pouvez déverrouiller la file d'attente en lançant une session privilégiée pour le compte membre dans IAM. Une fois que vous avez lancé une session privilégiée, vous pouvez supprimer la politique de file d'attente mal configurée pour retrouver l'accès à la file d'attente. Pour plus d'informations, voir Exécuter une tâche privilégiée sur un compte AWS Organizations membre dans le Guide de l'utilisateur IAM.

Utilisation de politiques avec Amazon SQS

Cette rubrique fournit des exemples de stratégies basées sur l'identité dans lesquelles un administrateur de compte peut associer des stratégies d'autorisation à des identités IAM (utilisateurs, groupes et rôles).

> **Important**
>
> Nous vous recommandons tout d'abord d'examiner les rubriques de présentation qui détaillent les concepts de base et les options disponibles pour gérer l'accès à vos ressources Amazon Simple Queue Service. Pour de plus amples informations, veuillez consulter Présentation de la gestion de l'accès dans Amazon SQS.
> À l'exception de `ListQueues`, toutes les actions Amazon SQS prennent en charge les autorisations de niveau ressource. Pour de plus amples informations, veuillez consulter Autorisations d'API Amazon SQS : référence des actions et ressources.

Utilisation des stratégies Amazon SQS et IAM

Il existe deux manières d'autoriser vos utilisateurs à accéder à vos ressources Amazon SQS : en utilisant le système de politiques Amazon SQS (politiques basées sur les ressources) et en utilisant le système de politique IAM (politiques basées sur l'identité). Vous pouvez utiliser l'une ou les deux

méthodes, à l'exception de l'`ListQueues`action, qui est une autorisation régionale qui ne peut être définie que dans une politique IAM.

Par exemple, le schéma suivant illustre une stratégie IAM et une stratégie Amazon SQS équivalente. La politique IAM accorde les droits sur Amazon `ReceiveMessage` SQS `SendMessage` et les actions relatives à la file d'attente `queue_xyz` appelée dans AWS votre compte, et la politique est attachée aux utilisateurs nommés Bob et Susan (Bob et Susan ont les autorisations indiquées dans la politique). Cette stratégie Amazon SQS donne également à Bob et à Susan des droits sur les actions `ReceiveMessage` et `SendMessage` pour cette même file d'attente.

> **Note**
>
> L'exemple suivant montre des politiques simples sans conditions. Vous pouvez spécifier une condition particulière dans l'une ou l'autre des stratégies et aboutir au même résultat.

Il existe une différence majeure entre les politiques IAM et Amazon SQS : le système de politiques Amazon SQS vous permet d'accorder des autorisations à AWS d'autres comptes, contrairement à IAM.

Il vous incombe de décider si vous voulez utiliser les deux systèmes conjointement pour gérer vos autorisations. Les exemples suivants illustrent la façon dont les deux systèmes de politique interagissent.

- Dans le premier exemple, Bob possède une stratégie IAM et une stratégie Amazon SQS qui s'appliquent à son compte. La stratégie IAM accorde à son compte l'autorisation d'effectuer l'action `ReceiveMessage` sur `queue_xyz`, tandis que la stratégie Amazon SQS autorise son compte à effectuer l'action `SendMessage` sur cette même file d'attente. Le diagramme suivant illustre le concept.

  Si Bob envoie une demande `ReceiveMessage` à `queue_xyz`, la stratégie IAM autorise l'action. Si Bob envoie une demande `SendMessage` à `queue_xyz`, la stratégie Amazon SQS autorise l'action.

- Dans le deuxième exemple, Bob abuse de son accès à la file d'attente `queue_xyz` de telle sorte qu'il devient nécessaire de supprimer cet accès. La méthode la plus simple consiste à ajouter une stratégie qui lui refuse l'accès à toutes les actions pour cette file d'attente. Cette stratégie prévaut sur les deux autres car une action `deny` explicite prévaut toujours sur une action `allow`. Pour plus d'informations sur la logique d'évaluation de stratégie, consultez la section Utilisation de stratégies personnalisées avec le langage de la stratégie d'accès Amazon SQS. Le diagramme suivant illustre le concept.

Vous pouvez également ajouter une instruction supplémentaire à la stratégie Amazon SQS, qui refuse à Bob tout type d'accès à la file d'attente. Cette approche a le même effet que l'ajout d'une stratégie IAM refusant à Bob l'accès à la file d'attente. Pour obtenir des exemples de stratégies couvrant les actions et ressources Amazon SQS, consultez Exemples de base de stratégies Amazon SQS. Pour plus d'informations sur l'écriture de stratégies Amazon SQS, consultez Utilisation de stratégies personnalisées avec le langage de la stratégie d'accès Amazon SQS.

Autorisations requises pour utiliser la console Amazon SQS

Un utilisateur désireux d'utiliser la console Amazon SQS doit disposer d'un ensemble minimal d'autorisations pour utiliser les files d'attente Amazon SQS dans son Compte AWS. Par exemple, l'utilisateur doit avoir l'autorisation d'appeler l'action `ListQueues` pour répertorier les files d'attente ou l'action `CreateQueue` pour créer des files d'attente. Outre les autorisations Amazon SQS pour abonner une file d'attente Amazon SQS à une rubrique Amazon SNS, la console exige également des autorisations pour les actions Amazon SNS.

Si vous créez une stratégie IAM plus restrictive que les autorisations minimales requises, la console peut ne pas fonctionner comme prévu pour les utilisateurs dotés de la stratégie IAM.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement les actions AWS CLI ou Amazon SQS.

Exemples de stratégies basées sur l'identité pour Amazon SQS

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou à modifier des ressources Amazon SQS. Ils ne peuvent pas non plus effectuer de tâches à l'aide de l'API AWS Management Console, AWS Command Line Interface (AWS CLI) ou de AWS l'API. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Pour apprendre à créer une politique basée sur l'identité IAM à l'aide de ces exemples de documents de politique JSON, consultez Création de politiques IAM (console) dans le Guide de l'utilisateur IAM.

Pour en savoir plus sur les actions et les types de ressources définis par Amazon SQS, y compris le format de chaque type de ressource, consultez la section Actions, ressources et clés de condition ARNs pour Amazon Simple Queue Service dans la référence d'autorisation du service.

> **Note**
>
> Lorsque vous configurez des hooks de cycle de vie pour Amazon EC2 Auto Scaling, vous n'avez pas besoin de rédiger de politique pour envoyer des messages à une file d'attente Amazon SQS. Pour plus d'informations, consultez Amazon EC2 Auto Scaling Lifecycle Hooks dans le guide de EC2 l'utilisateur Amazon.

Bonnes pratiques en matière de politiques

Les stratégies basées sur l'identité déterminent si une personne peut créer, consulter ou supprimer des ressources Amazon SQS dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d'informations, consultez politiques gérées par AWS ou politiques gérées par AWS pour les activités professionnelles dans le Guide de l'utilisateur IAM.

- Accordez les autorisations de moindre privilège : lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation d'IAM pour appliquer des autorisations, consultez politiques et autorisations dans IAM dans le Guide de l'utilisateur IAM.
- Utilisez des conditions dans les politiques IAM pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que AWS CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.
- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez Validation de politiques avec IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez Sécurisation de l'accès aux API avec MFA dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.

Utilisation de la console Amazon SQS

Pour accéder à la console Amazon Simple Queue Service, vous devez disposer d'un ensemble minimum d'autorisations. Ces autorisations doivent vous permettre de répertorier et de consulter les informations relatives aux ressources Amazon SQS de votre. Compte AWS Si vous créez une politique basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la

console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l' AWS API. Autorisez plutôt l'accès à uniquement aux actions qui correspondent à l'opération d'API qu'ils tentent d'effectuer.

Pour garantir que les utilisateurs et les rôles peuvent toujours utiliser la console Amazon SQS, associez également la politique gérée par Amazon `AmazonSQSReadOnlyAccess` AWS SQS aux entités. Pour plus d'informations, consultez Ajout d'autorisations à un utilisateur dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
```

```
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Autoriser un utilisateur à créer des files d'attente

Dans l'exemple suivant, nous créons une stratégie qui permet à Bob d'accéder à toutes les actions Amazon SQS, mais seulement avec les files d'attente dont le nom comporte la chaîne littérale `alice_queue_` en préfixe.

Amazon SQS n'accorde pas automatiquement au créateur d'une file d'attente les autorisations de l'utiliser. Par conséquent, dans la stratégie IAM, nous devons explicitement accorder à Bob les autorisations d'utiliser toutes les actions Amazon SQS en plus de l'action `CreateQueue`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
   }]
}
```

Permettre aux développeurs d'écrire des messages dans une file d'attente partagée

Dans l'exemple suivant, nous créons un groupe pour les développeurs et y attachons une politique qui permet au groupe d'utiliser l'`SendMessage`action Amazon SQS, mais uniquement avec la file d'attente qui appartient à la file spécifiée Compte AWS et qui est nommée. `MyCompanyQueue`

JSON

```
{
```

```
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
   }]
}
```

Vous pouvez utiliser `*` au lieu de `SendMessage` pour attribuer les actions suivantes à un mandataire sur une file d'attente partagée : `ChangeMessageVisibility`, `DeleteMessage`, `GetQueueAttributes`, `GetQueueUrl`, `ReceiveMessage` et `SendMessage`.

> **Note**
>
> Bien que `*` comprenne l'accès fourni par d'autres types d'autorisation, Amazon SQS examine les autorisations séparément. Par exemple, il est possible d'accorder à la fois les autorisations `*` et `SendMessage` à un utilisateur, même si le symbole `*` inclut l'accès fourni par `SendMessage`.
> Ce concept s'applique également quand vous supprimez une autorisation. Si un mandataire dispose uniquement d'une autorisation `*`, toute demande de suppression de l'autorisation `SendMessage` ne signifie pas que l'utilisateur peut tout faire, sauf cette action. Au lieu de cela, cette demande n'a aucun effet, car le mandataire n'a aucune autorisation `SendMessage` explicite. Pour attribuer uniquement l'autorisation `ReceiveMessage` au mandataire, commencez par ajouter l'autorisation `ReceiveMessage`, puis supprimez l'autorisation `*`.

Permettre aux gestionnaires d'obtenir la taille générale des files d'attente

Dans l'exemple suivant, nous créons un groupe pour les responsables et y attachons une politique qui permet au groupe d'utiliser l'`GetQueueAttributes`action Amazon SQS avec toutes les files d'attente appartenant au compte spécifié. AWS

JSON

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

```
      "Effect": "Allow",
      "Action": "sqs:GetQueueAttributes",
      "Resource": "*"
   }]
}
```

Autoriser un partenaire à envoyer des messages à une file d'attente spécifique

Vous pouvez accomplir cette tâche à l'aide d'une stratégie Amazon SQS ou IAM. Si votre partenaire dispose d'une politique Amazon SQS Compte AWS, il peut être plus facile d'utiliser une politique Amazon SQS. Cependant, tout utilisateur de l'entreprise du partenaire qui possède les informations de AWS sécurité peut envoyer des messages à la file d'attente. Si vous souhaitez limiter l'accès à un utilisateur ou une application en particulier, vous devez traiter le partenaire comme un utilisateur de votre propre entreprise et utiliser une stratégie IAM au lieu d'une stratégie Amazon SQS.

L'exemple suivant effectue les actions suivantes :

1. Créez un groupe appelé WidgetCo pour représenter l'entreprise partenaire.
2. Création d'un utilisateur pour la personne ou l'application spécifique qui a besoin d'un accès dans l'entreprise du partenaire.
3. Ajoutez l'utilisateur au groupe .
4. Association d'une stratégie qui donne au groupe l'accès à l'action `SendMessage`, mais uniquement pour la file d'attente `WidgetPartnerQueue`.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "sqs:SendMessage",
         "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
   }]
}
```

Exemples de base de stratégies Amazon SQS

Cette section présente des exemples de stratégies pour les cas d'utilisation Amazon SQS les plus courants.

Vous pouvez utiliser la console pour vérifier les effets de chaque politique lorsque vous les associez à l'utilisateur. Au départ, l'utilisateur n'a pas les autorisations requises et ne peut donc effectuer aucune action dans la console. A mesure que vous lui associez des stratégies, vous pouvez vérifier que l'utilisateur peut exécuter diverses actions dans la console.

> **Note**
>
> Nous vous recommandons d'utiliser deux fenêtres de navigateur : l'une pour accorder des autorisations et l'autre pour vous connecter à l' AWS Management Console aide des informations d'identification de l'utilisateur afin de vérifier les autorisations que vous lui accordez.

Exemple 1 : accorder une autorisation à une Compte AWS

L'exemple de politique suivant accorde à Compte AWS number `111122223333` l'`SendMessage`autorisation pour la file d'attente nommée `444455556666/queue1` dans la région USA Est (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"
   }]
```

```
}
```

Exemple 2 : accorder deux autorisations à une Compte AWS

L'exemple de politique suivant accorde à la `111122223333` fois le Compte AWS numéro `SendMessage` et l'`ReceiveMessage`autorisation pour la file d'attente nommée`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"
   }]
}
```

Exemple 3 : accorder toutes les autorisations à deux Comptes AWS

L'exemple de politique suivant accorde deux Comptes AWS numéros différents (`111122223333`et`444455556666`) l'autorisation d'utiliser toutes les actions auxquelles Amazon SQS autorise un accès partagé pour la file d'attente nommée `123456789012/queue1` dans la région USA Est (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"
    }]
}
```

Exemple 4 : Accorder des autorisations inter-comptes à un rôle et à un nom d'utilisateur

L'exemple de politique suivant accorde`role1`, `username1` sous un Compte AWS numéro`111122223333`, l'autorisation entre comptes d'utiliser toutes les actions auxquelles Amazon SQS autorise un accès partagé pour la file d'attente `123456789012/queue1` nommée dans la région USA Est (Ohio).

Les autorisations intercompte ne s'appliquent pas aux actions suivantes :

- `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"
   }]
}
```

Exemple 5 : Accorder une autorisation à tous les utilisateurs

L'exemple de stratégie suivant accorde à tous les utilisateurs (anonymes) l'autorisation `ReceiveMessage` pour la file d'attente dénommée `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"
   }]
}
```

Exemple 6 : Accorder une autorisation limitée dans le temps à tous les utilisateurs

L'exemple de stratégie suivant accorde à tous les utilisateurs (anonymes) l'autorisation `ReceiveMessage` pour la file d'attente dénommée `111122223333/queue1`, mais seulement entre 12 h (midi) et 15 h le 31 janvier 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"
         }
      }
   }]
}
```

Exemple 7 : Accorder toutes les autorisations à tous les utilisateurs d'une plage d'adresses CIDR

L'exemple de stratégie suivant accorde à tous les utilisateurs (anonymes) l'autorisation d'utiliser toutes les actions Amazon SQS qui peuvent être partagées pour la file d'attente nommée `111122223333/queue1`, mais uniquement si la demande provient de la plage d'adresses 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"
         }
      }
   }]
}
```

Exemple 8 : Ajouter les utilisateurs de différentes plages d'adresses CIDR à une liste d'autorisations ou à une liste de blocage pour leur permettre ou les empêcher d'effectuer une action

L'exemple de stratégie suivant comporte deux instructions :

- La première instruction accorde à tous les utilisateurs (anonymes) de la plage d'adresses CIDR `192.0.2.0/24` (à l'exception de `192.0.2.188`) l'autorisation d'utiliser l'action `SendMessage` pour la file d'attente dénommée `111122223333`/queue1.
- La deuxième instruction empêche tous les utilisateurs (anonymes) de la plage d'adresses CIDR `12.148.72.0/23` d'utiliser la file d'attente en les ajoutant à une liste de blocage.

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"
            }
         }
      }]
   }
```

Utilisation de stratégies personnalisées avec le langage de la stratégie d'accès Amazon SQS

Pour accorder des autorisations de base (telles que `SendMessage`ou `ReceiveMessage`) basées uniquement sur un Compte AWS identifiant, vous n'avez pas besoin de rédiger une politique personnalisée. Utilisez plutôt l'action Amazon SQS. `AddPermission`

Pour autoriser ou refuser l'accès en fonction de conditions spécifiques, telles que l'heure de la demande ou l'adresse IP du demandeur, vous devez créer une politique Amazon SQS personnalisée et la télécharger à l'aide SetQueueAttributesde l'action.

Rubriques

- Architecture de contrôle d'accès Amazon SQS
- Flux de travail des processus de contrôle d'accès Amazon SQS
- Concepts clés du langage de la stratégie d'accès Amazon SQS
- Logique d'évaluation du langage de la stratégie d'accès Amazon SQS
- Relations entre les refus explicites et les refus par défaut dans le langage de la stratégie d'accès Amazon SQS
- Limites des politiques personnalisées d'Amazon SQS
- Exemples de langage de la stratégie d'accès Amazon SQS personnalisé

Architecture de contrôle d'accès Amazon SQS

Le schéma suivant décrit le système de contrôle d'accès pour vos ressources Amazon SQS.

1

Vous-même, le propriétaire de la ressource.

2

Vos ressources contenues dans le AWS service (par exemple, les files d'attente Amazon SQS).

3

Vos stratégies. Il est recommandé d'avoir une stratégie par ressource. Le AWS service fournit une API que vous utilisez pour télécharger et gérer vos politiques.

4

Les demandeurs et leurs demandes entrantes au service AWS .

5

Code d'évaluation du langage de la stratégie d'accès. Il s'agit de l'ensemble de code du AWS service qui évalue les demandes entrantes par rapport aux politiques applicables et détermine si le demandeur est autorisé à accéder à la ressource.

Flux de travail des processus de contrôle d'accès Amazon SQS

Le schéma suivant décrit le flux de travail général du contrôle d'accès avec le langage de la politique d'accès Amazon SQS.

1

Vous écrivez une stratégie Amazon SQS pour votre file d'attente.

2

Vous téléchargez votre politique sur AWS. Le AWS service fournit une API que vous utilisez pour télécharger vos politiques. Par exemple, vous utilisez l'action Amazon SQS `SetQueueAttributes` pour importer une stratégie pour une file d'attente Amazon SQS spécifique.

3

Quelqu'un envoie une demande d'utilisation de votre file d'attente Amazon SQS.

4

Amazon SQS examine toutes les stratégies Amazon SQS disponibles et détermine lesquelles sont applicables.

5

Amazon SQS évalue les stratégies et détermine si le demandeur est autorisé à utiliser votre file d'attente.

6

En fonction du résultat de l'évaluation de la stratégie, Amazon SQS renvoie une erreur `Access denied` au demandeur ou continue à traiter la demande.

Concepts clés du langage de la stratégie d'accès Amazon SQS

Pour écrire vos propres stratégies, vous devez être familiarisé avec le langage JSON et un certain nombre de concepts clés.

Autorisation

Résultat d'une Instruction dont l'Effet est défini sur `allow`.

Action

Activité que le Principal est autorisé à effectuer, généralement une demande à AWS.

Default-deny

Résultat d'une Instruction ne disposant d'aucun paramètre Autorisation ou Explicit-deny.

Condition

Tout détail ou restriction concernant une Autorisation. Les conditions typiques sont liées à la date et à l'heure, ainsi qu'aux adresses IP.

Effet

Résultat que vous souhaitez que la Instruction d'une Stratégie renvoie au moment de l'évaluation. Vous spécifiez la valeur `deny` ou `allow` lorsque vous écrivez la déclaration de stratégie. Trois résultats sont possibles lors de l'évaluation de stratégie : Default-deny, Autorisation ou Explicit-deny.

Explicit-deny

Résultat d'une Instruction dont l'Effet est défini sur `deny`.

Evaluation

Processus utilisé par Amazon SQS pour déterminer si une demande entrante doit être refusée ou autorisée en fonction d'une Stratégie.

Emetteur

Utilisateur qui écrit une Stratégie pour accorder des autorisations à une ressource. Par définition, l'émetteur est toujours le propriétaire de la ressource. AWS n'autorise pas les utilisateurs d'Amazon SQS à créer des politiques pour des ressources dont ils ne sont pas propriétaires.

Clé

Caractéristique spécifique à la base d'une restriction d'accès.

Autorisation

Concept consistant à autoriser ou refuser l'accès à une ressource à l'aide d'une Condition et d'une Clé.

Stratégie

Document jouant le rôle de conteneur pour une ou plusieurs déclarations.

Amazon SQS utilise la stratégie pour déterminer s'il convient d'autoriser un utilisateur à accéder à une ressource.

Principal

Utilisateur qui reçoit l'Autorisation dans la Stratégie.

Ressource

Objet auquel le Principal demande l'accès.

Instruction

Description formelle d'une autorisation unique, écrite dans le langage de la stratégie d'accès, dans le cadre d'un document de Stratégie plus vaste.

Demandeur

Utilisateur qui envoie une demande d'accès à une Ressource.

Logique d'évaluation du langage de la stratégie d'accès Amazon SQS

Au moment de l'évaluation, Amazon SQS détermine si une demande d'un utilisateur qui n'est pas le propriétaire de la ressource doit être autorisée ou refusée. La logique d'évaluation suit plusieurs règles de base :

- Par défaut, toutes les demandes d'utilisation de votre ressource ne provenant pas de vous-même sont refusées.
- La valeur Autorisation prévaut sur Default-deny.
- La valeur Explicit-deny prévaut sur allow.
- L'ordre dans lequel les stratégies sont évaluées n'a pas d'importance.

Le schéma suivant décrit en détail la façon dont Amazon SQS évalue les décisions relatives aux autorisations d'accès.

1 La décision commence par un refus par défaut (default-deny).

2 Le code d'application évalue toutes les stratégies qui s'appliquent à la demande (en se basant sur la ressource, le mandataire, l'action et les conditions). L'ordre dans lequel le code d'application évalue les stratégies n'a pas d'importance.

3 Le code d'application recherche une instruction explicit-deny qui peut s'appliquer à la demande. S'il

en trouve une, le code d'application renvoie une décision de type deny (refus) et le processus se termine.

4

En l'absence d'instruction explicit-deny (refus explicite), le code d'application recherche des instructions allow (autorisation) pouvant s'appliquer à la demande. S'il en trouve une, il renvoie une décision de type allow (autoriser) et le processus se termine (le service continue à traiter la demande).

5

Si aucune instruction allow n'est détectée, la décision finale est un refus (deny). En l'absence de refus explicite (explicit-deny) ou d'autorisation (allow), nous parlons d'un refus par défaut (default-deny).

Relations entre les refus explicites et les refus par défaut dans le langage de la stratégie d'accès Amazon SQS

Si une stratégie Amazon SQS ne s'applique pas directement à une demande, celle-ci se conclut par un Default-deny. Par exemple, si un utilisateur demande l'autorisation d'utiliser Amazon SQS, mais que la seule stratégie qui s'applique à l'utilisateur indique qu'il peut utiliser DynamoDB, la demande se conclut par un default-deny.

Si une condition de la déclaration n'est pas respectée, la demande se conclut par un default-deny. Si toutes les conditions d'une déclaration sont respectées, la demande se conclut par une décision de type Autorisation ou Explicit-deny, en fonction de la valeur de l'élément Effet de la stratégie. Les stratégies ne spécifient pas comment procéder si une condition n'est pas respectée. Le résultat par défaut est donc un default-deny dans ce cas. Supposons par exemple que vous souhaitez refuser les demandes provenant de l'Antarctique. Vous créez une stratégie Policy A1 qui autorise une demande uniquement si elle ne provient pas de l'Antarctique. Le schéma suivant illustre la stratégie Amazon SQS.

Si un utilisateur envoie une demande depuis les États-Unis, la condition est respectée (la demande ne provient pas de l'Antarctique) et la demande se conclut par une décision allow. Par contre, si un

utilisateur envoie une demande depuis l'Antarctique, la condition n'est pas respectée et la demande se conclut par défaut par un default-deny. Vous pouvez modifier le résultat et le faire passer en explicit-deny en écrivant une stratégie Policy A2 qui refuse explicitement toute demande provenant de l'Antarctique. Le schéma suivant illustre la politique.

Si un utilisateur envoie une demande depuis l'Antarctique, la condition est remplie et la demande se conclut par un explicit-deny.

Il est important de bien faire la distinction entre un default-deny et un explicit-deny, car une décision allow peut prévaloir sur le premier mais pas sur le second. Par exemple, la stratégie Policy B autorise les demandes si elles arrivent le 1er juin 2010. Le schéma suivant compare l'association de cette stratégie avec une stratégie Policy A1 et une stratégie Policy A2.

Dans le scénario 1, la stratégie Policy A1 se conclut par un default-deny et la stratégie Policy B par un allow, car la stratégie autorise les demandes qui arrivent le 1er juin 2010. L'autorisation allow de Policy B remplace le refus par défaut (default-deny) de Policy A1 et, par conséquent, la demande est autorisée.

Dans le scénario 2, la stratégie Policy B2 génère un explicit-deny et la stratégie Policy B génère une décision allow. Le refus explicite (explicit-deny) de Policy A2 remplace l'autorisation (allow) de Policy B et, par conséquent, la demande est refusée.

Limites des politiques personnalisées d'Amazon SQS

Accès intercomptes

Les autorisations intercompte ne s'appliquent pas aux actions suivantes :

- `AddPermission`
- `CancelMessageMoveTask`
- `CreateQueue`
- `DeleteQueue`
- `ListMessageMoveTask`
- `ListQueues`
- `ListQueueTags`
- `RemovePermission`
- `SetQueueAttributes`
- `StartMessageMoveTask`
- `TagQueue`
- `UntagQueue`

Clés de condition

Actuellement, Amazon SQS prend en charge uniquement un sous-ensemble limité des clés de condition disponibles dans IAM. Pour de plus amples informations, veuillez consulter Autorisations d'API Amazon SQS : référence des actions et ressources.

Exemples de langage de la stratégie d'accès Amazon SQS personnalisé

Voici des exemples classiques de stratégies d'accès Amazon SQS.

Exemple 1 : Accorder une autorisation à un compte

L'exemple de stratégie Amazon SQS suivant accorde au compte Compte AWS 111122223333 l'autorisation d'envoyer et de recevoir la file d'attente `queue2` détenue par le compte Compte 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"
   }]
}
```

Exemple 2 : Accorder une autorisation à un ou plusieurs comptes

L'exemple suivant de politique Amazon SQS donne un ou plusieurs Comptes AWS accès aux files d'attente détenues par votre compte pendant une période spécifique. Il est nécessaire d'écrire cette stratégie et de la télécharger dans Amazon SQS à l'aide de l'action `SetQueueAttributes`, car l'action `AddPermission` ne permet pas de spécifier une restriction de durée lors de l'octroi de l'accès à une file d'attente.

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"
         }
      }
   }]
}
```

Exemple 3 : donner l'autorisation aux demandes provenant d' EC2 instances Amazon

L'exemple suivant de politique Amazon SQS donne accès aux demandes provenant d'instances Amazon EC2 . Cet exemple repose sur l'exemple « Exemple 2 : Accorder une autorisation à un ou plusieurs comptes » : il restreint l'accès aux demandes envoyées avant le 30 juin 2009 à midi (UTC), ainsi qu'à la plage d'adresses IP `203.0.113.0/24`. Il est nécessaire d'écrire cette stratégie et de la télécharger dans Amazon SQS à l'aide de l'action `SetQueueAttributes`, car l'action `AddPermission` ne permet pas de spécifier une restriction d'adresse IP lors de l'octroi de l'accès à une file d'attente.

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"
            }
         }
      }]
}
```

Exemple 4 : Refus d'accès à un compte spécifique

L'exemple suivant de politique Amazon SQS refuse un Compte AWS accès spécifique à votre file d'attente. Cet exemple s'appuie sur l'exemple Exemple 1 : Accorder une autorisation à un compte « » : il refuse l'accès au spécifié Compte AWS. Il est nécessaire d'écrire cette stratégie et de la télécharger dans Amazon SQS à l'aide de l'action `SetQueueAttributes`, car l'action `AddPermission` ne permet pas de refuser l'accès à une file d'attente (elle permet uniquement d'accorder l'accès à une file d'attente).

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"
```

```
   }]
}
```

Exemple 5 : Refuser l'accès s'il n'émane pas d'un point de terminaison de VPC

L'exemple suivant de stratégie Amazon SQS restreint l'accès à `queue1` : 111122223333 peut effectuer les actions `SendMessage` et `ReceiveMessage` uniquement à partir de l'ID de point de terminaison d'un VPC `vpce-1a2b3c4d` (spécifié à l'aide de la condition `aws:sourceVpce`). Pour de plus amples informations, veuillez consulter Points de terminaison Amazon Virtual Private Cloud pour Amazon SQS.

> **Note**
>
> - La condition `aws:sourceVpce` ne requiert pas d'ARN pour la ressource du point de terminaison de VPC, uniquement l'ID du point de terminaison de VPC.
> - Vous pouvez modifier l'exemple suivant pour restreindre toutes les actions au point de terminaison d'un VPC spécifique en refusant toutes les actions Amazon SQS (`sqs:*`) dans la deuxième instruction. Toutefois, une telle déclaration de stratégie stipulerait que toutes les actions (y compris les actions administratives requises pour modifier les autorisations de la file d'attente) doivent être effectuées via le point de terminaison de VPC spécifique défini dans la stratégie, ce qui pourrait empêcher l'utilisateur de modifier les autorisations de la file d'attente par la suite.

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"
               }
            }
         }
      ]
}
```

Utilisation d'informations d'identification de sécurité temporaires avec Amazon SQS

En plus de créer des utilisateurs dotés de leurs propres identifiants de sécurité, IAM vous permet également d'octroyer des identifiants de sécurité temporaires à n'importe quel utilisateur, lui permettant ainsi d'accéder à vos AWS services et ressources. Vous pouvez gérer les utilisateurs qui possèdent des Comptes AWS. Vous pouvez également gérer les utilisateurs de votre système qui n'en ont pas Comptes AWS (utilisateurs fédérés). En outre, les applications que vous créez pour accéder à vos AWS ressources peuvent également être considérées comme des « utilisateurs ».

Vous pouvez utiliser ces informations d'identification de sécurité temporaires pour effectuer des demandes à Amazon SQS. Les bibliothèques d'API calculent la valeur de signature nécessaire en utilisant ces informations d'identification pour authentifier votre demande. Si vous envoyez des demandes en utilisant des informations d'identification expirées, Amazon SQS les rejette.

> **Note**
>
> Vous ne pouvez pas définir une stratégie en fonction d'informations d'identification temporaires.

Prérequis

1. Utilisez IAM pour créer des informations d'identification de sécurité temporaires :
   - Jeton de sécurité
   - ID de clé d'accès
   - Clé d'accès secrète
2. Préparez votre chaîne pour signer avec l'ID de clé d'accès temporaire et le jeton de sécurité.
3. Utilisez la clé d'accès secrète temporaire au lieu de votre propre clé d'accès secrète pour signer votre demande d'API de requête.

> **Note**
>
> Lorsque vous envoyez la demande d'API de requête signée, utilisez l'ID de clé d'accès temporaire au lieu de votre propre ID de clé d'accès, et incluez le jeton de sécurité. Pour plus d'informations sur la prise en charge par IAM des informations d'identification de sécurité temporaires, consultez la section Accorder un accès temporaire à vos AWS ressources dans le guide de l'utilisateur IAM.

Pour appeler une action d'API de requête Amazon SQS à l'aide d'informations d'identification de sécurité temporaires

1. Demandez un jeton de sécurité temporaire en utilisant AWS Identity and Access Management. Pour plus d'informations, consultez Création d'informations d'identification de sécurité temporaires pour activer l'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

   IAM renvoie un jeton de sécurité, un ID de clé d'accès et une clé d'accès secrète.
2. Préparez votre requête en utilisant l'ID de clé d'accès temporaire à la place de votre propre ID de clé d'accès, et incluez le jeton de sécurité. Signez votre demande à l'aide de la clé d'accès secrète temporaire au lieu de la vôtre.

3. Soumettez votre chaîne de requête signée avec l'ID de clé d'accès temporaire et le jeton de sécurité.

   L'exemple suivant montre comment utiliser des informations d'identification de sécurité temporaires pour authentifier une demande Amazon SQS. La structure de *AUTHPARAMS* dépend de la signature de la demande d'API. Pour plus d'informations, consultez Signing AWS API Requests dans le manuel Amazon Web Services General Reference.

```
https://sqs.us-east-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Attribute.1.Name=VisibilityTimeout
&Attribute.1.Value=40
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

   L'exemple suivant utilise des informations d'identification de sécurité temporaires pour envoyer deux messages à l'aide de l'action `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
```

Gestion de l'accès pour les files d'attente Amazon SQS chiffrées avec des politiques de moindre privilège

Vous pouvez utiliser Amazon SQS pour échanger des données sensibles entre les applications à l'aide du chiffrement côté serveur (SSE) intégré àAWS Key Management Service (KMS). Grâce à l'intégration d'Amazon SQS AWS KMS, vous pouvez gérer de manière centralisée les clés qui protègent Amazon SQS, ainsi que les clés qui protègent vos autres ressources. AWS

Plusieurs AWS services peuvent agir comme des sources d'événements qui envoient des événements à Amazon SQS. Pour permettre à une source d'événements d'accéder à la file d'attente cryptée Amazon SQS, vous devez configurer la file d'attente avec une clé gérée par le client AWS KMS . Utilisez ensuite la politique clé pour autoriser le service à utiliser les méthodes d' AWS KMS API requises. Le service requiert également des autorisations pour authentifier l'accès et permettre à la file d'attente d'envoyer des événements. Pour cela, vous pouvez utiliser une stratégie Amazon SQS, qui est basée sur les ressources que vous pouvez utiliser pour contrôler l'accès à la file d'attente Amazon SQS et à ses données.

Les sections suivantes fournissent des informations sur la façon de contrôler l'accès à votre file d'attente Amazon SQS chiffrée par le biais de la politique Amazon SQS et AWS KMS de la politique clé. Les stratégies décrites dans ce guide vous aideront à respecter le principe de moindre privilège.

Ce guide décrit également comment les stratégies basées sur les ressources résolvent le problème de député confus en utilisant les clés contextuelles de condition IAM globales `aws:SourceArn`, `aws:SourceAccount` et `aws:PrincipalOrgID`.

Rubriques

- Présentation
- Stratégie de clé respectant le principe du moindre privilège pour Amazon SQS
- Instructions de stratégie Amazon SQS relatives à la file d'attente de lettres mortes
- Prévention des problèmes de député confus entre services
- Utiliser IAM Access Analyzer pour examiner l'accès intercompte

Présentation

Dans cette rubrique, nous allons vous présenter un cas d'utilisation courant pour illustrer comment créer la stratégie de clé et la stratégie de file d'attente Amazon SQS. Ce cas d'utilisation est représenté dans l'image suivante.

Dans cet exemple, le producteur du message est une rubrique Amazon Simple Notification Service (SNS) configurée pour diffuser en éventail les messages vers votre file d'attente Amazon SQS chiffrée. Le consommateur de messages est un service informatique, tel qu'une AWS Lambdafonction, une instance Amazon Elastic Compute Cloud (EC2) ou un AWS Fargateconteneur. Votre file d'attente Amazon SQS est ensuite configurée pour envoyer les messages en échec à une file d'attente de lettres mortes (DLQ). Cela est utile pour le débogage de votre application ou de votre système de messagerie, car DLQs cela vous permet d'isoler les messages non consommés afin de déterminer pourquoi leur traitement a échoué. Dans la solution définie dans cette rubrique, un service de calcul tel qu'une fonction Lambda est utilisé pour traiter les messages stockés dans la file d'attente Amazon SQS. Si le destinataire du message se trouve dans un cloud privé virtuel (VPC), l'instruction de la stratégie `DenyReceivingIfNotThroughVPCE` incluse dans ce guide vous permet de limiter la réception des messages à ce VPC spécifique.

> **Note**
>
> Ce guide contient uniquement les autorisations IAM requises sous forme d'instructions de stratégie. Pour élaborer la politique, vous devez ajouter les instructions à votre politique Amazon SQS ou à votre politique AWS KMS clé. Ce guide ne fournit pas d'instructions sur la façon de créer la file d'attente Amazon SQS ou la AWS KMS clé. Pour savoir comment créer ces ressources, consultez les sections Création d'une file d'attente Amazon SQS et Création de clés.

La stratégie Amazon SQS définie dans ce guide ne permet pas de rediriger les messages directement vers la même file d'attente Amazon SQS ou vers une autre.

Stratégie de clé respectant le principe du moindre privilège pour Amazon SQS

Dans cette section, nous décrivons les autorisations de moindre privilège requises AWS KMS pour la clé gérée par le client que vous utilisez pour chiffrer votre file d'attente Amazon SQS. Avec ces autorisations, vous pouvez limiter l'accès aux seules entités prévues en implémentant le moindre privilège. La stratégie de clé doit comprendre les instructions de stratégie suivantes, que nous décrivons en détail ci-dessous :

- Accorder des autorisations d'administrateur à la AWS KMS clé
- Accorde l'accès en lecture seule aux métadonnées de clés
- Accorder des autorisations KMS Amazon SNS à Amazon SNS pour la publication de messages dans la file d'attente
- Permettre aux consommateurs de déchiffrer les messages de la file d'attente

Accorder des autorisations d'administrateur à la AWS KMS clé

Pour créer une AWS KMS clé, vous devez fournir des autorisations d' AWS KMS administrateur au rôle IAM que vous utilisez pour déployer la AWS KMS clé. Ces autorisations d'administrateur sont définies dans l'instruction de stratégie `AllowKeyAdminPermissions` suivante. Lorsque vous ajoutez cette déclaration à votre politique AWS KMS clé, assurez-vous de la *`<admin-role ARN>`* remplacer par le nom de ressource Amazon (ARN) du rôle IAM utilisé pour déployer la AWS KMS clé, gérer la AWS KMS clé, ou les deux. Il peut s'agir du rôle IAM de votre pipeline de déploiement ou du rôle d'administrateur de votre organisation dans 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**
>
> Dans une politique AWS KMS clé, la valeur de l'`Resource`élément doit être`*`, ce qui signifie « cette AWS KMS clé ». L'astérisque (`*`) identifie la AWS KMS clé à laquelle la politique clé est attachée.

Accorde l'accès en lecture seule aux métadonnées de clés

Pour accorder à d'autres rôles IAM un accès en lecture seule à vos métadonnées de clés, ajoutez l'instruction `AllowReadAccessToKeyMetaData` à votre stratégie de clé. Par exemple, l'instruction suivante vous permet de répertorier toutes les AWS KMS clés de votre compte à des fins d'audit. Cette instruction accorde à l'utilisateur AWS root un accès en lecture seule aux métadonnées clés. Par conséquent, tout principal IAM du compte peut avoir accès aux métadonnées de clés lorsque ses stratégies basées sur l'identité disposent des autorisations répertoriées dans l'instruction suivante : `kms:Describe*`, `kms:Get*` et `kms:List*`. Assurez-vous de les remplacer *`<account-ID>`* par vos propres informations.

```
{
  "Sid": "AllowReadAcesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
```

```
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}
```

Accorder des autorisations KMS Amazon SNS à Amazon SNS pour la publication de messages dans la file d'attente

Pour permettre à votre rubrique Amazon SNS de publier des messages dans votre file d'attente Amazon SQS chiffrée, ajoutez l'instruction de stratégie `AllowSNSToSendToSQS` à votre stratégie de clé. Cette déclaration accorde à Amazon SNS l'autorisation d'utiliser la AWS KMS clé pour publier dans votre file d'attente Amazon SQS. Assurez-vous de les remplacer *<account-ID>* par vos propres informations.

> **Note**
>
> La déclaration `Condition` in the limite l'accès au seul service Amazon SNS sur le même AWS compte.

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
```

```
  }
}
```

Permettre aux consommateurs de déchiffrer les messages de la file d'attente

L'instruction `AllowConsumersToReceiveFromTheQueue` suivante accorde au consommateur de messages Amazon SQS les autorisations requises pour déchiffrer les messages reçus de la file d'attente Amazon SQS chiffrée. Lorsque vous joignez la déclaration de politique, remplacez-la *`<consumer's runtime role ARN>`* par l'ARN du rôle d'exécution IAM du destinataire du message.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

Stratégie Amazon SQS du moindre privilège

Cette section décrit les stratégies de file d'attente Amazon SQS relatives au moindre privilège pour le cas d'utilisation couvert par ce guide (par exemple, Amazon SNS vers Amazon SQS). La stratégie définie est conçue pour empêcher tout accès involontaire en utilisant à la fois des instructions `Deny` et `Allow`. Les instructions `Allow` donnent accès à l'entité ou aux entités prévues. Les instructions `Deny` évitent que d'autres entités indésirables n'accèdent à la file d'attente Amazon SQS, tout en excluant l'entité prévue dans la condition de stratégie.

La stratégie Amazon SQS inclut les instructions suivantes, que nous décrivons en détail ci-dessous :

- Restreindre les autorisations de gestion Amazon SQS
- Restreindre les actions de file d'attente Amazon SQS pour l'organisation spécifiée
- Accorder des autorisations Amazon SQS aux consommateurs
- Application du chiffrement en transit

- Limiter la transmission de messages à une rubrique Amazon SNS spécifique
- (Facultatif) Restreindre la réception des messages au point de terminaison d'un VPC spécifique

Restreindre les autorisations de gestion Amazon SQS

L'instruction de stratégie `RestrictAdminQueueActions` suivante limite les autorisations de gestion Amazon SQS uniquement au(x) rôle(s) IAM que vous utilisez pour déployer la file d'attente, la gérer, ou les deux. Veillez à remplacer les *<placeholder values>* par vos propres informations. Spécifiez l'ARN du rôle IAM utilisé pour déployer la file d'attente Amazon SQS, ainsi que celui de tous ARNs les rôles d'administrateur devant disposer d'autorisations de gestion 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>"
      ]
    }
  }
}
```

Restreindre les actions de file d'attente Amazon SQS pour l'organisation spécifiée

Pour protéger vos ressources Amazon SQS contre tout accès externe (accès par une entité extérieure à votre organisation AWS), utilisez l'instruction suivante. Cette instruction limite l'accès à la file d'attente Amazon SQS pour l'organisation que vous spécifiez dans la `Condition`. Assurez-vous de le *<SQS queue ARN>* remplacer par l'ARN du rôle IAM utilisé pour déployer la file d'attente Amazon SQS, puis par *<org-id>* l'ID de votre organisation.

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

Accorder des autorisations Amazon SQS aux consommateurs

Pour recevoir des messages de la file d'attente Amazon SQS, vous devez fournir les autorisations nécessaires au consommateur des messages. L'instruction de stratégie suivante accorde au consommateur spécifié les autorisations requises pour consommer les messages de la file d'attente Amazon SQS. Lorsque vous ajoutez cette déclaration à votre politique Amazon SQS, assurez-vous de la `<consumer's IAM runtime role ARN>` remplacer par l'ARN du rôle d'exécution IAM utilisé par le consommateur et `<SQS queue ARN>` par l'ARN du rôle IAM utilisé pour déployer la file d'attente Amazon SQS.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": "<consumer's IAM execution role ARN>"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
```

```
    "sqs:DeleteMessage",
    "sqs:GetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>"
}
```

Pour éviter que d'autres entités ne reçoivent des messages de la file d'attente Amazon SQS, ajoutez l'instruction `DenyOtherConsumersFromReceiving` à la stratégie de file d'attente Amazon SQS. Cette instruction limite la consommation des messages au consommateur spécifié et n'autorise aucun autre consommateur à y avoir accès, même lorsque ses autorisations d'identité l'y autorisent. Assurez-vous de remplacer *<SQS queue ARN>* et *<consumer's runtime role ARN>* par vos propres informations.

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

Application du chiffrement en transit

L'instruction de stratégie `DenyUnsecureTransport` suivante oblige les consommateurs et les producteurs à utiliser des canaux sécurisés (connexions TLS) pour envoyer et recevoir des messages depuis la file d'attente Amazon SQS. Assurez-vous de le *<SQS queue ARN>* remplacer par l'ARN du rôle IAM utilisé pour déployer la file d'attente Amazon SQS.

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}
```

Limiter la transmission de messages à une rubrique Amazon SNS spécifique

L'instruction de stratégie `AllowSNSToSendToTheQueue` suivante permet à la rubrique Amazon SNS spécifiée d'envoyer des messages à la file d'attente Amazon SQS. Assurez-vous de le *<SQS queue ARN>* remplacer par l'ARN du rôle IAM utilisé pour déployer la file d'attente Amazon SQS *<SNS topic ARN>* et par l'ARN de la rubrique 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>"
    }
  }
}
```

L'instruction de stratégie `DenyAllProducersExceptSNSFromSending` suivante empêche les autres producteurs d'envoyer des messages à la file d'attente. Remplacez *<SQS queue ARN>* et *<SNS topic ARN>* par vos propres informations.

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(Facultatif) Restreindre la réception des messages au point de terminaison d'un VPC spécifique

Pour limiter la réception de messages au point de terminaison d'un VPC spécifique, ajoutez l'instruction de stratégie suivante à votre stratégie de file d'attente Amazon SQS. Cette instruction empêche un consommateur de messages de recevoir des messages de la file d'attente, sauf si les messages proviennent du point de terminaison d'un VPC souhaité. *<SQS queue ARN>*Remplacez-le par l'ARN du rôle IAM utilisé pour déployer la file d'attente Amazon SQS *<vpce_id>* et par l'ID du point de terminaison VPC.

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
```

```
  }
}
```

Instructions de stratégie Amazon SQS relatives à la file d'attente de lettres mortes

Ajoutez les instructions de stratégie suivantes, identifiées par leur ID d'instruction, à votre stratégie d'accès DLQ :

- `RestrictAdminQueueActions`
- `DenyQueueActionsOutsideOrg`
- `AllowConsumersToReceiveFromTheQueue`
- `DenyOtherConsumersFromReceiving`
- `DenyUnsecureTransport`

Outre l'ajout des instructions de stratégie précédentes à votre stratégie d'accès DLQ, vous devez également ajouter une instruction pour restreindre la transmission de messages aux files d'attente Amazon SQS, comme décrit dans la section suivante.

Restreindre la transmission de messages vers des files d'attente Amazon SQS

Pour restreindre l'accès aux files d'attente Amazon SQS provenant du même compte, ajoutez l'instruction de stratégie `DenyAnyProducersExceptSQS` suivante à la stratégie de file d'attente DLQ. Cette instruction ne limite pas la transmission de messages à une file d'attente spécifique, car vous devez déployer le DLQ avant de créer la file d'attente principale. Vous ne connaîtrez donc pas l'ARN Amazon SQS lorsque vous créerez le DLQ. Si vous devez limiter l'accès à une seule file d'attente Amazon SQS, modifiez `aws:SourceArn` dans la `Condition` avec l'ARN de votre file d'attente source Amazon SQS lorsque vous le connaîtrez.

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
```

```
    }
  }
}
```

> **Important**
>
> Les stratégies de file d'attente Amazon SQS définies dans ce guide ne limitent pas l'action `sqs:PurgeQueue` à un ou plusieurs rôles IAM spécifiques. L'action `sqs:PurgeQueue` vous permet de supprimer tous les messages de la file d'attente Amazon SQS. Vous pouvez également utiliser cette action pour modifier le format du message sans remplacer la file d'attente Amazon SQS. Lors du débogage d'une application, vous pouvez effacer la file d'attente Amazon SQS pour supprimer les messages potentiellement erronés. Lorsque vous testez l'application, vous pouvez générer un volume élevé de messages dans la file d'attente Amazon SQS, puis purger la file d'attente pour repartir à zéro avant de passer à la production. La raison pour laquelle cette action n'est pas limitée à un certain rôle est que ce rôle peut ne pas être connu lors du déploiement de la file d'attente Amazon SQS. Vous devrez ajouter cette autorisation à la stratégie basée sur l'identité du rôle pour pouvoir purger la file d'attente.

Prévention des problèmes de député confus entre services

Le problème de député confus est un problème de sécurité dans lequel une entité qui n'a pas l'autorisation d'effectuer une action peut contraindre une entité plus privilégiée à effectuer cette action. Pour éviter cela, AWS fournit des outils qui vous aident à protéger votre compte si vous fournissez à des tiers (comptes croisés) ou à d'autres AWS services (appelés interservices) un accès aux ressources de votre compte. Les instructions de stratégie de cette section peuvent vous aider à éviter le problème de député confus entre services.

L'usurpation d'identité entre services peut se produire lorsqu'un service (le service appelant) appelle un autre service (le service appelé). Le service appelant peut être manipulé pour utiliser ses autorisations afin d'agir sur les ressources d'un autre client de sorte qu'il n'y aurait pas accès autrement. Pour éviter ce problème, les stratégies basées sur les ressources définies dans cet article utilisent les clés contextuelles de condition IAM globales `aws:SourceArn`, `aws:SourceAccount` et `aws:PrincipalOrgID`. Cela limite les autorisations dont dispose un service pour une ressource spécifique, un compte spécifique ou une organisation spécifique dans AWS Organizations.

Utiliser IAM Access Analyzer pour examiner l'accès intercompte

Vous pouvez utiliser AWS IAM Access Analyzer pour examiner vos politiques de file d'attente Amazon SQS et vos politiques clés AWS KMS et vous avertir lorsqu'une file d'attente ou AWS KMS une clé Amazon SQS autorise l'accès à une entité externe. IAM Access Analyzer vous aide à identifier les ressources de votre organisation et de vos comptes partagés avec une entité située en dehors de la zone de confiance. Cette zone de confiance peut être un AWS compte ou l'organisation au sein d' AWS Organizations que vous spécifiez lorsque vous activez IAM Access Analyzer.

IAM Access Analyzer identifie les ressources partagées avec des acteurs externes en utilisant un raisonnement basé sur la logique pour analyser les politiques basées sur les ressources dans votre environnement. AWS Pour chaque instance d'une ressource qui est partagée en dehors de votre zone de confiance, Access Analyzer génère un résultat. Les résultats comprennent des renseignements sur l'accès et le principal externe à qui il est accordé. Révisez les résultats pour déterminer si l'accès est intentionnel et sûr, ou s'il est non intentionnel et représente un risque pour la sécurité. En cas d'accès involontaire, consultez la stratégie concernée et corrigez-la. Consultez ce billet de blog pour plus d'informations sur la manière dont AWS IAM Access Analyzer identifie les accès involontaires à vos ressources. AWS

Pour plus d'informations sur AWS IAM Access Analyzer, consultez la documentation d'AWS IAM Access Analyzer.

Autorisations d'API Amazon SQS : référence des actions et ressources

Vous pouvez utiliser le tableau ci-dessous comme référence lorsque vous configurez le Contrôle d’accès et que vous écrivez des stratégies d'autorisation que vous pouvez associer à une identité IAM. Le chaque action d'Amazon Simple Queue Service, les actions correspondantes pour lesquelles vous pouvez accorder des autorisations pour effectuer l'action et la AWS ressource pour laquelle vous pouvez accorder les autorisations.

Spécifiez les actions dans le champ `Action` de la stratégie, et la valeur des ressources dans le champ `Resource` de la stratégie. Pour spécifier une action, utilisez le préfixe `sqs:` suivi du nom de l'action (par exemple, `sqs:CreateQueue`).

Amazon SQS prend actuellement en charge les clés contextuelles de condition globales disponibles dans IAM.

API Amazon Simple Queue Service et autorisations requises pour les actions

AddPermission

Action(s) : `sqs:AddPermission`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

ChangeMessageVisibility

Action(s) : `sqs:ChangeMessageVisibility`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

ChangeMessageVisibilityBatch

Action(s) : `sqs:ChangeMessageVisibilityBatch`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

CreateQueue

Action(s) : `sqs:CreateQueue`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

DeleteMessage

Action(s) : `sqs:DeleteMessage`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

DeleteMessageBatch

Action(s) : `sqs:DeleteMessageBatch`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

DeleteQueue

Action(s) : `sqs:DeleteQueue`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

GetQueueAttributes

Action(s) : `sqs:GetQueueAttributes`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

GetQueueUrl

Action(s) : `sqs:GetQueueUrl`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

ListDeadLetterSourceQueues

Action(s) : `sqs:ListDeadLetterSourceQueues`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

ListQueues

Action(s) : `sqs:ListQueues`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

ListQueueTags

Action(s) : `sqs:ListQueueTags`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

PurgeQueue

Action(s) : `sqs:PurgeQueue`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

ReceiveMessage

Action(s) : `sqs:ReceiveMessage`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

RemovePermission

Action(s) : `sqs:RemovePermission`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

SendMessage et SendMessageBatch

Action(s) : `sqs:SendMessage`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

SetQueueAttributes

Action(s) : `sqs:SetQueueAttributes`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

TagQueue

Action(s) : `sqs:TagQueue`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

UntagQueue

Action(s) : `sqs:UntagQueue`

Ressource : `arn:aws:sqs:region:account_id:queue_name`

Journalisation et surveillance dans Amazon SQS

Amazon Simple Queue Service est intégré à AWS CloudTrailun service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un Service AWS. CloudTrail capture tous les appels d'API pour Amazon SQS sous forme d'événements. Les appels capturés incluent des appels provenant de la console Amazon SQS et des appels de code vers les opérations de l'API Amazon SQS. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande envoyée à Amazon SQS, l'adresse IP à partir de laquelle la demande a été effectuée, la date à laquelle elle a été faite et des informations supplémentaires.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l’identité permettent de déterminer :

- Si la demande a été effectuée avec des informations d’identification d’utilisateur root ou d’utilisateur root.
- Si la demande a été faite au nom d'un utilisateur du centre d'identité IAM.
- Si la demande a été effectuée avec les informations d’identification de sécurité temporaires d’un rôle ou d’un utilisateur fédéré.
- Si la requête a été effectuée par un autre Service AWS.

CloudTrail est actif dans votre compte Compte AWS lorsque vous créez le compte et vous avez automatiquement accès à l'historique des CloudTrail événements. L'historique des CloudTrail événements fournit un enregistrement consultable, consultable, téléchargeable et immuable des 90 derniers jours des événements de gestion enregistrés dans un. Région AWS Pour plus d'informations, consultez la section Utilisation de l'historique des CloudTrail événements dans le guide de AWS CloudTrail l'utilisateur. La consultation de CloudTrail l'historique des événements est gratuite.

Pour un enregistrement continu des événements de vos 90 Compte AWS derniers jours, créez un magasin de données sur les événements de Trail ou CloudTrailLake.

CloudWatch Alarmes Amazon

Surveillez une seule métrique sur une période que vous spécifiez et effectuez une ou plusieurs actions en fonction de la valeur de la métrique par rapport à un seuil défini sur plusieurs périodes. Par exemple, vous pouvez configurer une CloudWatch alarme pour envoyer une notification à une rubrique Amazon SNS ou déclencher une action pour envoyer un message à une file d'attente Amazon SQS. CloudWatch les alarmes n'exécutent pas d'actions simplement parce qu'elles sont dans un état spécifique ; l'état doit changer et rester dans cet état pendant un certain nombre de périodes définies.

Pour plus d'informations, consultez Création d' CloudWatch alarmes pour les métriques Amazon SQS et Création d'alarmes pour les files d'attente de lettres mortes à l'aide d'Amazon CloudWatch.

Amazon CloudWatch Logs

Surveillez, stockez et accédez aux fichiers journaux liés à Amazon SQS en configurant vos applications ou les fonctions Lambda qui traitent les messages pour envoyer des journaux à Logs. CloudWatch Vous pouvez utiliser ces journaux pour analyser le traitement des messages, résoudre les problèmes et surveiller les performances de vos flux de travail Amazon SQS.

Pour de plus amples informations, veuillez consulter Journalisation des appels d'API Amazon Simple Queue Service à l'aide de AWS CloudTrail.

CloudWatch Événements Amazon

Utilisez Amazon CloudWatch Events pour détecter des modifications ou des événements spécifiques dans votre AWS environnement et les acheminer vers une file d'attente Amazon SQS. Cela vous permet de capturer des données d'événements, de déclencher des flux de travail ou de stocker des événements pour les traiter ultérieurement.

Pour plus d'informations, consultez Automatiser les notifications envoyées par les AWS services à Amazon SQS à l'aide d'Amazon EventBridge ce guide et découvrez EventBridge l'évolution d'Amazon CloudWatch Events dans le guide de EventBridge l'utilisateur Amazon.

AWS CloudTrail Journaux

CloudTrail capture un enregistrement détaillé des actions effectuées sur Amazon SQS par les utilisateurs, les rôles ou. Services AWS Ces journaux vous permettent de suivre les appels d'API,

tels que `SendMessageReceiveMessage`, ou `DeleteQueue`, et fournissent des informations clés telles que l'auteur de la demande, le moment où elle s'est produite et l'adresse IP d'origine.

Pour de plus amples informations, veuillez consulter Journalisation des appels d'API Amazon Simple Queue Service à l'aide de AWS CloudTrail.

AWS Trusted Advisor

Trusted Advisor utilise les meilleures pratiques développées par les AWS clients pour optimiser votre utilisation d'Amazon SQS. Il passe en revue vos files d'attente Amazon SQS et fournit des recommandations pratiques pour renforcer la sécurité, améliorer la fiabilité du traitement des messages et réduire les coûts. Par exemple, il peut suggérer d'activer les files d'attente contenant des lettres mortes ou d'améliorer vos politiques d'accès aux files d'attente afin de garantir la sécurité des opérations.

Pour plus d'informations, consultez AWS Trusted Advisor dans le Guide de l'utilisateur Support .

CloudTrail sentiers

Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Tous les sentiers créés à l'aide du AWS Management Console sont multirégionaux. Vous ne pouvez créer un journal de suivi en une ou plusieurs régions à l'aide de l' AWS CLI. Il est recommandé de créer un parcours multirégional, car vous capturez l'activité dans l'ensemble Régions AWS de votre compte. Si vous créez un journal de suivi pour une seule région, il convient de n'afficher que les événements enregistrés dans le journal de suivi pour une seule région Région AWS. Pour plus d'informations sur les journaux de suivi, consultez Créez un journal de suivi dans vos Compte AWS et Création d'un journal de suivi pour une organisation dans le AWS CloudTrail Guide de l'utilisateur.

Vous pouvez envoyer une copie de vos événements de gestion en cours dans votre compartiment Amazon S3 gratuitement CloudTrail en créant un journal. Toutefois, des frais de stockage Amazon S3 sont facturés. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification. Pour obtenir des informations sur la tarification Amazon S3, consultez Tarification Amazon S3.

CloudTrail Stockages de données sur les événements du lac

CloudTrail Lake vous permet d'exécuter des requêtes SQL sur vos événements. CloudTrail Lake convertit les événements existants au format JSON basé sur les lignes au format Apache ORC. ORC est un format de stockage en colonnes qui est optimisé pour une récupération rapide des données. Les événements sont agrégés dans des magasins de données d'événement. Ceux-ci constituent des collections immuables d'événements basées sur des critères que vous

sélectionnez en appliquant des sélecteurs d'événements avancés. Les sélecteurs que vous appliquez à un magasin de données d'événement contrôlent les événements qui persistent et que vous pouvez interroger. Pour plus d'informations sur CloudTrail Lake, consultez la section Travailler avec AWS CloudTrail Lake dans le guide de AWS CloudTrail l'utilisateur.

CloudTrail Les stockages et requêtes de données sur les événements de Lake entraînent des coûts. Lorsque vous créez un magasin de données d'événement, vous choisissez l'option de tarification que vous voulez utiliser pour le magasin de données d'événement. L'option de tarification détermine le coût d'ingestion et de stockage des événements, ainsi que les périodes de conservation par défaut et maximale pour le magasin de données d'événement. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification.

Journalisation des appels d'API Amazon Simple Queue Service à l'aide de AWS CloudTrail

CloudTrail vous permet de consigner et de surveiller les opérations Amazon SQS à l'aide de deux types d'événements : les événements liés aux données et les événements de gestion. Cela facilite le suivi et l'audit de l'activité Amazon SQS sur votre compte.

Événements liés aux données Amazon SQS dans CloudTrail

Les événements de données fournissent des informations sur les opérations de ressources effectuées sur ou dans une ressource (par exemple, l'envoi de messages à un objet Amazon SQS). Ils sont également connus sous le nom opérations de plans de données. Les événements de données sont souvent des activités dont le volume est élevé. Par défaut, CloudTrail n'enregistre pas les événements liés aux données. L'historique des CloudTrail événements n'enregistre pas les événements liés aux données.

Des frais supplémentaires s'appliquent pour les événements de données. Pour plus d'informations sur la CloudTrail tarification, consultez la section AWS CloudTrail Tarification.

Vous pouvez enregistrer les événements de données pour les types de ressources Amazon SQS à l'aide de la CloudTrail console ou des AWS CLI opérations d' CloudTrail API. Pour plus d'informations sur la façon de journaliser les événements de données, consultez Journalisation des événements de données avec la AWS Management Console et Journalisation des événements de données avec l' AWS Command Line Interface dans le Guide de l'utilisateur AWS CloudTrail .

Pour enregistrer les événements liés aux données Amazon SQS CloudTrail, vous devez utiliser des sélecteurs d'événements avancés pour configurer les ressources ou actions Amazon SQS

spécifiques que vous souhaitez enregistrer. Incluez le type de ressource `AWS::SQS::Queue`pour capturer les actions liées à la file d'attente. Vous pouvez affiner encore vos préférences de journalisation en utilisant des filtres tels que `eventName` (tels que `SendMessage`les événements). Pour plus d'informations, consultez AdvancedEventSelector dans la Référence d'API CloudTrail .

Type d'événement de données (console)	valeur resources.type	Données APIs enregistrées sur CloudTrail
File d'attente Amazon SQS	`AWS::SQS::Queue`	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch • GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

Utilisez des sélecteurs d'événements avancés pour filtrer les champs et n'enregistrer que les événements importants. Pour plus d'informations sur ces champs, consultez AdvancedFieldSelector dans la Référence d'API AWS CloudTrail .

Événements de gestion Amazon SQS dans CloudTrail

Les événements de gestion fournissent des informations sur les opérations de gestion effectuées sur les ressources de votre Compte AWS. Ils sont également connus sous le nom opérations de plan de contrôle. Par défaut, CloudTrail enregistre les événements de gestion.

Amazon SQS enregistre les opérations du plan de contrôle suivantes en CloudTrail tant qu'événements de gestion.

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTasks
- PurgeQueue
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Exemple d'événement Amazon SQS

Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'opération d'API demandée, la date et l'heure de l'opération, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics. Les événements n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant montre un CloudTrail événement illustrant l'`SendMessage`opération.

```
{
  "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**
>
> L'`ListQueues`opération est un cas unique car elle n'agit pas sur une ressource spécifique. Par conséquent, le champ ARN n'inclut pas de nom de file d'attente et utilise un caractère générique (*) à la place.

Pour plus d'informations sur le contenu des CloudTrail enregistrements, voir le contenu des CloudTrail enregistrements dans le Guide de AWS CloudTrail l'utilisateur.

Surveillance des files d'attente Amazon SQS à l'aide de CloudWatch

Amazon SQS et Amazon CloudWatch sont intégrés afin que vous puissiez les utiliser CloudWatch pour consulter et analyser les métriques de vos files d'attente Amazon SQS. Vous pouvez consulter et analyser les métriques de vos files d'attente depuis la console Amazon SQS, CloudWatch la console, en utilisant ou en utilisant AWS CLIl'API. CloudWatch Vous pouvez également définir des CloudWatch alarmes pour les métriques Amazon SQS.

CloudWatch les métriques relatives à vos files d'attente Amazon SQS sont automatiquement collectées et transmises à CloudWatch intervalles d'une minute. Ces statistiques sont collectées sur toutes les files d'attente qui répondent aux CloudWatch directives relatives à l'activité. CloudWatch considère qu'une file d'attente est active pendant six heures au maximum si elle contient des messages ou si une action y accède.

Lorsqu'une file d'attente Amazon SQS est inactive pendant plus de six heures, le service Amazon SQS est considéré comme inactif et cesse de fournir des métriques au service. CloudWatch Les données manquantes, ou les données représentant zéro, ne peuvent pas être visualisées dans les CloudWatch métriques d'Amazon SQS pendant la période pendant laquelle votre file d'attente Amazon SQS était inactive.

> **Note**
>
> - Une file d'attente Amazon SQS peut être activée lorsque l'utilisateur qui appelle une API depuis la file d'attente n'est pas autorisé et que la demande échoue.

- La console Amazon SQS exécute un appel d'`GetQueueAttributes`API lorsque la page de la file d'attente est ouverte. La demande `GetQueueAttributes` d'API active la file d'attente.
- Un délai pouvant aller jusqu'à 15 minutes se produit dans CloudWatch les métriques lorsqu'une file d'attente est activée à partir d'un état inactif.
- Les métriques Amazon SQS indiquées dans le document sont gratuites. CloudWatch Elles sont fournies dans le cadre du service Amazon SQS.
- CloudWatch les métriques sont prises en charge à la fois pour les files d'attente standard et FIFO.

Accès aux CloudWatch métriques pour Amazon SQS

Amazon SQS et Amazon CloudWatch sont intégrés afin que vous puissiez les utiliser CloudWatch pour consulter et analyser les métriques de vos files d'attente Amazon SQS. Vous pouvez consulter et analyser les métriques de vos files d'attente depuis la console Amazon SQS, CloudWatch la console, en utilisant ou en utilisant AWS CLIl'API. CloudWatch Vous pouvez également définir des CloudWatch alarmes pour les métriques Amazon SQS.

Utilisation de la console Amazon SQS

Utilisez la console Amazon SQS pour accéder aux métriques d'un maximum de 10 files d'attente Amazon SQS et les analyser.

1. Connectez-vous à la console Amazon SQS.
2. Dans la liste des files d'attente, sélectionnez (cochez) les files d'attente pour lesquelles vous souhaitez accéder aux métriques. Vous pouvez afficher les métriques de jusqu'à 10 files d'attente.
3. Sélectionnez l'onglet Monitoring (Surveillance).

   Plusieurs graphiques sont affichés dans la section métriques SQS.
4. Pour comprendre un graphique particulier, passez la souris sur

   en regard du graphique souhaité ou consultez CloudWatch Métriques disponibles pour Amazon SQS.
5. Pour modifier la plage de temps de tous les graphiques en même temps, sélectionnez la plage de temps souhaitée dans Plage de temps (par exemple, Dernière heure).

6. Pour afficher les statistiques supplémentaires d'un graphique individuel, sélectionnez ce dernier.
7. Dans la boîte de dialogue Détails de supervision de CloudWatch , sélectionnez une statistique, (par exemple, Somme). Pour obtenir une liste des statistiques prises en charge, consultez la section CloudWatch Métriques disponibles pour Amazon SQS.
8. Pour modifier la plage de temps et l'intervalle de temps affichés par un graphique individuel (par exemple, pour afficher une plage de temps des 24 dernières heures au lieu des 5 dernières minutes, ou pour afficher une période de toutes les heures au lieu de toutes les 5 minutes), la boîte de dialogue du graphique étant toujours affichée, définissez la plage de temps souhaitée dans Plage de temps (par exemple, 24 dernières heures). Pour Période, sélectionnez la période souhaitée dans la plage de temps spécifiée (par exemple, 1 heure). Lorsque vous en avez terminé avec le graphe, cliquez sur Fermer.
9. (Facultatif) Pour utiliser des CloudWatch fonctionnalités supplémentaires, dans l'onglet Surveillance, choisissez Afficher toutes les CloudWatch mesures, puis suivez les instructions de la Utilisation de la CloudWatch console Amazon procédure.

Utilisation de la CloudWatch console Amazon

Utilisez la CloudWatch console pour accéder aux métriques Amazon SQS et les analyser.

1. Connectez-vous à la console CloudWatch.
2. Dans le volet de navigation, choisissez Métriques.
3. Sélectionnez l'espace de nom de métrique SQS.

4. Sélectionnez la dimension de métrique Queue Metrics.

5. Vous pouvez désormais examiner vos métriques Amazon SQS :

   - Pour trier les métriques, utilisez l'en-tête de colonne.
   - Pour représenter graphiquement une métrique, cochez la case en regard de la métrique.
   - Pour filtrer par métrique, choisissez le nom de la métrique, puis Add to search (Ajouter à la recherche).

Pour plus d'informations et des options supplémentaires, consultez Graph Metrics et Using Amazon CloudWatch Dashboards dans le guide de l' CloudWatch utilisateur Amazon.

À l'aide du AWS Command Line Interface

Pour accéder aux métriques Amazon SQS à l'aide de AWS CLI, exécutez la `get-metric-statistics` commande.

Pour plus d'informations, consultez Obtenir des statistiques pour une métrique dans le guide de CloudWatch l'utilisateur Amazon.

Utilisation de l' CloudWatch API

Pour accéder aux métriques Amazon SQS à l'aide de l' CloudWatch API, utilisez l'`GetMetricStatistics`action.

Pour plus d'informations, consultez Obtenir des statistiques pour une métrique dans le guide de CloudWatch l'utilisateur Amazon.

Création d' CloudWatch alarmes pour les métriques Amazon SQS

CloudWatch vous permet de déclencher des alarmes lorsqu'une métrique atteint un seuil spécifié. Par exemple, vous pouvez créer une alarme pour la métrique `NumberOfMessagesSent`. Par exemple, si plus de 100 messages sont envoyés à la file d'attente `MyQueue` en 1 heure, une notification par e-mail est envoyée. Pour plus d'informations, consultez la section Création d' CloudWatch alarmes Amazon dans le guide de CloudWatch l'utilisateur Amazon.

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Choisissez Alarmes, puis Créer une alarme.
3. Dans la section Sélectionner une métrique de la boîte de dialogue Créer une alarme, choisissez Parcourir les métriques, SQS.
4. Pour SQS > Metrics de file d'attente, choisissez le QueueNamenom de la métrique pour laquelle vous souhaitez définir une alarme, puis choisissez Next. Pour obtenir la liste des métriques disponibles, consultez la section CloudWatch Métriques disponibles pour Amazon SQS.

   Dans l'exemple suivant, la sélection est destinée à une alarme pour la métrique `NumberOfMessagesSent` pour la file d'attente `MyQueue`. L'alarme se déclenche lorsque le nombre de messages envoyés dépasse 100.
5. Dans la section Définir une alarme de la boîte de dialogue Créer une alarme, procédez comme suit :

   a. Sous Seuil de l'alarme, tapez un Nom et une Description pour l'alarme.

b. Définissez is sur > 100.

c. Définissez pour sur 1 out of 1 datapoints (1 sur 1 point de données).

d. Sous Aperçu de l'alarme, définissez Période sur 1 heure.

e. Définissez Statistique sur Standard, Somme.

f. Sous Actions, définissez Chaque fois que cette alarme, sur L'état est ALARME.

   Si vous souhaitez CloudWatch envoyer une notification lorsque l'alarme est déclenchée, sélectionnez une rubrique Amazon SNS existante ou choisissez Nouvelle liste et entrez les adresses e-mail séparées par des virgules.

   > **Note**
   >
   > Si vous créez une rubrique Amazon SNS, les adresses e-mail doivent être vérifiées avant que vous ne puissiez recevoir des notifications. Si l'état de l'alarme change avant la vérification des adresses e-mail, les notifications ne sont pas remises.

6. Sélectionnez Create Alarm (Créer une alerte).

   L'alarme est créée.

CloudWatch Métriques disponibles pour Amazon SQS

Amazon SQS envoie les métriques suivantes à. CloudWatch

> **Note**
>
> Pour certaines métriques, le résultat est approximatif en raison de l'architecture distribuée d'Amazon SQS. Dans la plupart des cas, le nombre devrait être proche du nombre réel de messages dans la file d'attente.

Métriques Amazon SQS

Amazon SQS publie automatiquement les métriques opérationnelles sur Amazon CloudWatch sous l'`AWS/SQS`espace de noms. Ces indicateurs vous aident à surveiller l'état et les performances des files d'attente. En raison de la nature distribuée de SQS, de nombreuses valeurs sont approximatives, mais suffisamment précises pour la plupart des décisions opérationnelles.

> **Note**
>
> - Toutes les métriques émettent des valeurs non négatives uniquement lorsque la file d'attente est active.
> - Certaines métriques (telles que`SentMessageSize`) ne sont pas émises tant qu'au moins un message n'est envoyé.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateAgeOfOldestMessage`	L'âge du plus vieux message non traité de la file d'attente.	Secondes	Signalé si la file d'attente contient au moins un message actif.	• Pour les files d'attente standard, si un message est reçu trois fois ou plus et qu'il n'est pas supprimé, SQS le déplace en fin de file d'attente. La métrique reflète ensuite l'âge du message suivant qui n'a pas dépassé le seuil de réception. Cette réorganisation se produit même lorsqu'une politique de redrive est en place. • Les messages contenant des pilules empoisonnées (ceux reçus à plusieurs reprises mais jamais supprimés) sont exclus de cette métrique jusqu'à ce qu'ils soient traités avec succès. • Lorsqu'un message est déplacé vers une DLQ après avoir dépassé

Métrique	Description	Unités	Signaler un comportement	Remarques clés
				le `maxReceiveCount` , l'âge est réinitialisé. Dans ce cas, la métrique du DLQ reflète l'heure à laquelle le message a été déplacé, et non la date à laquelle il a été initialement envoyé. • Les files d'attente FIFO ne réorganisent pas les messages pour préserver l'ordre. Un message défaillant bloque son groupe de messages jusqu'à ce qu'il soit supprimé ou qu'il expire. Si un DLQ est configuré, le message y est envoyé une fois le seuil de réception atteint.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateNumberOfGroupsWithInflightMessages`	Pour FIFO uniquement. Le nombre de groupes de messages contenant un ou plusieurs messages en cours de vol.	Nombre	Signalé si la file d'attente FIFO est active.	• Un message est considéré comme en cours de vol une fois qu'il a été reçu de la file d'attente par un consommateur, mais qu'il n'a pas encore été supprimé ou expiré. • Cette métrique vous aide à résoudre les problèmes et à optimiser le débit des files d'attente FIFO. Les valeurs élevées indiquent généralement une forte simultanéité. • Si la file d'attente a un important arriéré et que cette valeur reste faible, envisagez de redimensionner le nombre de clients ou d'augmenter le nombre de groupes de messages actifs. • Pour le débit et les limites en vol, voir. Quotas Amazon SQS

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateNumberOfMessagesDelayed`	Nombre de messages de la file d'attente qui sont retardés et ne peuvent pas être récupérés immédiatement.	Nombre	Signalé si des messages différés existent dans la file d'attente.	• S'applique aux files d'attente configurées avec un délai par défaut et aux messages individuels envoyés avec un `DelaySeconds` paramètre . • Les messages différés restent cachés aux consommateurs jusqu'à l'expiration de leur délai, ce qui peut avoir une incidence sur le nombre de files d'attente ou le débit perçus.
`ApproximateNumberOfMessagesNotVisible`	Le nombre de messages en vol qui ont été reçus mais qui n'ont pas encore été supprimés ou ont expiré.	Nombre	Signalé si des messages existent en vol.	• Les messages entrent dans l'état de vol après avoir été envoyés à un consommateur via l'`ReceiveMessage` API. • Ces messages sont temporairement masqués aux autres consommateurs pendant la période d'expiration du délai de visibilité. • Utilisez cette métrique pour suivre les retards de traitement des messages ou les clients bloqués.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateNumberOfMessagesVisible`	Le nombre de messages actuellement disponibles pour la récupération et le traitement.	Nombre	Signalé si la file d'attente est active.	• Reflète le retard de traitement actuel dans la file d'attente. • Il n'existe aucune limite stricte quant au nombre de messages pouvant être accumulés, mais ils sont soumis à la période de rétention configurée dans la file d'attente. • Une valeur constamment élevée peut indiquer des consommateurs sous-approvisionnés ou une logique de traitement bloquée.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`NumberOfEmptyReceives` [1]	Le nombre d'appels d'ReceiveMessageAPI n'ayant renvoyé aucun message.	Nombre	Signalé lors des opérations de réception.	• Cette métrique peut aider à identifier les inefficiences liées au comportement des utilisateurs lors des sondages ou la sous-utilisation des instances destinées aux consommateurs. • Des valeurs élevées peuvent apparaître lorsque la file d'attente est vide, lorsque le consommateur utilise des interrogations courtes ou lorsque les messages sont traités plus rapidement qu'ils ne sont produits. • Il ne s'agit pas d'un indicateur précis de l'état de la file d'attente. Il reflète le comportement côté service et peut inclure de nouvelles tentatives.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`NumberOfDeduplicatedSentMessages`	Pour FIFO uniquement. Nombre de messages envoyés qui ont été dédupliqués et qui n'ont pas été ajoutés à la file d'attente.	Nombre	Signalé si `MessageDeduplicationId` des valeurs ou du contenu dupliqués sont détectés.	• SQS déduplique les messages en fonction du hachage `MessageDeduplicationId` ou du hachage basé sur le contenu (si activé). • Une valeur élevée peut indiquer qu'un producteur envoie à plusieurs reprises le même message dans le délai de déduplication de 5 minutes. • Utilisez cette métrique pour résoudre les problèmes liés à la logique redondante du producteur ou pour vérifier que la déduplication fonctionne comme prévu.

Métrique	Description	Unités	Signaler un comportement	Remarques clés

`NumberOfMessagesDeleted` [1]	Le nombre de messages supprimés de la file d'attente avec succès.	Nombre	Signalé pour chaque demande de suppression avec un identifiant de réception valide.	• Cette métrique compte toutes les opérations de suppression réussies, même si le même message est supprimé plusieurs fois.
				• Les raisons les plus courantes justifiant higher-than-expected les valeurs sont les suivantes :
				  • Suppressions multiples du même message à l'aide de différentes poignées de réception, une fois le délai de visibilité expiré et le message reçu à nouveau.
				  • Suppressions dupliquées à l'aide du même descripteur de réception, qui renvoient toujours un statut de réussite et incrémentent la métrique.
				• Utilisez cet indicateur pour suivre le succès du traitement des messages, mais ne le considérez pas comme le nombre exact de messages uniques supprimés.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`NumberOfMessagesReceived` [1]	Le nombre de messages renvoyés par l'ReceiveMessageAPI.	Nombre	Signalé lors des opérations de réception.	• Cela inclut tous les messages renvoyés aux consommateurs, y compris ceux qui sont ensuite renvoyés dans la file d'attente en raison de l'expiration du délai de visibilité. • Un seul message peut être reçu plusieurs fois s'il n'est pas supprimé, ce qui peut entraîner un dépassement de cette métrique par rapport au nombre de messages envoyés. • Utilisez-le pour suivre l'activité des consommateurs, mais ne le considérez pas comme le nombre de messages uniques traités.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`NumberOfMessagesSent` [1]	Le nombre de messages ajoutés avec succès à une file d'attente.	Nombre	Signalé pour chaque envoi manuel réussi.	• Les appels manuels vers `SendMessage` ou vers un DLQ `SendMessageBatch` sont comptabilisés, y compris ceux qui ciblent directement un DLQ. • Les messages qui sont automatiquement déplacés vers une DLQ après avoir dépassé le ne `maxReceiveCount` sont pas inclus dans cette métrique. • Par conséquent, il `NumberOfMessagesSent` peut être inférieur à, `NumberOfMessagesReceived` en particulier si les politiques de redrive déplacent de nombreux messages vers les DLQs coulisses.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`SentMessageSize` [1]	Taille des messages envoyés avec succès à la file d'attente.	Octets	Non émis tant qu'au moins un message n'a pas été envoyé.	• Cette métrique n'apparaîtra pas dans la CloudWatch console tant que la file d'attente n'aura pas reçu son premier message. • Utilisez cette métrique pour suivre la taille de chaque message en octets. Cela est utile pour analyser les tendances de la charge utile ou estimer le coût du débit. • La taille maximale des messages pour SQS est de 1 MiB.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateNumberOfNoisyGroups`	Nombre de groupes de messages considérés comme bruyants dans une file d'attente normale. Un groupe de messages bruyant représente un locataire voisin bruyant d'une file d'attente à locataires multiples.	Nombre	Une valeur non négative est signalée si la file d'attente est active.	• Permet d'identifier les problèmes potentiels liés au bruit des voisins dans les environnements à locataires multiples en suivant les groupes de messages consommant des ressources disproportionnées. • Utilisez cette métrique pour définir des alarmes qui se déclenchent lorsque le nombre de groupes bruyants dépasse votre seuil acceptable, ce qui indique des problèmes potentiels d'équité des files d'attente.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateNumberOfMessagesVisibleInQuietGroups`	Le nombre de messages visibles, à l'exception des messages provenant de groupes de messages bruyants.	Nombre	Une valeur non négative est signalée si la file d'attente est active.	• Fournit une visibilité sur le backlog des files d'attente pour les groupes de messages à débit standard, à l'exception des messages provenant de voisins bruyants. • Permet d'identifier le véritable retard de traitement pour les groupes de messages classiques en filtrant l'impact des voisins bruyants.
`ApproximateNumberOfMessagesNotVisibleInQuietGroups`	Le nombre de messages en cours, à l'exception des messages provenant de groupes de messages bruyants.	Nombre	Une valeur non négative est signalée si la file d'attente est active.	• Suit les messages en cours de vol (en cours de traitement mais pas encore supprimés) provenant de groupes de messages bien gérés. • Utilisez cette métrique pour surveiller le débit de traitement des groupes de messages normaux et détecter les blocages de traitement qui ne sont pas causés par des voisins bruyants.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateNumberOfMessagesDelayedInQuietGroups`	Le nombre de messages, à l'exception des messages provenant de groupes de messages bruyants, qui sont retardés et ne peuvent pas être lus immédiatement. Les messages différés apparaissent lorsque la file d'attente est configurée comme file d'attente différée ou lorsqu'un message a été envoyé avec un paramètre de délai.	Nombre	Une valeur non négative est signalée si la file d'attente est active.	• Permet de contrôler l'arriéré de messages différés provenant de groupes de messages présentant des modèles de débit normaux ou attendus (par opposition aux groupes à volume élevé ou bruyants) • Utile pour comprendre les futures exigences de traitement et planifier la capacité pour les charges de travail typiques.

Métrique	Description	Unités	Signaler un comportement	Remarques clés
`ApproximateAgeOfOldestMessageInQuietGroups`	L'âge du plus vieux message non supprimé de la file d'attente, à l'exception des messages provenant de groupes de messages bruyants.	Secondes	Une valeur non négative est signalée si la file d'attente est active.	• Utilisé pour surveiller la conformité aux SLA et détecter les goulots d'étranglement liés au traitement dans les groupes de messages présentant des modèles de débit normaux ou attendus (par opposition aux groupes de messages à volume élevé ou bruyants qui pourraient fausser la métrique). • Utilisez cette métrique pour définir des alarmes en cas d'expiration des délais de traitement des messages qui ignorent les messages vieillis artificiellement provenant de voisins bruyants.

[1] Ces mesures reflètent l'activité au niveau du système et peuvent inclure de nouvelles tentatives, des doublons ou des messages différés. N'utilisez pas de chiffres bruts pour estimer l'état de la file d'attente en temps réel sans tenir compte du comportement du cycle de vie des messages.

Files d'attente pour les lettres mortes () DLQs et statistiques CloudWatch

Lorsque vous travaillez avec DLQs, il est important de comprendre le comportement des métriques Amazon SQS :

- **`NumberOfMessagesSent`**— Cette métrique se comporte différemment pour DLQs :
  - Envoi manuel — Les messages envoyés manuellement à un DLQ sont capturés par cette métrique.

- Redrive automatique : les messages déplacés automatiquement vers un DLQ en raison d'échecs de traitement ne sont pas capturés par cette métrique. Par conséquent, les `NumberOfMessagesReceived` métriques `NumberOfMessagesSent` et peuvent présenter des divergences pour DLQs.
- Mesure recommandée pour DLQs : pour surveiller l'état d'une DLQ, utilisez la `ApproximateNumberOfMessagesVisible` métrique. Cette métrique indique le nombre de messages actuellement disponibles pour traitement dans le DLQ.

Des files d'attente et CloudWatch des indicateurs équitables

Lorsque vous utilisez des files d'attente équitables, Amazon SQS émet les statistiques supplémentaires suivantes :

- `ApproximateNumberOfNoisyGroups`
- `ApproximateNumberOfMessagesVisibleInQuietGroups`
- `ApproximateNumberOfMessagesNotVisibleInQuietGroups`
- `ApproximateNumberOfMessagesDelayedInQuietGroups`
- `ApproximateAgeOfOldestMessageInQuietGroups`

> **Note**
>
> Chaque `QuietGroup` métrique est un sous-ensemble de la `Approximate` métrique standard équivalente au niveau de la file d'attente, mais exclut les messages provenant de groupes voisins bruyants.

Groupes bruyants

Un groupe de messages bruyant représente un locataire voisin bruyant d'une file d'attente à locataires multiples.

Groupes calmes

Groupes de messages à l'exception des groupes bruyants.

Observation du comportement équitable des files d'attente SQS

Pour surveiller l'effet des files d'attente équitables sur Amazon SQS, vous pouvez comparer les `Approximate..InQuietGroups` statistiques avec les mesures standard au niveau des files d'attente. Lors d'une augmentation du trafic pour un locataire en particulier, les indicateurs généraux relatifs aux files d'attente peuvent révéler des arriérés croissants ou un vieillissement des messages. Toutefois, en examinant les groupes silencieux séparément, vous pouvez identifier que la plupart des groupes de messages ou des locataires peu bruyants ne sont pas concernés et fournir une estimation du nombre total de groupes de messages concernés.

Bien que ces nouveaux indicateurs fournissent une bonne vue d'ensemble du comportement des files d'attente équitables d'Amazon SQS, il peut être utile de comprendre quel locataire spécifique est à l'origine de la charge. Amazon CloudWatch Contributor Insights vous permet de consulter des statistiques concernant les principaux contributeurs, le nombre total de contributeurs uniques et leur utilisation. Cela est particulièrement utile dans les scénarios où vous avez affaire à des milliers de locataires qui, autrement, produiraient des données de cardinalité (et des coûts) élevés lors de l'émission de métriques traditionnelles.

Pour un exemple de configuration de surveillance pour les files d'attente régulières, consultez l'exemple sur GitHub.

Dimensions pour les métriques Amazon SQS

Les métriques Amazon SQS CloudWatch utilisent une seule dimension :. **`QueueName`** Toutes les données métriques sont regroupées et filtrées selon le nom de la file d'attente.

Conseils de surveillance

Surveillez efficacement le SQS à l'aide d'indicateurs et d' CloudWatch alarmes clés pour détecter les arriérés de files d'attente, optimiser les performances et respecter les limites de service.

- Définissez CloudWatch des alarmes en fonction `ApproximateNumberOfMessagesVisible` de la croissance du backlog.
- Surveillez `NumberOfEmptyReceives` pour ajuster la fréquence des sondages et réduire le coût des API.
- `ApproximateNumberOfGroupsWithInflightMessages`À utiliser dans les files d'attente FIFO pour diagnostiquer les limites de débit.
- Passez en revue les quotas SQS pour comprendre les seuils métriques et les limites de service.

Validation de conformité pour Amazon SQS

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. AWS fournit les ressources suivantes pour faciliter la mise en conformité :

- Conformité et gouvernance de la sécurité : ces guides de mise en œuvre de solutions traitent des considérations architecturales et fournissent les étapes à suivre afin de déployer des fonctionnalités de sécurité et de conformité.
- Référence des services éligibles HIPAA : liste les services éligibles HIPAA. Tous ne Services AWS sont pas éligibles à la loi HIPAA.
- AWS Ressources de https://aws.amazon.com/compliance/resources/ de conformité — Cette collection de classeurs et de guides peut s'appliquer à votre secteur d'activité et à votre région.
- AWS Guides de conformité destinés aux clients — Comprenez le modèle de responsabilité partagée sous l'angle de la conformité. Les guides résument les meilleures pratiques en matière de sécurisation Services AWS et décrivent les directives relatives aux contrôles de sécurité dans de nombreux cadres (notamment le National Institute of Standards and Technology (NIST), le Payment Card Industry Security Standards Council (PCI) et l'Organisation internationale de normalisation (ISO)).
- Évaluation des ressources à l'aide des règles du guide du AWS Config développeur : le AWS Config service évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- AWS Security Hub— Cela Service AWS fournit une vue complète de votre état de sécurité interne AWS. Security Hub utilise des contrôles de sécurité pour évaluer vos ressources AWS et vérifier votre conformité par rapport aux normes et aux bonnes pratiques du secteur de la sécurité. Pour obtenir la liste des services et des contrôles pris en charge, consultez Référence des contrôles Security Hub.

- Amazon GuardDuty — Cela Service AWS détecte les menaces potentielles qui pèsent sur vos charges de travail Comptes AWS, vos conteneurs et vos données en surveillant votre environnement pour détecter toute activité suspecte et malveillante. GuardDuty peut vous aider à répondre à diverses exigences de conformité, telles que la norme PCI DSS, en répondant aux exigences de détection des intrusions imposées par certains cadres de conformité.
- AWS Audit Manager— Cela vous Service AWS permet d'auditer en permanence votre AWS utilisation afin de simplifier la gestion des risques et la conformité aux réglementations et aux normes du secteur.

Résilience dans Amazon SQS

L'infrastructure AWS mondiale est construite autour des AWS régions et des zones de disponibilité. AWS Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données. Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section Infrastructure AWS mondiale.

Outre l'infrastructure AWS mondiale, Amazon SQS propose des files d'attente distribuées.

Files d'attente distribuées

Un système de messagerie distribué comporte trois parties principales : les composants de votre système distribué, votre file d'attente (distribuée sur les serveurs Amazon SQS) et les messages de la file d'attente.

Dans le scénario suivant, le système comprend plusieurs producteurs (composants qui envoient des messages à la file d'attente) et plusieurs consommateurs (composants qui reçoivent des messages de la file d'attente). La file d'attente (qui contient les messages A à E) stocke les messages de manière redondante sur plusieurs serveurs Amazon SQS.

Sécurité de l'infrastructure dans Amazon SQS

En tant que service géré, Amazon SQS est protégé par les procédures de sécurité du réseau AWS mondial décrites dans le livre blanc Amazon Web Services : présentation des processus de sécurité.

Vous utilisez des actions d'API AWS publiées pour accéder à Amazon SQS via le réseau. Les clients doivent prendre en charge le protocole TLS (Transport Layer Security) 1.2 ou version ultérieure. Les clients doivent également prendre en charge les suites de chiffrement PFS (Perfect Forward Secrecy) comme Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)

En outre, vous devez signer les demandes à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un mandataire IAM. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires afin de signer les demandes.

Vous pouvez appeler ces actions d'API à partir de n'importe quel emplacement sur le réseau, mais Amazon SQS prend en charge les stratégies d'accès basées sur les ressources, ce qui peut inclure des restrictions en fonction de l'adresse IP source. Vous pouvez également utiliser les politiques Amazon SQS pour contrôler l'accès depuis des points de terminaison Amazon VPC spécifiques ou spécifiques. VPCs Cela permet d'isoler efficacement l'accès réseau à une file d'attente Amazon SQS donnée uniquement du VPC spécifique au sein du réseau. AWS Pour de plus amples informations, veuillez consulter Exemple 5 : Refuser l'accès s'il n'émane pas d'un point de terminaison de VPC.

Bonnes pratiques de sécurité pour Amazon SQS

AWS fournit de nombreuses fonctionnalités de sécurité pour Amazon SQS, que vous devriez examiner dans le contexte de votre propre politique de sécurité. Voici les bonnes pratiques de sécurité préventive pour Amazon SQS.

> **Note**
>
> Les conseil de mise en œuvre spécifiques fournies concernent les implémentations et les cas d'utilisation courants. Nous vous suggérons de consulter ces bonnes pratiques dans le contexte de votre cas d'utilisation, de votre architecture et de votre modèle de menace spécifique.

S'assurer que les files d'attente ne sont pas accessibles publiquement

À moins que vous ne demandiez explicitement à quiconque sur Internet de lire ou d'écrire dans votre file d'attente Amazon SQS, vous devez vous assurer que votre file d'attente n'est pas accessible au public (accessible par tout le monde dans le monde ou par tout utilisateur authentifié AWS).

- Évitez de créer des stratégies avec `Principal` défini sur `""`.
- Évitez d'utiliser un caractère générique (`*`). Nommez plutôt un ou plusieurs utilisateurs spécifiques.

Implémentation d'un accès sur la base du moindre privilège

Lorsque vous accordez des autorisations, vous décidez qui les reçoit, pour quelles files d'attente celles-ci sont destinées et quelles actions d'API spécifiques vous souhaitez autoriser pour ces files d'attente. La mise en œuvre du moindre privilège est importante pour réduire les risques de sécurité et atténuer l'effet des erreurs ou des intentions malveillantes.

Suivez les conseils de sécurité standard pour accorder le moindre privilège. Autrement dit, accordez uniquement les autorisations requises pour effectuer une tâche spécifique. Vous pouvez implémenter ceci à l'aide d'une combinaison de stratégies de sécurité.

Amazon SQS utilise le modèle producteur-consommateur nécessitant trois types d'accès de compte utilisateur :

- Administrateurs - Accès à la création, à la modification et à la suppression de files d'attente. Les administrateurs contrôlent également les stratégies de file d'attente.
- Producteurs : accès à l'envoi de messages dans les files d'attente.
- Consommateurs : accès à la réception et à la suppression des messages dans les files d'attente.

Pour plus d'informations, consultez les sections suivantes :

- Gestion des identités et des accès dans Amazon SQS
- Autorisations d'API Amazon SQS : référence des actions et ressources
- Utilisation de stratégies personnalisées avec le langage de la stratégie d'accès Amazon SQS

Utiliser les rôles IAM pour les applications et les AWS services qui nécessitent un accès Amazon SQS

Pour que des applications ou AWS des services tels qu'Amazon puissent accéder EC2 aux files d'attente Amazon SQS, ils doivent utiliser des informations d' AWS identification valides dans leurs AWS demandes d'API. Ces informations d'identification ne faisant pas l'objet d'une rotation automatique, vous ne devez pas les stocker AWS directement dans l'application ou l' EC2 instance.

Vous devez utiliser un rôle IAM pour gérer des informations d'identification temporaires pour les applications ou services devant accéder à Amazon SQS. Lorsque vous utilisez un rôle, vous n'avez pas à distribuer des informations d'identification à long terme (telles qu'un nom d'utilisateur, un mot de passe et des clés d'accès) à une EC2 instance ou à un AWS service tel que AWS Lambda. Le rôle fournit plutôt des autorisations temporaires que les applications peuvent utiliser lorsqu'elles appellent d'autres AWS ressources.

Pour de plus amples informations, veuillez consulter Rôles IAMet Scénarios courants pour les rôles : utilisateurs, applications et services dans le guide de l'utilisateur.

Mise en œuvre du chiffrement côté serveur

Pour atténuer les problèmes de fuite de données, utilisez le chiffrement au repos pour chiffrer vos messages à l'aide d'une clé stockée dans un emplacement différent de celui où les messages sont stockés. Le chiffrement côté serveur (SSE) fournit le chiffrement des données au repos. Amazon SQS chiffre vos données au niveau des messages lorsqu'il les stocke et déchiffre les messages pour vous lorsque vous y accédez. SSE utilise des clés gérées dans AWS Key

Management Service. Tant que vous authentifiez votre demande et que vous avez des autorisations d'accès, il n'y a aucune différence entre l'accès aux données chiffrées et aux données déchiffrées.

Pour plus d'informations, consultez Chiffrement au repos dans Amazon SQS et Gestion des clés Amazon SQS.

Application du chiffrement des données en transit

Sans HTTPS (TLS), un attaquant basé sur le réseau peut espionner le trafic réseau ou le manipuler, en utilisant une attaque telle que. man-in-the-middle Autorisez uniquement les connexions chiffrées via HTTPS (TLS) en utilisant la condition `aws:SecureTransport` de la stratégie de file d'attente pour forcer les demandes à utiliser SSL.

Réflexion sur l'utilisation des points de terminaison de VPC pour accéder à Amazon SQS

Si vous avez des files d'attente avec lesquelles vous devez pouvoir interagir mais qui ne doivent absolument pas être exposées à Internet, utilisez des points de terminaison de VPC pour mettre en file d'attente l'accès uniquement aux hôtes au sein d'un VPC particulier. Vous pouvez utiliser des politiques de file d'attente pour contrôler l'accès aux files d'attente depuis des points de terminaison Amazon VPC spécifiques ou depuis des points spécifiques. VPCs

Les points de terminaison d'un VPC Amazon SQS offrent deux façons de contrôler l'accès à vos messages :

- Vous pouvez contrôler les demandes, les utilisateurs ou les groupes autorisés à traverser un point de terminaison d'un VPC spécifique.
- Vous pouvez contrôler quels points de terminaison VPCs ou quels points de terminaison VPC ont accès à votre file d'attente à l'aide d'une politique de file d'attente.

Pour plus d'informations, consultez Points de terminaison Amazon Virtual Private Cloud pour Amazon SQS et Création d'une stratégie de point de terminaison d'un VPC Amazon pour Amazon SQS.

Ressources Amazon SQS associées

Le tableau suivant répertorie les ressources connexes qui peuvent vous être utiles lors de l'utilisation de ce service.

Ressource	Description
Référence d'API Amazon Simple Queue Service	Descriptions des actions, paramètres et types de données, et liste des erreurs renvoyées par le service.
Amazon SQS dans la référence des commandes de l'AWS CLI	Descriptions des AWS CLI commandes que vous pouvez utiliser pour gérer les files d'attente.
Régions et points de terminaison	Informations sur les régions et les points de terminaison Amazon SQS
Page produit	Page web principale pour de plus amples informations sur Amazon SQS.
Forum de discussion	Un forum communautaire pour les développeurs où ils peuvent discuter des questions techniques liées à Amazon SQS.
AWS Informations de support Premium	Page Web principale contenant des informations sur le Support AWS Premiumone-on-one, un canal d'assista nce rapide destiné à vous aider à créer et à exécuter des applications sur des services AWS d'infrastructure.

Historique de la documentation

Le tableau suivant décrit les modifications importantes apportées au Guide du développeur Amazon Simple Queue Service depuis janvier 2019. Pour recevoir les notifications des mises à jour de cette documentation, abonnez-vous au flux RSS.

Les fonctionnalités du service sont parfois déployées progressivement AWS dans les régions où un service est disponible. Nous mettons à jour cette documentation pour la première version uniquement. Nous ne fournissons pas d'informations sur la disponibilité des régions et n'annonçons pas les déploiements régionaux ultérieurs. Pour plus d'informations sur la disponibilité des fonctionnalités du service dans les régions et pour vous abonner aux notifications concernant les mises à jour, voir Quelles sont les nouveautés AWS ? .

Modification	Description	Date
Support pour Fair Queues	Le support Fair Queues a été ajouté pour les files d'attente standard.	21 juillet 2025
Ajout de la prise en charge des terminaux à double pile (IPv4 et IPv6)	Amazon SQS prend désormais en charge les points de terminaison à double pile (IPv4 et IPv6), ce qui permet d'accéder aux files d'attente via les deux protocole s IP.	17 avril 2025
CloudTrail intégration pour tous les Amazon SQS APIs	CloudTrail intégration ajoutée pour tous les Amazon SQS APIs.	10 janvier 2025
SQSUnlockQueuePolicy	Amazon SQS a ajouté une nouvelle politique AWS gérée appelée `SQSUnlock QueuePolicy` à déverroui ller une file d'attente et à supprimer une politique de file	15 novembre 2024

	d'attente mal configurée qui empêche tous les principaux d'accéder à une file d'attente Amazon SQS.	
AWS kms:Decrypt	Amazon SQS n'a plus besoin d'`kms:Decrypt` autorisation pour l'`SendMessage` API. Les clients n'ont désormais besoin que de l'`kms:GenerateDataKey` autorisation sur la clé KMS utilisée pour chiffrer la file d'attente, mais ils ont toujours besoin de `kms:Decrypt` l'autorisation d'appeler`ReceiveMessage` .	24 juillet 2024
Mise à jour des métriques FIFO	Support `NumberOfDeduplicatedSentMessages` et `ApproximateNumberOfGroupsWithInflightMessages` ajout aux métriques FIFO d'Amazon SQS.	3 juillet 2024
`ListQueueTags` action prise en charge dans la politique SQSRead OnlyAccess gérée par Amazon	La politique SQSRead OnlyAccess gérée par Amazon permet `ListQueueTags` de récupérer toutes les balises associées à une file d'attente Amazon SQS spécifiée.	2 mai 2024
AWS Protocole JSON	Effectuez des demandes d'API à l'aide du protocole AWS JSON.	27 juillet 2023

Nouvelle section décrivant les politiques AWS gérées pour Amazon SQS et les mises à jour de ces politiques	Amazon SQS a ajouté une nouvelle action qui vous permet de répertorier les tâches de transfert de messages les plus récentes (jusqu'à 10) dans une file d'attente source spécifique. Cette action est associée à l'opération d'API `ListMessageMoveTasks` .	7 juin 2023
Redrive la file d'attente des lettres mortes en utilisant APIs	Configurez le redrive des files d'attente contenant des lettres mortes à l'aide d'Amazon SQS. APIs	7 juin 2023
ABAC pour Amazon SQS	Contrôle d'accès par attributs (ABAC) à l'aide de balises de file d'attente pour des autorisations d'accès flexibles et évolutives.	10 novembre 2022
La limite de débit élevé FIFO augmente	Augmentation des quotas par défaut pour le mode débit élevé FIFO dans les régions commerciales, ainsi que pour l'optimisation des documents à haut débit FIFO.	20 octobre 2022
Le chiffrement côté serveur (SSE) par défaut est disponible	Chiffrement côté serveur (SSE) à l'aide du chiffrement propre à SQS (SSE-SQS) par défaut.	26 septembre 2022

La prise en charge de la protection contre le problème de député confus d'Amazon SQS est disponible	La protection contre le problème de député confus vous permet de spécifier de nouveaux en-têtes dans les demandes, qui sont vérifiés par rapport aux conditions de la stratégie KMS lors de l'utilisation du SSE géré par Amazon SQS.	29 décembre 2021
Le chiffrement SSE géré est disponible	Le chiffrement SSE géré par Amazon SQS (SSE-SQS) est un chiffrement géré côté serveur qui utilise des clés de chiffrement appartenant à Amazon SQS pour protéger les données sensibles envoyées via des files d'attente de messages.	23 novembre 2021
La redirection des files d'attente de lettres mortes est disponible	Amazon SQS prend en charge la redirection des files d'attente de lettres mortes pour les files d'attente standard.	10 novembre 2021
Un débit élevé pour les messages dans les files d'attente FIFO est disponible	Le débit élevé pour les files d'attente FIFO Amazon SQS permet d'augmenter le nombre de transactions par seconde (TPS) pour les messages dans les files d'attente FIFO. Pour plus d'informations sur les quotas de débit, consultez la section Quotas relatifs aux messages.	27 mai 2021

Un débit élevé pour les messages dans les files d'attente FIFO est disponible dans la version préliminaire	Le débit élevé pour les files d'attente FIFO Amazon SQS est en version préliminaire et est susceptible d'être modifié. Cette fonctionnalité fournit un plus grand nombre de transactions par seconde (TPS) pour les messages dans les files d'attente FIFO. Pour plus d'informations sur les quotas de débit, consultez la section Quotas relatifs aux messages.	17 décembre 2020
Nouvelle conception de la console Amazon SQS	Pour simplifier les flux de travail de développement et de production, la console Amazon SQS propose une nouvelle expérience utilisateur.	8 juillet 2020
Amazon SQS prend en charge la pagination pour les ListQueues et listDeadLetter SourceQueues	Vous pouvez spécifier le nombre maximum de résultats à renvoyer à partir d'une listDeadLetterSourceQueuesListQueues ou d'une requête.	22 juin 2020
Amazon SQS prend en charge les métriques CloudWatch Amazon d'une minute dans AWS toutes les régions, à AWS GovCloud l'exception des régions (États-Unis)	La CloudWatch métrique d'une minute pour Amazon SQS est disponible dans toutes les régions, à l'exception AWS GovCloud (US) des régions.	9 janvier 2020

Amazon SQS prend en charge les métriques d'une minute CloudWatch	La CloudWatch métrique d'une minute pour Amazon SQS n'est actuellement disponibl e que dans les régions suivantes : USA Est (Ohio), Europe (Irlande), Europe (Stockholm) et Asie-Pacifique (Tokyo).	25 novembre 2019
AWS Lambda des déclenche urs pour les files d'attente FIFO Amazon SQS sont disponibles	Vous pouvez configurer les messages arrivant dans une file d'attente FIFO en tant que déclencheur d'une fonction Lambda.	25 novembre 2019
Le chiffrement côté serveur (SSE) pour Amazon SQS est disponible dans les régions de Chine	SSE pour Amazon SQS est disponible dans les régions de Chine.	13 novembre 2019
Les files d'attente FIFO sont disponibles dans la région Moyen-Orient (Bahreïn)	Les files d'attente FIFO sont disponibles dans la région Moyen-Orient (Bahreïn).	10 octobre 2019
Les points de terminaison Amazon Virtual Private Cloud (Amazon VPC) pour Amazon SQS sont disponibles dans les régions AWS GovCloud (USA Est) et (USA Ouest) AWS GovCloud	Vous pouvez envoyer des messages à vos files d'attente Amazon SQS depuis Amazon VPC dans les régions AWS GovCloud (USA Est) et (USA Ouest). AWS GovCloud	5 septembre 2019

Amazon SQS permet de résoudre les problèmes de files d'attente à l' AWS X-Ray aide des attributs du système de messagerie	Vous pouvez résoudre les problèmes liés aux messages transitant par des files d'attente Amazon SQS à l'aide de X-Ray. Cette version ajoute le paramètre de demande `MessageSystemAttribute` (qui vous permet d'envoyer des en-têtes de suivi X-Ray via Amazon SQS) aux opérations d'API `SendMessage` et `SendMessageBatch`, l'attribut `AWSTraceHeader` à l'opération d'API `ReceiveMessage` et le type de données `MessageSystemAttributeValue`.	28 août 2019
Vous pouvez baliser les files d'attente Amazon SQS lors de leur création	Vous pouvez utiliser un seul appel d'API Amazon SQS, une fonction AWS SDK ou une commande AWS Command Line Interface (AWS CLI) pour créer simultanément une file d'attente et spécifier ses balises. En outre, Amazon SQS prend en charge les clés `aws:TagKeys` et `aws:RequestTag` AWS Identity and Access Management (IAM).	22 août 2019

Le client de file d'attente temporaire pour Amazon SQS est désormais disponible	Les files d'attente temporair es vous permettent de gagner du temps de développement et de réduire les coûts de déploiement lorsque vous utilisez des modèles de messages courants tels que demande-réponse. Vous pouvez utiliser le Client de file d'attente temporaire pour créer des files d'attente temporaires à haut débit, économiques et gérées par l'application.	25 juillet 2019
SSE pour Amazon SQS est disponible dans la région AWS GovCloud (USA Est)	Le chiffrement côté serveur (SSE) pour Amazon SQS est disponible dans AWS GovCloud la région (USA Est).	20 juin 2019
Les files d'attente FIFO sont disponibles dans les régions Asie-Pacifique (Hong Kong), Chine (Pékin), AWS GovCloud (USA Est) et AWS GovCloud (USA Ouest)	Les files d'attente FIFO sont disponibles dans les régions Asie-Pacifique (Hong Kong), Chine (Pékin), AWS GovCloud (USA Est) et AWS GovCloud (USA Ouest).	15 mai 2019
Les stratégies de point de terminaison d'un VPC Amazon sont disponibles pour Amazon SQS	Vous pouvez créer des stratégies de point de terminaison d'un VPC Amazon pour Amazon SQS.	4 avril 2019
Les files d'attente FIFO sont disponibles dans les régions Europe (Stockholm) et Chine (Ningxia)	Les files d'attente FIFO sont disponibles dans les régions Europe (Stockholm) et Chine (Ningxia).	14 mars 2019

Les files d'attente FIFO sont disponibles dans toutes les régions où Amazon SQS est disponible	Les files d'attente FIFO sont disponibles dans les régions USA Est (Ohio), USA Est (Virginie du Nord), USA Ouest (Californie du Nord), USA Ouest (Oregon), Asie-Pacifique (Mumbai), Asie-Pacifique (Séoul), Asie-Pacifique (Singapour), Asie-Pacifique (Sydney), Asie-Pacifique (Tokyo), Canada (Centre), Europe (Francfort), Europe (Irlande), Europe (Londres), Europe (Paris) et Amérique du Sud (São Paulo).	7 février 2019

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.