



開發人員指南

Amazon Simple Queue Service



Amazon Simple Queue Service: 開發人員指南

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能隸屬於 Amazon，或與 Amazon 有合作關係，或由 Amazon 贊助。

Table of Contents

什麼是 Amazon SQS ?	1
使用 Amazon SQS 的優勢	1
基本架構	1
分散式佇列	1
訊息生命週期	2
Amazon SQS、Amazon MQ 和 Amazon SNS 之間的差異	4
開始使用	6
設定	6
步驟 1：建立 AWS 帳戶 和 IAM 使用者	6
步驟 2：授予程式設計存取權	8
步驟 3：準備好開始使用範本程式碼	9
後續步驟	10
了解 Amazon SQS 主控台	10
佇列類型	11
在 Amazon SQS 中實作請求回應系統	13
建立標準佇列	13
建立佇列	14
使用標準佇列傳送訊息	15
建立 FIFO 佇列	16
建立佇列	16
使用 FIFO 佇列傳送訊息	18
一般任務	18
管理佇列	20
編輯佇列	20
接收和刪除訊息	20
確認佇列是空的	22
刪除佇列	23
清除佇列	24
標準佇列	25
Amazon SQS at-least-once交付	25
佇列和訊息識別碼	25
標準佇列的識別碼	26
FIFO 佇列	27
FIFO 佇列金鑰術語	28

FIFO 交付邏輯	29
傳送訊息	29
接收訊息	30
多次重試	30
有關 FIFO 行為的其他備註	31
更好地理解範例	31
恰好一次的處理	32
從標準佇列移到 FIFO 佇列	32
FIFO 佇列和 Lambda 並行行為	33
FIFO 佇列訊息分組	34
Lambda 與 FIFO 佇列並行	34
使用案例範例	34
FIFO 佇列的高輸送量	35
使用案例	35
分割區與資料分配	35
啟用 FIFO 佇列的高輸送量	38
佇列和訊息識別碼	38
FIFO 佇列的識別碼	26
FIFO 佇列的其他識別碼	40
配額	41
FIFO 佇列配額	41
Amazon SQS 配額	41
標準佇列配額	42
訊息配額	44
政策配額	49
特性和功能	50
無效字母佇列	50
針對無效字母佇列使用政策	50
了解無效字母佇列的訊息保留期間	51
設定無效字母佇列	51
設定無效字母佇列再驅動	52
CloudTrail 更新和許可需求	58
使用 Amazon CloudWatch 建立無效字母佇列的警示	62
Amazon SQS 的訊息中繼資料	62
訊息屬性	62
訊息系統屬性	66

處理訊息所需的資源	66
列出佇列分頁	67
成本分配標籤	67
短和長輪詢	68
以短輪詢消耗訊息	68
以長輪詢消耗訊息	69
長短輪詢之間的差異	69
可見性逾時	70
可見性逾時使用案例	70
設定和調整可見性逾時	71
傳輸中訊息和配額	71
了解標準 和 FIFO 佇列中的可見性逾時	71
處理失敗	72
變更和終止可見性逾時	72
最佳實務	72
延遲佇列	73
暫時佇列	74
虛擬佇列	74
請求-回應訊息模式 (虛擬佇列)	75
範例情況：處理登入請求	76
清除佇列	78
訊息計時器	79
存取 EventBridge Pipes	79
管理大型訊息	81
使用適用於 Java 的擴充用戶端程式庫	81
使用適用於 Python 的擴充用戶端程式庫	87
設定 Amazon SQS	90
Amazon SQS 的 ABAC	90
ABAC 是什麼？	90
為什麼一定要在 Amazon SQS 中使用 ABAC？	91
為存取控制加上標籤	91
建立 IAM 使用者和 Amazon SQS 佇列	92
測試屬性型存取控制	95
設定佇列參數	96
設定存取政策	98
為佇列設定 SSE-SQS	99

為佇列設定 SSE-KMS	100
為佇列設定標籤	101
訂閱佇列到主題	102
跨帳戶訂閱	103
跨區域訂閱	103
設定 Lambda 觸發條件	104
先決條件	104
使用 EventBridge 自動化通知	105
訊息屬性	106
最佳實務	108
錯誤處理和有問題的訊息	108
在 Amazon SQS 中處理請求錯誤	108
在 Amazon SQS 中擷取有問題的訊息	108
在 Amazon SQS 中設定無效字母佇列保留	109
訊息重複資料刪除和分組	109
避免在 Amazon SQS 中處理不一致的訊息	110
使用訊息重複資料刪除 ID	110
使用訊息群組 ID	112
使用接收請求嘗試 ID	113
訊息處理和計時	113
在 Amazon SQS 中及時處理訊息	114
在 Amazon SQS 中設定長輪詢	114
在 Amazon SQS 中使用適當的輪詢模式	115
Java 開發套件範例	116
使用伺服器端加密	116
將 SSE 新增到現有佇列	116
停用佇列的 SSE	117
使用 SSE 建立佇列	118
擷取 SSE 屬性	118
設定標籤	118
列出標籤	119
新增或更新標籤	119
移除標籤	120
傳送訊息屬性	120
定義屬性	121
傳送具有屬性的訊息	122

使用 APIs	124
使用 AWS JSON 通訊協定提出查詢 API 請求	125
建構端點	125
提出 POST 請求	126
解譯 Amazon SQS JSON API 回應	127
Amazon SQS AWS JSON 通訊協定FAQs	128
使用查詢通訊協定提出 AWS 查詢 API 請求	131
建構端點	131
提出 GET 請求	132
提出 POST 請求	126
解譯 Amazon SQS XML API 回應	133
對請求進行身分驗證	134
使用 HMAC-SHA 的基本身分驗證流程	135
第 1 部分：來自使用者的請求	136
第 2 部分：來自 的回應 AWS	137
批次動作	138
批次訊息動作	138
使用 Amazon SQS 啟用用戶端緩衝和請求批次處理	139
使用 Amazon SQS 使用水平擴展和動作批次來增加輸送量	147
使用 AWS SDKs	159
使用 JMS	161
先決條件	161
使用 Java Messaging Library	162
建立 JMS 連線	162
建立 Amazon SQS 佇列	163
同步傳送訊息	164
同步接收訊息	165
異步接收訊息	167
使用用戶端認可模式	168
使用無順序認可模式	169
使用 JMS Client 搭配其他 Amazon SQS 用戶端	170
搭配標準佇列使用 JMS 的工作 Java 範例	171
ExampleConfiguration.java	171
TextMessageSender.java	174
SyncMessageReceiver.java	176
AsyncMessageReceiver.java	177

SyncMessageReceiverClientAcknowledge.java	179
SyncMessageReceiverUnorderedAcknowledge.java	183
SpringExampleConfiguration.xml	187
SpringExample.java	188
ExampleCommon.java	190
支援的 JMS 1.1 實作	192
支援的常用介面	192
支援的訊息類型	192
支援的訊息認可模式	193
JMS 定義標頭和預訂屬性	193
教學課程	195
使用 建立 Amazon SQS 佇列 AWS CloudFormation	195
從 VPC 傳送訊息	197
步驟 1：建立 Amazon EC2 金鑰對	198
步驟 2：建立 AWS 資源	198
步驟 3：確認 EC2 執行個體不是供公開存取	199
步驟 4：建立 Amazon SQS 的 Amazon VPC 端點	200
步驟 5：傳送訊息到您的 Amazon SQS 佇列	201
程式碼範例	203
基本概念	216
您好，Amazon SQS	217
動作	229
案例	381
建立簡訊應用程式	382
建立傳訊應用程式	383
建立 Amazon Textract Explorer 應用程式	384
建立並發布到 FIFO 主題	385
偵測映像中的人物和物件	397
使用 S3 管理大型訊息	398
處理 S3 事件通知	402
將訊息發佈至佇列	406
傳送和接收批次的訊息	521
使用適用於 .NET AWS 的訊息處理架構搭配 Amazon SQS	526
使用 Amazon SQS Java Messaging Library 來使用 JMS 介面	527
使用佇列標籤	549
無伺服器範例	552

使用 Amazon SQS 觸發條件調用 Lambda 函數	552
使用 Amazon SQS 觸發條件報告 Lambda 函數的批次項目失敗	561
故障診斷	572
存取遭拒錯誤	572
Amazon SQS 佇列政策和 IAM 政策	573
AWS Key Management Service (AWS KMS) 許可	573
VPC 端點政策	575
組織服務控制政策	575
API 錯誤	575
QueueDoesNotExist 錯誤	575
InvalidAttributeValue 錯誤	576
ReceiptHandle 錯誤	576
DLQ 和 DLQ 重新驅動問題	577
DLQ 問題	577
DLQ 重新驅動問題	578
FIFO 調節問題	580
ReceiveMessage API 呼叫未傳回的訊息	581
空佇列	581
已達到飛行中限制	581
訊息延遲	581
訊息正在處理中	581
輪詢方法	581
網路錯誤	582
ETIMEOUT error	582
UnknownHostException error	583
使用 X-Ray 對佇列進行故障診斷	584
安全	585
資料保護	585
資料加密	586
網際網路流量隱私權	596
身分與存取管理	598
目標對象	598
使用身分驗證	598
使用政策管理存取權	601
概觀	603
如何搭配 IAM 使用 Amazon Simple Queue Service	609

AWS 受管政策	615
故障診斷	617
使用 政策	618
日誌記錄和監控	660
記錄 API 呼叫	662
監控佇列	665
法規遵循驗證	676
恢復能力	677
分散式佇列	677
基礎架構安全	678
最佳實務	678
確定佇列無法公開存取	679
實作最低權限存取	679
針對需要 Amazon SQS 存取的應用程式 AWS 和服務使用 IAM 角色	680
實作伺服器端加密	680
強制加密傳輸中的資料	680
考慮使用 VPC 端點來存取 Amazon SQS	680
相關資源	681
文件歷史記錄	682
.....	dclxxxviii

什麼是 Amazon Simple Queue Service ?

Amazon Simple Queue Service (Amazon SQS) 提供安全、耐用且可用的託管佇列，可讓您整合與分離分散式軟體系統和元件。Amazon SQS 提供通用結構，例如[無效字母佇列](#)和[成本分配標籤](#)。它提供一般 Web 服務 API，您可以使用 AWS SDK 支援的任何程式設計語言來存取。

使用 Amazon SQS 的優勢

- 安全性 – [您可以控制](#)誰能傳送訊息到 Amazon SQS 佇列，並從佇列接收訊息。您可以選擇使用預設的 Amazon SQS 受管伺服器端加密 (SSE) 保護佇列中訊息的內容，也可以使用 AWS Key Management Service 中管理的自訂 [SSE](#) 金鑰 (AWS KMS) 來傳輸敏感資料。
- 耐用性 - 為了確保訊息的安全，Amazon SQS 將它們存放在多部伺服器上。標準佇列支援[至少傳遞訊息一次](#)，而 FIFO 佇列支援[恰好處理一次訊息](#)和[高輸送量](#)模式。
- 可用性 – Amazon SQS 使用[冗餘基礎設施](#)以提供對訊息的高度並行存取，以及產生和消耗訊息的高可用性。
- 可擴展性 – Amazon SQS 可以個別處理每個[緩衝的請求](#)，以透明的方式進行擴展來處理負載增加或尖峰，無須任何佈建指示。
- 可靠性 – Amazon SQS 在處理訊息時會鎖定訊息，以便同一時間多個生產者可以傳送、多個消費者可以接收訊息。
- 自訂 - 您的佇列不需完全一樣；舉例來說，您可以[在佇列上設定預設延遲](#)。您可以[使用 Amazon Simple Storage Service \(Amazon S3\)](#) 或 Amazon DynamoDB 存放大於 256 KB 的訊息內容，並讓 Amazon SQS 保有指向該 Amazon SQS 物件的指標，或者也可以將大型訊息分割成較小的訊息。

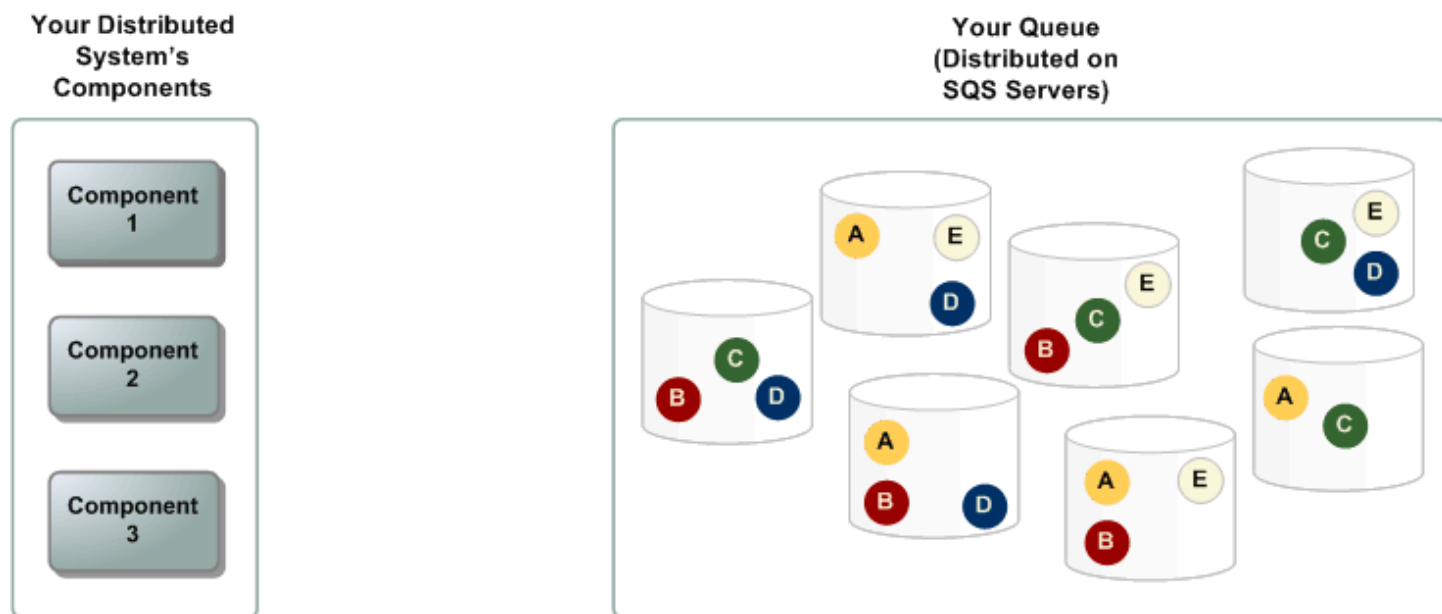
基本 Amazon SQS 架構

本節說明分散式傳訊系統的元件，並說明 Amazon SQS 訊息的生命週期。

分散式佇列

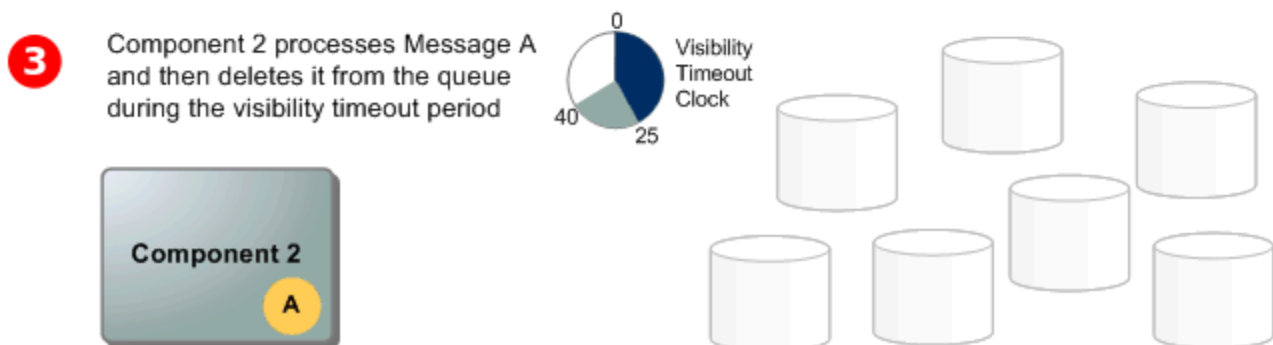
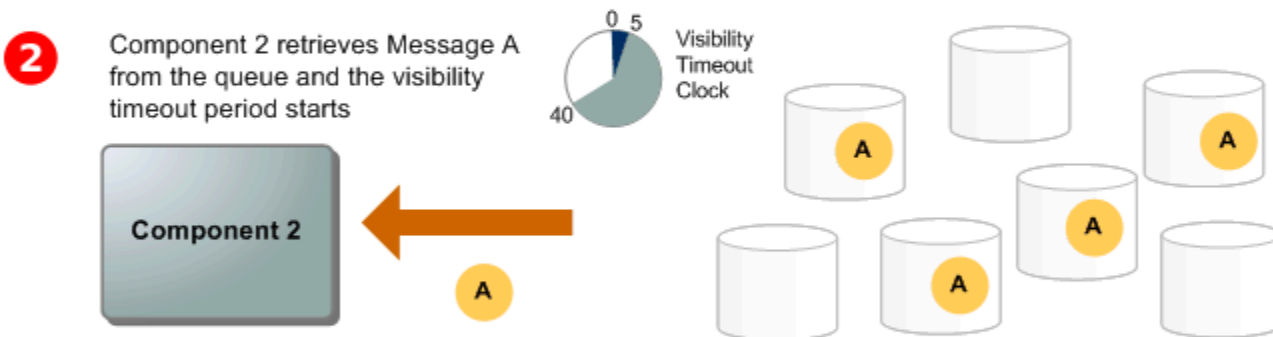
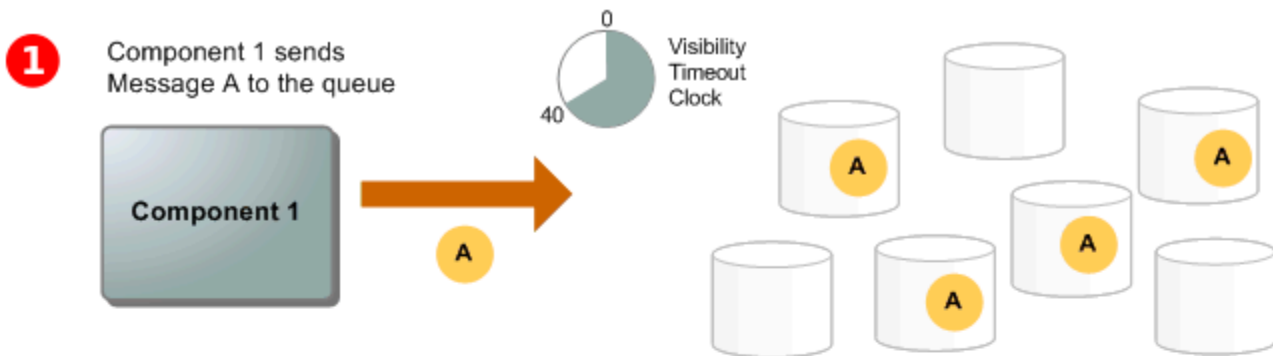
分散式傳訊系統中有三個主要部分：分散式系統的元件、佇列（在 Amazon SQS 伺服器上分佈），以及佇列中的訊息。

在以下情況中，您的系統有數個生產者（可傳送訊息到佇列的元件）以及消費者（從佇列接收訊息的元件）。佇列（存放訊息 A 到 E）將訊息以備援方式存放在多個 Amazon SQS 伺服器上。



訊息生命週期

以下情況說明佇列中 Amazon SQS 訊息從建立到刪除的生命週期。



1 生產者（元件 1）會將訊息 A 傳送至佇列，而訊息會以備援方式分散到 Amazon SQS 伺服器。

2 當取用者（元件 2）準備好處理訊息時，它會使用佇列中的訊息，並傳回訊息 A。在處理訊息 A 時，訊息 A 仍會留在佇列上，在[可見性逾時](#)期間內，不會傳回至後續的接收請求。

3 取用者（元件 2）從佇列刪除訊息 A，以防止可見性逾時過期時再次接收和處理訊息。

Note

Amazon SQS 會自動刪除在佇列上存在超過訊息保留期間上限的訊息。預設的訊息保留期間為 4 天。不過可以使用 [SetQueueAttributes](#) 動作，將訊息保留期間設為 60 秒至 1,209,600 秒 (14 天)。

Amazon SQS、Amazon MQ 和 Amazon SNS 之間的差異

Amazon SQS、[Amazon SNS](#) 和 [Amazon MQ](#) 提供高度可擴展且 easy-to-use 受管傳訊服務，每個服務都專為分散式系統中的特定角色而設計。以下是這些服務之間差異的增強概觀：

Amazon SQS 會將分散式軟體系統和元件分離並擴展為佇列服務。它通常透過單一訂閱者處理訊息，非常適合需要預防訂單和損失的工作流程。為了更廣泛地分佈，整合 Amazon SQS 與 Amazon SNS 可實現 [廣發訊息模式](#)，有效地一次將訊息推送給多個訂閱者。

Amazon SNS 允許發佈者透過做為通訊管道的主題，將訊息傳送給多個訂閱者。訂閱者會使用支援的端點類型接收已發佈的訊息，例如 [Amazon Data Firehose](#)、[Amazon SQS](#)、[Lambda](#)、HTTP、電子郵件、行動推播通知和行動簡訊 (SMS)。此服務非常適合需要即時通知的情況，例如即時使用者參與或警示系統。為了防止訂閱者離線時訊息遺失，整合 Amazon SNS 與 Amazon SQS 佇列訊息可確保一致的交付。

Amazon MQ 最適合希望從傳統訊息代理程式遷移的企業，支援標準訊息通訊協定，例如 AMQP 和 MQTT，以及 [Apache ActiveMQ](#) 和 [RabbitMQ](#)。它與需要穩定、可靠訊息的舊版系統相容，無需進行大量重新設定。

下表提供每個 服務資源類型的概觀：

資源類型	Amazon SNS	Amazon SQS	Amazon MQ
同步	否	否	是
異步	是	是	是
佇列	否	是	是
發佈者-訂閱者訊息	是	否	是
訊息代理程式	否	否	是

建議新應用程式使用 Amazon SQS 和 Amazon SNS，因為它們可以從近乎無限的可擴展性和簡單的 API 中受益。他們通常會透過 pay-as-you-go 定價，為大量應用程式提供更具成本效益的解決方案。我們建議使用 Amazon MQ 從現有訊息代理程式遷移應用程式，這些訊息代理程式依賴 JMS 等 API 或 Advanced Message Queuing Protocol (AMQP)、MQTT、OpenWire 和 Simple Text Oriented Message Protocol (STOMP) 等通訊協定的相容性。

Amazon SQS 入門

本主題會引導您使用 Amazon SQS 主控台來建立和管理標準佇列和 FIFO 佇列。您將了解如何導覽主控台、檢視佇列屬性，以及區分佇列類型。關鍵任務包括傳送、接收和設定訊息、調整可見性逾時和訊息保留等參數，以及透過 政策管理佇列存取。

主題

- [設定 Amazon SQS](#)
- [了解 Amazon SQS 主控台](#)
- [Amazon SQS 佇列類型](#)
- [建立 Amazon SQS 標準佇列並傳送訊息](#)
- [建立 Amazon SQS FIFO 佇列並傳送訊息](#)
- [Amazon SQS 入門常見任務](#)

設定 Amazon SQS

首次使用 Amazon SQS 之前，您必須先完成下列步驟：

步驟 1：建立 AWS 帳戶 和 IAM 使用者

若要存取任何 AWS 服務，您必須先建立 [AWS 帳戶](#)，這是可使用 AWS 產品的 Amazon.com 帳戶。您可以使用 AWS 帳戶 來檢視您的活動和用量報告，以及管理身分驗證和存取。

為了避免將 AWS 帳戶 根使用者用於 Amazon SQS 動作，最佳實務是為需要管理 Amazon SQS 存取權的每個人建立 IAM 使用者。

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

註冊程序的一部分包括接聽電話或文字訊息，並在電話鍵盤上輸入驗證碼。

當您註冊時，AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 <https://aws.amazon.com/> 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center 和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者[AWS Management Console](#)身分登入。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的[以根使用者身分登入](#)。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

如需說明，請參閱《IAM 使用者指南》中的[為您的 AWS 帳戶 根使用者（主控台）啟用虛擬 MFA 裝置](#)。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[啟用 AWS IAM Identity Center](#)。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

如需使用 IAM Identity Center 目錄做為身分來源的教學課程，請參閱 AWS IAM Identity Center 《使用者指南》中的[使用預設值設定使用者存取權 IAM Identity Center 目錄](#)。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《使用者指南》中的[登入 AWS 存取入口網站](#)。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[建立許可集](#)。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[新增群組](#)。

步驟 2：授予程式設計存取權

若要使用 Amazon SQS 動作（例如，使用 Java 或透過 AWS Command Line Interface），您需要存取金鑰 ID 和私密存取金鑰。

Note

存取金鑰 ID 和私密存取金鑰是特定的 AWS Identity and Access Management。請勿將它們與其他服務的登入 AWS 資料混淆，例如 Amazon EC2 金鑰對。

如果使用者想要與 AWS 外部互動，則需要程式設計存取 AWS Management Console。授予程式設計存取權的方式取決於正在存取的使用者類型 AWS。

若要授與使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	到	根據
人力資源身分 (IAM Identity Center 中管理的使用者)	使用暫時登入資料簽署對 AWS CLI、AWS SDKs程式設計請求。AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱AWS Command Line Interface 《使用者指南》中的設定

哪個使用者需要程式設計存取權？	到	根據
		AWS CLI 要使用 AWS IAM Identity Center的。 • AWS SDKs、工具和 AWS APIs，請參閱 AWS SDKs 和工具參考指南中的 IAM Identity Center 身分驗證。
IAM	使用暫時登入資料簽署對 AWS CLI、AWS SDKs 程式設計請求。AWS APIs	遵循《IAM 使用者指南》中 將臨時登入資料與 AWS 資源搭配使用 的指示。
IAM	(不建議使用) 使用長期登入資料來簽署對 AWS CLI、AWS SDKs 程式設計請求。AWS APIs	請依照您要使用的介面所提供的指示操作。 • 如需 AWS CLI，請參閱 AWS Command Line Interface 《使用者指南》中的使用 IAM 使用者憑證進行身分驗證。 • AWS SDKs 和工具，請參閱 AWS SDKs 和工具參考指南中的使用長期憑證進行身分驗證。 • 對於 AWS APIs，請參閱《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。

步驟 3：準備好開始使用範本程式碼

本指南包含使用適用於 Java 的 AWS 開發套件的範例。若要執行範例程式碼，請遵循[適用於 Java 2.0 的 AWS 開發套件入門](#)中的設定指示進行。

您可以開發其他程式設計語言 AWS 的應用程式，例如 Go、JavaScript、Python 和 Ruby。如需詳細資訊，請參閱[要建置的工具 AWS](#)。

Note

您無需撰寫程式碼，就可以使用 AWS Command Line Interface (AWS CLI) 或 Windows PowerShell 等工具探索 Amazon SQS。您可以在 AWS CLI 命令參考的 [Amazon SQS 區段](#) 中找到 AWS CLI 範例。您可以在 [AWS Tools for PowerShell Cmdlet 參考](#) 的 Amazon Simple Queue Service 區段中找到 Windows PowerShell 範例。

後續步驟

您現在已準備好[開始](#)使用 AWS Management Console 管理 Amazon SQS 佇列和訊息。

了解 Amazon SQS 主控台

當您開啟 Amazon SQS 主控台時，請從導覽窗格中選擇佇列。佇列頁面提供作用中區域中所有佇列的相關資訊。

每個佇列項目都會提供有關佇列的基本資訊，包括其類型和金鑰屬性。[標準佇列](#)針對最大輸送量和最佳訊息排序進行最佳化，與[First-In-First-Out \(FIFO\)](#) 佇列不同，其會優先考慮訊息排序，以及需要嚴格訊息排序的應用程式唯一性。

Queues (2)		↻	Edit	Delete	Send and receive messages	Actions ▼	Create queue
🔍 Search queues by prefix		< 1 > ⚙️					
	Name ▲	Type ▼	Created ▼	Messages available ▼	Messages in flight ▼	Encryption ▼	Content-based deduplication ▼
☐	MyTestQueue	Standard	6/27/2022, 13:03:08 EDT	0	0	Disabled	-
☐	testFifo1.fifo	FIFO	6/27/2022, 13:03:41 EDT	0	0	Disabled	Disabled

互動式元素和動作

在佇列頁面中，您有多種管理佇列的選項：

1. 快速動作 – 與每個佇列名稱相鄰的下拉式功能表可讓您快速存取常見動作，例如傳送訊息、檢視或刪除訊息、設定觸發，以及刪除佇列本身。
2. 詳細檢視和組態 – 按一下佇列名稱會開啟其詳細資訊頁面，您可以在其中深入了解佇列設定和組態。在這裡，您可以調整訊息保留期間、可見性逾時和訊息大小上限等參數，以根據應用程式的需求自訂佇列。

MyTestQueue

Edit Delete Purge Send and receive messages Start DLQ redrive

Details Info

Name	Type	ARN
MyTestQueue	Standard	arn:aws:sqs:us-east-1:269704527654:MyTestQueue
Encryption	URL	Dead-letter queue
Disabled	https://sqs.us-east-1.amazonaws.com/269704527654/MyTestQueue	-

► More

SNS subscriptions Lambda triggers Dead-letter queue Monitoring Tagging Access policy Encryption Dead-letter queue redrive tasks

區域選擇和資源標籤

請確定您已正確 AWS 區域 存取和管理佇列。此外，請考慮使用資源標籤來組織和分類佇列，以便在 AWS 共用環境中實現更好的資源管理、成本分配和存取控制。

透過利用 Amazon SQS 主控台中提供的功能，您可以有效地管理您的訊息基礎設施、最佳化佇列效能，並確保應用程式可靠的訊息傳遞。

Amazon SQS 佇列類型

Amazon SQS 支援兩種佇列類型：[標準佇列](#)和 [FIFO](#) 佇列。使用下表來判斷哪個佇列最符合您的需求。

標準佇列	FIFO 佇列
無限制輸送量 – 標準佇列支援每秒每個動作 (SendMessage、ReceiveMessage 或) 非常高、幾乎無限制的 API 呼叫次數 DeleteMessage。這種高輸送量使它們非常適合需要快速處理大量訊息的使用案例，例如即時資料串流或大規模應用程式。雖然標準佇列會隨需求自動擴展，但必須監控使用模式以確保最佳效能，尤其是工作負載較高的區域。 At-least-once交付 – at-least-once交付，這表示每個訊息至少交付一次，但在某些情況下，由於	高輸送量 – 當您使用批次處理時，每個 API 方法 (ReceiveMessage、或) 的 FIFO 佇列每秒最多處理 3SendMessageBatch，000 則訊息 DeleteMessageBatch。此輸送量依賴每秒 300 個 API 呼叫，每個 API 呼叫處理一批 10 個 messages.By，啟用高輸送量模式，您可以擴展到每秒 30,000 個交易 (TPS)，並在訊息群組內輕鬆排序。沒有批次處理，FIFO 佇列支援每個 API 方法 (SendMessage、ReceiveMessage 或) 每秒最多 300 個 API 呼叫 DeleteMessage。如果您需要更多輸

標準佇列

重試或網路延遲，訊息可能會交付一次以上。您應該設計應用程式以使用等冪操作來處理潛在的重複訊息，以確保多次處理相同的訊息不會影響系統的狀態。

最高效的排序 – 提供最高效的排序，這表示當 Amazon SQS 嘗試按照傳送順序傳遞訊息時，不保證這一點。在某些情況下，訊息可能無法按順序送達，尤其是在高輸送量或故障復原的情況下。對於訊息處理順序至關重要的應用程式，您應該處理應用程式中的重新排序邏輯，或使用 FIFO 佇列進行嚴格排序保證。

耐用性和備援 – 標準佇列透過跨多個 AWS 可用區域存放每個訊息的多個副本，以確保高耐用性。這可確保訊息不會遺失，即使在基礎設施故障時也是如此。

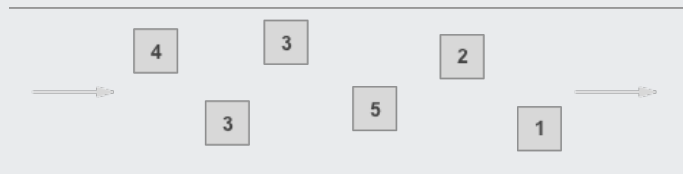
可見性逾時 – Amazon SQS 可讓您設定可見性逾時，以控制訊息在收到訊息後會隱藏多久，確保其他消費者在完全處理訊息或逾時過期之前不會處理訊息。

FIFO 佇列

送量，您可以透過 [AWS Support Center](#) 請求增加配額。若要啟用高輸送量模式，請參閱 [在 Amazon SQS 中啟用 FIFO 佇列的高輸送量](#)。

精確處理一次 – FIFO 佇列會傳送每則訊息一次，並保持可用狀態，直到您處理和刪除訊息為止。透過使用 [MessageDeduplicationId](#) 或內容型重複資料刪除等功能，即使因網路問題或逾時而重試，您也能防止重複的訊息。

First-in-first-out交付 – FIFO 佇列會確保您依每個訊息群組內傳送的順序接收訊息。透過將訊息分散到多個群組，您可以平行處理訊息，同時仍維持每個群組中的順序。



標準佇列	FIFO 佇列
當輸送量至關重要時，使用標準佇列在應用程式之間傳送資料，例如： - 將即時使用者請求與密集背景工作分離。允許使用者在處理背景中調整大小或編碼等任務時快速上傳媒體，確保快速回應時間而不會使系統過載。 - 將任務配置到多個工作者節點。將大量信用卡驗證請求分散到多個工作者節點，並使用等冪操作處理重複的訊息，以避免處理錯誤。 - 批次訊息以供未來處理。將多個項目排入佇列，以便批次新增至資料庫。由於無法保證訊息順序，因此請設計您的系統，以在必要時處理 out-of-order 的處理。	當事件順序很重要時，使用 FIFO 佇列在應用程式之間傳送資料，例如： - 請確定以正確的順序執行使用者輸入的命令。這是 FIFO 佇列的金鑰使用案例，其中命令順序至關重要。例如，如果使用者在應用程式中執行一系列動作，FIFO 佇列會確保動作的處理順序與輸入的順序相同。 - 以正確的順序傳送價格修改，以顯示正確的產品價格。FIFO 佇列可確保產品價格的多次更新到達並依序處理。如果沒有 FIFO，價格降低可能會在價格增加後處理，導致顯示不正確的資料。 - 在註冊帳戶之前，防止學生註冊課程。透過使用 FIFO 佇列，您可以確保註冊程序以正確的順序進行。系統會先處理帳戶註冊，然後處理課程註冊，防止註冊請求過早執行。

在 Amazon SQS 中實作請求回應系統

實作請求回應或遠端程序呼叫 (RPC) 系統時，請記住下列最佳實務：

- 在啟動時建立回覆佇列 – 而不是在每個訊息建立回覆佇列，而是在啟動時為每個生產者建立回覆佇列。使用關聯 ID 訊息屬性，有效率地將回應映射至請求。
- 避免在生產者之間共用回覆佇列 – 確保每個生產者都有自己的回覆佇列。共用回覆佇列可能會導致生產者接收其他生產者的回應訊息。

如需使用暫時佇列用戶端實作請求-回應模式的相關資訊，請參閱 [請求-回應訊息模式 \(虛擬佇列\)](#)。

建立 Amazon SQS 標準佇列並傳送訊息

您可以使用 Amazon SQS 主控台建立 [標準佇列](#) 並傳送訊息。本主題也強調最佳實務，包括避免佇列名稱中的敏感資訊，以及利用受管伺服器端加密。

使用 Amazon SQS 主控台建立標準佇列

Important

在 2022 年 8 月 17 日，預設伺服器端加密 (SSE) 已套用至所有 Amazon SQS 佇列。請勿在佇列名稱中新增個人身分識別資訊 (PII) 或其他機密或敏感資訊。許多 Amazon Web Services 都可以存取佇列名稱，包括計費和 CloudWatch 日誌。佇列名稱不適用於私有或敏感資料。

若要建立 Amazon SQS 標準佇列

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 選擇建立佇列。
3. 針對類型，依預設會設定標準佇列類型。

Note

您無法在建立佇列之後變更佇列類型。

4. 輸入佇列的名稱。
5. (選用) 主控台會設定佇列組態參數的預設值。在組態下，您可以為下列參數設定新值：
 - a. 在可見性逾時中，輸入持續時間和單位。範圍是從 0 秒至 12 小時。預設值為 30 秒。
 - b. 在訊息保留期間中，輸入持續時間和單位。範圍從 1 分鐘到 14 天。預設值為 4 天。
 - c. 對於遞送延遲，請輸入持續時間和單位。範圍是從 0 秒至 15 分鐘。預設值為 0 秒。
 - d. 在訊息大小上限中，輸入一個值。範圍介於 1 KB 到 256 KB 之間。預設值為 256 KB。
 - e. 針對接收訊息等待時間，輸入值。範圍是從 0 秒至 20 秒。預設值為 0 秒，它會設定短輪詢。任何非零值都會設定長輪詢。
6. (選擇性) 定義存取政策。存取政策會定義可存取佇列的帳戶、使用者和角色。存取政策也會定義使用者可存取的動作 (例如 [SendMessage](#)、[ReceiveMessage](#) 或 [DeleteMessage](#))。預設政策只允許佇列擁有者傳送和接收訊息。

若要複製存取政策，請執行下列其中一項動作：

- 選擇基本以設定誰可以將訊息傳送到佇列，以及誰可以從佇列接收訊息。主控台會根據您的選擇建立政策，並在唯讀 JSON 面板中顯示產生的存取政策。

- 選擇進階以直接修改 JSON 存取政策。這可讓您指定每個主體 (帳戶、使用者或角色) 可以執行的自訂動作集。
7. 對於再驅動允許政策，選擇啟用。選取下列其中一項：全部允許、依佇列或全部拒絕。選擇依佇列時，請依 Amazon Resource Name (ARN) 指定最多 10 個來源佇列的清單。
 8. Amazon SQS 預設會提供受管伺服器端加密。若要選擇加密金鑰類型，或停用 Amazon SQS 受管伺服器端加密，請展開加密。如需加密金鑰類型的詳細資訊，請參閱 [使用 SQS 受管加密金鑰設定佇列的伺服器端加密](#) 和 [使用 Amazon SQS 主控台設定佇列的伺服器端加密](#)。
-  **Note**

啟用 SSE 後，對加密佇列的匿名 SendMessage 和 ReceiveMessage 請求將被拒絕。Amazon SQS 安全性最佳實務建議您不要使用匿名請求。如果您希望將匿名請求傳送到 Amazon SQS 佇列，請務必停用 SSE。
9. (選用) 若要設定[無效字母佇列](#)以接收無法傳遞的訊息，請展開無效字母佇列。
 10. (選擇性) 若要將[標籤](#)新增至佇列，請展開標籤。
 11. 選擇建立佇列。Amazon SQS 會建立佇列並顯示佇列的詳細資訊頁面。

Amazon SQS 會在整個系統中傳播有關新佇列的資訊。由於 Amazon SQS 是分散式系統，因此在主控台在佇列頁面上顯示佇列之前，您可能會遇到輕微的延遲。

使用標準佇列傳送訊息

建立佇列之後，您可以傳送訊息給佇列。

1. 在左側導覽窗格中，選擇佇列。在佇列清單中，選取您剛建立的佇列。
2. 在動作中選擇傳送及接收訊息。

主控台會顯示傳送和接收訊息頁面。

3. 在訊息內文中，輸入訊息文字。
4. 對於標準佇列，您可以輸入遞送延遲的值並選擇單位。例如，輸入 60 並選擇秒數。如需詳細資訊，請參閱 [Amazon SQS 訊息計時器](#)。
5. 選擇傳送訊息。

訊息傳送完畢後，主控台會顯示成功訊息。選擇檢視詳細資訊以顯示已傳送訊息的相關資訊。

建立 Amazon SQS FIFO 佇列並傳送訊息

您可以使用 主控台 建立 Amazon SQS FIFO 佇列並傳送訊息。本主題說明如何設定佇列參數，包括可見性逾時、訊息保留和重複資料刪除，同時遵循安全最佳實務，例如避免佇列名稱中的敏感資訊並啟用伺服器端加密。它還涵蓋定義存取政策、設定無效字母佇列，以及使用 FIFO 特定屬性傳送訊息，例如訊息群組 ID 和重複資料刪除 ID。

使用 Amazon SQS 主控台建立 FIFO 佇列

您可以使用 Amazon SQS 主控台來建立 [FIFO 佇列](#)。主控台提供所有設定的預設值 (佇列名稱除外)。

Important

在 2022 年 8 月 17 日，預設伺服器端加密 (SSE) 已套用至所有 Amazon SQS 佇列。請勿在佇列名稱中新增個人身分識別資訊 (PII) 或其他機密或敏感資訊。許多 Amazon Web Services 都可以存取佇列名稱，包括計費和 CloudWatch 日誌。佇列名稱不適用於私有或敏感資料。

若要建立 Amazon SQS FIFO 佇列

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 選擇建立佇列。
3. 針對類型，依預設會設定標準佇列類型。若要建立 FIFO 佇列，請選擇 FIFO。

Note

您無法在建立佇列之後變更佇列類型。

4. 輸入佇列的名稱。

FIFO 佇列名稱結尾必須是 `.fifo` 尾碼。尾碼會計入 80 個字元的佇列名稱配額中。若要判斷佇列是否為 [FIFO](#)，可以檢查佇列名稱是否以尾碼結尾。

5. (選用) 主控台會設定佇列 [組態參數](#) 的預設值。在組態下，您可以為下列參數設定新值：
 - a. 在可見性逾時中，輸入持續時間和單位。範圍是從 0 秒至 12 小時。預設值為 30 秒。
 - b. 在訊息保留期間中，輸入持續時間和單位。範圍從 1 分鐘到 14 天。預設值為 4 天。
 - c. 對於遞送延遲，請輸入持續時間和單位。範圍是從 0 秒至 15 分鐘。預設值為 0 秒。

- d. 在訊息大小上限中，輸入一個值。範圍介於 1 KB 到 256 KB 之間。預設值為 256 KB。
- e. 針對接收訊息等待時間，輸入值。範圍是從 0 秒至 20 秒。預設值為 0 秒，它會設定[短輪詢](#)。任何非零值都會設定長輪詢。
- f. 對於 FIFO 佇列，請選擇內容型重複資料刪除以啟用內容型重複資料刪除。預設設定為停用。
- g. (選用) 若要讓 FIFO 佇列啟用更高輸送量以傳送和接收佇列中的訊息，請選擇啟用高輸送量 FIFO。

選擇此選項會將相關選項 (重複資料刪除範圍和 FIFO 輸送量限制) 變更為啟用 FIFO 佇列高輸送量的必要設定。如果您變更使用高輸送量 FIFO 所需的任何設定，則佇列的正常輸送量將生效，而重複資料刪除會依指定方式執行。如需詳細資訊，請參閱[Amazon SQS 中 FIFO 佇列的高輸送量](#)及[Amazon SQS 訊息配額](#)。

6. (選用) 定義存取政策。[存取政策](#)會定義可存取佇列的帳戶、使用者和角色。存取政策也會定義使用者可存取的動作 (例如 [SendMessage](#)、[ReceiveMessage](#) 或 [DeleteMessage](#))。預設政策只允許佇列擁有者傳送和接收訊息。

若要複製存取政策，請執行下列其中一項動作：

- 選擇基本以設定誰可以將訊息傳送到佇列，以及誰可以從佇列接收訊息。主控台會根據您的選擇建立政策，並在唯讀 JSON 面板中顯示產生的存取政策。
 - 選擇進階以直接修改 JSON 存取政策。這可讓您指定每個主體 (帳戶、使用者或角色) 可以執行的自訂動作集。
7. 對於再驅動允許政策，選擇啟用。選取下列其中一項：全部允許、依佇列或全部拒絕。選擇依佇列時，請依 Amazon Resource Name (ARN) 指定最多 10 個來源佇列的清單。
 8. Amazon SQS 預設會提供受管伺服器端加密。若要選擇加密金鑰類型，或停用 Amazon SQS 受管伺服器端加密，請展開加密。如需加密金鑰類型的詳細資訊，請參閱 [使用 SQS 受管加密金鑰設定佇列的伺服器端加密](#) 和 [使用 Amazon SQS 主控台設定佇列的伺服器端加密](#)。

 Note

啟用 SSE 後，對加密佇列的匿名 `S SendMessage` 和 `R ReceiveMessage` 請求將被拒絕。Amazon SQS 安全性最佳實務建議您不要使用匿名請求。如果您希望將匿名請求傳送到 Amazon SQS 佇列，請務必停用 SSE。

9. (選用) 若要設定[無效字母佇列](#)以接收無法傳遞的訊息，請展開無效字母佇列。
10. (選擇性) 若要將[標籤](#)新增至佇列，請展開標籤。
11. 選擇建立佇列。Amazon SQS 會建立佇列並顯示佇列的詳細資訊頁面。

Amazon SQS 會在整個系統中傳播有關新佇列的資訊。由於 Amazon SQS 是分散式系統，因此在主控台在佇列頁面上顯示佇列之前，您可能會遇到輕微的延遲。

建立佇列之後，您可以向其[傳送訊息](#)，以及[接收和刪除訊息](#)。您也可以[編輯](#)佇列類型以外的任何佇列組態設定。

使用 FIFO 佇列傳送訊息

建立佇列之後，您可以傳送訊息給佇列。

1. 在左側導覽窗格中，選擇佇列。在佇列清單中，選取您剛建立的佇列。
2. 在動作中選擇傳送及接收訊息。

主控台會顯示傳送和接收訊息頁面。

3. 在訊息內文中，輸入訊息文字。
4. 對於先出 (FIFO) 佇列，請輸入訊息群組 ID。如需詳細資訊，請參閱 [Amazon SQS 中的 FIFO 佇列交付邏輯](#)。
5. (選擇性) 對於 FIFO 佇列，您可以輸入訊息重複資料刪除 ID。如果您啟用佇列的內容型重複資料刪除功能，則不需要訊息重複資料刪除 ID。如需詳細資訊，請參閱 [Amazon SQS 中的 FIFO 佇列交付邏輯](#)。
6. FIFO 佇列不支援個別訊息的計時器。如需詳細資訊，請參閱 [Amazon SQS 訊息計時器](#)。
7. 選擇傳送訊息。

訊息傳送完畢後，主控台會顯示成功訊息。選擇檢視詳細資訊以顯示已傳送訊息的相關資訊。

Amazon SQS 入門常見任務

建立佇列並了解如何傳送、接收和刪除訊息後，建議您嘗試下列動作：

- 觸發 [Lambda 函數](#) 以自動處理傳入訊息，啟用事件驅動的工作流程，而不需要持續輪詢。
- [設定佇列，包括 SSE 和其他功能](#)。
- [傳送具有屬性的訊息](#)。
- [從 VPC 傳送訊息](#)。
- 探索 Amazon SQS [的功能](#) 和 [架構](#)。
- 探索可協助您充分利用 Amazon SQS 的 [指導方針和警告](#)。
- 探索 AWS SDK 的 Amazon SQS 範例，例如 [AWS SDK for Java 2.x 開發人員指南](#)。

- 了解 [Amazon SQS AWS CLI 命令](#)。
- 了解 [Amazon SQS API 動作](#)。
- 了解如何以程式設計方式與 Amazon SQS 互動。請參閱[使用 APIs](#)和探索[AWS 開發中心](#)：
 - [Java](#)
 - [JavaScript](#)
 - [PHP](#)
 - [Python](#)
 - [Ruby](#)
 - [Windows 與 .NET](#)
- 了解如何監控[成本和資源](#)。
- 了解如何[保護您的資料](#)。
- 進一步了解 [Amazon SQS 工作流程](#)。

管理 Amazon SQS 佇列

了解如何使用主控台管理 Amazon SQS 佇列，包括編輯佇列設定、接收和刪除訊息、確認佇列清空，以及刪除或清除佇列。了解有效處理訊息的最佳實務，例如使用長輪詢、管理可見性逾時，以及透過監控儀表板或 驗證指標 AWS CLI。遵循實際步驟來維護佇列並有效處理訊息，同時將中斷降至最低。

使用主控台編輯 Amazon SQS 佇列

您可以使用 Amazon SQS 主控台來編輯佇列組態參數（佇列類型除外），並視需要修改或移除功能。

若要編輯 Amazon SQS 佇列 (主控台)

1. 開啟 Amazon SQS 主控台的[佇列](#)頁面。
2. 選取佇列，然後選擇編輯。
3. (選用) 在組態下，更新佇列的[組態參數](#)。
4. (選用) 若要更新[存取政策](#)，請在存取政策下修改 JSON 政策。
5. (選用) 若要更新無效字母佇列[再驅動允許政策](#)，請展開再驅動允許政策。
6. (選用) 若要更新或移除[加密](#)，請展開加密。
7. (選用) 若要新增、更新或移除[無效字母佇列](#) (可讓您接收無法傳遞的訊息)，請展開無效字母佇列。
8. (選用) 若要新增、更新或移除佇列的[標籤](#)，請展開標籤。
9. 選擇 Save (儲存)。

- 主控台會顯示佇列的詳細資訊頁面。

在 Amazon SQS 中接收和刪除訊息

傳送訊息至 Amazon SQS 佇列後，您可以擷取和刪除它們，以處理您的應用程式工作流程。此程序可確保安全可靠的訊息處理。本主題會逐步引導您使用 Amazon SQS 主控台擷取和刪除訊息，並說明金鑰設定以最佳化此操作。以下是接收和刪除訊息的重要概念：

1. 接收訊息

- 當您從 Amazon SQS 佇列擷取訊息時，您無法將特定訊息設為目標。反之，指定單一請求中要擷取的訊息數量上限（最多 10 個）。
- 由於 Amazon SQS 的分散式性質，從訊息很少的佇列擷取可能會傳回空的回應。若要緩解此問題：

- 使用長輪詢，這會等到訊息可用或輪詢逾時。這種方法可減少不必要的輪詢成本，並提高效率。
- 視需要重新發出請求。

2. 訊息可見性和刪除

- 擷取後不會自動刪除訊息。此功能可確保您可以在應用程式故障或網路中斷時重新處理訊息。
- 處理後，您必須明確傳送刪除請求，以永久移除訊息。此動作會確認處理成功。
- 使用 Amazon SQS 主控台擷取的訊息會保持可見，以供重新擷取。調整自動化環境的可見性逾時設定，以便在處理其他消費者的訊息時暫時隱藏訊息。

3. 可見性逾時

- 此設定會決定訊息在擷取後會保持隱藏的時間長度。設定適當的逾時，以確保訊息只處理一次，並防止在分散式處理期間重複。

使用主控台接收和刪除訊息

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 在佇列頁面上，選擇您要接收訊息的佇列，然後選擇傳送和接收訊息。
4. 在傳送和接收訊息頁面上，選取訊息的輪詢。

Amazon SQS 會顯示進度列，指出輪詢持續時間。擷取的訊息會出現在訊息區段中，顯示：

- 訊息 ID
 - 傳送日期
 - 大小
 - 接收計數
5. 若要刪除訊息，請選擇您要移除的訊息，然後選取刪除。

在刪除訊息對話方塊中，選擇刪除以確認刪除。

如需進階操作的詳細資訊，包括以 API 為基礎的訊息擷取和刪除，請參閱 [Amazon SQS API 參考指南](#)。

確認 Amazon SQS 佇列是空的

在大多數情況下，您可以使用[長輪詢](#)來判斷佇列是否為空。在極少數情況下，即使佇列仍包含訊息，您也可能會收到空白回應，尤其是當您在建立佇列時為接收訊息等待時間指定較低的值時。本節說明如何確認佇列是空的。

若要確認佇列是空的 (主控台)

1. 停止所有生產者傳送訊息。
2. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
3. 在導覽窗格中，選擇 Queues (佇列)。
4. 在佇列頁面上，選擇佇列。
5. 選擇 Monitoring (監控) 索引標籤。
6. 在監控儀表板的右上角，選擇 [重新整理] 符號旁邊的向下箭頭。從下拉式選單中，選擇自動重新整理。將重新整理間隔保留為 1 分鐘。
7. 請觀察下列儀表板：
 - 延遲訊息的大約數量
 - 不可見訊息的大約數量
 - 可見訊息的大約數量

當它們都顯示幾分鐘的 0 值時，佇列為空白。

確認佇列是空的 (AWS CLI，AWS API)

1. 停止所有生產者傳送訊息。
2. 重複執行下列其中一個命令：
 - AWS CLI: [get-queue-attributes](#)
 - AWS API : [GetQueueAttributes](#)
3. 觀察下列屬性的指標：
 - ApproximateNumberOfMessagesDelayed
 - ApproximateNumberOfMessagesNotVisible
 - ApproximateNumberOfMessagesVisible

當它們都位於 0 幾分鐘後，佇列為空白。

如果您依賴 Amazon CloudWatch 指標，請確保您看到多個連續的零資料點，然後再考慮該佇列為空白。如需 CloudWatch 指標的相關資訊，請參閱 [適用於 Amazon SQS 的可用 CloudWatch 指標](#)。

刪除 Amazon SQS 佇列

如果您不再使用 Amazon SQS 佇列，而且不打算在近期使用它，請刪除佇列。

Tip

如果您想在刪除佇列之前先確認佇列是空的，請參閱 [確認 Amazon SQS 佇列是空的](#)。

您可以刪除佇列，即使佇列不是空的。如果您要刪除佇列中的訊息但不刪除佇列本身，您可以[清除佇列](#)。

若要刪除佇列 (主控台)

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 在佇列頁面中，選擇要刪除的佇列。
4. 選擇 刪除。
5. 在刪除佇列對話方塊中，輸入 **delete** 以確認刪除。
6. 選擇 刪除。

刪除佇列 (AWS CLI 和 API)

根據您的需求選擇適當的方法來刪除佇列：

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

使用 Amazon SQS 主控台從佇列清除訊息

若要保留 Amazon SQS 佇列，但移除其所有訊息，您可以清除佇列。這將刪除所有訊息，包括目前隱藏的訊息（在飛行中）。清除程序最多可能需要 60 秒，因此無論佇列的大小為何，請等待整整 60 秒。

Important

清除佇列後，您無法擷取任何已刪除的訊息。

清除佇列（主控台）

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 在佇列頁面上，選擇要清除的佇列。
4. 從動作中，選擇清除。
5. 在清除佇列對話方塊中，輸入 **purge** 並選擇清除，以確認清除。
 - 佇列中的所有訊息都已清除。主控台會顯示確認橫幅。

Amazon SQS 標準佇列

Amazon SQS 提供標準佇列做為預設佇列類型，支援諸如 [SendMessage](#)、[ReceiveMessage](#) 和 等動作的每秒幾乎無限次數的 API 呼叫 [DeleteMessage](#)。標準佇列可確保 at-least-once 訊息傳遞，但由於高度分佈的架構，可能會傳遞多個訊息副本，且訊息偶爾可能會未依順序送達。儘管如此，標準佇列會盡力維持訊息傳遞的順序。

當您使用 傳送訊息時 [SendMessage](#)，Amazon SQS 會在確認訊息之前，以備援方式將訊息存放在多個可用區域 (AZs) 中。此備援可確保沒有任何單一電腦、網路或 AZ 失敗會導致無法存取訊息。

您可以使用 Amazon SQS 主控台建立和設定佇列。如需詳細說明，請參閱 [使用 Amazon SQS 主控台建立標準佇列](#)。如需 Java 特定範例，請參閱 [Amazon SQS Java 開發套件範例](#)。

標準佇列的使用案例

標準訊息佇列適用於各種案例，只要應用程式可以處理可能多次送達或無法排序的訊息。範例包括：

- 從密集的背景工作解耦即時使用者請求 – 使用者可以在系統調整大小或在背景編碼媒體時上傳媒體。
- 將任務配置到多個工作者節點 – 例如，處理大量的信用卡驗證請求。
- 批次處理訊息以供未來處理 – 排程要在稍後新增至資料庫的多個項目。

如需標準佇列相關配額的資訊，請參閱 [Amazon SQS 標準佇列配額](#)。

關於使用標準佇列的最佳實務，請參閱 [Amazon SQS 最佳實務](#)。

Amazon SQS at-least-once 交付

Amazon SQS 會在多個伺服器上存放訊息的副本，以供備援使用並提供高可用性。偶爾在接收或刪除訊息時，存放訊息副本的其中一個伺服器可能會無法使用。

如果發生這種情況，訊息的複本不會在無法使用的伺服器上刪除，而且您可能會在收到訊息時再次收到該訊息複本。請將您的應用程式設為等冪 (若相同訊息處理一次以上應不會有不良影響)。

Amazon SQS 佇列和訊息識別碼

本主題說明標準 和 FIFO 佇列的識別符。這些識別碼可以協助您尋找和操作特定的佇列和訊息。

Amazon SQS 標準佇列的識別符

如需下列識別碼的相關資訊，請參閱 [Amazon Simple Queue Service API 參考](#)。

佇列名稱和 URL

建立新的佇列時，必須指定在您 AWS 帳戶和區域中的唯一佇列名稱。Amazon SQS 會為您建立的佇列指派一個識別碼，稱為佇列 URL，內含佇列名稱及其他 Amazon SQS 元件。當您想要在佇列上執行動作時，即需提供佇列 URL。

以下為是 AWS 帳號為 123456789012 的使用者所擁有之 MyQueue 佇列的佇列 URL。

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

您可以列出佇列並剖析跟在帳戶號碼後面的字串，便可以程式設計方式來擷取佇列的 URL。如需詳細資訊，請參閱 [ListQueues](#)。

訊息 ID

每則訊息會收到 Amazon SQS 在 [SendMessage](#) 回應中傳回給您的系統指定訊息 ID。此識別碼可以用來辨識訊息。訊息 ID 的長度上限為 100 個字元。

接收控點

每次從佇列接收訊息時，便會收到該訊息的接收控點。控點是與接收訊息的動作有所關聯，而非訊息本身。若要刪除訊息或變更訊息的可見性，請務必提供接收控點 (而非訊息 ID)。因此必須先接收訊息才能刪除訊息 (無法將訊息放進佇列然後再將之收回)。接收控點的長度上限為 1024 個字元。

Important

若某則訊息收到一次以上，每次收到訊息時都會收到一個不同的接收控點。當您提出刪除訊息的請求時，必須提供最近接收的接收控點 (否則無法刪除訊息)。

以下是分成三行的接收控點範例。

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

Amazon SQS FIFO 佇列

FIFO First-In-First-Out佇列具有[標準佇列](#)的所有功能，但旨在增強應用程式之間的傳訊，當操作順序和事件至關重要，或無法容忍重複時。

FIFO 佇列最重要的功能是 [FIFO First-In-First-Out交付](#)和[一次性處理](#)：

- 傳送和接收訊息的順序會嚴格保留，且訊息會傳送一次，並持續無法使用，直到取用者處理和刪除為止。
- 此種佇列不會出現重複的情況。

此外，FIFO 佇列支援允許單一佇列中多個排序訊息群組的訊息群組。FIFO 佇列中的訊息群組數目沒有配額。

您可能使用 FIFO 佇列的情況範例如下：

1. 訂單至關重要的電子商務訂單管理系統
2. 與需要按順序處理事件的第三方系統整合
3. 依照輸入的順序處理使用者輸入的內容
4. 通訊和聯網 - 依相同的順序傳送和接收資料和資訊
5. 電腦系統 - 確保使用者輸入的指令以正確的順序執行
6. 教育機構 - 學生必須先註冊帳戶，否則無法註冊課程
7. 網上售票系統 - 門票依照先到先得的原則分發

Note

FIFO 佇列也提供恰好一次 (exactly-once) 的處理方式，但每秒交易次數 (TPS) 有所限制。您可以將 Amazon SQS 高輸送量模式搭配 FIFO 佇列使用，以提高交易限制。如需使用高輸送量模式的詳細資訊，請參閱 [Amazon SQS 中 FIFO 佇列的高輸送量](#)。如需有關輸送量配額的詳細資訊，請參閱 [the section called “訊息配額”](#)。

Amazon SQS FIFO 佇列可在 Amazon SQS 可用的所有區域中使用。

如需使用複雜排序的 FIFO 佇列的詳細資訊，請參閱[使用 Amazon SQS FIFO 佇列解決複雜的訂購挑戰](#)。

如需有關如何使用 Amazon SQS 主控台建立和設定佇列的資訊，請參閱 [使用 Amazon SQS 主控台建立標準佇列](#)。如需 Java 範例，請參閱 [Amazon SQS Java 開發套件範例](#)。

如需使用 FIFO 佇列的最佳實務，請參閱 [Amazon SQS 最佳實務](#)。

Amazon SQS FIFO 佇列金鑰術語

以下重要用語有助於更加了解 FIFO 佇列的功能。如需詳細資訊，請參閱 [Amazon Simple Queue Service API 參考](#)。

用戶端

Amazon SQS 緩衝非同步用戶端目前不支援 FIFO 佇列。

訊息重複資料刪除 ID

用於 Amazon SQS FIFO 佇列的字符，可唯一識別訊息並防止重複。如果在 5 分鐘的重複資料刪除間隔內傳送具有相同重複資料刪除 ID 的多個訊息，這些訊息會視為重複，而且只會傳送一個副本。如果您未指定重複資料刪除 ID，且已啟用內容型重複資料刪除，Amazon SQS 會透過雜湊訊息內文來產生重複資料刪除 ID。此機制透過消除指定時間範圍內的重複訊息，確保準確交付一次。

Note

即使訊息已接收和刪除，Amazon SQS 仍會繼續追蹤重複資料刪除 ID。

訊息群組 ID

MessageGroupId 是只在 Amazon SQS FIFO (First-In-First-Out) 佇列中使用的屬性，可將訊息組織成不同的群組。相同訊息群組內的訊息一律以嚴格順序一次處理一個訊息，確保不會同時處理來自相同群組的兩個訊息。標準佇列不會使用 MessageGroupId，也不會提供訂購保證。如果需要嚴格排序，請改用 FIFO 佇列。

接受請求嘗試 ID

接收請求嘗試 ID 是用來刪除 Amazon SQS 中重複 [ReceiveMessage](#) 呼叫的唯一權杖。

序號

Amazon SQS 指派給每則訊息的大型非連續數字。

服務

如果您的應用程式使用多個 AWS 服務，或混合使用 AWS 和外部服務，請務必了解哪些服務功能不支援 FIFO 佇列。

雖然允許您將 FIFO 佇列設定為目標，但傳送通知至 Amazon SQS 的部分 AWS 或外部服務可能與 FIFO 佇列不相容。

下列 AWS 服務的功能目前與 FIFO 佇列不相容：

- [Amazon S3 事件通知](#)
- [Auto Scaling Lifecycle Hooks](#)
- [AWS IoT 規則動作](#)
- [AWS Lambda 無效字母佇列](#)

如需有關其他服務與 FIFO 佇列之間相容性的資訊，請參閱您的服務說明文件。

Amazon SQS 中的 FIFO 佇列交付邏輯

下列概念說明 Amazon SQS FIFO 佇列如何處理訊息的傳送和接收，特別是在處理訊息排序和訊息群組 IDs 時。

傳送訊息

Amazon SQS FIFO 佇列會使用唯一的重複資料刪除 IDs 和訊息群組 IDs 保留訊息順序。本主題強調訊息群組 IDs 在群組內維持嚴格排序的重要性，並強調確保跨多個生產者可靠、有序訊息交付的最佳實務。

1. 訂單保留

- 當多個訊息連續傳送到具有唯一訊息重複資料刪除 IDs FIFO 佇列時，Amazon SQS 會儲存它們並確認其傳輸。然後，這些訊息會依其傳輸的確切順序接收和處理。

2. 訊息群組 ID

- 在 FIFO 佇列中，訊息會根據其訊息群組 ID 排序。如果多個生產者或執行緒以相同的訊息群組 ID 傳送訊息，Amazon SQS 會確保它們按照到達順序進行儲存和處理。
- 最佳實務：為保證多個生產者之間的嚴格訊息順序，請為每個生產者的所有訊息指派唯一的訊息群組 ID。

3. 每個群組的排序

- FIFO 佇列邏輯適用於每個訊息群組 ID：
 - 每個訊息群組 ID 代表一組不同的排序訊息。
 - 在訊息群組 ID 中，所有訊息都會以嚴格順序傳送和接收。
 - 具有不同訊息群組 IDs 的訊息可能會抵達，或以彼此相對的順序處理。
- 要求 - 您必須將訊息群組 ID 與每個訊息建立關聯。如果傳送訊息時沒有群組 ID，動作會失敗。
- 單一群組案例 - 如果您需要嚴格處理所有訊息，請針對每則訊息使用相同的訊息群組 ID。

接收訊息

Amazon SQS FIFO 佇列會處理訊息擷取，包括批次處理、FIFO 訂單保證，以及請求特定訊息群組 IDs 的限制。本主題說明 Amazon SQS 如何擷取訊息群組 IDs 同時維持嚴格的排序和可見性規則。

1. 批次擷取

- 從具有多個訊息群組 IDs FIFO 佇列接收訊息時，Amazon SQS：
 - 在單一呼叫中嘗試以相同的訊息群組 ID 傳回盡可能多的訊息。
 - 允許其他取用者同時處理來自不同訊息群組 IDs 的訊息。
- 重要釐清
 - 您可能會在一個批次中收到來自相同訊息群組 ID 的多則訊息（使用 `MaxNumberOfMessages` 參數在單一呼叫中最多 10 則訊息）。
 - 不過，您無法在後續請求中接收來自相同訊息群組 ID 的其他訊息，直到：
 - 目前接收的訊息已刪除，或
 - 它們會再次顯示（例如，在可見性逾時過期之後）。

2. FIFO 訂單保證

- 批次中擷取的訊息會在群組中保留其 FIFO 順序。
- 如果同一訊息群組 ID 可用的訊息少於 10 則 Amazon SQS 可能會在相同批次中包含來自其他訊息群組 IDs 的訊息，但每個群組會保留 FIFO 順序。

3. 消費者限制

- 您無法明確請求接收來自特定訊息群組 ID 的訊息。

多次重試

生產者和消費者可以在 Amazon SQS FIFO 佇列中安全地重試失敗的動作，而不會中斷訊息順序或引入重複項目。本主題重點介紹重複資料刪除 IDs 和可見性逾時如何在重試期間確保訊息完整性。

1. 生產者重試

- 如果 [SendMessage](#) 動作失敗，生產者可以使用相同的訊息重複資料刪除 ID 重試傳送訊息多次。
- 只要生產者在重複資料刪除間隔到期之前收到至少一個確認，就會重試：
 - 請勿引入重複的訊息。
 - 請勿中斷訊息順序。

2. 消費者重試

- 如果 [ReceiveMessage](#) 動作失敗，消費者可以使用相同的接收請求嘗試 ID，視需要重試多次。
- 只要消費者在可見性逾時到期之前收到至少一個確認，就會重試：
 - 請勿中斷訊息順序。

有關 FIFO 行為的其他備註

了解如何處理可見性逾時、使用多個訊息群組 IDs 啟用平行處理，並確保在單一群組案例中進行嚴格的循序處理。

1. 處理可見性逾時

- 當訊息被擷取但未刪除時，在可見性逾時過期之前，它會保持隱藏狀態。
- 在刪除第一個訊息或再次顯示之前，不會傳回來自相同訊息群組 ID 的其他訊息。

2. 並行和平行處理

- FIFO 佇列允許平行處理不同訊息群組 IDs 的訊息。
- 若要最大化並行，請使用多個訊息群組 IDs 為獨立工作流程設計系統。

3. 單一群組案例

- 若要嚴格循序處理 FIFO 佇列中的所有訊息，請針對佇列中的所有訊息使用單一訊息群組 ID。

更好地理解範例

以下是說明 Amazon SQS 中 FIFO 佇列行為的實際案例。

1. 案例 1：單一群組 ID

- 生產者使用相同的訊息群組 ID 群組 A 傳送五則訊息。
- 消費者以 FIFO 順序接收這些訊息。在消費者刪除這些訊息或可見性逾時過期之前，不會收到群組 A 的其他訊息。

2. 案例 2：多個群組 IDs

- 生產者將五則訊息傳送至群組 A，並將 5 傳送至群組 B。
- 消費者 1 處理來自群組 A 的訊息，而消費者 2 處理來自群組 B 的訊息。這可讓平行處理在每個群組中維持嚴格的順序。

3. 案例 3：批次擷取

- 生產者將七則訊息傳送至群組 A，並將三則訊息傳送至群組 B。
- 單一取用者最多擷取 10 則訊息。如果佇列允許，可能會傳回：
 - 來自群組 A 的七則訊息，以及來自群組 B 的三則訊息（如果單一群組可用的訊息較少，則為更少）。

Amazon SQS 中的精確一次處理

與標準佇列不同，FIFO 佇列不會引進重複的訊息。FIFO 佇列有助於避免將重複的資訊傳送至佇列。若是在 5 分鐘的刪除重複資料的間隔內重試 `SendMessage` 動作，Amazon SQS 不會將任何重複資料引進佇列內。

若要設定重複資料刪除，您必須執行以下其中一項：

- 啟用內容型重複資料刪除功能。如此可指示 Amazon SQS 使用 SHA-256 雜湊，以訊息的本文來產生訊息重複資料刪除 ID—而非使用訊息的屬性。如需詳細資訊，請參閱《Amazon Simple Queue Service API 參考》中有關 [CreateQueue](#)、[GetQueueAttributes](#) 和 [SetQueueAttributes](#) 動作的文件。
- 請明確提供該訊息的訊息重複資料刪除 ID (或檢視序號)。如需詳細資訊，請參閱《Amazon Simple Queue Service API 參考》中有關 [SendMessage](#)、[SendMessageBatch](#) 和 [ReceiveMessage](#) 動作的文件。

從標準佇列移至 Amazon SQS 中的 FIFO 佇列

如果您現有的應用程式使用標準佇列，而且您想要利用 FIFO 佇列的排序或恰好一次的處理功能，則需要正確設定佇列和應用程式。

重要考量事項

- 建立 FIFO 佇列：您無法將現有的標準佇列轉換為 FIFO 佇列。您必須為應用程式建立新的 FIFO 佇列，或刪除現有的標準佇列，並將其重新建立為 FIFO 佇列。
- 延遲參數：FIFO 佇列不支援每個訊息的延遲，僅支援每個佇列的延遲。如果您的應用程式在每則訊息上設定 `DelaySeconds` 參數，您必須改為修改參數以設定在整個佇列 `DelaySeconds` 上。

- 訊息群組 ID：為每個傳送的訊息提供訊息群組 ID。此 ID 可平行處理訊息，同時維持其個別順序。使用訊息群組 ID 的精細商業維度，以更好地擴展 FIFO 佇列。您分發訊息的目標訊息群組 IDs 越多，可供取用的訊息數量就越多。
- 高輸送量模式：針對 FIFO 佇列使用建議的[高輸送量模式](#)，以達到更高的輸送量。如需簡訊配額的詳細資訊，請參閱 [Amazon SQS 訊息配額](#)。

移至 FIFO 佇列的檢查清單

將訊息傳送至 FIFO 佇列前，請先確認以下項目：

1. 設定延遲設定

- 修改您的應用程式以移除每個訊息的延遲。
- 在整個佇列上設定 DelaySeconds 參數。

2. 設定訊息群組 IDs

- 根據業務維度指定訊息群組 ID，將訊息組織成訊息群組。
- 使用更精細的業務維度來改善可擴展性。

3. 處理重複訊息刪除

- 如果您的應用程式無法使用相同的訊息內文傳送訊息，請為每個訊息提供唯一的訊息重複資料刪除 ID。
- 如果您的應用程式傳送具有唯一訊息內文的訊息，請啟用內容型重複資料刪除。

4. 設定取用者

- 一般而言，消費者不需要變更程式碼。
- 如果處理訊息需要很長的時間，且可見性逾時設為高，請考慮將接收請求嘗試 ID 新增至每個 ReceiveMessage 動作。這有助於在聯網失敗時重試接收嘗試，並防止佇列因接收嘗試失敗而暫停。

透過遵循這些步驟，您可以確保您的應用程式正確使用 FIFO 佇列，充分利用其排序和恰好一次的處理功能。如需詳細資訊，請參閱 [Amazon Simple Queue Service API 參考](#)。

Amazon SQS FIFO 佇列和 Lambda 並行行為

透過將 FIFO First-In-First-Out 佇列與 Lambda 搭配使用，您可以確保在每個訊息群組內依序處理訊息。Lambda 函數不會同時為相同的訊息群組執行多個執行個體，因此會維護順序。不過，它可以擴展

以平行處理多個訊息群組，確保有效處理佇列的工作負載。以下幾點說明處理來自 Amazon SQS FIFO 佇列的訊息時，Lambda 函數在訊息群組 IDs 方面的行為：

- 每個訊息群組的單一執行個體：在任何時間點，只有一個 Lambda 執行個體會處理來自特定訊息群組 ID 的訊息。這可確保相同群組中的訊息會依序處理，以維持 FIFO 序列的完整性。
- 同時處理不同的群組：Lambda 可以使用多個執行個體同時處理來自不同訊息群組 IDs 的訊息。這表示當 Lambda 函數的一個執行個體正在處理來自一個訊息群組 ID 的訊息時，其他執行個體可以同時處理來自其他訊息群組 IDs 的訊息，利用 Lambda 的並行功能來平行處理多個群組。

FIFO 佇列訊息分組

FIFO 佇列可確保以傳送訊息的確切順序處理訊息。他們使用訊息群組 ID 來分組應依序處理的訊息。

相同訊息群組內的訊息會依序處理，且一次只會處理每個群組中的訊息，以維持此順序。

Lambda 與 FIFO 佇列並行

建立佇列之後，您可以傳送訊息給佇列。

當您設定 Lambda 函數來處理來自 Amazon SQS FIFO 佇列的訊息時，Lambda 會遵守 FIFO 佇列提供的排序保證。下列要點說明在使用訊息群組 IDs 時，處理來自 Amazon SQS FIFO 佇列的訊息時，Lambda 函數在並行和擴展方面的行為。

- 訊息群組內的並行：一次只有一個 Lambda 執行個體會處理特定訊息群組 ID 的訊息。這可確保群組內的訊息會循序處理。
- 擴展和多個訊息群組：雖然 Lambda 可以同時擴展以處理訊息，但此擴展發生在不同的訊息群組之間。如果您有多個訊息群組，Lambda 可以平行處理多個群組，每個群組都由個別的 Lambda 執行個體處理。

如需詳細資訊，請參閱《AWS Lambda 操作員指南》中的 [Lambda 中的擴展和並行](#)。

使用案例範例

假設您的 FIFO 佇列收到具有相同訊息群組 ID 的訊息，且您的 Lambda 函數具有較高的並行限制（最多 1000 個）。

如果來自群組 ID 'A' 的訊息正在處理中，而來自群組 ID 'A' 的另一個訊息送達，則第二個訊息在第一個訊息完全處理之前，不會觸發新的 Lambda 執行個體。

不過，如果來自群組 IDs 'A' 和 'B' 的訊息送達，則兩個訊息都可以由不同的 Lambda 執行個體同時處理。

Amazon SQS 中 FIFO 佇列的高輸送量

Amazon SQS 中的高輸送量 FIFO 佇列可有效管理高訊息輸送量，同時維持嚴格的訊息順序，確保處理大量訊息的應用程式的可靠性和可擴展性。此解決方案非常適合需要高輸送量和有序訊息交付的案例。

在訊息排序不重要且傳入訊息數量相對較低或偶爾的情況下，不需要 Amazon SQS 高輸送量 FIFO 佇列。例如，如果您有處理不常或非循序訊息的小型應用程式，則與高輸送量 FIFO 佇列相關聯的額外複雜性和成本可能不合理。此外，如果您的應用程式不需要高輸送量 FIFO 佇列提供的增強輸送量功能，選擇標準 Amazon SQS 佇列可能更具成本效益且更容易管理。

若要增強高輸送量 FIFO 佇列中的請求容量，建議增加訊息群組的數量。如需高輸送量訊息配額的詳細資訊，請參閱 Amazon Web Services 一般參考 中的 [Amazon SQS 服務配額](#)。

如需每個佇列配額和資料分佈策略的資訊，請參閱 [Amazon SQS 訊息配額](#) 和 [SQS FIFO 佇列高輸送量的分割區和資料分配](#)。

Amazon SQS FIFO 佇列高輸送量的使用案例

下列使用案例強調高輸送量 FIFO 佇列的各種應用程式，展示其在各產業和情境中的效率：

1. 即時資料處理：處理即時資料串流的應用程式，例如事件處理或遙測資料擷取，可以受益於高輸送量 FIFO 佇列，以處理訊息的持續流入，同時保留其順序以進行準確分析。
2. 電子商務訂單處理：在維護客戶交易訂單至關重要的電子商務平台中，高輸送量 FIFO 佇列可確保訂單會循序處理且不會延遲，即使在尖峰購物季節也一樣。
3. 金融服務：處理高頻率交易或交易資料的金融機構依賴高輸送量 FIFO 佇列來處理市場資料和交易，同時遵守嚴格的訊息排序法規要求。
4. 媒體串流：串流平台和媒體分發服務利用高輸送量 FIFO 佇列來管理媒體檔案和串流內容的交付，確保使用者能夠順暢播放體驗，同時維持內容交付的正確順序。

SQS FIFO 佇列高輸送量的分割區和資料分配

Amazon SQS 將 FIFO 佇列資料存放在分割區中。分割區是佇列的儲存體配置，該佇列會自動複寫到 AWS 區域內的多個可用區域。您未管理分割區。相反地，Amazon SQS 會處理分割區管理。

對於 FIFO 佇列，Amazon SQS 會在下列情況下修改佇列中的分割區數目：

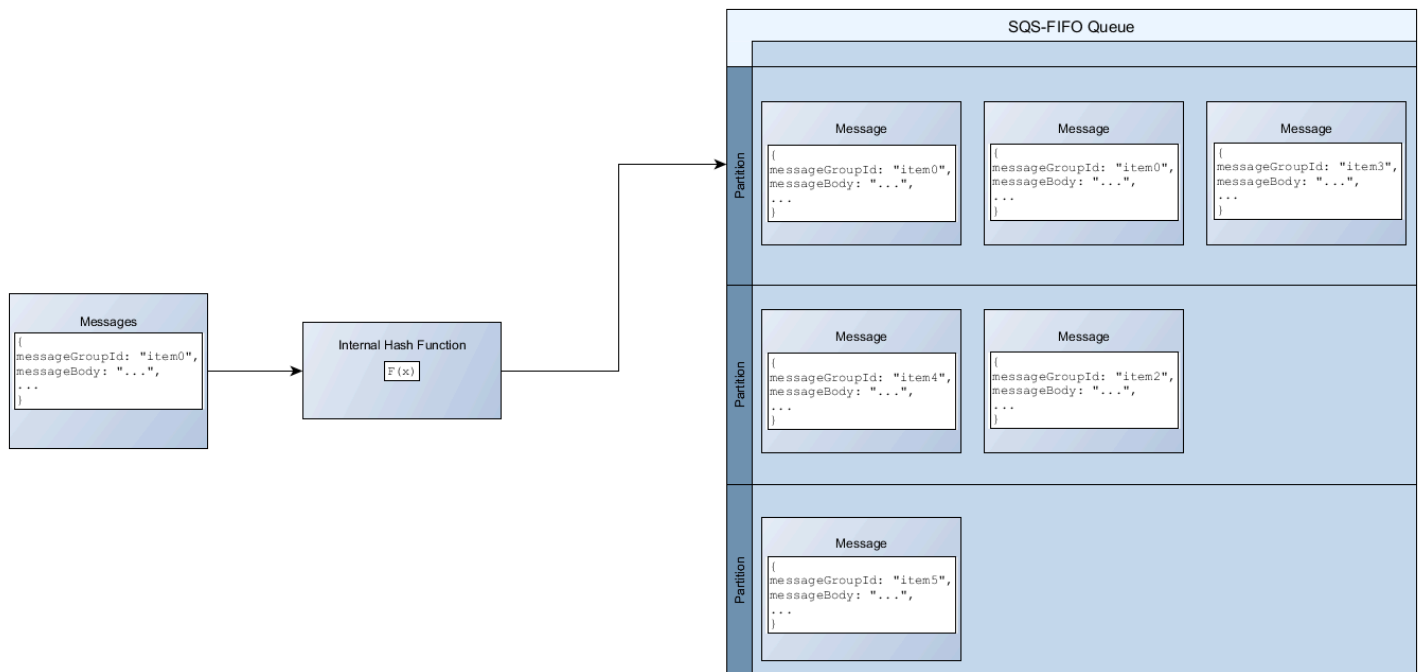
- 如果目前的要求率接近或超過現有分割區可支援的值，則會配置額外的分割區，直到佇列達到地區配額為止。如需配額的相關資訊，請參閱 [Amazon SQS 訊息配額](#)。
- 如果目前的分割區使用率低，則可能會減少分割區的數目。

分割區管理會在背景自動進行，而且對您的應用程式是透明的。您的佇列和訊息隨時都可用。

依訊息群組 ID 分發資料

若要將訊息新增至 FIFO 佇列，Amazon SQS 會使用每則訊息群組 ID 的值做為內部雜湊函數的輸入。雜湊函數的輸出值決定要存放項目的分割區。

下圖顯示跨多個分割區的佇列。佇列的訊息群組 ID 是以項目編號為基礎。Amazon SQS 使用其雜湊函數來判斷新項目的存放位置，在此例中是依據字串 `item0` 的雜湊值。請注意，這些項目的儲存順序與新增到佇列的順序相同。每個項目的位置取決於其訊息群組 ID 的雜湊值。



Note

無論您 AWS 使用的訊息群組 IDs 可以有大量不同的值，Amazon SQS 都已針對 FIFO 佇列分割區的項目統一分佈進行最佳化。

最佳化分割區利用率

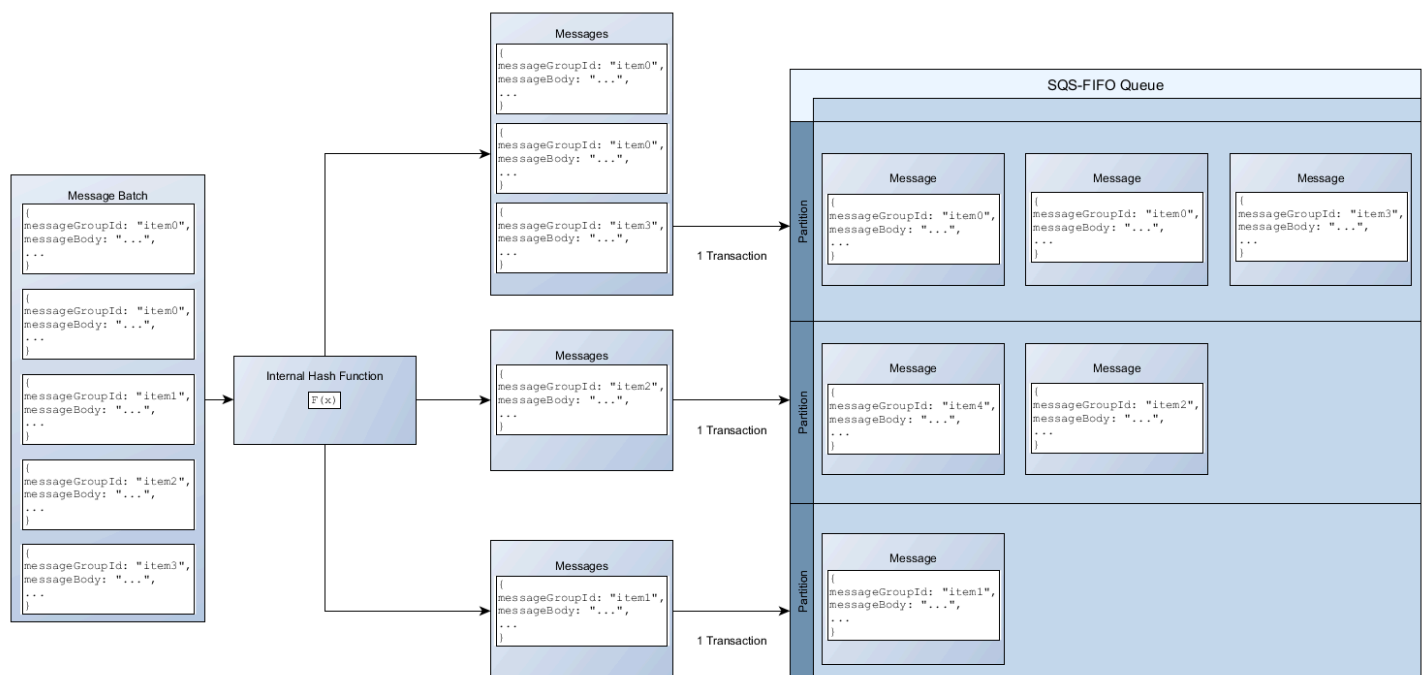
每個分割區最多支援每秒 3,000 則訊息進行批次處理，或支援的區域中傳送、接收和刪除作業，每秒最多可支援 300 則訊息。如需高輸送量訊息配額的詳細資訊，請參閱 Amazon Web Services 一般參考中的 [Amazon SQS 服務配額](#)。

使用批次 API 時，系統會根據 [依訊息群組 ID 分發資料](#) 中所述的程序路由傳送每封訊息。路由至相同分割區的訊息會在單一交易中分組和處理。

若要最佳化 SendMessageBatch API 的分割區使用率，AWS 建議盡可能使用相同的訊息群組 IDs 批次處理訊息。

若要最佳化 DeleteMessageBatch 和 ChangeMessageVisibilityBatch APIs 的分割區使用率，AWS 建議使用 MaxNumberOfMessages 參數設定為 10 的 ReceiveMessage 請求，並對單一 ReceiveMessage 請求傳回的接收處理常式進行批次處理。

在下列範例中，會傳送一批具有不同訊息群組識別碼的訊息。批次會分成三個群組，每個群組都會計入分割區的配額。



 Note

Amazon SQS 僅保證具有相同訊息群組 ID 內部雜湊函數的訊息會在批次請求中分組。視內部雜湊函數的輸出和分割區數量而定，具有不同訊息群組 ID 的訊息可能會被分組。由於雜湊函數或分割區數目可以隨時變更，因此在一個點上分組的訊息可能不會在稍後分組。

在 Amazon SQS 中啟用 FIFO 佇列的高輸送量

您可以為任何新的或現有的 FIFO 佇列啟用高輸送量。當您建立和編輯 FIFO 佇列時，此功能包括三個新選項：

- 啟用高輸送量 FIFO – 為目前 FIFO 佇列中的訊息提供更高的輸送量。
- 重複資料刪除範圍 - 指定要在佇列或訊息群組層級執行重複資料刪除。
- FIFO 輸送量限制 - 指定是在佇列或訊息群組層級設定 FIFO 佇列中訊息的輸送量配額。

若要啟用 FIFO 佇列的高輸送量 (主控台)

1. 開始[建立](#)或[編輯](#) FIFO 佇列。
2. 指定佇列的選項時，請選擇啟用高輸送量 FIFO。

為 FIFO 佇列啟用高輸送量可設定相關選項，如下所示：

- 重複資料刪除範圍設定為訊息群組，這是使用 FIFO 佇列高輸送量的必要設定。
- FIFO 輸送量限制設定為每個訊息群組識別碼，這是使用 FIFO 佇列高輸送量的必要設定。

如果您變更使用高輸送量 FIFO 佇列所需的任何設定，則佇列的正常輸送量將生效，而重複資料刪除會依指定方式執行。

3. 繼續指定佇列的所有選項。完成時，請選擇建立佇列或儲存。

建立或編輯 FIFO 佇列後，您可用更高的 TPS [傳送訊息](#)至該佇列，以及[接收和刪除訊息](#)。如需高輸送量配額，請參閱 [Amazon SQS 訊息配額](#) 中的訊息輸送量。

Amazon SQS 中的 FIFO 佇列和訊息識別符

本節說明 FIFO 佇列的識別碼。這些識別碼可以協助您尋找和操作特定的佇列和訊息。

Amazon SQS 中 FIFO 佇列的識別符

如需下列識別碼的相關資訊，請參閱 [Amazon Simple Queue Service API 參考](#)。

佇列名稱和 URL

建立新的佇列時，必須指定在您 AWS 帳戶和區域中的唯一佇列名稱。Amazon SQS 會為您建立的佇列指派一個識別碼，稱為佇列 URL，內含佇列名稱及其他 Amazon SQS 元件。當您想要在佇列上執行動作時，即需提供佇列 URL。

FIFO 佇列名稱結尾必須是 `.fifo` 尾碼。尾碼會計入 80 個字元的佇列名稱配額中。若要判斷佇列是否為 [FIFO](#)，可以檢查佇列名稱是否以尾碼結尾。

以下是 AWS 帳戶號碼為 123456789012 的使用者 MyQueue 所擁有之名為 FIFO 佇列的佇列 URL 123456789012。

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

您可以列出佇列並剖析跟在帳戶號碼後面的字串，便可以程式設計方式來擷取佇列的 URL。如需詳細資訊，請參閱 [ListQueues](#)。

訊息 ID

每則訊息會收到 Amazon SQS 在 [SendMessage](#) 回應中傳回給您的系統指定訊息 ID。此識別碼可以用來辨識訊息。訊息 ID 的長度上限為 100 個字元。

接收控點

每次從佇列接收訊息時，便會收到該訊息的接收控點。控點是與接收訊息的動作有所關聯，而非訊息本身。若要刪除訊息或變更訊息的可見性，請務必提供接收控點 (而非訊息 ID)。因此必須先接收訊息才能刪除訊息 (無法將訊息放進佇列然後再將之收回)。接收控點的長度上限為 1024 個字元。

Important

若某則訊息收到一次以上，每次收到訊息時都會收到一個不同的接收控點。當您提出刪除訊息的請求時，必須提供最近接收的接收控點 (否則無法刪除訊息)。

以下為接收控點的範例 (分散在三行內)。

```
MbZj6wDWli+JvwWJaBV+3dcjk2YW2vA3+STFF1jTM8tJJg6HRG6PYSasuWXPJB+Cw
```

```
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq20KpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE  
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon SQS FIFO 佇列的其他識別碼

如需下列識別碼的相關資訊，請參閱 [Amazon SQS 中的精確一次處理](#) 和《[Amazon Simple Queue Service API 參考](#)》。

訊息重複資料刪除 ID

用於 Amazon SQS FIFO 佇列的字符，可唯一識別訊息並防止重複。如果在 5 分鐘的重複資料刪除間隔內傳送具有相同重複資料刪除 ID 的多個訊息，這些訊息會視為重複，而且只會傳送一個副本。如果您未指定重複資料刪除 ID，且已啟用內容型重複資料刪除，Amazon SQS 會透過雜湊訊息內文來產生重複資料刪除 ID。此機制透過消除指定時間範圍內的重複訊息，確保準確交付一次。

訊息群組 ID

MessageGroupId 是只在 Amazon SQS FIFO (First-In-First-Out) 佇列中使用的屬性，可將訊息組織成不同的群組。相同訊息群組內的訊息一律以嚴格順序一次處理一個訊息，確保不會同時處理來自相同群組的兩個訊息。標準佇列不會使用 MessageGroupId，也不會提供訂購保證。如果需要嚴格排序，請改用 FIFO 佇列。

序號

Amazon SQS 指派給每則訊息的大型非連續數字。

Amazon SQS 配額

本主題說明 Amazon SQS FIFO 和標準佇列的配額和限制，詳細說明它們如何影響佇列建立、組態和訊息處理。了解訊息保留限制、傳輸中訊息上限和傳輸量閾值等限制，以及透過批次處理、API 呼叫最佳化和長輪詢將效率最大化的策略。本主題也涵蓋命名慣例、標記規則，以及請求提高配額以滿足高需求工作負載的方法，以確保有效的佇列管理和最佳效能。

Amazon SQS FIFO 佇列配額

Amazon SQS 配額

下表列出 FIFO 佇列相關的配額。

配額	描述
延遲佇列	佇列的預設 (最小) 延遲時間為 0 秒。最大值為 15 分鐘。
列出的佇列	每個 ListQueues 請求 1,000 個佇列。
長輪詢等待時間	最長輪詢等待時間上限為 20 秒。
訊息群組	FIFO 佇列中的訊息群組數目沒有配額。
每個佇列的訊息數 (後端記錄)	Amazon SQS 佇列可以存放的訊息數量不受限制。
每個佇列的訊息數 (傳輸中)	FIFO 佇列最多支援 120,000 個傳輸中訊息 (消費者收到但尚未刪除的訊息)。如果達到此限制，Amazon SQS 不會傳回錯誤，但處理可能會受到影響。您可以聯絡 AWS Support 來請求增加超過此限制。
佇列名稱	FIFO 佇列名稱結尾必須是 <code>.fifo</code> 尾碼。尾碼會計入 80 個字元的佇列名稱配額中。若要判斷佇列是否為 FIFO ，可以檢查佇列名稱是否以尾碼結尾。
佇列標籤	我們不建議在佇列中新增超過 50 個標籤。標記支援 UTF-8 Unicode 字元。 標籤 Key 為必要，但標籤 Value 是選用的。

配額	描述
	標籤 Key 和標籤 Value 區分大小寫。
	標籤 Key 和標籤 Value 可以包含 UTF-8 中的 Unicode 英數字元和空格。允許使用以下特殊字元：_ . : / = + - @
	標籤 Key 或 Value 不得包含保留字首 aws：(您無法刪除具有此字首的標籤鍵或值)。
	最大標籤 Key 長度為 128 個 UTF-8 形式的 Unicode 字元。標籤 Key 不能為空或 null。
	最大標籤 Value 長度為 256 個 UTF-8 形式的 Unicode 字元。標籤 Value 可以為空或 null。
	標記動作每個限制為 30 TPS AWS 帳戶。如果您的應用程式需要更高的輸送量，請 提交要求 。

Amazon SQS 標準佇列配額

下表列出標準佇列相關的配額。

配額	描述
延遲佇列	佇列的預設 (最小) 延遲時間為 0 秒。最大值為 15 分鐘。
列出的佇列	每個 ListQueues 請求 1,000 個佇列。
長輪詢等待時間	最長輪詢等待時間上限為 20 秒。
每個佇列的訊息數 (後端記錄)	Amazon SQS 佇列可以存放的訊息數量不受限制。
每個佇列的訊息數 (傳輸中)	對於大部分的標準佇列 (視佇列流量和未處理訊息而定)，最多可有約 120,000 則傳送中訊息 (消費者從佇列接收，但尚未從佇列中刪除)。如果您在使用 短輪詢 時，達到此配額，Amazon SQS 會傳回 OverLimit 錯誤訊息。如果您使用 長輪詢 ，Amazon SQS 不會傳回任何錯誤訊息。為

配額	描述
	了避免達到配額，請在訊息處理過後，將佇列上的訊息刪除。也可以增加用於處理訊息的佇列數量。如需申請提高配額，請 提交支援申請 。
佇列名稱	佇列名稱的長度上限為 80 個字元。接受以下字元：英數字元、連字號 (-) 和底線 (_)。  Note 佇列名稱區分大小寫 (例如 Test-queue 、 test-queue 是不同的佇列)。
佇列標籤	我們不建議在佇列中新增超過 50 個標籤。標記支援 UTF-8 Unicode 字元。 標籤 Key 為必要，但標籤 Value 是選用的。 標籤 Key 和標籤 Value 區分大小寫。 標籤 Key 和標籤 Value 可以包含 UTF-8 中的 Unicode 英數字元和空格。允許使用以下特殊字元：_ . : / = + - @ 標籤 Key 或 Value 不得包含保留字首 aws: (您無法刪除具有此字首的標籤鍵或值)。 最大標籤 Key 長度為 128 個 UTF-8 形式的 Unicode 字元。標籤 Key 不能為空或 null。 最大標籤 Value 長度為 256 個 UTF-8 形式的 Unicode 字元。標籤 Value 可以為空或 null。 標記動作每個限制為 30 TPS AWS 帳戶。如果您的應用程式需要更高的輸送量，請提交要求。

Amazon SQS 訊息配額

下表列出訊息相關的配額。

配額	描述
批次訊息 ID	批次處理訊息 ID 最多可有 80 個字元。接受以下字元：英數字元、連字號 (-) 和底線 (_)。
訊息屬性	一個訊息最多可包含 10 個中繼資料屬性。
訊息批次	單一訊息批次請求最多可包含 10 個訊息。如需詳細資訊，請參閱 Amazon SQS 批次動作 一節的 設定 AmazonSQS BufferedAsyncClient 。
訊息內容	訊息可以包含 XML、JSON 和無格式文字。允許使用以下 Unicode 字元：#x9 #xA #xD #x20 到 #xD7FF #xE000 到 #xFFFD #x10000 到 #x10FFFF 此清單中不包含的任何字元都會被拒絕。如需詳細資訊，請參閱字元的 W3C 規格。
訊息群組 ID	取用待處理的訊息，避免堆積具有相同訊息群組 ID 的大量待處理訊息。 MessageGroupId 對於 FIFO 佇列而言是必需的。您無法將它用於標準佇列。 您必須將非空白 MessageGroupId 與訊息相關聯。若未提供 MessageGroupId，該動作會失敗。 MessageGroupId 的長度上限為 128 個字元。有效值：英數字元和標點符號 (!"#\$%&'()*+,-./:;<=>?@[\]^_`{ }~)。
訊息保留	在預設情況下，訊息會保留 4 天。最低為 60 秒 (1 分鐘)。最高為 1,209,600 秒 (14 天)。
訊息輸送量	標準佇列

配額	描述
	每個動作 (、 SendMessage ReceiveMessage 或) 的標準佇列支援非常高、幾乎無限制的每秒 API 呼叫次數 DeleteMessage 。這種高輸送量使它們非常適合需要快速處理大量訊息的使用案例，例如即時資料串流或大規模應用程式。雖然標準佇列會隨需求自動擴展，但必須監控使用模式以確保最佳效能，尤其是工作負載較高的區域。

配額	描述
	FIFO 佇列 - 每個 API 動作 (SendMessage 、 ReceiveMessage 和)，FIFO 佇列中的每個分割區每秒限制為 300 個交易DeleteMessage 。此限制特別適用於非高輸送量模式。透過切換到高輸送量模式，您可以超過此預設限制。若要啟用高輸送量模式，請參閱 在 Amazon SQS 中啟用 FIFO 佇列的高輸送量。 - 如果您使用批次處理，則非高輸送量 FIFO 佇列支援每個 API 動作 (SendMessage 、 ReceiveMessage 和) 每秒最多 3,000 個訊息DeleteMessage 。每秒 3000 則訊息代表 300 個 API 呼叫，每個呼叫具有一個含 10 則訊息的批次。 FIFO 佇列的高輸送量 Amazon SQS FIFO 限制是根據 API 請求的數量，而非訊息限制。對於高輸送量模式，這些 API 請求限制如下所示： 交易輸送量限制（非批次 API 呼叫） 這些限制會定義每個 API 操作（例如 SendMessage、ReceiveMessage 或 DeleteMessage）可以獨立執行的頻率，以確保每秒允許的交易 (TPS) 內的系統效能有效率。 下列限制是以非批次 API 呼叫為基礎： - 美國東部（維吉尼亞北部）、美國西部（奧勒岡）和歐洲（愛爾蘭）：每秒最多 70,000 筆交易 (TPS)。 - 美國東部（俄亥俄）和歐洲（法蘭克福）：最高 19,000 TPS。 - 亞太區域（孟買）、亞太區域（新加坡）、亞太區域（雪梨）和亞太區域（東京）：高達 9,000 TPS。

配額	描述
	- 歐洲（倫敦）和南美洲（聖保羅）：最高 4,500 TPS。 - 所有其他 AWS 區域：預設輸送量為 2,400 TPS。 使用批次處理將輸送量最大化 在單一 API 呼叫中處理多則訊息，進而大幅提升效率。批次處理可讓您在單一 API 請求中傳送、接收或刪除最多 10 則訊息，而不是個別處理每則訊息。這可減少 API 呼叫的總數，可讓您每秒處理更多訊息，同時保持在區域的交易限制 (TPS) 內，將輸送量和系統效能最大化。如需詳細資訊，請參閱使用 Amazon SQS 使用水平擴展和動作批次來增加輸送量。 下列限制是以批次 API 呼叫為基礎： - 美國東部（維吉尼亞北部）、美國西部（奧勒岡）和歐洲（愛爾蘭）：每秒最多 700,000 則訊息 (70,000 TPS 的非批次限制的 10 倍)。 - 美國東部（俄亥俄）和歐洲（法蘭克福）：每秒最多 190,000 則訊息。 - 亞太區域（孟買）、亞太區域（新加坡）、亞太區域（雪梨）和亞太區域（東京）：每秒最多 90,000 則訊息。 - 歐洲（倫敦）和南美洲（聖保羅）：每秒最多 45,000 則訊息。 - 所有其他 AWS 區域：每秒最多 24,000 則訊息。 最佳化批次以外的輸送量 雖然批次處理可以大幅提高輸送量，但請務必考慮其他策略來最佳化 FIFO 效能： - 將訊息分散到多個訊息群組 IDs – 由於單一群組內的訊息會依序處理，因此將工作負載分散到多個訊息群組，

配額	描述
	可實現更佳的平行處理和更高的整體輸送量。如需詳細資訊，請參閱SQS FIFO 佇列高輸送量的分割區和資料分配。 - 有效使用 API 呼叫 – 將不必要的 API 呼叫降至最低，例如頻繁的可見性變更或重複訊息刪除，以最佳化可用 TPS 的使用並提高效率。 - 使用長輪詢接收 – 在接收請求 <code>WaitTimeSeconds</code> 中設定，以在沒有訊息可用時減少空回應、減少不必要的 API 呼叫，以及更好地利用 TPS 配額，藉此利用長輪詢。 - 請求提高輸送量 – 如果您的應用程式需要高於預設限制的輸送量，請使用 Service Quotas 主控台請求提高輸送量。這對於高需求工作負載或具有較低預設限制的區域而言可能是必要的。若要啟用高輸送量模式，請參閱 在 Amazon SQS 中啟用 FIFO 佇列的高輸送量。
訊息計時器	訊息的預設 (最小值) 延遲時間為 0 秒。最大值為 15 分鐘。
訊息大小	最小訊息大小為 1 位元組 (1 個字元)。最大為 262,144 位元組 (256 KiB)。 若要傳送訊息大於 256 KiB，您可以使用適用於 Java 的 Amazon SQS 延伸用戶端程式庫和適用於 Python 的 Amazon SQS 延伸用戶端程式庫。此程式庫可讓您傳送包含 Amazon S3 中訊息承載參考的 Amazon SQS 訊息。承載大小上限為 2 GB。  Note 此延伸程式庫僅適用於同步用戶端。
訊息可見性逾時	訊息的預設可見性逾時為 30 秒。最小值為 0 秒。時間最長可設為 12 小時。

配額	描述
政策資訊	最大配額是 8,192 位元組、20 個陳述式、50 個委託人，或 10 個條件。如需詳細資訊，請參閱 Amazon SQS 政策配額 。

Amazon SQS 政策配額

下表列出政策相關的配額。

名稱	最大
位元組	8,192
條件	10
委託人	50
聲明	20
每個陳述式的動作	7

Amazon SQS 特性和功能

本主題提供 Amazon SQS 中常用的功能，用於管理訊息佇列、最佳化效能、確保可靠的訊息交付，以及有效率地處理訊息。

在 Amazon SQS 中使用無效字母佇列

Amazon SQS 支援無效字母佇列 (DLQs)，這些來源佇列可以鎖定未成功處理的訊息。DLQs 有助於偵錯您的應用程式，因為您可以隔離未使用的訊息，以判斷為什麼處理不成功。為了獲得最佳效能，最佳實務是將來源佇列和 DLQ 保持在相同 AWS 帳戶和區域內。一旦訊息進入無效字母佇列，您就可以：

- 檢查導致訊息移至無效字母佇列的例外狀況記錄。
- 分析移至無效字母佇列的訊息內容，以診斷應用程式問題。
- 判定您是否有給予消費者儲存的訊息處理時間。
- 使用無效字母佇列 [再驅動](#)，[將訊息移出無效字母佇列](#)。

您必須先建立新的佇列，才能將其設定為無效字母佇列。如需使用 Amazon SQS 主控台設定無效字母佇列的相關資訊，請參閱 [使用 Amazon SQS 主控台設定無效字母佇列](#)。如需無效字母佇列的說明，例如如何為移至無效字母佇列的任何訊息設定警示，請參閱 [使用 Amazon CloudWatch 建立無效字母佇列的警示](#)。

Note

若不想打斷訊息或操作的準確順序，請不要將無效字母佇列搭配 FIFO 佇列使用。例如，影像編輯套件的 Edit Decision List (EDL) 中的指示請勿搭配無效字母佇列使用，因為變更編輯順序會改變後續編輯的環境。

針對無效字母佇列使用政策

使用再驅動政策來指定 `maxReceiveCount`。`maxReceiveCount` 是取用者在移至無效字母佇列之前，可以從來源佇列接收訊息的次數。例如，如果 `maxReceiveCount` 設定為低值，例如 1，則無法接收訊息會導致訊息移至無效字母佇列。為了確保您的系統能夠抵禦錯誤，請將 `maxReceiveCount` 設定得夠高，以允許足夠的重試。

再驅動允許政策指定哪些來源佇列能存取無效字母佇列。您可以選擇是否允許所有來源佇列、允許特定來源佇列，或拒絕所有來源佇列使用無效字母佇列。預設值允許所有來源佇列使用無效字母佇列。如果

您選擇使用 `byQueue` 選項允許特定佇列，您可以使用來源佇列 Amazon Resource Name (ARN) 指定最多 10 個來源佇列。若指定 `denyAll`，佇列就無法當作無效字母佇列。

了解無效字母佇列的訊息保留期間

對於標準佇列，訊息的到期一律會以其原始排入佇列時間戳記為基礎。訊息被移至無效字母佇列時，排入佇列時間戳記不會變更。`ApproximateAgeOfOldestMessage` 指標指出訊息何時移至無效字母佇列，而不是訊息最初傳送的時間。例如，假設訊息在移至無效字母佇列之前，在原始佇列中花費 1 天時間。如果無效字母佇列的保留期限為 4 天，則會在 3 天後從無效字母佇列中刪除訊息，而且 `ApproximateAgeOfOldestMessage` 為 3 天。因此，最佳實務是一律將無效字母佇列的保留期間設定為長於原始佇列的保留期間。

若為 FIFO 佇列，訊息移到無效字母佇列時，會重設訊息。此 `ApproximateAgeOfOldestMessage` 測量結果會指出訊息移到無效字母佇列的時間。在上述相同範例中，訊息會在四天後從無效字母佇列中刪除，而 `ApproximateAgeOfOldestMessage` 是四天。

使用 Amazon SQS 主控台設定無效字母佇列

無效字母佇列 (DLQ) 是接收未成功從另一個佇列處理的訊息的佇列，稱為來源佇列。Amazon SQS 不會自動建立無效字母佇列。您必須先建立佇列，才能將其指派為無效字母佇列。設定 DLQ 時，佇列類型必須符合來源佇列類型：[FIFO 佇列](#) 只能使用 FIFO DLQ，而[標準佇列](#) 只能使用標準 DLQ。您可以在建立或編輯佇列時設定無效字母佇列。如需詳細資訊，請參閱[在 Amazon SQS 中使用無效字母佇列](#)。

若要為既有佇列設定無效字母佇列 (主控台)

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 選取來源佇列（將失敗訊息傳送至無效字母佇列的佇列），然後選擇編輯。
4. 捲動至無效字母佇列區段並切換已啟用。
5. 在無效字母佇列設定下，選擇您要用作無效字母佇列之現有佇列的 Amazon Resource Name (ARN)。
6. 設定接收值上限，此值會定義訊息在傳送至無效字母佇列之前可以接收多少次（有效範圍：1 到 1,000）。
7. 選擇 Save (儲存)。

了解如何在 Amazon SQS 中設定無效字母佇列重新驅動

使用無效字母佇列重新驅動，將未使用的訊息從無效字母佇列移至另一個目的地進行處理。根據預設，無效字母佇列再驅動會將訊息從無效字母佇列移至來源佇列。不過，如果兩種佇列類型相同，您也可以將任何其他佇列設定為再驅動目的地。例如，如果無效字母佇列是 FIFO 佇列，則再驅動目的地佇列也必須是 FIFO 佇列。此外，您可以設定再驅動速度，以設定 Amazon SQS 移動訊息的速率。

Note

將訊息從 FIFO 佇列移至 FIFO DLQ 時，原始訊息的重複資料刪除 ID 將會取代為原始訊息的 ID。這是為了確保 DLQ 重複資料刪除功能不會阻止儲存共用重複資料刪除 ID 的兩則獨立訊息。

死字母佇列會依收到訊息的順序重新驅動訊息，從最舊的訊息開始。但是，目的地佇列會根據接收的順序擷取重新推動的訊息，以及來自其他程序的新訊息。例如，如果生產者在同時從無效字母佇列接收重新驅動的訊息時，將訊息傳送到來源 FIFO 佇列，則重新驅動的訊息會與生產者的新訊息交集。

Note

再驅動任務會重設保留期間。所有重新驅動訊息都會被視為具有新訊息，messageID 並 enqueueTime 指派給重新驅動訊息。

使用 Amazon SQS API 為現有標準佇列設定無效字母佇列重新驅動

您可以使用 `StartMessageMoveTask`、`ListMessageMoveTasks` 和 `CancelMessageMoveTask` API 動作來設定無效字母佇列重新驅動：

API 動作	描述
StartMessageMoveTask	啟動非同步工作，將訊息從指定的來源佇列移至指定的目的地佇列。
ListMessageMoveTasks	取得特定來源佇列下的最新訊息移動任務 (最多 10 個)。
CancelMessageMoveTask	取消指定的訊息移動任務。只有當目前狀態為 <code>RUNNING</code> 時，才能取消訊息移動。

使用 Amazon SQS 主控台為現有標準佇列設定無效字母佇列重新驅動

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 選擇您已設定為[無效字母](#)佇列的佇列名稱。
4. 選擇啟動 DLQ 再驅動。
5. 在再驅動組態下，針對訊息目的地，執行下列任一項作業：
 - 若要將訊息再驅動到其來源佇列，請選擇再驅動至來源佇列。
 - 若要將訊息再驅動到其他佇列，請選擇再驅動至自訂目的地。然後，輸入現有目的地佇列的 Amazon Resource Name (ARN)。
6. 在速度控制設定下，選擇下列其中一項：
 - 系統最佳化 - 以每秒最大訊息數再驅動無效字母佇列訊息。
 - 自訂最大速度 - 以每秒自訂的最大訊息速率再驅動無效字母佇列訊息。允許的最大速率為每秒 500 則訊息。
 - 建議從自訂最大速度的較小值開始，並驗證來源佇列不會被訊息淹沒。從那裡開始，逐漸提高自訂最大速度值，並繼續監控來源佇列的狀態。
7. 完成設定無效字母佇列再驅動時，請選擇再驅動訊息。

Important

Amazon SQS 不支援在從無效字母佇列再驅動訊息的同時篩選和修改訊息。
無效字母佇列再驅動任務最多可執行 36 小時。Amazon SQS 支援每個帳戶最多有 100 個作用中再驅動任務。

8. 如果您要取消訊息再驅動任務，請在佇列的詳細資料頁面上，選擇取消 DLQ 再驅動。取消進行中的訊息再驅動時，任何已成功移至其移動目的地佇列的訊息都會保留在目的地佇列中。

設定無效字母佇列再驅動的佇列許可

您可以將許可新增至政策，讓使用者存取特定無效字母佇列動作。無效字母佇列再驅動所需的最低許可如下：

最低許可	必要的 API 方法
若要啟動訊息再驅動	- 新增無效字母佇列的 <code>sqs:StartMessageMoveTask</code>、<code>sqs:ReceiveMessage</code>、<code>sqs:DeleteMessage</code> 和 <code>sqs:GetQueueAttributes</code>。如果無效字母佇列或原始來源佇列已加密 (也稱為 SSE 佇列)，則也 <code>kms:Decrypt</code> 需要用來加密訊息的任何 KMS 金鑰。 - 新增目的地佇列的 <code>sqs:SendMessage</code>。如果目的地佇列已加密，則 <code>kms:GenerateDataKey</code> 和 <code>kms:Decrypt</code> 也是必要的。
若要取消進行中的訊息再驅動	新增無效字母佇列的 <code>sqs:CancelMessageMoveTask</code>、<code>sqs:ReceiveMessage</code>、<code>sqs:DeleteMessage</code> 和 <code>sqs:GetQueueAttributes</code>。如果無效字母佇列已加密 (也稱為 SSE 佇列)，<code>kms:Decrypt</code> 也需要。
若要顯示訊息移動狀態	新增無效字母佇列的 <code>sqs:ListMessageMoveTasks</code> 和 <code>sqs:GetQueueAttributes</code>。

若要設定加密佇列配對 (具有無效字母佇列的來源佇列) 的許可

使用下列步驟來設定無效字母佇列 (DLQ) 重新驅動的最低許可：

- 開啟位於 <https://console.aws.amazon.com/iam/> 的 IAM 主控台。
- 在導覽窗格中，選取 Policies (政策)。
- 建立新的 [政策](#) 並新增下列許可。將政策連接至將執行重新驅動操作的 IAM [使用者](#) 或 [角色](#)。
 - DLQ (來源佇列) 的許可：
 - `sqs:StartMessageMoveTask`
 - `sqs:CancelMessageMoveTask`
 - `sqs:ListMessageMoveTasks`
 - `sqs:ReceiveMessage`
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`

- `sqs:ListDeadLetterSourceQueues`
- 指定 DLQ 的資源 ARN（來源佇列）（例如 `"arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>"`）。
- 目的地佇列的許可：
 - `sqs:SendMessage`
 - 指定目的地佇列 Resource ARN 的（例如，「`arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>`」）。
- KMS 金鑰的許可：
 - `kms:Decrypt`（需要解密 DLQ 中的訊息。）
 - `kms:GenerateDataKey`（需要加密目的地佇列中的訊息。）
 - Resource ARNs：
 - 用來加密 DLQ（來源佇列）中訊息的 KMS 金鑰 ARN（例如 `"arn:aws:kms:<region>:<accountId>:key/<SourceQueueKeyId>"`）。
 - 用來加密目的地佇列中訊息的 KMS 金鑰 ARN（例如 `"arn:aws:kms:<region>:<accountId>:key/<DestinationQueueKeyId>"`）。

您的存取政策應類似以下內容：

```
{
  "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>"
    ]
  }
]
}

```

若要使用非加密佇列配對 (具有無效字母佇列的來源佇列) 設定許可

請依照下列步驟，設定處理標準、未加密的無效字母佇列 (DLQ) 所需的最低許可。必要的最低許可是從無效字母佇列接收、刪除和取得屬性，以及將屬性傳送至來源佇列。

1. 開啟位於 <https://console.aws.amazon.com/iam/> 的 IAM 主控台。
2. 在導覽窗格中，選取 Policies (政策)。
3. 建立新的 [政策](#) 並新增下列許可。將政策連接至將執行重新驅動操作的 IAM [使用者](#) 或 [角色](#)。

- DLQ (來源佇列) 的許可：
 - sqs:StartMessageMoveTask
 - sqs:CancelMessageMoveTask
 - sqs:ListMessageMoveTasks

- sqs:ReceiveMessage
- sqs:DeleteMessage
- sqs:ListDeadLetterSourceQueues
- 指定 DLQ 的資源 ARN (來源佇列) (例如 "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>")。
- 目的地佇列的許可：
 - sqs:SendMessage
 - 指定目的地佇列Resource ARN的 (例如 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>")。

您的存取政策應類似以下內容：

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

Amazon SQS 無效字母佇列重新驅動的 CloudTrail 更新和許可要求

2023 年 6 月 8 日，Amazon SQS 為 SDK 和 (CLI) 推出無效字母佇列 AWS AWS Command Line Interface (DLQ) 重新驅動。此功能是 AWS 主控台已支援之 DLQ 重新驅動的新增功能。如果您先前已使用 AWS 主控台重新驅動無效字母佇列訊息，則可能會受到下列變更的影響：

CloudTrail 事件重新命名

2023 年 10 月 15 日，Amazon SQS 主控台上無效字母佇列再驅動的 CloudTrail 事件名稱將會變更。如果您已為這些 CloudTrail 事件設定警示，則必須立即更新警示。以下是 DLQ 再驅動的新 CloudTrail 事件名稱：

上一個事件名稱	新事件名稱
CreateMoveTask	StartMessageMoveTask
CancelMoveTask	CancelMessageMoveTask

更新許可

Amazon SQS 也隨附於 SDK 和 CLI 版本中，也更新了 DLQ 再驅動的佇列許可，以符合安全最佳實務。使用下列佇列許可類型來再驅動 DLQ 的訊息。

1. 動作型許可 (DLQ API 動作的更新)
2. 受管 Amazon SQS 政策許可
3. 使用 sqs:* 萬用字元的許可政策

⚠ Important

若要使用 SDK 或 CLI 的 DLQ 再驅動，您必須擁有符合上述其中一個選項的 DLQ 再驅動許可政策。

如果 DLQ 再驅動的佇列許可不符合上述其中一個選項，您必須在 2023 年 8 月 31 日前更新您的許可。從現在到 2023 年 8 月 31 日，您的帳戶將只能在您之前使用過 DLQ 再驅動的區域中，使用以 AWS 主控台設定的許可再驅動訊息。例如，假設您在 us-east-1 和 eu-west-1 中都有「帳戶 A」。在 2023 年 6 月 8 日之前，「帳戶 A」用於在 us-east-1 AWS 主控台上重新驅動訊息，但不適用於 eu-west-1。在 2023 年 6 月 8 日至 2023 年 8 月 31 日期間，如果「帳戶 A 的」政策許可不符合上述其中一個選項，則只能用於在 us-east-1 的 AWS 主控台上重新驅動訊息，而不是在 eu-west-1 中。

⚠ Important

如果您的 DLQ 再驅動許可可在 2023 年 8 月 31 日之後不符合這些選項之一，您的帳戶將無法再使用 AWS 主控台再驅動 DLQ 訊息。

不過，如果您在 2023 年 8 月期間在 AWS 主控台上使用 DLQ 重新驅動功能，則您有一個延長期限，直到 2023 年 10 月 15 日為止，才能根據其中一個選項採用新的許可。

如需詳細資訊，請參閱[the section called “識別受影響政策”](#)。

以下是每個 DLQ 再驅動選項的佇列許可範例。使用[伺服器端加密 \(SSE\) 佇列](#)時，需要對應的 AWS KMS 金鑰許可。

動作型

```
{
  "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>"
  }
]
}

```

受管政策

下列受管政策包含必要的更新許可：

- AmazonSQSFullAccess - 包含下列無效字母佇列再驅動工作：啟動、取消和清單。
- AmazonSQSReadOnlyAccess – 提供唯讀存取權限，並包含清單無效字母佇列再驅動工作。

Step 1

Add permissions

Step 2

Review

Add permissions

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

Permissions options

☐ Add user to group

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

☐ Copy permissions

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

☒ Attach policies directly

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

Permissions policies (1/1051)


[Create policy](#)



2 matches



1



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

Cancel

Next

使用 sqs* 萬用字元的許可政策

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

識別受影響政策

如果您使用客戶受管政策 (CMPs)，您可以使用 AWS CloudTrail 和 IAM 來識別受佇列許可更新影響的政策。

Note

如果您正在使用 `AmazonSQSFullAccess` 和 `AmazonSQSReadOnlyAccess`，則不需要進一步的動作。

1. 登入 AWS CloudTrail 主控台。
2. 在事件歷史記錄頁面的查詢屬性下，使用下拉式功能表選取事件名稱。然後搜尋 `CreateMoveTask`。
3. 選擇事件以開啟詳細資訊頁面。在事件記錄區段中，從 `userIdentity` ARN 擷取 `UserName` 或 `RoleName`。
4. 登入 IAM 主控台。
 - 對於使用者，請選擇 [使用者]。選取上一個步驟中標識 `UserName` 的使用者。
 - 對於角色，請選擇 [角色]。搜尋上一個步驟中標識 `RoleName` 的使用者。
5. 在詳細資訊頁面的許可區段中，檢閱任何在 `Action` 中包含 `sqs:` 字首的政策，或檢閱 `Resource` 中已定義 Amazon SQS 佇列的政策。

使用 Amazon CloudWatch 建立無效字母佇列的警示

設定 CloudWatch 警示，以使用 [ApproximateNumberOfMessagesVisible](#) 指標監控無效字母佇列中的訊息。如需詳細說明，請參閱 [建立 Amazon SQS 指標的 CloudWatch 警示](#)。當警示觸發時，指出訊息已移至無效字母佇列，您可以[輪詢](#)佇列來檢閱和擷取它們。

Amazon SQS 的訊息中繼資料

使用訊息屬性將自訂中繼資料新增至您應用程式的 Amazon SQS 訊息。使用訊息系統屬性來存放中繼資料以與其他整合 AWS 服務，例如 AWS X-Ray。

Amazon SQS 訊息屬性

Amazon SQS 可讓您使用訊息屬性將結構化中繼資料（例如時間戳記、地理空間資料、簽章和識別符）與訊息一起包含。每則訊息最多可以擁有 10 個屬性。訊息屬性為選用且與訊息內文分開（但與訊息內文一起傳送）。您的消費者可以使用訊息屬性以特定方式處理訊息，而不必先處理訊息內文。如需使用 Amazon SQS 主控台傳送具有屬性的訊息的資訊，請參閱 [使用 Amazon SQS 傳送具有屬性的訊息](#)。

Note

不要將訊息屬性與訊息系統屬性混淆：雖然您可以使用訊息屬性將自訂中繼資料連接至應用程式的 Amazon SQS 訊息，但您可以使用[訊息系統屬性](#)來存放其他服務的中繼資料 AWS，例如 AWS X-Ray。

主題

- [訊息屬性元件](#)
- [訊息屬性資料類型](#)
- [計算訊息屬性的 MD5 訊息摘要](#)

訊息屬性元件

Important

訊息屬性的所有元件都包含在 256 KB 的訊息大小限制內。
Name、Type、Value 和訊息內文不能為空或 null。

每項訊息屬性均是由以下元件組成：

- 名稱 – 訊息屬性名稱可包含以下字元：A-Z、a-z、0-9、底線 (_)、連字號 (-) 和句點 (.)。將適用以下限制：
 - 長度上限為 256 個字元
 - 開頭不能是 AWS. 或 Amazon. (或任何大小寫變化)
 - 區分大小寫
 - 在訊息的所有屬性名稱中必須是唯一的
 - 開頭或結尾不能是句號
 - 序列中不能有句號
- 類型 – 訊息屬性資料類型。支援的類型包括 String、Number 和 Binary。您也可以新增任何資料類型的自訂資訊。資料類型的限制與訊息內文相同（如需詳細資訊，請參閱《Amazon Simple Queue Service API 參考[SendMessage](#)》中的）。此外，適用下列限制：
 - 長度上限為 256 個字元
 - 區分大小寫
- 值 – 訊息屬性值。若為 String 資料類型，屬性值的限制與訊息內文相同。

訊息屬性資料類型

訊息屬性資料類型指定 Amazon SQS 如何處理對應的訊息屬性值。例如，如果類型是 Number，Amazon SQS 會驗證數值。

Amazon SQS 支援邏輯資料類型 String、Number 和 Binary，搭配格式為 *.custom-data-type* 的選用自訂類型標籤

- 字串 – String 屬性可以使用任何有效的 XML 字元儲存 Unicode 文字。
- 數字 - Number 屬性可以存放正或負的數值。數字的精準度最多可達 38 位數，可介於 10^{-128} 至 10^{+126} 之間。

Note

Amazon SQS 會移除前置和結尾的零。

- 二進位 – 二進位屬性可儲存任何二進位資料，例如壓縮資料、加密資料或影像。
- 自訂 – 若要建立自訂資料類型，請將自訂類型標籤附加到任何資料類型。例如：

- `Number.byte`、`Number.short`、`Number.int` 和 `Number.float` 可協助區分數字類型。
- `Binary.gif` 和 `Binary.png` 可協助區分檔案類型。

 Note

Amazon SQS 不會解譯、驗證或使用附加的資料。
自訂類型標籤的限制與訊息內文相同。

計算訊息屬性的 MD5 訊息摘要

如果您使用 適用於 Java 的 AWS SDK，則可以略過本節。適用於 Java 的開發套件的 `MessageMD5ChecksumHandler` 類別支援 Amazon SQS 訊息屬性的 MD5 訊息摘要。

如果您使用查詢 API 或其中一個不支援 Amazon SQS 訊息屬性的 MD5 訊息摘要的 AWS SDKs，則必須使用下列準則來執行 MD5 訊息摘要計算。

 Note

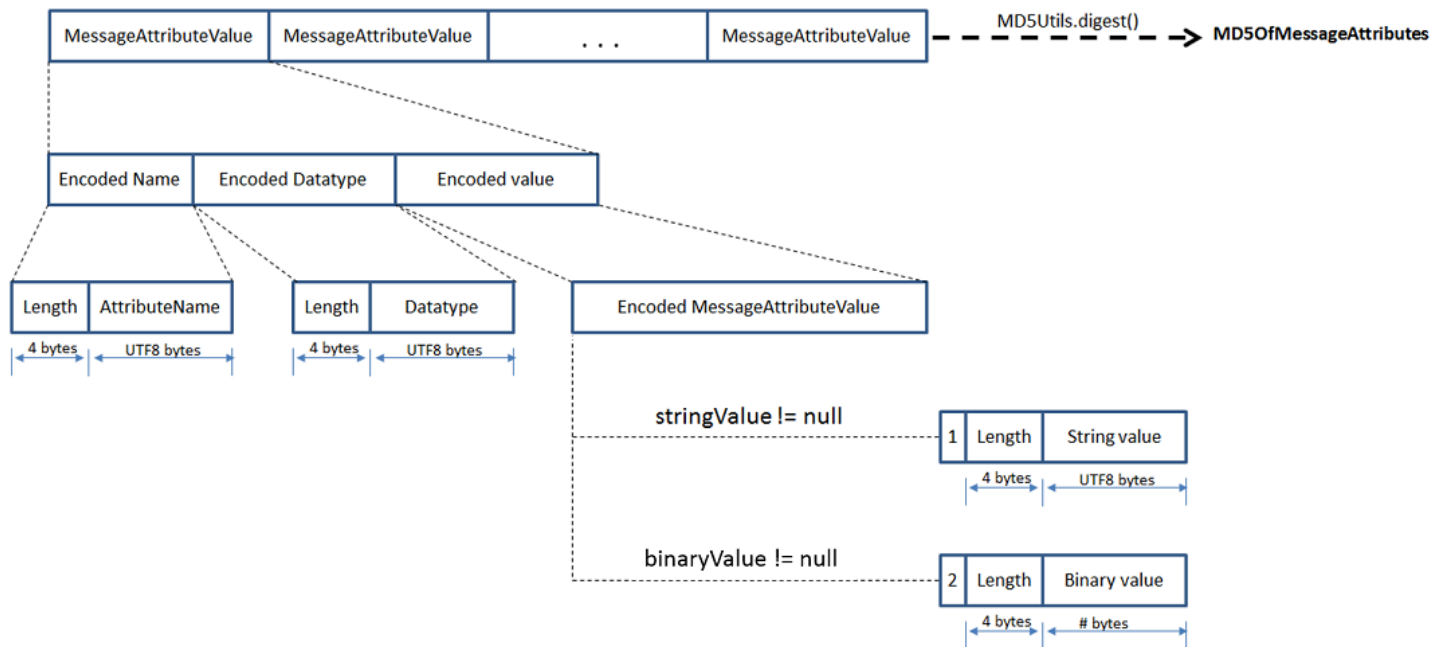
一律在 MD5 訊息摘要計算中包含自訂資料類型尾碼。

概觀

以下概述 MD5 訊息摘要計算演算法：

1. 以遞增順序根據名稱排列所有訊息屬性。
2. 將各屬性 (Name、Type 和 Value) 的個別部分編碼至緩衝區。
3. 運算整個緩衝區的訊息摘要。

下圖顯示的是單一訊息屬性的 MD5 訊息摘要編碼：



若要對單個 Amazon SQS 訊息屬性進行編碼

1. 編碼名稱：長度 (4 個位元組) 和名稱的 UTF-8 位元組。
2. 編碼資料類型：長度 (4 個位元組) 和資料類型的 UTF-8 位元組。
3. 編碼值的傳輸類型 (String 或 Binary) (1 個位元組)。

Note

String 和 Number 的邏輯資料類型使用 String 傳輸類型。
Binary 邏輯資料類型使用 Binary 傳輸類型。

- a. 若為 String 傳輸類型，為編碼 1。
- b. 若為 Binary 傳輸類型，為編碼 2。
4. 為屬性值編碼。
 - a. 若為 String 傳輸類型，請編碼屬性值：長度 (4 個位元組) 和值的 UTF-8 位元組。
 - b. 若為 Binary 傳輸類型，請編碼屬性值：長度 (4 個位元組) 和值的原始位元組。

Amazon SQS 訊息系統屬性

雖然您可以使用[訊息屬性](#)將應用程式的自訂中繼資料連接到 Amazon SQS 訊息，但您仍可以使用訊息系統屬性來存放 AWS X-Ray 等其他 AWS 服務的中繼資料。如需詳細資訊，請參閱 [SendMessage](#) 和 [SendMessageBatch](#) API 動作的 `MessageSystemAttribute` 請求參數、[ReceiveMessage](#) API 動作的 `AWSTraceHeader` 屬性，以及 Amazon Simple Queue Service API 參考中的 [MessageSystemAttributeValue](#) 資料類型。

訊息系統屬性的結構與訊息屬性完全相同，但下列屬性除外：

- 目前，唯一支援的訊息系統屬性是 `AWSTraceHeader`。其類型必須為 `String`，且其值必須是正確格式的 AWS X-Ray 追蹤標頭字串。
- 訊息系統屬性的大小不會計入訊息的總大小。

處理 Amazon SQS 訊息所需的資源

Amazon SQS 提供佇列中延遲、可見和不可見訊息的預估數量，協助您評估處理所需的資源。如需可見性的相關資訊，請參閱 [Amazon SQS 可見性逾時](#)。

Note

對於某些指標，結果是近似值，因為 Amazon SQS 的分散式架構。在大多數情況下，計數應接近佇列中的實際訊息數目。

下表列出可搭配 [GetQueueAttributes](#) 動作使用的屬性名稱：

任務	屬性名稱
取得可從佇列擷取的大約訊息數。	<code>ApproximateNumberOfMessagesVisible</code>
取得佇列中延遲且無法立即讀取的大約訊息數。這種情況會發生在將佇列設定為延遲佇列，或以延遲參數傳送訊息時。	<code>ApproximateNumberOfMessagesDelayed</code>

任務	屬性名稱
取得傳送中的大約訊息。如果訊息已傳送至用戶端，但尚未刪除或尚未達到可見性期間的結束時間，則訊息將被視為傳送中。	ApproximateNumberOfMessages NotVisible

Amazon SQS 清單佇列分頁

[listQueues](#) 和 [listDeadLetterQueues](#) API 方法支援選用分頁控制。根據預設，這些 API 方法會在回應訊息中傳回最多 1000 個佇列。您可以設定 `MaxResults` 參數，在每個回應中傳回較少的結果。

在 `listQueues` 或 `listDeadLetterQueues` 請求中設定參數 `MaxResults`，以指定要在回應中傳回的結果數量上限。如果您未設定 `MaxResults`，則回應最多包含 1,000 個結果，且回應中的 `NextToken` 值為 `null`。

如果設定 `MaxResults`，如果有其他結果可顯示，回應會包含 `NextToken` 的值。在下一個對 `listQueues` 的請求中使用 `NextToken` 做為參數，以接收結果的下一頁。如果沒有其他結果可顯示，則回應中的 `NextToken` 值為 `null`。

Amazon SQS 成本分配標籤

若要整理和辨識 Amazon SQS 成本分配標籤，可以新增中繼資料標籤，用於識別佇列的用途、擁有者或環境等。擁有許多佇列時特別實用。若要使用 Amazon SQS 主控台設定標籤，請參閱 [the section called “為佇列設定標籤”](#)

您可以使用成本分配標籤來整理 AWS 帳單，以反映您自己的成本結構。若要執行此操作，請註冊以取得 AWS 帳戶帳單，以包含標籤索引鍵和值。如需詳細資訊，請參閱 AWS Billing 使用者指南中的 [設定每月成本分配報告](#)。

每個標籤都是由您定義的鍵值配對所構成。舉例而言，如果您以下列方式標記您的佇列，即可輕鬆辨識您的生產和測試佇列：

佇列	金鑰	值
MyQueueA	QueueType	Production
MyQueueB	QueueType	Testing

Note

使用佇列標籤時，請牢記下列準則：

- 我們不建議在佇列中新增超過 50 個標籤。標記支援 UTF-8 Unicode 字元。
- 標籤沒有任何語義意義。Amazon SQS 會將標籤解譯為字元字串。
- 標籤會區分大小寫。
- 與現有標籤相同的索引鍵的新標籤會覆寫現有標籤。
- 標記動作每個限制為 30 TPS AWS 帳戶。如果您的應用程式需要更高的輸送量，請[提交要求](#)。

如需標籤限制的完整清單，請參閱 [Amazon SQS 標準佇列配額](#)。

Amazon SQS 短輪詢和長輪詢

Amazon SQS 提供短期和長期輪詢選項，用於從佇列接收訊息。在這兩個輪詢選項之間進行選擇時，請考慮應用程式對回應能力和成本效益的需求：

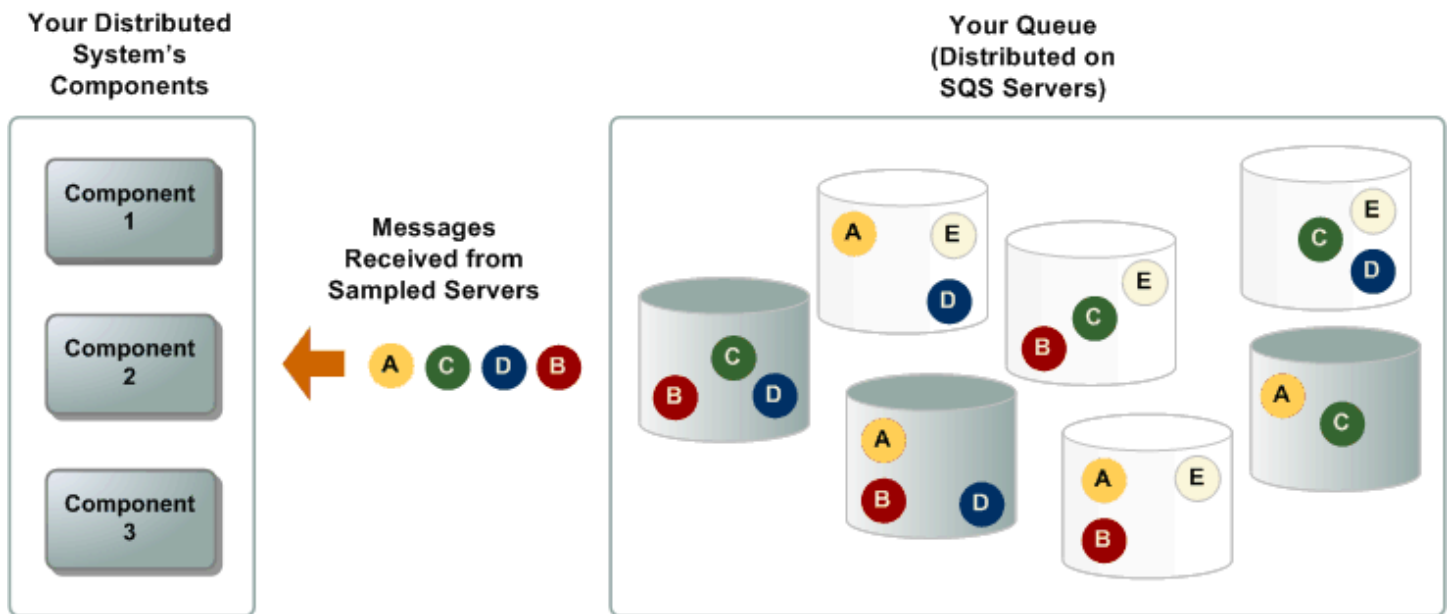
- 短輪詢（預設）– [ReceiveMessage](#) 請求會查詢伺服器子集（根據加權隨機分佈），以尋找可用的訊息，並傳送立即回應，即使找不到訊息也一樣。
- 長輪詢 – [ReceiveMessage](#) 查詢所有伺服器的訊息，在至少一個訊息可用時傳送回應，最高可達指定的上限。只有在輪詢等待時間過期時，才會傳送空的回應。此選項可以減少空的回應數量，並可能降低成本。

以下各節說明短輪詢和長輪詢的詳細資訊。

以短輪詢消耗訊息

當您使用短輪詢從佇列 (FIFO 或標準) 取用訊息時，Amazon SQS 會取樣其伺服器子集（根據加權隨機分佈），並僅傳回來自這些伺服器的訊息。因此，特定的 [ReceiveMessage](#) 請求可能不會傳回您的所有訊息。但是，若佇列中的訊息少於 1,000 則，後續的請求就會將訊息傳回。如果您持續自佇列消耗訊息，Amazon SQS 便會對其所有伺服器進行採樣，以致您將收到全部的訊息。

下圖為系統元件之一在提出接收請求後，從標準佇列傳回訊息的短輪詢行為。Amazon SQS 會對其若干伺服器 (顯示為灰色) 進行採樣，再從這幾個伺服器傳回訊息 A、C、D 和 B。此次的請求並不會傳回訊息 E，但後續的請求會將其傳回。



以長輪詢消耗訊息

當 [ReceiveMessage](#) API 動作的等待時間大於 0 時，長輪詢就會生效。最長輪詢等待時間上限為 20 秒。長輪詢可減少空回應 (沒有 [ReceiveMessage](#) 請求可用的訊息) 和 False 空回應 (有可用訊息但不包含在回應中) 的數目，藉以降低使用 Amazon SQS 的成本。如需使用 Amazon SQS 主控台為新的或現有的佇列啟用長輪詢的資訊，請參閱 [使用 Amazon SQS 主控台設定佇列參數](#)。如需最佳實務做法，請參閱「[在 Amazon SQS 中設定長輪詢](#)」。

長輪詢可提供下列好處：

- 允許 Amazon SQS 等待佇列中有可用的訊息後再傳送回應，藉此減少空回應的數量。除非連線逾時，否則對 [ReceiveMessage](#) 請求的回應至少會包含一則可用的訊息，最多為 [ReceiveMessage](#) 動作中指定的訊息數量上限。在極少數情況下，即使佇列仍包含訊息，您可能會收到空的回應，特別是當您為 [ReceiveMessageWaitTimeSeconds](#) 參數指定低值時。
- 透過查詢所有 - 而非一部分 Amazon SQS 伺服器來減少誤報。
- 訊息一旦可供使用時，立即傳回訊息。

如需如何確認佇列為空的資訊，請參閱 [確認 Amazon SQS 佇列是空的](#)。

長短輪詢之間的差異

當 [WaitTimeSeconds](#) 請求的 [ReceiveMessage](#) 參數以兩種方式之一設為 0 時，便會發生短輪詢：

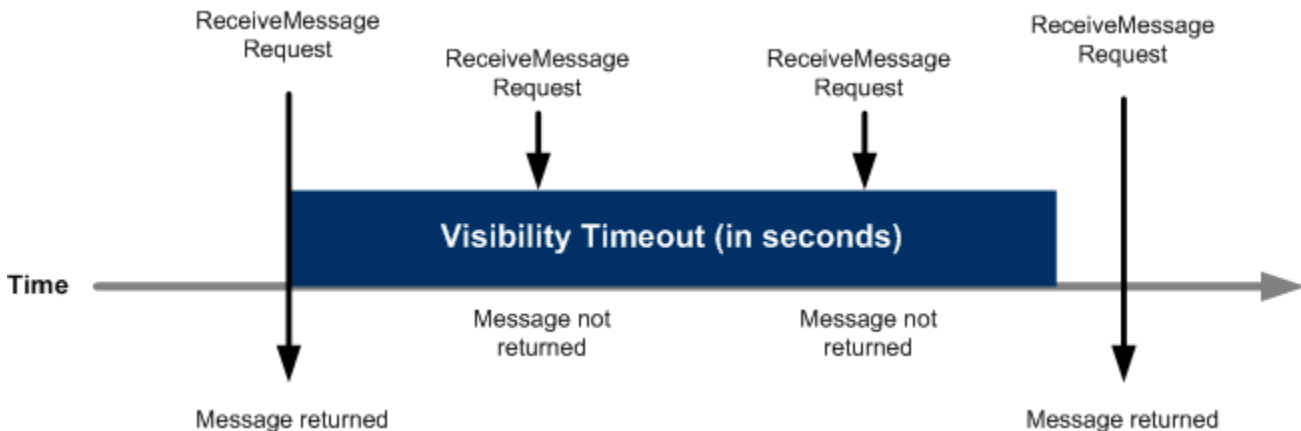
- [ReceiveMessage](#) 呼叫將 [WaitTimeSeconds](#) 設為 0。

- `ReceiveMessage` 呼叫未設定 `WaitTimeSeconds`，但佇列屬性 [ReceiveMessageWaitTimeSeconds](#) 設為 0。

Amazon SQS 可見性逾時

當您收到來自 Amazon SQS 佇列的訊息時，它會保留在佇列中，但其他消費者暫時看不到它。此可見性是由可見性逾時所控制，可確保其他取用者在您處理訊息時無法處理相同的訊息。Amazon SQS 提供兩種在處理後刪除訊息的選項：

- 手動刪除 – 您可以使用 [DeleteMessage](#) 動作明確刪除訊息。
- 自動刪除 – 某些 AWS SDKs 支援，訊息會在成功處理時自動刪除，簡化工作流程。



可見性逾時使用案例

管理長時間執行的任務 – 使用可見性逾時來處理需要延長處理時間的任務。為需要延長處理時間的訊息設定適當的可見性逾時。這可確保其他消費者在處理訊息時不會挑選相同的訊息，防止重複的工作並維持系統效率。

實作重試機制 – 針對無法在初始逾時內完成的任務，以程式設計方式延伸可見性逾時。如果任務無法在初始可見性逾時內完成，您可以透過程式設計方式延長逾時。這可讓您的系統重試處理訊息，而不會讓其他消費者看到訊息，進而改善容錯能力和可靠性。結合無效字母佇列 (DLQs) 來管理持續故障。

協調分散式系統 – 使用 SQS 可見性逾時來協調分散式系統的任務。設定可見性逾時，以符合不同元件的預期處理時間。這有助於維持一致性，並防止複雜分散式架構中的競爭條件。

最佳化資源使用率 – 調整 SQS 可見性逾時，以最佳化應用程式中的資源使用率。透過設定適當的逾時，您可以確保有效處理訊息，而不會不必要的綁定資源。這可提高整體系統效能和成本效益。

設定和調整可見性逾時

可見性逾時會在訊息傳送給您時立即開始。在此期間，您需要處理和刪除訊息。如果您在逾時過期之前未將其刪除，訊息會再次顯示在佇列中，並且可由其他取用者擷取。佇列的預設可見性逾時為 30 秒，但您可以根據應用程式需要處理和刪除訊息的時間進行調整。您也可以為個別訊息設定特定的可見性逾時，而無需變更佇列的整體設定。視需要使用 [ChangeMessageVisibility](#) 動作，以程式設計方式延長或縮短逾時。

傳輸中訊息和配額

在 Amazon SQS 中，傳輸中訊息是消費者已收到但尚未刪除的訊息。對於標準佇列，根據佇列流量和訊息待處理項目，大約有 120,000 個傳輸中訊息的限制。如果您達到此限制，Amazon SQS 會傳回 `OverLimit` 錯誤，指出在刪除一些進行中訊息之前，無法接收其他訊息。對於 FIFO 佇列，限制取決於作用中的訊息群組。

- 使用短輪詢時 – 如果在使用短輪詢時達到此限制，Amazon SQS 將傳回 `OverLimit` 錯誤，指出在刪除一些進行中訊息之前，無法接收其他訊息。
- 使用長輪詢 – 如果您使用長輪詢，Amazon SQS 不會在達到傳輸中訊息限制時傳回錯誤。相反地，它不會傳回任何新訊息，直到傳輸中訊息的數量低於限制。

若要有效管理傳輸中訊息：

1. 提示刪除 – 處理後刪除訊息（手動或自動），以減少傳輸中計數。
2. 使用 CloudWatch 監控 - 設定高傳輸中計數的警示，以防止達到限制。
3. 分佈負載 – 如果您正在處理大量訊息，請使用其他佇列或取用者來平衡負載並避免瓶頸。
4. 請求提高配額 – 如果需要更高的限制，請向 [AWS Support](#) 提交請求。

了解標準 和 FIFO 佇列中的可見性逾時

在標準和 FIFO First-In-First-Out 佇列中，可見性逾時有助於防止多個消費者同時處理相同的訊息。不過，由於 Amazon SQS at-least-once 交付模型，因此無法保證在可見性逾時期間不會多次交付訊息。

- 標準佇列 – 標準佇列中的可見性逾時可防止多個取用者同時處理相同的訊息。不過，由於 at-least-once 交付模型，Amazon SQS 不保證在可見性逾時期間內不會交付超過一次訊息。
- FIFO 佇列 – 對於 FIFO 佇列，具有相同訊息群組 ID 的訊息會以嚴格順序處理。當訊息群組 ID 為的訊息正在處理中時，該群組中的後續訊息在處理中訊息被刪除或可見性逾時過期之前，將無法使用。

不過，這不會無限期鎖定群組 – 每個訊息都會依序處理，而且只有當每個訊息刪除或再次顯示時，該群組中的下一個訊息才會供消費者使用。此方法可確保群組內的有序處理，而不會不必要地鎖定群組，使其無法傳送訊息。

處理失敗

如果您在可見性逾時到期之前未處理和刪除訊息，因為應用程式錯誤、當機或連線問題，訊息會再次顯示在佇列中。然後，相同或不同的取用者可以擷取它，以進行另一個處理嘗試。這可確保即使初始處理失敗，訊息也不會遺失。不過，設定過高的可見性逾時可能會延遲未處理訊息的重新出現，進而可能減慢重試速度。請務必根據預期的處理時間設定適當的可見性逾時，以便及時處理訊息。

變更和終止可見性逾時

您可以使用 `ChangeMessageVisibility` 動作來變更或終止可見性逾時：

- 變更逾時 – 使用 動態調整可見性逾時 [ChangeMessageVisibility](#)。這可讓您延長或縮短逾時持續時間，以符合處理需求。
- 終止逾時 – 如果您決定不處理接收的訊息，請透過 `ChangeMessageVisibility` 動作將 `VisibilityTimeout` 設定為 0 秒，以終止其可見性逾時。這可立即讓其他消費者處理訊息。

最佳實務

使用下列最佳實務來管理 Amazon SQS 中的可見性逾時，包括設定、調整和延長逾時，以及使用無效字母佇列 (DLQs) 處理未處理的訊息。

- 設定和調整逾時。首先，設定可見性逾時，以符合您的應用程式通常需要處理和刪除訊息的時間上限。如果您不確定確切的處理時間，請從較短的逾時開始（例如 2 分鐘），並視需要延長。實作活動訊號機制，以定期延長可見性逾時，確保訊息在處理完成之前不會顯示。這可將重新處理未處理訊息的延遲降至最低，並防止過早可見。
- 延長逾時並處理 12-Hour 限制。如果您的處理時間不同或可能超過最初設定的逾時，請使用 `ChangeMessageVisibility` 動作在處理訊息時延長可見性逾時。請記住，可見性逾時的上限為從第一次收到訊息起 12 小時。延長逾時不會重設此 12 小時限制。如果您的處理需要比此限制更長的時間，請考慮使用或將任務 AWS Step Functions 分成較小的步驟。
- 處理未處理的訊息。若要管理多次處理嘗試失敗的訊息，請設定無效字母佇列 (DLQ)。這可確保在單獨擷取多次重試後無法處理的訊息，以供進一步分析或處理，防止它們在主佇列中重複傳播。

Amazon SQS 延遲佇列

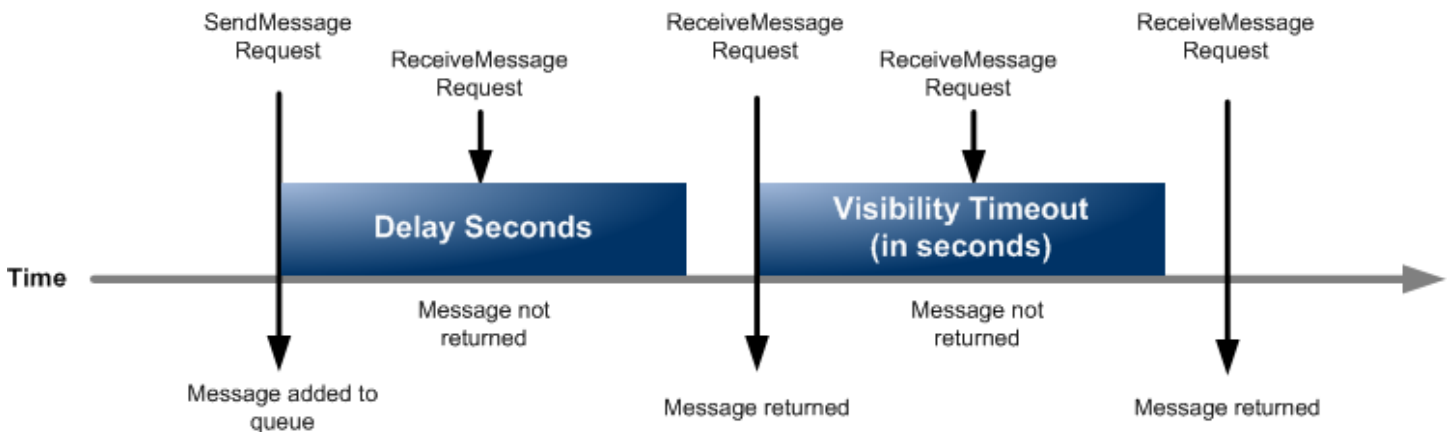
延遲佇列可讓您將新訊息傳遞給消費者延後幾秒鐘，例如，當您的消費者應用程式需要額外時間來處理訊息時。若您建立延遲佇列，則在延遲期間，您傳送到佇列的任何訊息對消費者都保持不可見。佇列的預設 (最小) 延遲時間為 0 秒。最大值為 15 分鐘。如需有關使用主控台來設定延遲佇列的詳細資訊，請參閱 [使用 Amazon SQS 主控台設定佇列參數](#)。

Note

若為標準佇列，個別佇列的延遲設定不會追溯—變更設定也不會影響到已經存在於佇列中的訊息延遲時間。

若為 FIFO 佇列，個別佇列的延遲設定會追溯 - 變更設定也會影響到已經存在於佇列中的訊息延遲時間。

延遲佇列訊息類似於[可見性逾時](#)，這兩種功能均能在指定期間內讓消費者無法看到訊息。延遲佇列和可見性逾時的差異在於，延遲佇列是在訊息首度新增至佇列時便隱藏起來，而可見性逾時是在佇列上的訊息被消耗時才會將訊息隱藏。下圖顯示了延遲佇列與可見性逾時之間的關係。



延長排程選項

雖然 Amazon SQS 延遲佇列和訊息計時器允許在未來最多 15 分鐘內排程訊息傳遞，但您可能需要更靈活的排程功能。在這種情況下，請考慮使用 [EventBridge 排程器](#)，這可讓您排程數十億個一次性或重複的 API 動作，而不受時間限制。EventBridge 排程器是進階訊息排程使用案例的建議解決方案。

若要設定個別訊息的延遲秒數，而不是整個佇列，請使用[訊息計時器](#)來允許 Amazon SQS 使用訊息計時器DelaySeconds的值，而不是延遲佇列DelaySeconds的值。[EventBridge 排程器](#)也支援排程個別訊息。

Amazon SQS 臨時佇列

使用常見的訊息模式 (例如要求-回應) 時，暫存佇列可協助您節省開發時間和部署成本。您可以使用[暫存佇列用戶端](#)來建立高輸送量、符合成本效益的應用程式管理暫存佇列。

用戶端會自動將多個臨時佇列映射至單一 Amazon SQS 佇列，這些佇列是根據特定程序的需求所建立。如此可讓您的應用程式減少 API 呼叫次數，並在各暫時佇列的流量變低時提高輸送量。不再使用暫時佇列時，用戶端會自動清除暫時佇列，即使使用用戶端的部分處理序未完全關閉。

以下是暫時佇列的優點：

- 它們可以做為特定執行緒或處理序的輕量型通訊頻道。
- 不須額外費用，即可建立並刪除它們。
- 它們是相容於靜態 (一般) Amazon SQS 佇列的 API。這表示傳送與接收訊息的現有程式碼可以傳送訊息至虛擬佇列，並接收來自虛擬佇列的訊息。

虛擬佇列

虛擬佇列是暫時佇列用戶端建立的本機資料結構。虛擬佇列可讓您將多個流量不足的目的地合併成單一 Amazon SQS 佇列。如需最佳實務，請參閱避免將相同的訊息群組 ID 與虛擬佇列重複使用。

Note

- 建立虛擬佇列時，只會為消費者建立接收訊息的暫時資料結構。由於虛擬佇列不會進行 Amazon SQS 的 API 呼叫，因此虛擬佇列不須任何費用。
- TPS 配額會套用到單一主機佇列中的所有虛擬佇列。如需詳細資訊，請參閱 [Amazon SQS 訊息配額](#)。

AmazonSQSVirtualQueuesClient 包裝類別新增對於虛擬佇列相關的屬性支援。若要建立虛擬佇列，您必須使用 HostQueueURL 屬性呼叫 CreateQueue API 動作。此屬性指定託管虛擬佇列的現有佇列。

虛擬佇列的 URL 採用下列格式。

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

生產者呼叫虛擬佇列 URL 上的 `SendMessage` 或 `SendMessageBatch` API 動作時，暫時佇列用戶端會執行下列動作：

1. 擷取虛擬佇列名稱。
2. 將虛擬佇列名稱做為其他訊息名稱連接。
3. 傳送訊息至託管佇列。

生產者傳送訊息時，背景執行緒會輪詢託管佇列，然後根據對應的訊息屬性將已接收的訊息傳送至虛擬佇列。

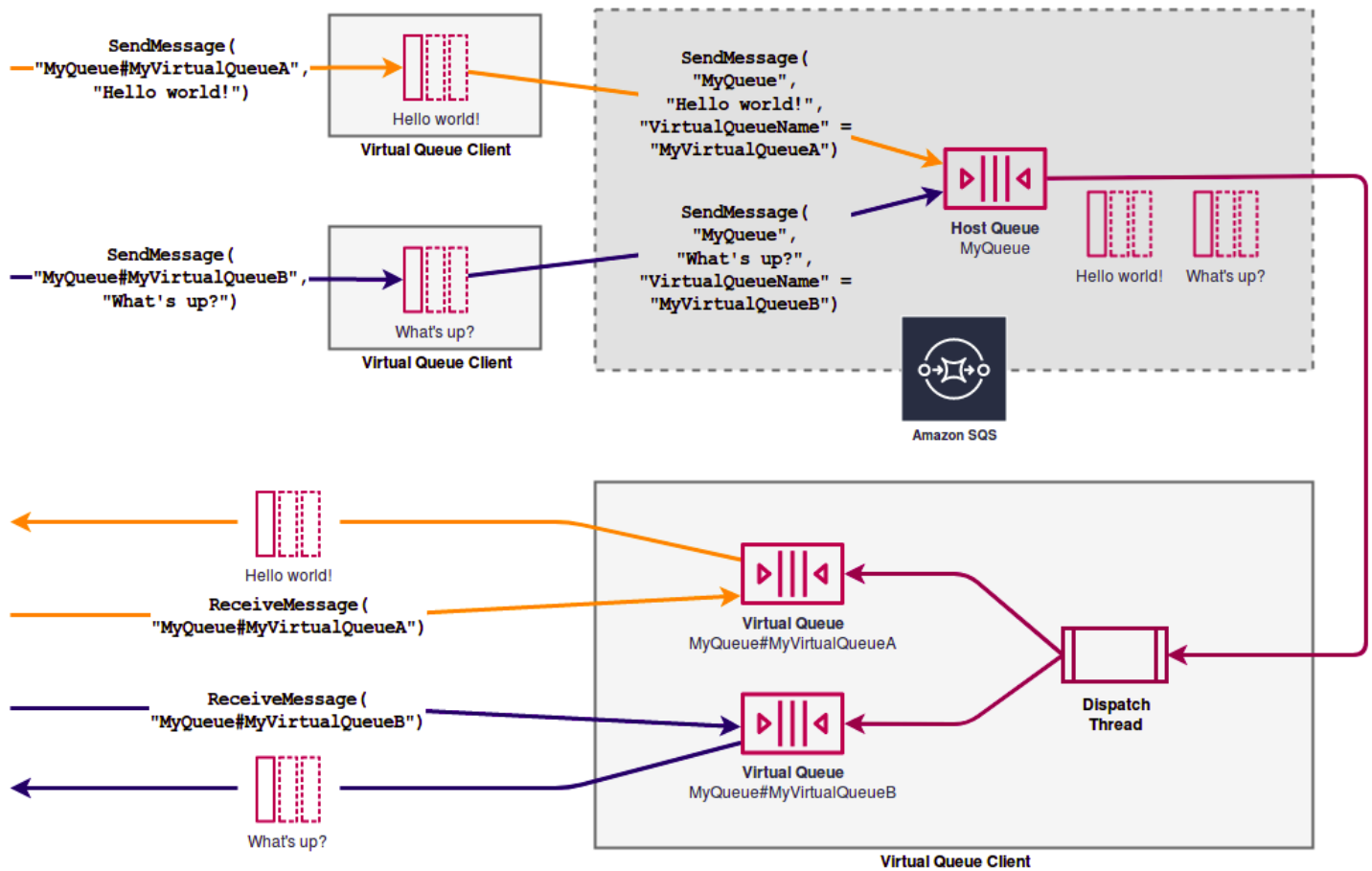
消費者呼叫虛擬佇列 URL 上的 `ReceiveMessage` API 動作時，暫時佇列用戶端便會在本機封鎖呼叫，直到背景執行緒傳送訊息至虛擬佇列。(此程序與[緩衝處理過的非同步用戶端](#)中的訊息預先提取相似：單一 API 動作最多可以提供訊息給 10 個虛擬佇列。) 刪除虛擬佇列時，會移除任何用戶端側資源，不會自行呼叫 Amazon SQS。

`AmazonSQSTemporaryQueuesClient` 類別會自動將其建立的所有佇列轉成暫時佇列。它還會自動依需求建立佇列屬性相同的託管佇列。這些佇列的名稱共用常見、可設定的字首 (預設為 `__RequesterClientQueues__`)，藉以表明為暫時佇列。這可讓用戶端做為便利替代方案，將建立與刪除佇列的現有程式碼最佳化。用戶端也包括允許佇列間雙向通訊的 `AmazonSQSRequester` 和 `AmazonSQSResponder` 介面。

請求-回應訊息模式 (虛擬佇列)

暫時佇列最常見的使用案例是請求-回應訊息模式，其中請求者會建立用於接收各回應訊息的暫時佇列。為了避免各回應訊息都會建立 Amazon SQS 佇列，暫時佇列用戶端可讓您建立與刪除多個暫時佇列，而無須進行任何 Amazon SQS API 呼叫。如需詳細資訊，請參閱實作請求回應系統。

下圖顯示使用此模式的常見組態。



範例情況：處理登入請求

以下範例情況顯示如何使用 `AmazonSQSRequester` 和 `AmazonSQSResponder` 介面來處理使用者的登入請求。

在用戶端

```
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();
}
}
```

傳送登入請求時，會進行下列動作：

1. 建立暫時佇列。
2. 將暫時佇列的 URL 連接至做為屬性的訊息。
3. 傳送訊息。
4. 接收來自暫時佇列的回應。
5. 刪除暫時佇列。
6. 傳回回應。

在伺服器端

以下範例假設在建構後，便會建立執行緒來輪詢佇列並呼叫各訊息的 `handleLoginRequest()` 方法，此外，`doLogin()` 為假設的方法。

```
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.sendMessage(MessageContent.fromMessage(message),
        new MessageContent(response));
}
}
```

清除佇列

若要確保 Amazon SQS 會回收虛擬佇列所使用的任何記憶體內資源，當您的應用程式不再需要暫時佇列用戶端時，它應呼叫 `shutdown()` 方法。您也可以使用 `AmazonSQSRequester` 介面的 `shutdown()` 方法。

暫時佇列用戶端也提供消除遺棄的託管佇列的方法。對於會在一段期間 (預設為 5 分鐘) 內接收 API 呼叫的各佇列而言，用戶端會使用 `TagQueue` API 動作來標記仍在使用的佇列。

Note

任何在佇列上採取的 API 動作都會把它標記為非閒置，包括不會傳回訊息的 `ReceiveMessage` 動作。

背景執行緒使用 `ListQueues` 和 `ListTags` API 動作來檢查包含已設定字首的所有佇列，刪除至少五分鐘未被標記的任何佇列。依此方式，如果一個用戶端未完全關閉，其他作用中用戶端會在之後清除。為減少工作重複，字首相同的所有用戶端都會透過以該字首命名的共用內部工作佇列進行通訊。

Amazon SQS 訊息計時器

訊息計時器可讓您在訊息新增至佇列時設定訊息的初始不可見期間。例如，如果您使用 45 秒計時器傳送訊息，則前 45 秒消費者會保持隱藏狀態。訊息的預設 (最小值) 延遲時間為 0 秒。最大值為 15 分鐘。如需有關使用主控台傳送附帶計時器的訊息的資訊，請參閱 [使用標準佇列傳送訊息](#)。

Note

FIFO 佇列不支援個別訊息的計時器。

若要設定整個佇列的延遲期間，而不是個別訊息，請使用 [延遲佇列](#)。個別訊息的訊息計時器設定會覆寫掉 Amazon SQS 延遲佇列上的任何 DelaySeconds 值。

延長排程選項

雖然 Amazon SQS 延遲佇列和訊息計時器允許在未來最多 15 分鐘內排程訊息傳遞，但您可能需要更靈活的排程功能。在這種情況下，請考慮使用 [EventBridge 排程器](#)，這可讓您排程數十億個一次性或重複的 API 動作，而不受時間限制。EventBridge 排程器是進階訊息排程使用案例的建議解決方案。

透過 Amazon SQS 主控台存取 Amazon EventBridge 管道

Amazon EventBridge Pipes 會將來源連接至目標。管道用於支援的來源和目標之間的點對點整合，並支援進階轉換和擴充。EventBridge Pipes 提供高度可擴展的方式，將您的 Amazon SQS 佇列連線至 Step Functions、Amazon SQS 和 API Gateway 等 AWS 服務，以及第三方軟體即服務 (SaaS) 應用程式，例如 Salesforce。

若要設定管道，您可以選擇來源、新增可選篩選、定義可選的擴充，以及選擇事件資料的目標。

在 Amazon SQS 佇列的詳細資料頁面上，您可以檢視使用該佇列做為其來源的管道。從那裡，您還可以：

- 啟動 EventBridge 主控台以檢視管道詳細資料。
- 啟動 EventBridge 主控台，以佇列作為其來源建立新管道。

如需將 Amazon SQS 佇列設定為管道來源的相關資訊，請參閱《Amazon EventBridge 使用者指南》中的 [Amazon SQS 佇列做為來源](#)。如需 EventBridge 管道的相關資訊，請參閱 [EventBridge 管道](#)。

若要存取指定 Amazon SQS 佇列的 EventBridge 管道

1. 開啟 Amazon SQS 主控台的[佇列](#)頁面。
2. 請選擇一個佇列。
3. 在佇列詳細資訊頁面上，選擇 EventBridge 管道索引標籤。

EventBridge 管道標籤包含目前設定為使用所選佇列作為來源的任何管道清單，包括：

- 管道名稱
 - 目前的狀態
 - 管道目標
 - 上次修改管道的時間
4. 檢視更多管道詳細資料或建立新管道 (如果需要)：
- 若要存取有關管的更多詳細資料：

選擇管道名稱。

這會啟動 EventBridge 主控台的管道詳細資料頁面。

- 若要建立新管道：

選擇將 Amazon SQS 佇列 Connect 到管道。

這會啟動 EventBridge 主控台的建立管道頁面，並將 Amazon SQS 佇列指定為管道來源。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[建立 EventBridge 管道](#)。

Important

Amazon SQS 佇列上的訊息會由單一管道讀取，然後在處理完畢後從佇列中刪除，無論訊息是否符合您可為該管道設定的篩選器。將多個管道設定為使用與其來源相同的佇列時，請務必小心操作。

使用擴充用戶端程式庫和 Amazon Simple Storage Service 管理大型 Amazon SQS 訊息

使用[適用於 Java 的 Amazon SQS 延伸用戶端程式庫](#)和[適用於 Python 的 Amazon SQS 延伸用戶端程式庫](#)來傳送大型訊息，特別是 256 KB 和 2 GB 之間的承載。這些程式庫會將訊息承載存放在 Amazon S3 儲存貯體中，並傳送參考給 Amazon SQS 佇列中儲存的物件。

Note

Amazon SQS 延伸用戶端程式庫與標準佇列和 FIFO 佇列相容。

使用 Java 和 Amazon S3 管理大型 Amazon SQS 訊息 Amazon S3

使用適用於 [Java 的 Amazon SQS 擴充用戶端程式庫](#)搭配 Amazon S3 來管理大型 Amazon SQS 訊息，特別是 256 KB 到 2 GB 的承載。程式庫會將訊息承載存放在 Amazon S3 儲存貯體中，並傳送訊息，其中包含對 Amazon SQS 佇列中存放物件的參考。

透過適用於 Java 的 Amazon SQS 擴充用戶端程式庫，您可以：

- 指定訊息是否一律儲存在 Amazon S3，或僅儲存大小超過 256 KB 的訊息。
- 傳送會參考儲存在 S3 儲存貯體內的單一訊息物件的訊息。
- 從 Amazon S3 儲存貯體擷取訊息物件
- 從 Amazon S3 儲存貯體刪除訊息物件

先決條件

下列範例使用 AWS Java 開發套件。若要安裝和設定 SDK，請參閱《適用於 Java 的 AWS SDK 開發人員指南》中的[設定適用於 Java 的 AWS SDK](#)。

執行範例程式碼之前，請設定您的 AWS 登入資料。如需詳細資訊，請參閱《適用於 Java 的 AWS SDK 開發人員指南》中的[設定開發的 AWS 登入資料和區域](#)。

[適用於 Java 的開發套件](#)和適用於 Java 的 Amazon SQS 擴充用戶端程式庫需要 J2SE 開發套件 8.0 或更新版本。

 Note

您可以使用適用於 Java 的 Amazon SQS 擴充用戶端程式庫來管理僅搭配 適用於 Java 的 AWS SDK使用 Amazon S3 的 Amazon SQS 訊息。您無法使用 AWS CLI、Amazon SQS 主控台、Amazon SQS HTTP API 或任何其他 AWS SDKs 來執行此操作。

AWS SDK for Java 2.x 範例：使用 Amazon S3 管理大型 Amazon SQS 訊息

下列適用於 Java 的 SDK 2.x 範例會建立具有隨機名稱的 Amazon S3 儲存貯體，並新增生命週期規則，以在 14 天後永久刪除物件。它也會建立名為 的佇列，MyQueue並將存放在 Amazon S3 儲存貯體中的隨機訊息傳送至佇列，並且超過 256 KB。最後，程式碼會擷取該訊息，傳回訊息的相關資訊，接著刪除訊息、佇列和儲存貯體。

```
/*
 * 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());
    }
}

```

您可以使用 [Apache Maven](#) 為您的 Java 專案設定和建置 Amazon SQS 延伸用戶端，或建置 SDK 本身。從您在應用程式中使用的 SDK 指定個別模組。

```

<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>
```

使用 Python 和 Amazon S3 管理大型 Amazon SQS 訊息 Amazon S3

使用 Amazon SQS [Amazon SQS Extended Client Library for Python](#) 搭配 Amazon S3 來管理大型 Amazon SQS 訊息，尤其是 256 KB 到 2 GB 之間的承載。程式庫會將訊息承載存放在 Amazon S3 儲存貯體中，並傳送訊息，其中包含對 Amazon SQS 佇列中存放物件的參考。

使用適用於 Python 的 Amazon SQS 擴充用戶端程式庫，您可以：

- 指定承載是否一律存放在 Amazon S3 中，或只在承載大小超過 256 KB 時才存放在 Amazon S3 中
- 傳送參考存放在 Amazon S3 儲存貯體中單一訊息物件的訊息
- 從 Amazon S3 儲存貯體擷取對應的承載物件
- 從 Amazon S3 儲存貯體刪除對應的承載物件

先決條件

以下是使用 Amazon SQS Extended Client Library for Python 的先決條件：

- 具有必要登入資料的 AWS 帳戶。若要建立 AWS 帳戶，請導覽至[AWS 首頁](#)，然後選擇建立 AWS 帳戶。遵循指示。如需登入資料的資訊，請參閱[登入](#)資料。
- SDK AWS：此頁面的範例使用 AWS Python SDK Boto3。若要安裝和設定 SDK，請參閱《適用於 [AWS Python 的 SDK](#) AWS 開發人員指南》中的適用於 Python 的 SDK 文件
- Python 3.x（或更新版本）和 pip。
- Amazon SQS Extended Client Library for Python，可從 [PyPI](#) 取得

Note

您可以使用適用於 Python 的 Amazon SQS 擴充用戶端程式庫，僅搭配適用於 Python 的 AWS SDK 使用 Amazon S3 來管理 Amazon SQS 訊息。您無法使用 AWS CLI、Amazon SQS 主控台、Amazon SQS HTTP API 或任何其他 AWS SDKs 來執行此操作。

設定訊息儲存

Amazon SQS 延伸用戶端可讓 使用下列訊息屬性來設定 Amazon S3 訊息儲存選項：

- `large_payload_support`：儲存大型訊息的 Amazon S3 儲存貯體名稱。
- `always_through_s3`：如果為 `True`，則所有訊息都會存放在 Amazon S3 中。如果 `False`，小於 256 KB 的訊息將不會序列化至 s3 儲存貯體。預設值為 `False`。
- `use_legacy_attribute`：如果為 `True`，則所有已發佈的訊息都會使用舊版預留訊息屬性 (`SQSLargePayloadSize`)，而非目前的預留訊息屬性 (`ExtendedPayloadSize`)。

使用適用於 Python 的擴展用戶端程式庫管理大型 Amazon SQS 訊息

下列範例會建立具有隨機名稱的 Amazon S3 儲存貯體。然後，它會建立名為 `MyQueue` 的 Amazon SQS 佇列，`MyQueue` 並將存放在 S3 儲存貯體中且超過 256 KB 的訊息傳送至佇列。最後，程式碼會擷取該訊息，傳回訊息的相關資訊，接著刪除訊息、佇列和儲存貯體。

```
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
```

使用 Amazon SQS 主控台設定 Amazon SQS 佇列

使用 Amazon SQS 主控台來設定和管理 Amazon SQS 佇列和功能。您也可以：

- 啟用伺服器端加密以增強安全性。
- 關聯無效字母佇列來處理未處理的訊息。
- 設定觸發來叫用 Lambda 函數以進行事件驅動的處理。

Amazon SQS 的屬性型存取控制

ABAC 是什麼？

屬性型存取控制 (ABAC) 是一種授權程序，可根據連接至使用者 AWS 和資源的標籤來定義許可。ABAC 根據屬性和值提供精細且靈活的存取控制，降低與重新設定角色型政策相關的安全風險，並集中進行稽核及存取政策管理。如需更多有關 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的[什麼是適用於 AWS 的 ABAC](#)。

Amazon SQS 支援 ABAC 的方式是讓您根據與 Amazon SQS 佇列相關聯的標籤和別名來控制對 Amazon SQS 佇列的存取。在 Amazon SQS 中啟用 ABAC 的標籤和別名條件金鑰，授權給 IAM 主體使用 Amazon SQS 佇列，而無需編輯政策或管理授與。若要進一步了解 ABAC 條件金鑰，請參閱服務授權參考中的[Amazon SQS 條件金鑰](#)。

若使用 ABAC，您可以使用標籤為 Amazon SQS 佇列設定 IAM 存取許可和政策，以協助您擴展許可權限管理範圍。您可以使用新增至每個商務角色的標籤，在 IAM 中建立單一許可政策，而不必在每次新增資源時更新政策。您也可以將標籤連接至 IAM 主體，以建立 ABAC 政策。當進行呼叫的 IAM 使用者角色上的標籤與 Amazon SQS 佇列標籤相符時，您可以設計 ABAC 政策來允許 Amazon SQS 作業。若要進一步了解如何標記 AWS，請參閱[AWS 標記策略](#)和[Amazon SQS 成本分配標籤](#)。

Note

ABAC for Amazon SQS 目前可在所有可使用 Amazon SQS AWS 的商業區域使用，但以下情況除外：

- 亞太區域 (海德拉巴)
- 亞太區域 (墨爾本)
- 歐洲 (西班牙)

- 歐洲 (蘇黎世)

為什麼一定要在 Amazon SQS 中使用 ABAC ？

以下是在 Amazon SQS 中使用 ABAC 的一些好處：

- Amazon SQS 的 ABAC 需要較少的許可政策。您不需要為不同的工作職能建立不同政策。您可以使用套用至一個以上的佇列的資源和請求標籤，如此可降低營運成本。
- 使用 ABAC 快速擴展團隊。當資源在建立期間適當標記，就會根據標籤自動授予新資源許可權。
- 使用 IAM 主體的許可權來限制資源存取。您可以為 IAM 主體建立標籤，並使用這些標籤來限制對符合 IAM 主體標籤之特定動作的存取。這可協助您將授與請求權限的程序自動化。
- 追蹤誰在存取您的資源。您可以透過查看 AWS CloudTrail 中的使用者屬性來判斷工作階段的身分識別。

主題

- [標記 Amazon SQS 中的存取控制](#)
- [建立 IAM 使用者和 Amazon SQS 佇列](#)
- [在 Amazon SQS 中測試屬性型存取控制](#)

標記 Amazon SQS 中的存取控制

以下是在 Amazon SQS 中使用標籤進行存取控制的範例。IAM 政策限制 IAM 使用者只能執行所有佇列的所有 Amazon SQS 操作，這些佇列包含鍵為 environment、值為 production 的資源標籤。如需詳細資訊，請參閱[使用標籤和 AWS Organizations 的屬性型存取控制](#)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessForProd",
      "Effect": "Deny",
      "Action": "sqs:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
```

```
        "aws:ResourceTag/environment": "prod"
      }
    }
  }
]
}
```

建立 IAM 使用者和 Amazon SQS 佇列

下列範例說明如何建立 ABAC 政策，以使用 AWS Management Console 和 控制對 Amazon SQS 的存取 AWS CloudFormation。

使用 AWS Management Console

建立 IAM 使用者

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/iam/> 開啟 IAM 主控台。
2. 從左側導覽窗格中，選擇使用者。
3. 選擇新增使用者，然後在使用者名稱文字方塊中輸入名稱。
4. 選取存取金鑰 - 程式設計存取方塊，然後選擇下一步：許可。
5. 選擇 Next: Add Tags (下一步：新增標籤)。
6. 新增鍵 environment 且值為 beta 的標籤。
7. 選擇下一步：檢閱，然後選擇建立使用者。
8. 請將存取金鑰 ID 和私密存取金鑰複製並存放在安全之處。

新增 IAM 使用者許可權

1. 選取您建立的 IAM 使用者。
2. 選擇 Add inline policy (新增內嵌政策)。
3. 在 JSON 標籤上，貼上下列政策。

```
{
  "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"
        }
    }
}
]
}

```

4. 選擇檢閱政策。

5. 選擇 建立政策。

使用 AWS CloudFormation

使用下列範例 AWS CloudFormation 範本建立已連接內嵌政策和 Amazon SQS 佇列的 IAM 使用者：

```
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"

```

在 Amazon SQS 中測試屬性型存取控制

以下範例顯示如何在 Amazon SQS 中測試屬性型存取控制。

使用設定為 `environment` 的標籤鍵和設定為 `prod` 的標籤值來建立佇列

執行此 AWS CLI 命令，以測試建立佇列時，標籤索引鍵設為 `環境`，標籤值設為 `prod`。如果您沒有 AWS CLI，您可以[下載並設定](#)機器的 CLI。

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

您收到來自 Amazon SQS 端點的 AccessDenied 錯誤訊息：

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

這是因為 IAM 使用者上的標籤值與 [CreateQueue](#) API 呼叫中傳遞的標籤不符。請記住，我們將標籤套用至具有鍵設定為 environment 且值設定為 beta 的 IAM 使用者。

以設定為 environment 的標籤鍵和設定為 beta 的標籤值建立佇列

執行此 CLI 命令，測試以設定為 environment 的標籤鍵和設定為 beta 的標籤值建立佇列。

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

您會收到一則訊息，確認佇列已成功建立，類似下列內容。

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

傳送訊息至佇列

執行此 CLI 命令以測試將訊息傳送至佇列。

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

回應顯示已成功將訊息傳遞至 Amazon SQS 佇列。IAM 使用者許可權允許您將訊息傳送至具有 beta 標籤的佇列。回應包括內含訊息的 MD5fMessageBody 和 MessageId。

```
{
  "MD5fMessageBody": "<MD5fMessageBody>",
  "MessageId": "<MessageId>"
}
```

使用 Amazon SQS 主控台設定佇列參數

[建立](#)或[編輯](#)佇列時，您可以設定下列參數：

- 可見性逾時 - 其他訊息消費者看不到從佇列 (由一位消費者) 接收訊息的時間長度。如需詳細資訊，請參閱[可見性逾時](#)。

Note

使用主控台設定可見性逾時，可設定佇列中所有訊息的逾時值。若要設定單一或多個訊息的逾時，您必須使用其中一個 AWS SDKs。

- 訊息保留期 - Amazon SQS 保留留在佇列中之訊息的時間量。在預設情況下，佇列會將訊息保留四天。您可以設定佇列為保留訊息最多 14 天。如需詳細資訊，請參閱[訊息保留期間](#)。
- 交付延遲 - Amazon SQS 在交付新增至佇列的訊息之前延遲的時間量。如需詳細資訊，請參閱[交付延遲](#)。
- 訊息大小上限 - 此佇列的訊息大小上限。如需詳細資訊，請參閱[訊息大小上限](#)。
- 接收訊息等待時間 - Amazon SQS 等待訊息在佇列收到接收請求後變為可用的時間上限。如需詳細資訊，請參閱 [Amazon SQS 短輪詢和長輪詢](#)。
- 啟用內容型重複資料刪除 - Amazon SQS 可以根據訊息內文自動建立重複資料刪除 ID。如需詳細資訊，請參閱 [Amazon SQS FIFO 佇列](#)。
- 啟用高輸送量 FIFO - 用於為佇列中的訊息啟用高輸送量。選擇此選項會將相關選項 ([重複資料刪除範圍](#)和 [FIFO 輸送量限制](#)) 變更為啟用 FIFO 佇列高輸送量的必要設定。如需詳細資訊，請參閱[Amazon SQS 中 FIFO 佇列的高輸送量](#)及[Amazon SQS 訊息配額](#)。
- 再驅動允許政策：定義哪些來源佇列能將此佇列用作無效字母佇列。如需詳細資訊，請參閱 [在 Amazon SQS 中使用無效字母佇列](#)。

若要設定現有佇列的佇列參數 (主控台)

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。選擇佇列，然後選擇編輯。
3. 捲動至組態區段。
4. 在可見性逾時中，輸入持續時間和單位。範圍介於 0 到 12 小時之間。預設值為 30 秒。
5. 在訊息保留期間中，輸入持續時間和單位。範圍介於 1 分鐘到 14 天之間。預設值為 4 天。
6. 對於標準佇列，請輸入接收訊息等待時間的值。範圍介於 0 到 20 秒之間。預設值為 0 秒，它會設定[短輪詢](#)。任何非零值都會設定長輪詢。
7. 對於遞送延遲，請輸入持續時間和單位。範圍介於 0 到 15 分鐘之間。預設值為 0 秒。
8. 在訊息大小上限中，輸入一個值。範圍介於 1 KB 至 256 KB 之間。預設值為 256 KB。

9. 對於 FIFO 佇列，請選擇啟用內容型重複資料刪除以啟用內容型重複資料刪除功能。預設設定為停用。
10. (選用) 若要讓 FIFO 佇列啟用更高輸送量以傳送和接收佇列中的訊息，請選擇啟用高輸送量 FIFO。

選擇此選項會將相關選項 (重複資料刪除範圍和 FIFO 輸送量限制) 變更為啟用 FIFO 佇列高輸送量的必要設定。如果您變更使用高輸送量 FIFO 所需的任何設定，則佇列的正常輸送量將生效，而重複資料刪除會依指定方式執行。如需詳細資訊，請參閱[Amazon SQS 中 FIFO 佇列的高輸送量](#)及[Amazon SQS 訊息配額](#)。

11. 對於再驅動允許政策，選擇啟用。從下列選項中選取：全部允許 (預設值)、依佇列或全部拒絕。選擇依佇列時，請依 Amazon Resource Name (ARN) 指定最多 10 個來源佇列的清單。
12. 完成佇列參數的設定後，請選擇儲存。

在 Amazon SQS 中設定存取政策

當您[編輯](#)佇列時，您可以設定其存取政策，以控制誰可以與其互動。

- 存取政策會定義哪些帳戶、使用者和角色具有存取佇列的許可。
- 它指定允許的動作，例如 [SendMessage](#)、[ReceiveMessage](#) 或 [DeleteMessage](#)。
- 根據預設，只有佇列擁有者具有傳送和接收訊息的許可。

設定現有佇列的存取政策（主控台）

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 選擇佇列，然後選擇編輯。
4. 捲動至存取政策區段。
5. 編輯輸入方塊中的存取政策陳述式。如需存取政策陳述式的詳細資訊，請參閱 [Amazon SQS 中的身分和存取管理](#)。
6. 完成設定存取政策後，請選擇儲存。

使用 SQS 受管加密金鑰設定佇列的伺服器端加密

除了預設的 Amazon SQS 受管伺服器端加密 (SSE) 選項之外，Amazon SQS 受管 SSE (SSE-SQS) 可讓您建立自訂的受管伺服器端加密，以使用 SQL 管理的加密金鑰來保護透過訊息佇列傳送的敏感資料。使用 SSE-SQS，您不需要建立和管理加密金鑰，也不需要修改程式碼來加密資料。SSE-SQS 可讓您安全地傳輸資料，並協助您符合嚴格的加密合規性和法規要求，而無需額外付費。

SSE-SQS 使用 256 位元進階加密標準 (AES-256) 加密法，來保護靜態資料。Amazon SQS 一收到訊息，SSE 就會將其加密。Amazon SQS 會以加密形式存放訊息，並且只有在將訊息傳送給授權的消費者時才會解密訊息。

Note

- 預設 SSE 選項只有在您建立佇列而未指定加密屬性時才有效。
- Amazon SQS 可讓您關閉所有佇列加密。因此，關閉 KMS-SSE，將不會自動啟用 SQS-SSE。如果您希望在關閉 KMS-SSE 之後啟用 SQS-SSE，您必須在請求中新增屬性變更。

若要設定佇列的 SSE-SQS 加密 (主控台)

Note

根據預設，使用 HTTP (非 TLS) 端點建立的任何新佇列都不會啟用 SSE-SQS 加密。使用 HTTPS 或[簽章版本 4](#) 端點建立 Amazon SQS 佇列是安全性最佳實務。

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 選擇佇列，然後選擇編輯。
4. 展開加密。
5. 在伺服器端加密下，選擇啟用。

Note

啟用 SSE 後，對加密佇列的匿名 SendMessage 和 ReceiveMessage 請求將被拒絕。Amazon SQS 安全性最佳實務建議您不要使用匿名請求。如果您希望將匿名請求傳送到 Amazon SQS 佇列，請務必停用 SSE。

6. 選取 Amazon SQS 金鑰 (SSE-SQS)。使用此選項無須額外付費。
7. 選擇 Save (儲存)。

使用 Amazon SQS 主控台設定佇列的伺服器端加密

為了保護佇列訊息中的資料，Amazon SQS 預設為所有新建立的佇列啟用伺服器端加密 (SSE)。Amazon SQS 已與 Amazon Web Services Key Management Service (Amazon Web Services KMS) 整合，以管理用於伺服器端加密 (SSE) 的 [KMS 金鑰](#)。如需使用 SSE 的詳細資訊，請參閱 [Amazon SQS 中的靜態加密](#)。

您指派給佇列的 KMS 金鑰必須具有金鑰政策，其中包含授權可使用佇列之所有主體的許可。如需詳細資訊，請參閱[金鑰管理](#)。

若您並非 KMS 金鑰的擁有者，或者您登入的帳戶並無 kms:ListAliases 和 kms:DescribeKey 的許可，即無法在 Amazon SQS 主控台上檢視 KMS 相關資訊。請要求 KMS 金鑰的擁有者授與您這些許可。如需詳細資訊，請參閱[金鑰管理](#)。

[建立](#)或[編輯](#)佇列時，可以設定 SSE-KMS。

若要設定現有佇列的 SSE-KMS (主控台)

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 選擇佇列，然後選擇編輯。
4. 展開加密。
5. 在伺服器端加密下，選擇啟用。

 Note

啟用 SSE 後，對加密佇列的匿名 SendMessage 和 ReceiveMessage 請求將被拒絕。Amazon SQS 安全性最佳實務建議您不要使用匿名請求。如果您希望將匿名請求傳送到 Amazon SQS 佇列，請務必停用 SSE。

6. 選取 AWS Key Management Service 金鑰 (SSE-KMS)。

主控台會顯示 KMS 金鑰的說明、帳戶和 KMS 金鑰 ARN。

7. 指定佇列的 KMS 金鑰 ID。如需詳細資訊，請參閱 [重要用語](#)。

a. 選擇選擇 KMS 金鑰別名選項。

b. 預設金鑰是 Amazon SQS 的 Amazon Web Services 託管 KMS 金鑰。若要使用此金鑰，請從 KMS 金鑰清單中選擇該金鑰。

c. 若要使用 Amazon Web Services 帳戶中的自訂 KMS 金鑰，請從 KMS 金鑰清單中選擇。如需建立自訂 KMS 金鑰的指示，請參閱《Amazon Web Services Key Management Service 開發人員指南》中的 [建立金鑰](#)。

d. 若要使用不在清單中的自訂 KMS 金鑰，或使用其他 Amazon Web Services 帳戶的自訂 KMS 金鑰，請選擇輸入 KMS 金鑰別名，然後輸入 KMS 金鑰 Amazon Resource Name (ARN)。

8. (選用) 對於資料金鑰重複使用期間，指定 1 分鐘到 24 小時之間的值。預設值為 5 分鐘。如需詳細資訊，請參閱 [了解資料金鑰重複使用期間](#)。

9. 完成 SSE-KMS 的設定後，請選擇儲存。

使用 Amazon SQS 主控台設定佇列的成本分配標籤

若要組織和識別 Amazon SQS 佇列，您可以新增成本分配標籤。如需詳細資訊，請參閱 [Amazon SQS 成本分配標籤](#)。

- 詳細資訊頁面上的標記標籤會顯示佇列的標籤。
- 您可以在 [建立](#) 或修改 [???](#) 佇列時新增或修改標籤。

若要設定現有佇列的標籤 (主控台)

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。

3. 選擇佇列，然後選擇編輯。
4. 捲動至標籤區段。
5. 新增、修改或移除佇列標籤：
 - a. 若要新增標籤，請選擇新增標籤、輸入鍵和值，然後選擇新增標籤。
 - b. 若要更新標籤，請變更其鍵和值。
 - c. 若要移除標籤，請選擇鍵/值對旁邊的移除。
6. 完成設定標籤後，請選擇儲存。

使用 Amazon SQS 主控台訂閱 Amazon SNS 主題的佇列 Amazon SQS

您可以訂閱一或多個 Amazon SQS 佇列至 Amazon SNS 主題。當您發佈訊息至主題時，Amazon SNS 會將訊息傳送至每個訂閱的佇列。Amazon SQS 會管理訂閱並處理所需的許可。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的[什麼是 Amazon SNS？](#)

當您訂閱 Amazon SQS 佇列至 Amazon SNS 主題時，Amazon SNS 會使用 HTTPS 將訊息轉送至 Amazon SQS。如需搭配加密的 Amazon SQS 佇列使用 Amazon SNS 的詳細資訊，請參閱[設定 AWS 服務的 KMS 許可](#)。Amazon SQS

Important

Amazon SQS 為每個存取政策最多支援 20 個陳述式。訂閱 Amazon SNS 主題會新增一個這類陳述式。超過此金額會導致主題訂閱交付失敗。

訂閱 Amazon SNS 主題的佇列（主控台）

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 在佇列清單中選擇要訂閱 Amazon SNS 主題的佇列。
4. 在 Actions (動作) 選單中選擇 Subscribe to Amazon SNS topic (訂閱 Amazon SNS 主題)。
5. 從指定此佇列功能表可用的 Amazon SNS 主題中，選擇佇列的 Amazon SNS 主題。

如果未列出 SNS 主題，請選擇輸入 Amazon SNS 主題 ARN，然後輸入主題的 Amazon Resource Name (ARN)。

6. 選擇 Save (儲存)。
7. 若要驗證訂閱，請將訊息發佈至主題，並檢視佇列中的訊息。如需詳細資訊，請參閱 [《Amazon Simple Notification Service 開發人員指南》](#) 中的 [Amazon SNS 訊息發佈](#)。

跨帳戶訂閱

如果您的 Amazon SQS 佇列和 Amazon SNS 主題不同 AWS 帳戶，則需要額外的許可。

主題擁有者（帳戶 A）

修改 Amazon SNS 主題的存取政策，AWS 帳戶 以允許 Amazon SQS 佇列的 訂閱。範例政策陳述式：

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

此政策允許帳戶111122223333訂閱 MyTopic。

佇列擁有者（帳戶 B）

修改 Amazon SQS 佇列的存取政策，以允許 Amazon SNS 主題傳送訊息。範例政策陳述式：

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

此政策允許 MyTopic 傳送訊息至 MyQueue。

跨區域訂閱

若要在不同的 中訂閱 Amazon SNS 主題 AWS 區域，請確定：

- Amazon SNS 主題的存取政策允許跨區域訂閱。
- Amazon SQS 佇列的存取政策允許 Amazon SNS 主題跨區域傳送訊息。

如需詳細資訊，請參閱 [《Amazon Simple Notification Service 開發人員指南》](#) 中的 [將 Amazon SNS 訊息傳送至不同區域中的 Amazon SQS 佇列或 AWS Lambda 函數](#)。

設定 Amazon SQS 佇列以觸發 AWS Lambda 函數

您可以使用 Lambda 函數來處理來自 Amazon SQS 佇列的訊息。Lambda 會輪詢佇列並同步叫用函數，以事件形式傳遞一批訊息。

設定可見性逾時

將佇列的可見性逾時設定為至少六倍的[函數逾時](#)。這可確保 Lambda 有足夠的時間在處理上一個批次時調節函數，以重試。

使用無效字母佇列 (DLQ)

指定無效字母佇列，以擷取 Lambda 函數無法處理的訊息。

處理多個佇列和函數

Lambda 函數可以透過為每個佇列建立單獨的事件來源來處理多個佇列。您也可以將多個 Lambda 函數與相同的佇列建立關聯。

加密佇列的許可

如果您將加密佇列與 Lambda 函數建立關聯，但 Lambda 不會輪詢訊息，請將 `kms:Decrypt` 許可新增至您的 Lambda 執行角色。

限制

佇列和 Lambda 函數必須位於相同位置 AWS 區域。

使用預設金鑰 (Amazon SQS 的 AWS 受管 KMS 金鑰) 的[加密佇列](#)無法在不同的 中叫用 Lambda 函數 AWS 帳戶。

如需實作詳細資訊，請參閱 [《AWS Lambda 開發人員指南》](#) 中的 [搭配使用 AWS Lambda 與 Amazon SQS](#)。

先決條件

若要設定 Lambda 函數觸發條件，您必須符合下列要求：

- 如果您使用的是使用者，您的 Amazon SQS 角色必須包含下列許可：
 - `lambda:CreateEventSourceMapping`
 - `lambda:ListEventSourceMappings`
 - `lambda:ListFunctions`
- Lambda 執行角色必須包含下列許可。
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `sqs:ReceiveMessage`
- 如果您將加密佇列與 Lambda 函數建立關聯，請將 `kms:Decrypt` 許可新增至 Lambda 執行角色。

如需詳細資訊，請參閱 [在 Amazon SQS 中管理存取的概觀](#)。

若要將佇列設定為觸發 Lambda 函數 (主控台)

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 在佇列頁面上，選擇要設定的佇列。
4. 在佇列頁面上，選擇 Lambda 觸發器索引標籤。
5. 在 Lambda 觸發程序頁面上，選擇 Lambda 觸發程序。

如果清單不包含您需要的 Lambda 觸發程序，請選擇設定 Lambda 函數觸發程序。輸入 Lambda 函數的 Amazon Resource Name (ARN)，或選擇現有的資源。然後選擇 Save (儲存)。

6. 選擇 Save (儲存)。主控台會儲存設定並顯示佇列的詳細資訊頁面。

在詳細資訊頁面上，Lambda 觸發程序索引標籤會顯示 Lambda 函數及其狀態。Lambda 函數大約需要 1 分鐘的時間才能與您的佇列產生關聯。

7. 若要驗證組態的結果，您可以[傳送訊息至佇列](#)，然後在 Lambda 主控台檢視觸發的 Lambda 函數。

使用 Amazon EventBridge 從 AWS 服務到 Amazon SQS 的通知自動化 Amazon SQS

Amazon EventBridge 可讓您近乎即時地自動化 AWS 服務 和回應事件，例如應用程式問題或資源變更。

- 您可以建立規則來篩選特定事件，並在規則符合事件時定義自動動作。
- EventBridge 支援多個目標，包括以 JSON 格式接收事件的 Amazon SQS 標準和 FIFO 佇列。

如需詳細資訊，請參閱 [Amazon EventBridge 使用者指南](#) 中的 [Amazon EventBridge 目標](#)。

使用 Amazon SQS 傳送具有屬性的訊息

對於標準和 FIFO 佇列，您可以將結構化中繼資料包含到訊息中，包括時間戳記、地理空間資料、簽章和識別符。如需詳細資訊，請參閱 [Amazon SQS 訊息屬性](#)。

使用 Amazon SQS 主控台將具有屬性的訊息傳送至佇列

1. 在 <https://console.aws.amazon.com/sqs/> 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 在佇列頁面上，選擇佇列。
4. 選擇傳送及接收訊息。
5. 輸入訊息屬性參數。
 - a. 在名稱文字方塊中，輸入最多 256 個字元的唯一名稱。
 - b. 針對屬性類型，選擇字串、數字或二進位。
 - c. （選用）輸入自訂資料類型。例如，您可以新增 **byte**、**int** 或 **float** 作為數字的自訂資料類型。
 - d. 在值文字方塊中，輸入訊息屬性值。

▼ Message attributes - Optional [Info](#)

Enter name	String ▼	Custom type	Enter value

6. 若要新增另一個訊息屬性，請選擇新增新屬性。

▼ Message attributes - Optional [Info](#)

Enter name

String ▼

Custom type

Enter value

Enter name

String ▼

Custom type

Enter value

Remove

Add new attribute

7. 您可以在傳送訊息之前隨時修改屬性值。
8. 若要刪除屬性，請選擇移除。若要刪除第一個屬性，請關閉訊息屬性。
9. 完成新增訊息屬性後，選擇傳送訊息。您的訊息已傳送，主控台會顯示成功訊息。若要檢視已傳送訊息之訊息屬性的相關資訊，請選擇檢視詳細資訊。選擇完成以關閉訊息詳細資訊對話方塊。

Amazon SQS 最佳實務

Amazon SQS 會管理和處理訊息佇列，讓應用程式的不同部分能夠可靠且大規模地交換訊息。本主題涵蓋關鍵操作最佳實務，包括使用長輪詢來減少空白回應、實作無效字母佇列來處理處理錯誤，以及最佳化佇列安全許可。

主題

- [Amazon SQS 錯誤處理和有問題的訊息](#)
- [Amazon SQS 訊息重複資料刪除和分組](#)
- [Amazon SQS 訊息處理和計時](#)

Amazon SQS 錯誤處理和有問題的訊息

本主題提供在 Amazon SQS 中管理和緩解錯誤的詳細說明，包括處理請求錯誤、擷取有問題的訊息，以及設定無效字母佇列保留以確保訊息可靠性的技術。

主題

- [在 Amazon SQS 中處理請求錯誤](#)
- [在 Amazon SQS 中擷取有問題的訊息](#)
- [在 Amazon SQS 中設定無效字母佇列保留](#)

在 Amazon SQS 中處理請求錯誤

若要處理請求錯誤，請使用以下其中一種策略：

- 如果您使用 AWS 開發套件，則已經有自動重試和退避邏輯可供您使用。如需詳細資訊，請參閱 Amazon Web Services 一般參考 中的 [AWS 中的錯誤重試與指數退避](#)。
- 如果您不使用 AWS SDK 功能進行重試和退避，請在未收到任何訊息、逾時或來自 Amazon SQS 的錯誤訊息後，先允許暫停（例如 200 毫秒）再重試 [ReceiveMessage](#) 動作。對於產生相同結果的後續 ReceiveMessage 使用，允許較長的暫停（例如 400 毫秒）。

在 Amazon SQS 中擷取有問題的訊息

若要擷取所有無法處理的訊息並收集準確的 CloudWatch 指標，請設定 [無效字母佇列](#)。

- Redrive policy 在來源佇列無法處理訊息指定的次數之後，會將訊息重新導向至無效信件佇列。
- 使用無效字母佇列可以降低訊息數量並降低讓您接觸到毒丸 (poison pill) 訊息 (已收到但無法處理的訊息) 的可能性。
- 佇列中包含毒丸訊息可能會曲解 [ApproximateAgeOfOldestMessage](#) CloudWatch 指標，因為這樣會提供毒丸訊息的錯誤存在時間。設定無效字母佇列有助於避免使用此指標時產生假警報。

在 Amazon SQS 中設定無效字母佇列保留

對於標準佇列，訊息的到期一律會以其原始排入佇列時間戳記為基礎。訊息被移至無效字母佇列時，排入佇列時間戳記不會變更。此 `ApproximateAgeOfOldestMessage` 測量結果會指出訊息移至無效字母佇列的時間，而非原始傳送訊息的時間。例如，假設訊息在移至無效字母佇列之前，在原始佇列中花費 1 天時間。如果無效字母佇列的保留期限為 4 天，則會在 3 天後從無效字母佇列中刪除訊息，而且 `ApproximateAgeOfOldestMessage` 為 3 天。因此，最佳實務是一律將無效字母佇列的保留期間設定為長於原始佇列的保留期間。

若為 FIFO 佇列，訊息移到無效字母佇列時，會重設訊息。此 `ApproximateAgeOfOldestMessage` 測量結果會指出訊息移到無效字母佇列的時間。在上面的相同範例中，訊息會在 4 天後從無效字母佇列中刪除，而且 `ApproximateAgeOfOldestMessage` 為 4 天。

Amazon SQS 訊息重複資料刪除和分組

本主題提供最佳實務，以確保 Amazon SQS 中的訊息處理一致。它說明如何使用：

- [MessageDeduplicationId](#) 以防止 FIFO 佇列中的重複訊息。
- [MessageGroupId](#) 在不同的訊息群組中管理訊息排序。

主題

- [避免在 Amazon SQS 中處理不一致的訊息](#)
- [在 Amazon SQS 中使用訊息重複資料刪除 ID](#)
- [使用 Amazon SQS 訊息群組 ID](#)
- [使用 Amazon SQS 接收請求嘗試 ID](#)

避免在 Amazon SQS 中處理不一致的訊息

因為 Amazon SQS 是分散式系統，所以即使 Amazon SQS 在 [ReceiveMessage](#) API 方法呼叫成功傳回時將訊息標示為已交付，消費者還是可能不會收到訊息。在此情況下，儘管消費者從未收到訊息，Amazon SQS 仍會將訊息記錄為交付至少一次。因為在這些情況下沒有其他嘗試交付訊息的動作，所以我們不建議將[無效字母佇列](#)的接收數目上限設定為 1。

在 Amazon SQS 中使用訊息重複資料刪除 ID

[MessageDeduplicationId](#) 是一種字符，僅用於 Amazon SQS FIFO 佇列，以防止重複的訊息傳遞。它可確保在 5 分鐘的重複資料刪除時段內，只會處理和交付具有相同重複資料刪除 ID 的訊息執行個體。

如果 Amazon SQS 已接受具有特定重複資料刪除 ID 的訊息，則任何具有相同 ID 的後續訊息都會獲得確認，但不會交付給消費者。

Note

即使訊息已接收和刪除，Amazon SQS 仍會繼續追蹤重複資料刪除 ID。

主題

- [何時在 Amazon SQS 中提供訊息重複資料刪除 ID](#)
- [在 Amazon SQS 中啟用單一生產者/消費者系統的重複資料刪除](#)
- [Amazon SQS 中的中斷復原案例](#)
- [在 Amazon SQS 中設定可見性逾時](#)

何時在 Amazon SQS 中提供訊息重複資料刪除 ID

在下列案例中，生產者應指定訊息重複資料刪除 ID：

- 傳送必須視為唯一的相同訊息內文時。
- 傳送具有相同內容但不同訊息屬性的訊息時，請確保個別處理每個訊息。
- 傳送具有不同內容的訊息時（例如，訊息內文中的重試計數器），但要求 Amazon SQS 將其識別為重複。

在 Amazon SQS 中啟用單一生產者/消費者系統的重複資料刪除

如果您有單一生產者和單一消費者，且訊息是唯一的，因為它們在內文中包含應用程式特定的訊息 ID，請遵循下列最佳實務：

- 為佇列啟用內容型重複資料刪除 (每個訊息都有唯一的內文)。生產者可省略訊息重複資料刪除 ID。
- 為 Amazon SQS FIFO 佇列啟用內容型重複資料刪除功能，並使用重複資料刪除 ID 傳送訊息時，重複資料刪除 ID [SendMessage](#) 會覆寫產生的內容型重複資料刪除 ID。
- 雖然消費者不需要為每個請求提供接收請求嘗試 ID，但最佳實務仍建議這麼做，因為這樣可讓失敗重試序列執行得更快。
- 您可以重試傳送或接收請求，因為它們不會干擾 FIFO 佇列中的訊息順序。

Amazon SQS 中的中斷復原案例

FIFO 佇列中的重複資料刪除程序分秒必爭。設計應用程式時，請確保生產者和消費者都可以從用戶端或網路中斷中復原，而不會導致重複或處理失敗。

生產者考量事項

- Amazon SQS 會強制執行 5 分鐘的重複資料刪除時段。
- 如果生產者在 5 分鐘後重試[SendMessage](#)請求，Amazon SQS 會將其視為新訊息，可能會建立重複項目。

消費者考量

- 如果取用者無法在可見性逾時到期之前處理訊息，另一個取用者可能會接收和處理訊息，導致重複處理。
- 根據應用程式的處理時間調整可見性逾時。
- 使用 [ChangeMessageVisibility](#) API 在訊息仍在處理時延長逾時。
- 如果訊息重複無法處理，請將其路由到[無效字母佇列 \(DLQ\)](#)，而不是允許它無限期地重新處理。
- 生產者必須注意到佇列的重複資料刪除間隔。Amazon SQS 的最低重複資料刪除間隔為 5 分鐘。在重複資料刪除間隔過期之後重試 [SendMessage](#) 請求，可能會將重複的訊息引進佇列中。例如，在汽車中的行動裝置傳送訊息，其順序很重要。如果在收到確認前汽車失去行動連線一段時間，則在重新連上行動連線後重試請求便會建立重複的訊息。

- 消費者必須擁有可見性逾時，以將可見性逾時過期前無法處理訊息的風險降至最低。訊息正在處理時，您可以呼叫 `ChangeMessageVisibility` 動作來延長可見性逾時。不過，如果可見性逾時已過期，另一個消費者可以立即開始處理訊息，如此會導致多次處理同一則訊息。若要避免這種情況，請設定[無效字母佇列](#)。

在 Amazon SQS 中設定可見性逾時

為了確保可靠的訊息處理，請將可見性逾時設定為比 AWS SDK 讀取逾時更長。這適用於搭配短輪詢和長輪詢使用 [ReceiveMessage](#) API 時。較長的可見性逾時可防止訊息在原始請求完成之前變成可供其他消費者使用，從而降低重複處理的風險。

使用 Amazon SQS 訊息群組 ID

[MessageGroupId](#) 是只在 Amazon SQS FIFO (First-In-First-Out) 佇列中使用的屬性，可將訊息組織成不同的群組。相同訊息群組內的訊息一律以嚴格順序一次處理一個訊息，確保不會同時處理來自相同群組的兩個訊息。標準佇列不會使用 `MessageGroupId`，也不會提供訂購保證。如果需要嚴格排序，請改用 FIFO 佇列。

主題

- [在 Amazon SQS 中交錯多個排序的訊息群組](#)
- [在 Amazon SQS 的多生產者/消費者系統中防止重複處理](#)
- [避免在 Amazon SQS 中使用相同訊息群組 ID 的大型訊息待處理項目](#)
- [避免將相同的訊息群組 ID 與 Amazon SQS 中的虛擬佇列重複使用](#)

在 Amazon SQS 中交錯多個排序的訊息群組

若要在單一 FIFO 佇列中交錯多個排序訊息群組，[MessageGroupId](#) 請將指派給每個群組（例如，不同使用者在工作階段資料）。這可讓多個取用者同時從佇列讀取，同時確保依順序處理相同群組中的訊息。

當具有特定的訊息 `MessageGroupId` 正在處理且隱藏時，直到可見性逾時過期或刪除訊息之前，其他取用者都無法處理來自相同群組的訊息。

在 Amazon SQS 的多生產者/消費者系統中防止重複處理

在訊息排序不是優先順序的高輸送量、低延遲系統中，生產者可以 [MessageGroupId](#) 為每個訊息指派唯一的。這可確保 Amazon SQS FIFO 佇列即使在多生產者/多消費者設定中也能消除重複項目。雖然此方法可防止重複的訊息，但它不保證訊息順序，因為每個訊息都被視為自己的獨立群組。

在具有多個生產者和消費者的任何系統中，一律會有重複交付的風險。如果消費者無法在可見性逾時到期之前處理訊息，Amazon SQS 會再次提供訊息，可能允許其他消費者接收訊息。若要緩解此問題，請根據處理時間確保適當的訊息確認和可見性逾時設定。

避免在 Amazon SQS 中使用相同訊息群組 ID 的大型訊息待處理項目

FIFO 佇列最多支援 120,000 個傳輸中訊息（消費者收到但尚未刪除的訊息）。如果達到此限制，Amazon SQS 不會傳回錯誤，但處理可能會受到影響。您可以聯絡 [AWS Support](#) 請求提高超過此限制。

FIFO 佇列會掃描前 120,000 則訊息，以判斷可用的訊息群組。如果大型待處理項目累積在單一訊息群組中，稍後傳送的其他群組的訊息將保持封鎖狀態，直到處理待處理項目為止。

Note

當消費者重複無法處理訊息時，可能會發生訊息待處理項目。這可能是因為訊息內容問題或消費者端故障。為了防止訊息處理延遲，請設定[無效字母佇列](#)，在多次失敗嘗試後移動未處理的訊息。這可確保可以處理相同訊息群組中的其他訊息，防止系統瓶頸。

避免將相同的訊息群組 ID 與 Amazon SQS 中的虛擬佇列重複使用

搭配共用主機佇列使用虛擬佇列時，請避免在不同虛擬佇列 [MessageGroupId](#) 中重複使用相同佇列。如果多個虛擬佇列共用相同的主機佇列，並包含具有相同的訊息 [MessageGroupId](#)，則這些訊息可以彼此封鎖，從而防止有效率的處理。為確保訊息處理順暢，請為不同虛擬佇列中的訊息指派唯一 [MessageGroupId](#) 值。

使用 Amazon SQS 接收請求嘗試 ID

接收請求嘗試 ID 是用來刪除 Amazon SQS 中重複 [ReceiveMessage](#) 呼叫的唯一權杖。在應用程式與 Amazon SQS 之間的網路中斷或連線問題期間，最佳實務是：

- [ReceiveMessage](#) 呼叫時提供接收請求嘗試 ID。
- 如果操作失敗，請使用相同的接收請求嘗試 ID 重試。

Amazon SQS 訊息處理和計時

本主題提供在 Amazon SQS 中最佳化訊息處理速度和效率的全方位指引，包括及時處理訊息的策略、選擇最佳輪詢模式，以及設定長輪詢以提高效能。

主題

- [在 Amazon SQS 中及時處理訊息](#)
- [在 Amazon SQS 中設定長輪詢](#)
- [在 Amazon SQS 中使用適當的輪詢模式](#)

在 Amazon SQS 中及時處理訊息

可見性逾時設定取決於您的應用程式需要花最久時間處理和刪除訊息。例如，如果您的應用程式需要 10 秒的時間來處理訊息，而您將可見性逾時設定為 15 分鐘，那麼如果先前的處理嘗試失敗，您必須等候相當長的時間才能再次嘗試處理訊息。或者，如果您的應用程式需要 10 秒的時間來處理訊息，但您將可見性逾時設定為只有 2 秒，那麼當原始消費者仍在處理訊息時，另一個消費者就會收到重複的訊息。

為了確保有足夠的時間來處理訊息，請使用以下其中一種策略：

- 如果您知道 (或可以合理預估) 處理訊息需要多長的時間，請將訊息的可見性逾時延長到處理和刪除訊息所需的最長時間。如需詳細資訊，請參閱[設定可見性逾時](#)。
- 如果您不知道處理一則訊息需要多久，請為您的消費者程序建立活動訊號：指定初始可見性逾時 (例如 2 分鐘)—接著只要您的消費者仍在處理訊息—每隔 1 分鐘就將可見性逾時延長 2 分鐘。

Important

最大可見性逾時是從 Amazon SQS 收到 `ReceiveMessage` 請求的時間起算 12 小時。延長可見性逾時不會重設 12 小時的最大值。

此外，您可能無法將個別訊息的逾時設定為完整 12 小時 (亦即 43,200 秒)，因為 `ReceiveMessage` 要求會啟動計時器。例如，如果您收到一則訊息，並透過傳送 `VisibilityTimeout` 等於 43,200 秒的 `ChangeMessageVisibility` 呼叫來設定 12 小時的最大值，則它可能會失敗。不過，除非透過 `ReceiveMessage` 要求訊息和更新可見性逾時之間有很大的延遲，否則使用值 43,195 秒就可以運作。如果您的消費者需要超過 12 小時，請考慮使用 Step Functions。

在 Amazon SQS 中設定長輪詢

當 [ReceiveMessage](#) API 動作的等待時間大於 0 時，長輪詢就會生效。最長輪詢等待時間上限為 20 秒。長輪詢可減少空回應 (沒有 `ReceiveMessage` 請求可用的訊息) 和 `False` 空回應 (有可用訊息但不

包含在回應中) 的數目，藉以降低使用 Amazon SQS 的成本。如需詳細資訊，請參閱 [Amazon SQS 短輪詢和長輪詢](#)。

為確保最佳訊息處理，請使用下列策略：

- 在大多數情況下，您可以設定 `ReceiveMessage` 等待時間為 20 秒。如果 20 秒對於您的應用程式來說太長，可以設定較短的 `ReceiveMessage` 等待時間 (最低 1 秒)。如果您不使用 AWS SDK 存取 Amazon SQS，或者如果您將 AWS SDK 設定為等待時間較短，則可能需要修改 Amazon SQS 用戶端，以允許更長的請求或使用更短的等待時間進行長輪詢。
- 如果您對多個佇列實作長輪詢，請為每個佇列使用一個執行緒，而非所有佇列使用單一執行緒。為每個佇列各使用一個執行緒可讓您的應用程式在訊息可供使用時，分別處理每個佇列中的訊息，而將單一執行緒用於輪詢多個佇列則可能導致您的應用程式無法處理其他佇列中的可用訊息，因為它必須等候沒有任何可用訊息的佇列 (最長 20 秒)。

Important

若要避免 HTTP 錯誤，請確定 `ReceiveMessage` 請求的 HTTP 回應逾時比 `WaitTimeSeconds` 參數長。如需詳細資訊，請參閱 [ReceiveMessage](#)。

在 Amazon SQS 中使用適當的輪詢模式

- 長輪詢可讓您在訊息一旦從 Amazon SQS 佇列提供使用時，立即使用訊息。
 - 若要降低使用 Amazon SQS 的成本並降低空佇列的空白接收數 (回應未傳回訊息的 `ReceiveMessage` 動作)，請啟用長輪詢。如需詳細資訊，請參閱 [Amazon SQS 長輪詢](#)。
 - 若要在輪詢具有多個接收訊息的多個執行緒時提高效率，請降低執行緒數量。
 - 在大部分情況下，長輪詢都優於短輪詢。
- 即使受輪詢的 Amazon SQS 佇列空白，短輪詢仍會立即傳回回應。
 - 若要滿足期望立即回應 `ReceiveMessage` 請求的應用程式要求，請使用短輪詢。
 - 短輪詢的計費與長輪詢相同。

Amazon SQS Java 開發套件範例

適用於 Java 的 AWS SDK 可讓您建置與 Amazon SQS 和其他 互動的 Java 應用程式 AWS 服務。

- 若要安裝和設定 SDK，請參閱《AWS SDK for Java 2.x 開發人員指南》中的[入門](#)。
- 如需基本佇列操作，例如建立佇列或傳送訊息，請參閱《AWS SDK for Java 2.x 開發人員指南》中的[使用 Amazon SQS 訊息佇列](#)。
- 本指南也包含其他 Amazon SQS 功能的範例，例如：
 - [搭配 Amazon SQS 佇列使用伺服器端加密](#)
 - [設定 Amazon SQS 佇列的標籤](#)
 - [將訊息屬性傳送至 Amazon SQS 佇列](#)

搭配 Amazon SQS 佇列使用伺服器端加密

使用 適用於 Java 的 AWS SDK 將伺服器端加密 (SSE) 新增至 Amazon SQS 佇列。每個佇列都使用 AWS Key Management Service (AWS KMS) KMS 金鑰來產生資料加密金鑰。此範例使用 Amazon SQS 的 AWS 受管 KMS 金鑰。

如需使用 SSE 和 KMS 金鑰之角色的相關資訊，請參閱 [Amazon SQS 中的靜態加密](#)。

將 SSE 新增到現有佇列

若要啟用現有佇列的伺服器端加密，請使用 [SetQueueAttributes](#) 方法來設定 KmsMasterKeyId 屬性。

下列程式碼範例會將 AWS KMS key 設定為 Amazon SQS 的 AWS 受管 KMS 金鑰。此範例也會將 [AWS KMS key 重複使用期間](#)設定為 140 秒。

執行範例程式碼之前，請確定您已設定 AWS 登入資料。如需詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的[設定用於開發的 AWS 登入資料和區域](#)。

```
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();
}
}
```

停用佇列的 SSE

若要停用現有佇列的伺服器端加密，請使用 `SetQueueAttributes` 方法將 `KmsMasterKeyId` 屬性設定為空白字串。

Important

`null` 不是 `KmsMasterKeyId` 的有效值。

使用 SSE 建立佇列

若要在建立佇列時啟用 SSE，請將 `KmsMasterKeyId` 屬性新增至 [CreateQueue](#) API 方法。

以下範例顯示如何建立啟用 SSE 的新佇列。佇列使用 Amazon SQS 的 AWS 受管 KMS 金鑰。此範例也會將 [AWS KMS key 重複使用期間](#) 設定為 160 秒。

執行範例程式碼之前，請確定您已設定 AWS 登入資料。如需詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的 [設定開發的 AWS 登入資料和區域](#)。

```
// 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);
```

擷取 SSE 屬性

如需擷取佇列屬性的相關資訊，請參閱《Amazon Simple Queue Service API 參考》中的 [範例](#)。

若要擷取特定佇列的 KMS 金鑰 ID 或資料金鑰重複使用期間，請執行 [GetQueueAttributes](#) 方法並擷取 `KmsMasterKeyId` 和 `KmsDataKeyReusePeriodSeconds` 值。

設定 Amazon SQS 佇列的標籤

使用成本分配標籤來幫助組織和識別您的 Amazon SQS 佇列。下列範例示範如何使用適用於 Java 的 AWS SDK 設定標籤。如需詳細資訊，請參閱 [Amazon SQS 成本分配標籤](#)。

執行範例程式碼之前，請確定您已設定登入 AWS 資料。如需詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的[設定開發的 AWS 登入資料和區域](#)。

列出標籤

若要列出佇列的標籤，請使用 `ListQueueTags` 方法。

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

新增或更新標籤

若要新增或更新佇列的標籤值，請使用 `TagQueue` 方法。

```
// 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);
```

移除標籤

若要移除佇列的一個或多個標籤，請使用 `UntagQueue` 方法。下列範例會移除 Accounting ID 標籤。

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

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

將訊息屬性傳送至 Amazon SQS 佇列

您可使用訊息屬性提供有關訊息的結構化中繼資料項目 (例如，時間戳記、地理空間資料、簽章和識別碼)。如需詳細資訊，請參閱 [Amazon SQS 訊息屬性](#)。

執行範例程式碼之前，請確定您已設定登入 AWS 資料。如需詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的 [設定開發的 AWS 登入資料和區域](#)。

定義屬性

為了定義訊息的屬性，請新增以下使用 [MessageAttributeValue](#) 資料類型的程式碼。如需詳細資訊，請參閱 [訊息屬性元件](#) 和 [訊息屬性資料類型](#)。

適用於 Java 的 AWS SDK 會自動計算訊息內文和訊息屬性檢查總和，並將其與 Amazon SQS 傳回的資料進行比較。如需詳細資訊，請參閱 [AWS SDK for Java 2.x 開發人員指南](#) 以及適用於其他程式設計語言的 [計算訊息屬性的 MD5 訊息摘要](#)。

String

此範例將名為 Name 的 String 屬性定義為 Jane 的值。

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

Number

此範例將名為 AccurateWeight 的 Number 屬性定義為 230.000000000000000001 的值。

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

Binary

此範例將名為 ByteArray 的 Binary 屬性定義為未初始化 10 位元組陣列的值。

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

String (custom)

此範例將名為 EmployeeId 的自訂屬性 String.EmployeeId 定義為 ABC123456 的值。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
```

```
messageAttributes.put("EmployeeId", new MessageAttributeValue()  
    .withDataType("String.EmployeeId")  
    .withStringValue("ABC123456"));
```

Number (custom)

此範例將名為 `AccountId` 的自訂屬性 `Number.AccountId` 定義為 `000123456` 的值。

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

Note

由於基本資料類型為 `Number`，[ReceiveMessage](#) 動作會傳回 `123456`。

Binary (custom)

此範例將名為 `ApplicationIcon` 的自訂屬性 `Binary.JPEG` 定義為未初始化 10 位元組陣列的值。

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

傳送具有屬性的訊息

此範例會在傳送訊息之前將屬性新增至 `SendMessageRequest`。

```
// 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

如果您將訊息傳送到 先進先出 (FIFO) 佇列，請確保 `sendMessage` 方法在您提供訊息群組 ID 之後才執行。

如果您使用 [SendMessageBatch](#) 方法，而非 [SendMessage](#)，您必須為批次中每個訊息指定訊息屬性。

搭配 Amazon SQS 使用 APIs

本主題提供建構 Amazon SQS 端點、使用 GET 和 POST 方法提出查詢 API 請求，以及使用批次 API 動作的相關資訊。如需 Amazon SQS [動作](#) (包括參數、錯誤、範例和[資料類型](#)) 的詳細資訊，請參閱《[Amazon Simple Queue Service API 參考](#)》。

您若是使用各種程式設計語言存取 Amazon SQS，還可以利用 [AWS 開發套件](#) 提供的下列自動化功能：

- 加密簽署服務請求
- 重試請求
- 處理錯誤回應

如需詳細資訊，請參閱[the section called “使用 AWS SDKs”](#)。

如需命令列工具的相關資訊，請參閱 [AWS CLI 命令參考](#) 以及 [AWS Tools for PowerShell Cmdlet 參考](#) 中的 Amazon SQS 章節。

Amazon SQS APIs 搭配 AWS JSON 通訊協定

Amazon SQS 使用 AWS JSON 通訊協定做為指定 [AWS SDK 版本](#) 上所有 Amazon SQS APIs 的傳輸機制。AWS JSON 通訊協定提供更高的輸送量、更低的延遲，以及更快 application-to-application 通訊。AWS JSON 通訊協定相較於 AWS 查詢通訊協定，在請求和回應的序列化/還原序列化方面更有效率。如果您仍偏好搭配 SQS APIs 使用 AWS 查詢通訊協定，請參閱 [Amazon SQS API 中使用的 AWS JSON 通訊協定支援哪些語言？](#) 以取得支援 Amazon SQS AWS 查詢通訊協定的 AWS SDK 版本。

Amazon SQS 使用 AWS JSON 通訊協定，在 AWS SDK 用戶端（例如 Java、Python、Golang、JavaScript）和 Amazon SQS 伺服器之間進行通訊。Amazon SQS API 操作的 HTTP 請求接受 JSON 格式的輸入。系統會執行 Amazon SQS 作業，執行回應會以 JSON 格式傳回 SDK 用戶端。與 AWS 查詢相比，AWS JSON 在用戶端和伺服器之間傳輸資料更簡單、更快速且更有效率。

- AWS JSON 通訊協定充當 Amazon SQS 用戶端和伺服器之間的中介裝置。
- 伺服器不了解建立 Amazon SQS 操作的程式設計語言，但了解 AWS JSON 通訊協定。
- AWS JSON 通訊協定使用 Amazon SQS 用戶端和伺服器之間的序列化（將物件轉換為 JSON 格式）和去序列化（將 JSON 格式轉換為物件）。

如需使用 Amazon SQS 的 AWS JSON 通訊協定詳細資訊，請參閱 [Amazon SQS AWS JSON 通訊協定FAQs](#)。

AWS JSON 通訊協定可在指定的 [AWS SDK 版本](#) 上使用。若要查看跨語言變體的 SDK 版本和發行日期，請參閱《AWS 開發套件和工具參考指南》中的 [AWS 開發套件及工具版本支援對照表](#)

在 Amazon SQS 中使用 AWS JSON 通訊協定提出查詢 API 請求

本主題說明如何建構 Amazon SQS 端點、提出 POST 請求，以及解譯回應。

Note

AWS 大多數語言變體都支援 JSON 通訊協定。如需支援的語言變體完整清單，請參閱 [Amazon SQS API 中使用的 AWS JSON 通訊協定支援哪些語言？](#)。

建構端點

為了使用 Amazon SQS 佇列，您必須建構端點。如需 Amazon SQS 端點的相關資訊，請參閱 Amazon Web Services 一般參考 中的以下頁面：

- [區域端點](#)
- [Amazon Simple Queue Service 端點和配額](#)

每個 Amazon SQS 端點都是獨立的。例如，若有兩個佇列同樣名為 MyQueue，其中一個佇列具有端點 `sqs.us-east-2.amazonaws.com`，而另一個佇列具有端點 `sqs.eu-west-2.amazonaws.com`，這兩個佇列便不會相互共享任何資料。

以下是某個端點提出請求建立佇列的範例。

```
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

佇列名稱和佇列 URL 區分大小寫。

AUTHPARAMS 的結構取決於 API 請求的簽署。如需詳細資訊，請參閱《Amazon Web Services 一般參考》中的[簽署 AWS API 請求](#)。

提出 POST 請求

Amazon SQS POST 請求是以 HTTP 請求本文的形式傳送查詢參數。

以下是 X-Amz-Target 設定為 AmazonSQS.<operationName> 的 HTTP 標頭範例，以及 Content-Type 設定為 application/x-amz-json-1.0 的 HTTP 標頭。

```
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"
}
```

這個 HTTP POST 請求會將訊息發佈至 Amazon SQS 佇列。

Note

這兩個 HTTP 標頭 X-Amz-Target 和 Content-Type 是必要的。

視用戶端的 HTTP 版本而定，HTTP 用戶端可能會對 HTTP 請求增加其他項目。

解譯 Amazon SQS JSON API 回應

當您將請求傳送至 Amazon SQS 時，它會傳回 JSON 回應與結果。回應結構取決於您使用的 API 動作。

若要了解這些回應的詳細資訊，請參閱：

- Amazon Simple Queue Service API 參考中 [API 動作中的特定 API 動作](#)
- [Amazon SQS AWS JSON 通訊協定FAQs](#)

成功的 JSON 回應結構

如果請求成功，則主要回應元素是 `x-amzn-RequestId`，其中包含請求的通用唯一標識符 (UUID) 以及其他附加回應欄位。例如，下列 [CreateQueue](#) 回應包含 `QueueUrl` 欄位，此欄位又包含建立的佇列的 URL。

```
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"
}
```

JSON 錯誤回應結構

如果請求未成功，Amazon SQS 會傳回主要回應，包括 HTTP 標頭和本文。

在 HTTP 標頭中，`x-amzn-RequestId` 包含請求的 UUID。`x-amzn-query-error` 包含兩項資訊：錯誤類型，以及錯誤是否為生產者或取用者錯誤。

在回應本文中，`"__type"` 指出其他錯誤詳細資訊，而 `Message` 則以可讀格式指出錯誤狀況。

以下為 JSON 格式的錯誤回應範例：

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: 66916324-67ca-54bb-a410-3f567a7a0571
```

```
x-amzn-query-error: AWS.SimpleQueueService.NonExistentQueue;Sender
Content-Length: <PayloadSizeBytes>
Date: <Date>
Content-Type: application/x-amz-json-1.0
{
  "__type": "com.amazonaws.sqs#QueueDoesNotExist",
  "message": "The specified queue does not exist."
}
```

Amazon SQS AWS JSON 通訊協定FAQs

本主題涵蓋搭配 Amazon SQS 使用 AWS JSON 通訊協定的常見問題。

什麼是 AWS JSON 通訊協定，它與現有的 Amazon SQS API 請求和回應有何不同？

JSON 是用於異質系統之間通訊的最廣泛使用和接受的佈線方法之一。Amazon SQS 使用 JSON 在 AWS SDK 用戶端 (例如 Java、Python、Golang、JavaScript) 和 Amazon SQS 伺服器之間進行通訊。Amazon SQS API 操作的 HTTP 請求接受 JSON 格式的輸入。系統會執行 Amazon SQS 操作，而回應會以 JSON 格式傳回 SDK 用戶端。與 AWS 查詢相比，JSON 可更有效率地在用戶端和伺服器之間傳輸資料。

- Amazon SQS AWS JSON 通訊協定充當 Amazon SQS 用戶端和伺服器之間的中介裝置。
- 伺服器不了解建立 Amazon SQS 操作的程式設計語言，但了解 AWS JSON 通訊協定。
- Amazon SQS AWS JSON 通訊協定使用 Amazon SQS 用戶端和伺服器之間的序列化（將物件轉換為 JSON 格式）和還原序列化（將 JSON 格式轉換為物件）。

如何開始使用 Amazon SQS 的 AWS JSON 通訊協定？

若要開始使用最新的 AWS SDK 版本，以實現 Amazon SQS 的更快傳訊，請將您的 AWS SDK 升級到指定的版本或任何後續版本。若要進一步了解 SDK 用戶端，請參閱下表中的「指南」資料欄。

以下是 AWS JSON 通訊協定與 Amazon SQS APIs 搭配使用之不同語言版本的 SDK 版本清單：

語言	SDK 用戶端儲存庫	必要的 SDK 用戶端版本	指南
C++	aws/aws-sdk-cpp	1.11.98	AWS 適用於 C++ 的 SDK

語言	SDK 用戶端儲存庫	必要的 SDK 用戶端版本	指南
Golang 1.x	aws/aws-sdk-go	v1.47.7	AWS 適用於 Go 的 SDK
Golang 2.x	aws/aws-sdk-go-v2	v1.28.0	AWS 適用於 Go V2 的 SDK
Java 1.x	aws/aws-sdk-java	1.12.585	AWS 適用於 Java 的開發套件
Java 2.x	aws/aws-sdk-java-v2	2.21.19	AWS 適用於 Java 的開發套件
JavaScript v2.x	aws/aws-sdk-js	上的 JavaScript AWS	
JavaScript v3.x	aws/aws-sdk-js-v3	v3.447.0	上的 JavaScript AWS
.NET	aws/aws-sdk-net	3.7.681.0	AWS 適用於 .NET 的 SDK
PHP	aws/aws-sdk-php	3.285.2	AWS 適用於 PHP 的 SDK
Python-boto3	boto/boto3	1.28.82	AWS 適用於 Python 的 SDK (Boto3)
Python-botocore	boto/botocore	1.31.82	AWS 適用於 Python 的 SDK (Boto3)
awscli	AWS CLI	1.29.82	AWS命令列介面

語言	SDK 用戶端儲存庫	必要的 SDK 用戶端版本	指南
Ruby	aws/aws-sdk-ruby	1.67.0	AWS 適用於 Ruby 的 SDK

為我的 Amazon SQS 工作負載啟用 JSON 通訊協定有何風險？

如果您使用 AWS SDK 的自訂實作或自訂用戶端和 AWS SDK 的組合來與產生以 AWS 查詢為基礎（又稱為 XML 類型）回應的 Amazon SQS 互動，則可能與 AWS JSON 通訊協定不相容。如果您遇到任何問題，請聯絡 AWS Support。

如果我已經使用最新的 AWS SDK 版本，但我的開放原始碼解決方案不支援 JSON，該怎麼辦？

您必須將 SDK 版本變更為您正在使用的版本之前的版本。[如何開始使用 Amazon SQS 的 AWS JSON 通訊協定？](#) 如需詳細資訊，請參閱。中列出的 AWS SDK 版本[如何開始使用 Amazon SQS 的 AWS JSON 通訊協定？](#) 使用 Amazon SQS APIs 的 JSON 線路通訊協定。如果您將 AWS SDK 變更為舊版，Amazon SQS APIs 將使用 AWS 查詢。

Amazon SQS API 中使用的 AWS JSON 通訊協定支援哪些語言？

Amazon SQS 支援所有正式推出 AWS SDKs 的語言變體 (GA)。目前，我們不支援 Kotlin、Rust 或 Swift。若要深入了解其他語言變體，請參閱[在 AWS 上建立的工具](#)。

Amazon SQS API 中使用的 AWS JSON 通訊協定支援哪些區域

Amazon SQS 在所有可使用 Amazon SQS 的[AWS 區域中](#)支援 AWS JSON 通訊協定。

使用 AWS JSON 通訊協定升級至 Amazon SQS 的指定 AWS SDK 版本時，可以預期哪些延遲改善？

AWS 與 AWS 查詢通訊協定相比，JSON 通訊協定在序列化和還原序列化請求和回應方面更有效率。根據 5 KB 訊息承載 AWS 的效能測試，Amazon SQS 的 JSON 通訊協定可將 end-to-end 訊息處理延遲降低高達 23%，並減少應用程式用戶端 CPU 和記憶體使用量。

AWS 查詢通訊協定是否會棄用？

AWS 查詢通訊協定將繼續受到支援。只要 AWS 軟體開發套件版本已設定 Amazon SQS 的 JSON 通訊協定入門中所列的任何先前版本，您就可以繼續使用 AWS 查詢通訊協定。 [AWS Amazon SQS](#)

在哪裡可找到 AWS JSON 通訊協定的更多資訊？

您可以在 Smithy 文件中的 [AWS JSON 1.0 通訊協定](#) 中找到有關 JSON 通訊協定的更多資訊。如需使用 AWS JSON 通訊協定的 Amazon SQS API 請求的詳細資訊，請參閱 [在 Amazon SQS 中使用 AWS JSON 通訊協定提出查詢 API 請求](#)。

在 Amazon SQS 中使用查詢通訊協定提出 AWS 查詢 API 請求

本主題說明如何建構 Amazon SQS 端點、提出 GET 和 POST 請求，以及解譯回應。

建構端點

為了能夠使用 Amazon SQS 佇列，您必須建構端點。如需 Amazon SQS 端點的相關資訊，請參閱 Amazon Web Services 一般參考 中的以下頁面：

- [區域端點](#)
- [Amazon Simple Queue Service 端點和配額](#)

每個 Amazon SQS 端點都是獨立的。例如，若有兩個佇列同樣名為 MyQueue，其中一個佇列具有端點 `sqs.us-east-2.amazonaws.com`，而另一個佇列具有端點 `sqs.eu-west-2.amazonaws.com`，這兩個佇列便不會相互共享任何資料。

以下是某個端點提出請求建立佇列的範例。

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

Note

佇列名稱和佇列 URL 區分大小寫。

AUTHPARAMS 的結構取決於 API 請求的簽署。如需詳細資訊，請參閱《Amazon Web Services 一般參考》中的[簽署 AWS API 請求](#)。

提出 GET 請求

Amazon SQS GET 請求的架構是由以下各部分組成的 URL：

- 端點 - 請求所作用的資源 ([佇列名稱和 URL](#))，例如：`https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`
- 動作 - 要在端點上執行的[動作](#)。端點與動作之間用問號 (?) 隔開，例如：`?Action=SendMessage&MessageBody=Your%20Message%20Text`
- 參數 - 任何請求的參數。每個參數以 ampersand (&) 分隔，例如：`&Version=2012-11-05&AUTHPARAMS`

以下是提出 GET 請求將訊息傳送至 Amazon SQS 佇列的範例。

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

Note

佇列名稱和佇列 URL 區分大小寫。

由於 GET 請求是 URL，所有參數值必須以 URL 編碼處理。由於 URL 不允許空格，每個空格都按 URL 編碼處理為 %20 為便於閱讀，範例的其餘部分並未以 URL 編碼處理。

提出 POST 請求

Amazon SQS POST 請求是以 HTTP 請求本文的形式傳送查詢參數。

以下是 Content-Type 設為 application/x-www-form-urlencoded 的 HTTP 標頭範例。

```
POST /123456789012/MyQueue HTTP/1.1
Host: sqs.us-east-2.amazonaws.com
Content-Type: application/x-www-form-urlencoded
```

標頭後面接著的是 [form-urlencoded](#) GET 請求，其將傳送訊息至 Amazon SQS 佇列。每個參數之間用 & 符號隔開。

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Expires=2020-10-15T12%3A00%3A00Z
&Version=2012-11-05
&AUTHPARAMS
```

Note

僅需要有 Content-Type HTTP 標頭。*AUTHPARAMS* 的作用與 GET 請求的情形相同。視用戶端的 HTTP 版本而定，HTTP 用戶端可能會對 HTTP 請求增加其他項目。

解譯 Amazon SQS XML API 回應

當您傳送請求至 Amazon SQS 時，它會傳回包含請求結果的 XML 回應。若要了解這些回應的結構和詳細資訊，請參閱 Amazon Simple Queue Service API 參考中的特定 [API 動作](#)。

成功的 XML 回應結構

若請求成功，主要回應元素將以請求的動作命名並加上 Response (例如 *ActionNameResponse*)。

此元素內含以下子元素：

- ***ActionNameResult*** – 內含專屬於動作的元素。例如，CreateQueueResult 元素內含 QueueUrl 元素，後者又內含建立的佇列所在 URL。
- **ResponseMetadata** – 包含 RequestId 其依序包含請求的通用唯一識別碼 (UUID)。

以下為 XML 格式的成功回應範例：

```
<CreateQueueResponse
  xmlns=https://sqs.us-east-2.amazonaws.com/doc/2012-11-05/
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:type=CreateQueueResponse>
  <CreateQueueResult>
    <QueueUrl>https://sqs.us-east-2.amazonaws.com/770098461991/queue2</QueueUrl>
  </CreateQueueResult>
  <ResponseMetadata>
```

```
<RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
</ResponseMetadata>
</CreateQueueResponse>
```

XML 錯誤回應結構

若請求不成功，Amazon SQS 一律會傳回主要回應元素 `ErrorResponse`。此元素內含 `Error` 元素和 `RequestId` 元素。

`Error` 元素內含以下子元素：

- **Type** – 指定此錯誤是來自生產者還是消費者。
- **Code** – 指定錯誤的類型。
- **Message** – 指定易讀格式的錯誤情況。
- **Detail** – (選用) 指定關於錯誤的額外詳細資訊。

`RequestId` 元素內含請求的 UUID。

以下為 XML 格式的錯誤回應範例：

```
<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-4c88daeeea257</RequestId>
</ErrorResponse>
```

驗證 Amazon SQS 的請求

身分驗證是辨識與確認傳送請求的當事方身分的過程。在身分驗證的第一階段，會 AWS 驗證生產者的身分，以及生產者是否已[註冊使用 AWS](#)（如需詳細資訊，請參閱[步驟 1：建立 AWS 帳戶和 IAM 使用者](#)）。接下來，依下列程序 AWS 刪除：

1. 生產者 (發送者) 必須取得必要的登入資料。

2. 生產者將請求和登入資料發送給消費者 (接收者)。
3. 消費者使用登入資料來確認是否為生產者發送請求。
4. 以下其中一種情況將發生：
 - 若身分驗證成功，消費者便會繼續進行流程。
 - 若身分驗證失敗，消費者便會拒絕請求並傳回錯誤。

使用 HMAC-SHA 的基本身分驗證流程

使用查詢 API 存取 Amazon SQS 時，必須提供下列項目以對您的請求進行身分驗證：

- 識別您的 AWS 存取金鑰 ID AWS 帳戶，其 AWS 使用來查詢您的私密存取金鑰。
- HMAC-SHA 請求簽章，使用您的私密存取金鑰（只有您已知的共用秘密 AWS，如需詳細資訊，請參閱 [RFC2104](#)）計算。[AWS 開發套件](#)會處理簽署程序，但若您透過 HTTP 或 HTTPS 提交查詢請求，則每一次查詢請求都必須附上簽章。

1. 衍生 Signature 第 4 版簽署金鑰。如需詳細資訊，請參閱[使用 Java 衍生簽署金鑰](#)。

Note

Amazon SQS 支援 Signature 第 4 版，此版本比起之前版本提供最佳的 SHA256 型安全性與效能。若您建立的新應用程式會用到 Amazon SQS，請使用 Signature 第 4 版。

2. 對請求簽章進行 Base64 編碼。以下範例 Java 程式碼執行此作業：

```
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);
    }
}
```

- 請求的時間戳記 (或過期時間)。在請求中使用的時間戳記必須是 `dateTime` 物件，有完整的日期，包含小時、分鐘、秒鐘。例如：2007-01-31T23:59:59Z。儘管並非必要，但建議您使用國際標準時間 (格林威治標準時間) 時區提供此物件。

Note

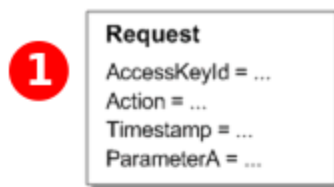
請確定您的伺服器時間設定正確。如果您指定時間戳記（而不是過期），請求會在指定時間後 15 分鐘自動過期 (AWS 不會處理時間戳記早於伺服器上目前時間 AWS 15 分鐘以上的請求)。

如果您是使用 .NET，切勿傳送過於精確的時間戳記 (因為該平台對於如何去除多餘的時間準度有不同的解釋)。在此情況下，應手動建構精準度不超過一毫秒的 `dateTime` 物件。

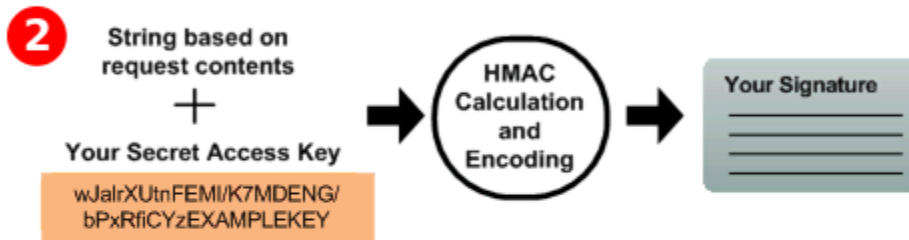
第 1 部分：來自使用者的請求

以下是使用 HMAC-SHA AWS 請求簽章驗證請求時必須遵循的程序。

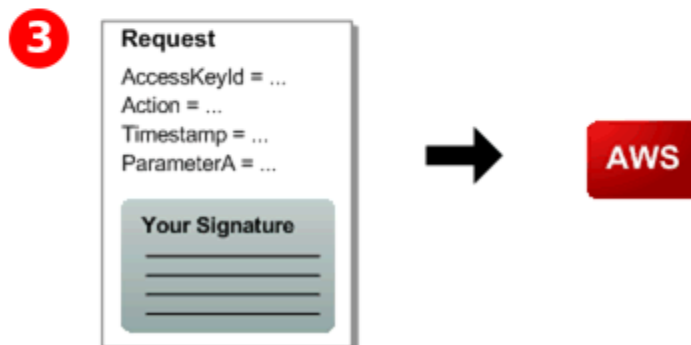
Create a request:



Create an HMAC-SHA signature:



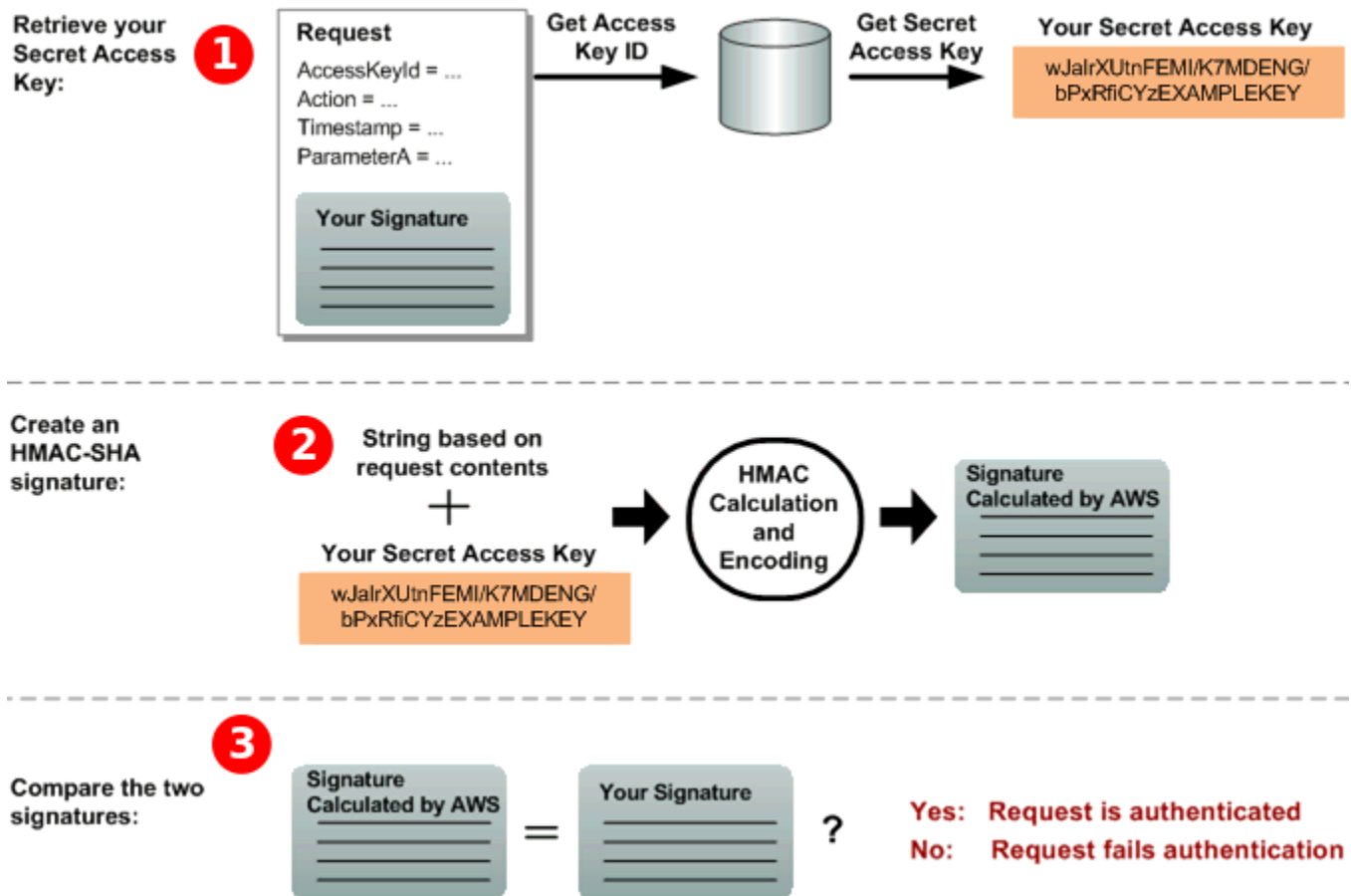
Send the request and signature to AWS:



1. 建構 請求 AWS。
2. 使用您的私密存取金鑰，計算金鑰式雜湊訊息身分驗證碼 (HMAC-SHA) 簽章。
3. 在請求中包含簽章和您的存取金鑰 ID，然後將請求傳送至 AWS。

第 2 部分：來自 的回應 AWS

AWS 會開始下列程序以回應。



1. AWS 使用存取金鑰 ID 來查詢您的私密存取金鑰。
2. AWS 會從請求資料和私密存取金鑰產生簽章，使用您在請求中傳送的簽章時所使用的相同演算法。
3. 以下其中一種情況將發生：
 - 如果 AWS 產生的簽章符合您在請求中傳送的簽章，會將請求 AWS 視為真實。
 - 如果比較失敗，則會捨棄請求，並 AWS 傳回錯誤。

Amazon SQS 批次動作

Amazon SQS 提供批次動作，可協助您降低成本，並使用單一動作處理最多 10 則訊息。這些批次動作包括：

- [SendMessageBatch](#)
- [DeleteMessageBatch](#)
- [ChangeMessageVisibilityBatch](#)

您可以使用批次動作，在單一 API 呼叫中執行多個操作，這有助於最佳化效能並降低成本。您可以使用查詢 API 或任何支援 Amazon SQS 批次動作的 AWS SDK 來利用批次功能。

重要詳細資訊

- 訊息大小限制：單一SendMessageBatch呼叫中傳送的所有訊息總大小不得超過 262,144 位元組 (256 KiB)。
- 許可：您無法明確設定 SendMessageBatch、DeleteMessageBatch或 的許可ChangeMessageVisibilityBatch。反之，為 SendMessage、DeleteMessage或 設定對應批次版本動作的ChangeMessageVisibility許可。
- 主控台支援：Amazon SQS 主控台不支援批次動作。您必須使用查詢 API 或 AWS SDK 來執行批次操作。

批次訊息動作

若要進一步最佳化成本和效率，請考慮下列批次處理訊息動作的最佳實務：

- 批次 API 動作：使用 [Amazon SQS 批次 API 動作](#)來傳送、接收和刪除訊息，以及使用單一動作變更多個訊息的訊息可見性逾時。這可減少 API 呼叫的數量和相關聯的成本。
- 用戶端緩衝和長輪詢：使用長輪詢搭配 隨附的[緩衝非同步用戶端](#)，將用戶端緩衝與請求批次結合 適用於 Java 的 AWS SDK。此方法有助於將請求數量降至最低，並最佳化大量訊息的處理。

Note

Amazon SQS 緩衝非同步用戶端目前不支援 FIFO 佇列。

使用 Amazon SQS 啟用用戶端緩衝和請求批次處理

[適用於 Java 的 AWS SDK](#) 包含可存取 Amazon SQS 的 `AmazonSQSBufferedAsyncClient`。此用戶端允許使用用戶端緩衝進行簡單的請求批次處理。從用戶端發出的呼叫會先緩衝，然後以批次請求的形式傳送至 Amazon SQS。

用戶端緩衝功能最多可緩衝處理 10 個請求並以批次請求的形式傳送，降低了使用 Amazon SQS 的成本且能減少傳送的請求數。`AmazonSQSBufferedAsyncClient` 會對同步和非同步的呼叫進行緩衝處理。批次處理的請求以及對[長輪詢](#)的支援也有助於提高傳輸量。如需詳細資訊，請參閱 [使用 Amazon SQS 使用水平擴展和動作批次來增加輸送量](#)。

由於 `AmazonSQSBufferedAsyncClient` 實作的介面與 `AmazonSQSAsyncClient` 相同，從 `AmazonSQSAsyncClient` 移轉到 `AmazonSQSBufferedAsyncClient` 通常只需對既有的程式碼進行最少的更動。

Note

Amazon SQS 緩衝非同步用戶端目前不支援 FIFO 佇列。

使用 AmazonSQSBufferedAsyncClient

開始之前，請完成 [設定 Amazon SQS](#) 中的步驟。

AWS 適用於 Java 的 SDK 1.x

對於適用於 Java 的 AWS SDK 1.x，您可以 `AmazonSQSBufferedAsyncClient` 根據下列範例建立新的：

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

新的 `AmazonSQSBufferedAsyncClient` 建立完成後，您可以使用它來傳送多個請求至 Amazon SQS (如同使用 `AmazonSQSAsyncClient`)，例如：

```
final CreateQueueRequest createRequest = new
    CreateQueueRequest().withQueueName("MyQueue");
```

```
final CreateQueueResult res = bufferedSqs.createQueue(createRequest);

final SendMessageRequest request = new SendMessageRequest();
final String body = "Your message text" + System.currentTimeMillis();
request.setRequestBody( body );
request.setQueueUrl(res.getQueueUrl());

final Future<SendMessageResult> sendResult = bufferedSqs.sendMessageAsync(request);

final ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
final ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

設定 AmazonSQSBufferedAsyncClient

AmazonSQSBufferedAsyncClient 已預先設為適合大多數使用案例的組態。您可以進一步設定 AmazonSQSBufferedAsyncClient，例如：

1. 使用必要的組態參數，建立 QueueBufferConfig 類別的執行個體。
2. 對 AmazonSQSBufferedAsyncClient 建構函式提供此執行個體。

```
// 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 組態參數

參數	預設值	描述
longPoll	true	若 longPoll 設為 true，AmazonSQS BufferedAsyncClient

參數	預設值	描述
		在消費訊息時會嘗試使用長輪詢。
<code>longPollWaitTimeoutSeconds</code>	20 秒	在傳回空的接收結果前，<code>ReceiveMessage</code> 呼叫留置於伺服器上等待訊息出現在佇列中的時間上限 (單位為秒)。  Note 停用長輪詢時，此設定沒有作用。
<code>maxBatchOpenMs</code>	200 毫秒	傳出呼叫對於其他也要批次處理同類型訊息的呼叫稍作等待的時間上限 (單位為毫秒)。 此設定值越高，執行相同工作量所需的批次數就越少 (但是，批次的第一次呼叫就需要花越長時間等待)。 若將此參數設為 0，提交的請求便不會等待其他請求，實際上即是停用了批次處理功能。

參數	預設值	描述
maxBatchSize	每批次 10 個請求	單次請求同時批次處理的訊息數上限。此設定值越高，執行相同請求數所需的批次數量就越少。  Note Amazon SQS 允許的上限值為每批次 10 個請求。
maxBatchSizeBytes	256 KiB	用戶端嘗試傳送至 Amazon SQS 之訊息批次的大小上限，單位為位元組。  Note 256 KiB 是 Amazon SQS 允許的最大值。

參數	預設值	描述
<code>maxDoneReceiveBatches</code>	10 個批次	AmazonSQSBufferedAsyncClient 在用戶端預取並存放的接收批次數上限。 此設定值越高，則不必呼叫 Amazon SQS 就能滿足越多的接收請求 (但是，預取的訊息越多，停留在緩衝區內的時間就越久，而導致訊息的可見性逾時會到期)。  Note 0 表示已停用所有訊息預先擷取，且訊息只會隨需使用。
<code>maxInflightOutboundBatches</code>	5 個批次	可同時處理的活動中傳出批次的上限。 此設定值越高，傳出批次的傳送速度越快 (受限於 CPU 或頻寬等的配額)，且 AmazonSQSBufferedAsyncClient 所耗用的執行緒越多。

參數	預設值	描述
<code>maxInflightReceive Batches</code>	10 個批次	可同時處理的活動中接收批次的上限。 此設定值越高，可接收的訊息數越多 (受限於 CPU 或頻寬等的配額)，且 AmazonSQS <code>BufferedAsyncClient</code> 所耗用的執行緒越多。  Note 0 表示已停用所有訊息預先擷取，且訊息只會隨需使用。
<code>visibilityTimeoutSeconds</code>	-1	若將此參數設為非零正值，其設定的可見性逾時值就會覆寫消費訊息的佇列所設的可見性逾時值。  Note -1 表示將選擇佇列預設的設定。 可見性逾時不可設為 0。

AWS 適用於 Java 的 SDK 2.x

對於適用於 Java 的 AWS SDK 2.x，您可以 `SqsAsyncBatchManager` 根據下列範例建立新的：

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

新的 `SqsAsyncBatchManager` 建立完成後，您可以使用它來傳送多個請求至 Amazon SQS (如同使用 `SqsAsyncClient`)，例如：

```
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);
```

設定 SqsAsyncBatchManager

SqsAsyncBatchManager 已預先設為適合大多數使用案例的組態。您可以進一步設定 SqsAsyncBatchManager，例如：

透過 建立自訂組態 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 參數

參數	預設值	描述
maxBatchSize	每批次 10 個請求	單次請求同時批次處理的訊息數上限。此設定值越高，執行相同請求數所需的批次數量就越少。
 Note Amazon SQS 的允許值上限為每個批次 10 個請求。		
sendRequestFrequency	200 毫秒	傳出呼叫對於其他也要批次處理同類型訊息的呼叫稍作等待的時間上限 (單位為毫秒)。

參數	預設值	描述
		此設定值越高，執行相同工作量所需的批次數就越少 (但是，批次的第一次呼叫就需要花越長時間等待)。 若將此參數設為 0，提交的請求便不會等待其他請求，實際上即是停用了批次處理功能。
<code>receiveMessageVisibilityTimeout</code>	-1	若將此參數設為非零正值，其設定的可見性逾時值就會覆寫消費訊息的佇列所設的可見性逾時值。  Note 1 表示將選擇佇列預設的設定。可見性逾時不可設為 0。
<code>receiveMessageMinWaitDuration</code>	50 毫秒	<code>receiveMessage</code> 呼叫等待擷取可用訊息的最短時間 (以毫秒為單位)。設定越高，執行相同數量的請求所需的批次就越少。

使用 Amazon SQS 使用水平擴展和動作批次來增加輸送量

Amazon SQS 支援高輸送量傳訊。如需輸送量限制的詳細資訊，請參閱 [Amazon SQS 訊息配額](#)。

若要最大化輸送量：

- 透過新增更多每個執行個體，水平**擴展**生產者和消費者。
- 使用**動作批次**處理，在單一請求中傳送或接收多則訊息，減少 API 呼叫的額外負荷。

水平擴展

由於您是透過 HTTP 請求-回應協定來存取 Amazon SQS，請求延遲 (自開始請求至收到回應的時間間隔) 會限制您透過單一連線使用單一執行緒所能達到的輸送量。例如，若 Amazon EC2 型的用戶端到 Amazon SQS 的延遲度在相同區域的平均值為 20 毫秒，透過單一連線使用單一執行緒可達到的最大輸送量平均為 50 TPS。

水平擴展意指增加訊息生產者 (提出 [SendMessage](#) 請求) 及消費者 (提出 [ReceiveMessage](#) 和 [DeleteMessage](#) 請求) 的數目，藉以提高佇列整體的傳輸量。水平擴展的方式有三種：

- 增加每個用戶端的執行緒數目
- 增加更多用戶端
- 增加每個用戶端的執行緒數目並增加更多用戶端

增加更多用戶端之後，基本上佇列傳輸量會獲得線性增益。例如，若用戶端數目加倍，傳輸量也會翻倍。

動作批次處理

批次處理可在每次往返服務期間處理更多工作 (例如，透過單次 [SendMessageBatch](#) 請求傳送多則訊息)。Amazon SQS 批次動作包括 [SendMessageBatch](#)、[DeleteMessageBatch](#) 和 [ChangeMessageVisibilityBatch](#)。若要在不變動生產者或消費者的情況下利用批次處理功能，您可以使用 [Amazon SQS 緩衝非同步用戶端](#)。

Note

由於 [ReceiveMessage](#) 一次可處理 10 則訊息，也就不必有 [ReceiveMessageBatch](#) 動作。

批次處理會將批次動作的延遲度分散到批次請求中的多則訊息，而非接受單則訊息 (例如 [SendMessage](#) 請求) 的所有延遲。由於每次往返所執行的工作較多，批次請求可以有效率地運用執行緒和連線，進而提高傳輸量。

您可以結合批次處理與水平擴展來提供傳輸量，所需的執行緒、連線和請求的數目會比單獨的訊息請求所需的量更少。您可以使用批次的 Amazon SQS 動作一次傳送、接收、刪除多達 10 則訊息。由於 Amazon SQS 收費是以請求數為準，批次處理可大幅降低您的成本。

批次處理可能會給您的應用程式帶來一些複雜性 (例如，您的應用程式必須將訊息累積起來後再傳送出去，或者有時候必須花更久的時間等待回應)。但是，批次處理在以下情況仍然頗具效益：

- 應用程式會在短時間內產生許多訊息，所以絕不會延遲太久。
- 訊息消費者自行從佇列擷取訊息，有別於典型的訊息生產者需要傳送訊息來回應其無法控制的事件。

Important

即使批次中的個別訊息失敗，批次請求也可能會成功。提出批次請求之後，務必確認個別訊息是否出現失敗情況，必要時重試動作。

單次操作和批次請求實際可行的 Java 範例

先決條件

新增 `aws-java-sdk-sqs.jar`、`aws-java-sdk-ec2.jar`，以及 `commons-logging.jar` 套件到您的 Java 建置類別路徑。以下範例顯示 Maven 專案 `pom.xml` 檔案中的此等依存關係。

```
<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

以下 Java 程式碼範例實作單純的生產者-消費者模式。主要的執行緒會產生數個生產者和消費者執行緒，負責在指定時間處理 1 KB 的訊息。此範例包括提出單次操作請求的生產者和消費者，以及提出批次請求的生產者和消費者。

```
/*
 * 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.
        }
    }
}
```

監控範本執行階段的數量指標

Amazon SQS 會自動為傳送、接收及刪除的訊息產生數量指標。您可以透過佇列的監控索引標籤或 [CloudWatch 主控台](#) 存取這類指標和其他項目。

Note

可能要在佇列開始執行達 15 分鐘後方可取得指標。

搭配 AWS SDK 使用 Amazon SQS

AWS 軟體開發套件 SDKs) 適用於許多熱門的程式設計語言。每個 SDK 都提供 API、程式碼範例和說明文件，讓開發人員能夠更輕鬆地以偏好的語言建置應用程式。

SDK 文件	代碼範例
適用於 C++ 的 AWS SDK	適用於 C++ 的 AWS SDK 程式碼範例
AWS CLI	AWS CLI 程式碼範例
適用於 Go 的 AWS SDK	適用於 Go 的 AWS SDK 程式碼範例
適用於 Java 的 AWS SDK	適用於 Java 的 AWS SDK 程式碼範例
適用於 JavaScript 的 AWS SDK	適用於 JavaScript 的 AWS SDK 程式碼範例
適用於 Kotlin 的 AWS SDK	適用於 Kotlin 的 AWS SDK 程式碼範例
適用於 .NET 的 AWS SDK	適用於 .NET 的 AWS SDK 程式碼範例
適用於 PHP 的 AWS SDK	適用於 PHP 的 AWS SDK 程式碼範例
AWS Tools for PowerShell	適用於 PowerShell 的工具程式碼範例
適用於 Python (Boto3) 的 AWS SDK	適用於 Python (Boto3) 的 AWS SDK 程式碼範例
適用於 Ruby 的 AWS SDK	適用於 Ruby 的 AWS SDK 程式碼範例
適用於 Rust 的 AWS SDK	適用於 Rust 的 AWS SDK 程式碼範例
適用於 SAP ABAP 的 AWS SDK	適用於 SAP ABAP 的 AWS SDK 程式碼範例
適用於 Swift 的 AWS SDK	適用於 Swift 的 AWS SDK 程式碼範例

可用性範例

找不到所需的內容嗎？請使用本頁面底部的提供意見回饋連結申請程式碼範例。

搭配 Amazon SQS 使用 JMS

Amazon SQS Java 訊息程式庫是適用於 Amazon SQS 的 Java 訊息服務 (JMS) 介面，可讓您在已經使用 JMS 的應用程式中充分運用 Amazon SQS。此介面可讓您盡可能不變更程式碼就能使用 Amazon SQS 做為 JMS 的供應者。搭配適用於 Java 的 AWS SDK，Amazon SQS Java 訊息程式庫可建立 JMS 連線和工作階段，以及傳送和接收 Amazon SQS 佇列訊息的生產者和消費者。

此程式庫支援符合 [JMS 1.1 規格](#) 的佇列訊息傳送和接收 (JMS 點對點模式)。此程式庫支援同步將文字、位元組、物件訊息傳送至 Amazon SQS 佇列。此程式庫也支援同步或非同步接收物件。

如需支援 JMS 1.1 規格的 Amazon SQS Java 訊息程式庫功能資訊，請參閱 [Amazon SQS 支援的 JMS 1.1 實作](#) 以及 [Amazon SQS 常見問答集](#)。

使用 JMS 和 Amazon SQS 的先決條件

開始之前，您必須準備好以下事項：

- 適用於 Java 的開發套件

有兩種方式可將適用於 Java 的開發套件納入您的專案：

- 下載並安裝適用於 Java 的開發套件。
- 使用 Maven 以獲得 Amazon SQS Java 訊息程式庫。

Note

適用於 Java 的開發套件會為相依性包含在內。

[適用於 Java 的開發套件](#) 和適用於 Java 的 Amazon SQS 擴充用戶端程式庫需要 J2SE 開發套件 8.0 或更新版本。

如需關於下載適用於 Java 的開發套件的資訊，請參閱 [適用於 Java 的開發套件](#)。

- Amazon SQS Java 訊息程式庫

若您不使用 Maven，您必須將 `amazon-sqs-java-messaging-lib.jar` 套件新增至 Java 類別路徑。如需下載程式庫的資訊，請參閱 [Amazon SQS Java 訊息程式庫](#)。

Note

Amazon SQS Java 訊息程式庫包含對 [Maven](#) 及 [Spring Framework](#) 的支援。如需使用 Maven、Spring Framework、Amazon SQS Java 訊息程式庫的範本程式碼，請參閱 [搭配 Amazon SQS 標準佇列使用 JMS 的工作 Java 範例](#)。

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
  <version>1.0.4</version>
  <type>jar</type>
</dependency>
```

- Amazon SQS 佇列

使用 Amazon SQS、CreateQueueAPI AWS Management Console 或包含在 Amazon SQS Java Messaging Library 中的包裝 Amazon SQS 用戶端來建立佇列。

- 如需使用 AWS Management Console 或 CreateQueue API 以 Amazon SQS 建立佇列的詳細資訊，請參閱[建立佇列](#)。
- 如需使用 Amazon SQS Java 訊息程式庫的詳細資訊，請參閱 [使用 Amazon SQS Java Messaging Library](#)。

使用 Amazon SQS Java Messaging Library

若要開始使用 Java 訊息服務 (JMS) 搭配 Amazon SQS，請使用本節中的程式碼範例。以下章節會示範如何建立 JMS 連線和工作階段，以及如何傳送和接收訊息。

包含在 Amazon SQS Java 訊息程式庫的包裝 Amazon SQS 用戶端物件會檢查 Amazon SQS 佇列是否存在。如果佇列不存在，用戶端便會建立佇列。

建立 JMS 連線

開始之前，請參閱 中的先決條件[使用 JMS 和 Amazon SQS 的先決條件](#)。

1. 請建立連線工廠並對工廠呼叫 createConnection 方法。

```
// Create a new connection factory with all defaults (credentials and region) set
automatically
```

```
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(  
    new ProviderConfiguration(),  
    AmazonSQSClientBuilder.defaultClient()  
);  
  
// Create the connection.  
SQSConnection connection = connectionFactory.createConnection();
```

`SQSConnection` 類別會將 `javax.jms.Connection` 延伸。搭配 JMS 標準連線方法，`SQSConnection` 會提供額外的方法，例如 `getAmazonSQSClient` 和 `getWrappedAmazonSQSClient`。兩種方法皆能執行不包含在 JMS 規格內的管理操作，例如建立新佇列。不過，`getWrappedAmazonSQSClient` 方法還可提供目前連線使用的 Amazon SQS 用戶端的包裝版本。包裝函式會將每個例外狀況從用戶端的轉換為 `JMSEException`，讓如此可更輕鬆地由預期 `JMSEException` 事件的既有程式碼使用。

2. 您可以使用 `getAmazonSQSClient` 及 `getWrappedAmazonSQSClient` 傳回的用戶端物件來執行未包含在 JMS 規格內的管理操作 (例如建立 Amazon SQS 佇列)。

若有會預期 JMS 例外狀況出現的既有程式碼，則應使用 `getWrappedAmazonSQSClient`：

- 若您使用 `getWrappedAmazonSQSClient`，傳回的用戶端物件會將所有例外狀況轉換為 JMS 例外狀況。
- 若您使用 `getAmazonSQSClient`，例外狀況全為 Amazon SQS 例外狀況。

建立 Amazon SQS 佇列

包裝用戶端物件會檢查 Amazon SQS 佇列是否存在。

如果佇列不存在，用戶端便會建立佇列。如果佇列存在，該函數不會傳回任何內容。如需詳細資訊，請參閱視需要建立佇列章節的 [TextMessageSender.java](#) 範例。

建立標準佇列

```
// 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");  
}
```

若要建立 FIFO 佇列

```
// 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

FIFO 佇列名稱結尾必須是 `.fifo` 尾碼。

如需 `ContentBasedDeduplication` 屬性的相關資訊，請參閱[Amazon SQS 中的精確一次處理](#)。

同步傳送訊息

1. 當連線和底層的 Amazon SQS 佇列準備好之時，請以 `AUTO_ACKNOWLEDGE` 模式建立非交易的 JMS 工作階段。

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. 若要傳送文字訊息到佇列，請建立 JMS 佇列身分和訊息生產者。

```
// 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. 請建立文字訊息，並傳送到佇列。
 - 若要傳送訊息到標準佇列，則無需設定任何額外參數。

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

- 若要傳送訊息到 FIFO 佇列，則必須設定訊息群組 ID。您也可以設定訊息重複資料刪除 ID。如需詳細資訊，請參閱 [Amazon SQS FIFO 佇列金鑰術語](#)。

```
// 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"));
```

同步接收訊息

1. 若要接收訊息，請建立相同佇列的消費者並呼叫 start 方法。

您可以隨時在連線上呼叫 start 方法。不過，消費者在您呼叫前不會接收訊息。

```
// Create a consumer for the 'MyQueue'
MessageConsumer consumer = session.createConsumer(queue);
// Start receiving incoming messages
connection.start();
```

2. 請在消費者上呼叫 receive 方法呼叫，將逾時設定為 1 秒，然後將接收訊息的內容列印出來。

- 從標準佇列接收訊息後，即可存取訊息內容。

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

- 從 FIFO 佇列接收訊息後，即可存取訊息的內容和其他 FIFO 特定的訊息屬性，例如訊息群組 ID、訊息重複資料刪除 ID 和序號。如需詳細資訊，請參閱 [Amazon SQS FIFO 佇列金鑰術語](#)。

```
// 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. 關閉連線和工作階段。

```
// Close the connection (and the session).
connection.close();
```

輸出結果類似如下：

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Note

您可以使用 Spring Framework 來初始化這些物件。

如需其他資訊，請參閱 `SpringExampleConfiguration.xml`、`SpringExample.java` 和 `ExampleConfiguration.java` 節的 `ExampleCommon.java`、[搭配 Amazon SQS 標準佇列使用 JMS 的工作 Java 範例](#) 的其他說明類別。

如需傳送和接收物件的完整範例，請參閱 [TextMessageSender.java](#) 和 [SyncMessageReceiver.java](#)。

異步接收訊息

在 [使用 Amazon SQS Java Messaging Library](#) 的範例中，訊息會傳送到 `MyQueue` 並同步接收。

以下範例顯示的是如何透過接聽程式異步接收訊息。

1. 實作 `MessageListener` 介面。

```
class MyListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to
            screen.
            System.out.println("Received: " + ((TextMessage) message).getText());
        } catch (JMSEException e) {
            e.printStackTrace();
        }
    }
}
```

收到訊息時，即會呼叫 `MessageListener` 介面的 `onMessage` 方法。在此接聽程式的實作中，會將儲存在訊息中的文字列印出來。

2. 消費者端不會明確地呼叫 `receive` 方法，而是將消費者的訊息接聽程式設定為 `MyListener` 實作的執行個體。主執行緒會等待一秒。

```
// 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);
```

接下來的步驟與 [使用 Amazon SQS Java Messaging Library](#) 範例相同。如需異步消費者的完整資訊，請參閱 `AsyncMessageReceiver.java` 的 [搭配 Amazon SQS 標準佇列使用 JMS 的工作 Java 範例](#)。

此範例的輸出結果類似如下：

```
JMS Message ID:8example-588b-44e5-bbcbf-d816example2
Received: Hello World!
```

使用用戶端認可模式

[使用 Amazon SQS Java Messaging Library](#) 中的範例使用 `AUTO_ACKNOWLEDGE` 模式，所有接收到的訊息會自動獲得認可 (因此會從底層的 Amazon SQS 佇列刪除)。

1. 若要明確在處理訊息後予以認可，則必須建立使用 `CLIENT_ACKNOWLEDGE` 模式的工作階段。

```
// Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

2. 收到訊息後，會顯示訊息再明確予以認可。

```
// 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

在此模式中，當訊息受到認可，所有在此之前收到的訊息也會預設為認可。舉例而言，若收到 10 則訊息，而只有第 10 則訊息受到認可 (按訊息接收順序)，則之前的九則訊息也會受到認可。

接下來的步驟與 [使用 Amazon SQS Java Messaging Library](#) 範例相同。如需異步消費者用戶端認可模式的完整範例，請參閱 `SyncMessageReceiverClientAcknowledge.java` 的 [搭配 Amazon SQS 標準佇列使用 JMS 的工作 Java 範例](#)。

此範例的輸出結果類似如下：

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

使用無順序認可模式

當使用 `CLIENT_ACKNOWLEDGE` 模式時，受到明確認可的訊息之前所有接收到的訊息均會自動受到認可。如需詳細資訊，請參閱 [使用用戶端認可模式](#)。

Amazon SQS Java 訊息程式庫提供了另一種認可模式。使用 `UNORDERED_ACKNOWLEDGE` 模式時，所有接收的訊息必須明確受到用戶端個別認可，無論接收順序。若要這樣做，請建立 `UNORDERED_ACKNOWLEDGE` 模式的工作階段。

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

其他步驟與 [使用用戶端認可模式](#) 範例相同。如需同步消費者 `UNORDERED_ACKNOWLEDGE` 模式的完整資訊，請參閱 `SyncMessageReceiverUnorderedAcknowledge.java`。

此範例的輸出結果類似如下：

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

搭配其他 Amazon SQS 用戶端使用 Java Message Service

搭配 AWS SDK 使用 Amazon SQS Java Message Service (JMS) 用戶端會將 Amazon SQS 訊息大小限制為 256 KB。不過，您可以使用任何 Amazon SQS 用戶端建立 JMS 提供者。例如，您可以使用 JMS Client 搭配適用於 Java 的 Amazon SQS 擴充用戶端程式庫來傳送 Amazon SQS 訊息，其中包含對 Amazon S3 中的訊息承載 (最多 2 GB) 的參考。如需詳細資訊，請參閱[使用 Java 和 Amazon S3 管理大型 Amazon SQS 訊息 Amazon S3](#)。

下列 Java 程式碼範例會為擴充用戶端程式庫建立 JMS 提供者。

在測試此範例[使用 JMS 和 Amazon SQS 的先決條件](#)之前，請參閱 中的先決條件。

```
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);
```

以下 Java 範本程式碼會建立連線工廠：

```
// 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();
```

搭配 Amazon SQS 標準佇列使用 JMS 的工作 Java 範例

下列程式碼範例示範如何搭配 Amazon SQS 標準佇列使用 Java 訊息服務 (JMS)。如需使用 FIFO 佇列的相關資訊，請參閱 [若要建立 FIFO 佇列](#)、[同步傳送訊息](#)、[同步接收訊息](#)。(對於標準和 FIFO 佇列，同步接收訊息是相同的。但是，FIFO 佇列中的訊息包含更多屬性。)

在測試下列範例[使用 JMS 和 Amazon SQS 的先決條件](#)之前，請參閱 中的先決條件。

ExampleConfiguration.java

下列 Java SDK v 1.x 程式碼範例會設定預設佇列名稱、區域，以及要與其他 Java 範例搭配使用的登入資料。

```
/*
 * 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

下列 Java 程式碼範例會先建立文字訊息生產者。

```
/*  
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
 *  
 * Licensed under the Apache License, Version 2.0 (the "License").  
 * You may not use this file except in compliance with the License.  
 * A copy of the License is located at  
 *  
 * https://aws.amazon.com/apache2.0  
 *  
 * or in the "license" file accompanying this file. This file is distributed  
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either  
 * express or implied. See the License for the specific language governing  
 * permissions and limitations under the License.  
 */  
  
public class TextMessageSender {  
    public static void main(String args[]) throws JMSEException {  
        ExampleConfiguration config =  
        ExampleConfiguration.parseConfig("TextMessageSender", args);  
  
        ExampleCommon.setupLogging();  
  
        // Create the connection factory based on the config  
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(  
            new ProviderConfiguration(),  
            AmazonSQSClientBuilder.standard()  
                .withRegion(config.getRegion().getName())  
                .withCredentials(config.getCredentialsProvider())  
            );  
    }
```



```
// Create the connection
SQSConnection connection = connectionFactory.createConnection();

// Create the queue if needed
ExampleCommon.ensureQueueExists(connection, config.getQueueName());

// Create the session
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
MessageProducer producer =
session.createProducer( session.createQueue( config.getQueueName() ) );

    sendMessages(session, producer);

// Close the connection. This closes the session automatically
connection.close();
System.out.println( "Connection closed" );
}

private static void sendMessages( Session session, MessageProducer producer ) {
    BufferedReader inputReader = new BufferedReader(
        new InputStreamReader( System.in, Charset.defaultCharset() ) );

    try {
        String input;
        while( true ) {
            System.out.print( "Enter message to send (leave empty to exit): " );
            input = inputReader.readLine();
            if( input == null || input.equals("") ) break;

            TextMessage message = session.createTextMessage(input);
            producer.send(message);
            System.out.println( "Send message " + message.getJMSMessageID() );
        }
    } catch (EOFException e) {
        // Just return on EOF
    } catch (IOException e) {
        System.err.println( "Failed reading input: " + e.getMessage() );
    } catch (JMSException e) {
        System.err.println( "Failed sending message: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
```

SyncMessageReceiver.java

下列 Java 程式碼範例會建立同步訊息消費者。

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

public class SyncMessageReceiver {
    public static void main(String args[]) throws JMSEException {
        ExampleConfiguration config =
            ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                .withRegion(config.getRegion().getName())
                .withCredentials(config.getCredentialsProvider())
        );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

```

    MessageConsumer consumer =
session.createConsumer( session.createQueue( config.getQueueName() ) );

    connection.start();

    receiveMessages(session, consumer);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println( "Connection closed" );
}

private static void receiveMessages( Session session, MessageConsumer consumer ) {
    try {
        while( true ) {
            System.out.println( "Waiting for messages");
            // Wait 1 minute for a message
            Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        }
    } catch (JMSEException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}

```

AsyncMessageReceiver.java

下列 Java 程式碼範例會建立異步訊息消費者。

```

/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at

```

```
*
* https://aws.amazon.com/apache2.0
*
* or in the "license" file accompanying this file. This file is distributed
* on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
* express or implied. See the License for the specific language governing
* permissions and limitations under the License.
*
*/

public class AsyncMessageReceiver {
    public static void main(String args[]) throws JMSEException, InterruptedException {
        ExampleConfiguration config =
            ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                .withRegion(config.getRegion().getName())
                .withCredentials(config.getCredentialsProvider())
        );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
            session.createConsumer( session.createQueue( config.getQueueName() ) );

        // No messages are processed until this is called
        connection.start();

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );
    }
}
```

```

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();

        public void waitForOneMinuteOfSilence() throws InterruptedException {
            for(;;) {
                long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
                long remainingTillOneMinuteOfSilence =
                    TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
                if( remainingTillOneMinuteOfSilence < 0 ) {
                    break;
                }
                TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
            }
        }

        @Override
        public void onMessage(Message message) {
            try {
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message " +
                    message.getJMSMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSEException e) {
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}

```

SyncMessageReceiverClientAcknowledge.java

以下 Java 程式碼範例會建立用戶端認可模式的異步消費者。

```

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

/**
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 * messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 * UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 * messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 * visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 * attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also
 * acknowledged.
 *
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 * SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
    for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSEException, InterruptedException {
        // Create the configuration for the example

```

```
ExampleConfiguration config =
ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

// Setup logging for the example
ExampleCommon.setupLogging();

// Create the connection factory based on the config
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
    new ProviderConfiguration(),
    AmazonSQSClientBuilder.standard()
        .withRegion(config.getRegion().getName())
        .withCredentials(config.getCredentialsProvider())
    );

// Create the connection
SQSConnection connection = connectionFactory.createConnection();

// Create the queue if needed
ExampleCommon.ensureQueueExists(connection, config.getQueueName());

// Create the session with client acknowledge mode
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

// Create the producer and consume
MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

// Open the connection
connection.start();

// Send two text messages
sendMessage(producer, session, "Message 1");
sendMessage(producer, session, "Message 2");

// Receive a message and don't acknowledge it
receiveMessage(consumer, false);

// Receive another message and acknowledge it
receiveMessage(consumer, true);

// Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
```

```
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
        receiving no messages since
        // we have acknowledged the second message. Although we didn't explicitly
        acknowledge the first message,
        // in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the
        explicitly acknowledged message
        // are also acknowledged. Therefore, we have implicitly acknowledged the first
        message.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSEException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSEException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
     (TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
     "Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
     acknowledged.
     * @throws JMSEException
     */
}
```



```
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSEException {
    // Receive a message
    Message message =
consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

    if (message == null) {
        System.out.println("Queue is empty!");
    } else {
        // Since this queue has only text messages, cast the message object and
        print the text
        System.out.println("Received: " + ((TextMessage) message).getText());

        // Acknowledge the message if asked
        if (acknowledge) message.acknowledge();
    }
}
}
```

SyncMessageReceiverUnorderedAcknowledge.java

以下 Java 程式碼範例會建立無順序認可模式的異步消費者。

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

/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for
 * received messages. This example
```

```
* complements the example given in {@link SyncMessageReceiverClientAcknowledge} for
CLIENT_ACKNOWLEDGE mode.
*
* First, a session, a message producer, and a message consumer are created. Then, two
messages are sent. Next, two messages
* are received but only the second one is acknowledged. After waiting for the
visibility time out period, an attempt to
* receive another message is made. It's shown that the first message received in the
prior attempt is returned again
* for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
explicitly acknowledged no matter what
* the order they're received.
*
* This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
SyncMessageReceiverClientAcknowledge}
* for an example.
*/
public class SyncMessageReceiverUnorderedAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
    for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSEException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
        ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                .withRegion(config.getRegion().getName())
                .withCredentials(config.getCredentialsProvider())
            );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());
```

```
// Create the session with unordered acknowledge mode
Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

// Create the producer and consume
MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

// Open the connection
connection.start();

// Send two text messages
sendMessage(producer, session, "Message 1");
sendMessage(producer, session, "Message 2");

// Receive a message and don't acknowledge it
receiveMessage(consumer, false);

// Receive another message and acknowledge it
receiveMessage(consumer, true);

// Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
System.out.println("Waiting for visibility timeout...");
Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

// Attempt to receive another message and acknowledge it. This results in
receiving the first message since
// we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE
mode, all the messages must
// be explicitly acknowledged.
receiveMessage(consumer, true);

// Close the connection. This closes the session automatically
connection.close();
System.out.println("Connection closed.");
}

/**
 * Sends a message through the producer.
 */
```

```
* @param producer Message producer
* @param session Session
* @param messageText Text for the message to be sent
* @throws JMSEException
*/
private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSEException {
    // Create a text message and send it
    producer.send(session.createTextMessage(messageText));
}

/**
 * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
 * If a message is received, the message is printed. If no message is received,
 "Queue is empty!" is
 * printed.
 *
 * @param consumer Message consumer
 * @param acknowledge If true and a message is received, the received message is
 acknowledged.
 * @throws JMSEException
 */
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSEException {
    // Receive a message
    Message message =
consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

    if (message == null) {
        System.out.println("Queue is empty!");
    } else {
        // Since this queue has only text messages, cast the message object and
print the text
        System.out.println("Received: " + ((TextMessage) message).getText());

        // Acknowledge the message if asked
        if (acknowledge) message.acknowledge();
    }
}
}
```

SpringExampleConfiguration.xml

以下 XML 程式碼範例是 [SpringExample.java](#) 的 Bean 組態檔案。

```
<!--
  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

以下 Java 程式碼範例會使用 Bean 組態檔案來初始化物件。

```

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

public class SpringExample {
    public static void main(String args[]) throws JMSEException {
        if( args.length != 1 || !args[0].endsWith(".xml")) {

```

```
        System.err.println( "Usage: " + SpringExample.class.getName() + " <spring
config.xml>" );
        System.exit(1);
    }

    File springFile = new File( args[0] );
    if( !springFile.exists() || !springFile.canRead() ) {
        System.err.println( "File " + args[0] + " doesn't exist or isn't
readable." );
        System.exit(2);
    }

    ExampleCommon.setupLogging();

    FileSystemXmlApplicationContext context =
        new FileSystemXmlApplicationContext( "file://" +
springFile.getAbsolutePath() );

    Connection connection;
    try {
        connection = context.getBean(Connection.class);
    } catch( NoSuchBeanDefinitionException e ) {
        System.err.println( "Can't find the JMS connection to use: " +
e.getMessage() );
        System.exit(3);
        return;
    }

    String queueName;
    try {
        queueName = context.getBean("QueueName", String.class);
    } catch( NoSuchBeanDefinitionException e ) {
        System.err.println( "Can't find the name of the queue to use: " +
e.getMessage() );
        System.exit(3);
        return;
    }

    if( connection instanceof SQSConnection ) {
        ExampleCommon.ensureQueueExists( (SQSConnection) connection, queueName );
    }

    // Create the session
    Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

```

        MessageConsumer consumer =
session.createConsumer( session.createQueue( queueName) );

        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                    break;
                }
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message" );
            }
        } catch (JMSEException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}

```

ExampleCommon.java

以下 Java 程式碼範例檢查 Amazon SQS 佇列是否存在，如果不存在，則建立此佇列。另外還包含範例記錄程式碼。

```

/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *

```



```
* https://aws.amazon.com/apache2.0
*
* or in the "license" file accompanying this file. This file is distributed
* on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
* express or implied. See the License for the specific language governing
* permissions and limitations under the License.
*
*/

public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is
     run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName)
    throws JMSEException {
        AmazonSQSMessagingClientWrapper client =
        connection.getWrappedAmazonSQSClient();

        /**
         * In most cases, you can do this with just a createQueue call, but
        GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
        queue
         * already exists. Also many users and roles have permission to call
        GetQueueUrl
         * but don't have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSEException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: " );
        if( message instanceof TextMessage ) {
```

```
        TextMessage txtMessage = ( TextMessage ) message;
        System.out.println( "\t" + txtMessage.getText() );
    } else if( message instanceof BytesMessage ){
        BytesMessage byteMessage = ( BytesMessage ) message;
        // Assume the length fits in an int - SQS only supports sizes up to 256k so
that
        // should be true
        byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
        byteMessage.readBytes(bytes);
        System.out.println( "\t" + Base64.encodeAsString( bytes ) );
    } else if( message instanceof ObjectMessage ) {
        ObjectMessage objMessage = (ObjectMessage) message;
        System.out.println( "\t" + objMessage.getObject() );
    }
}
}
```

Amazon SQS 支援的 JMS 1.1 實作

Amazon SQS Java 訊息程式庫支援以下 [JMS 1.1 實作](#)。如需 Amazon SQS Java 訊息程式庫支援的特性和功能相關資訊，請參閱 [Amazon SQS 常見問答集](#)。

支援的常用介面

- Connection
- ConnectionFactory
- Destination
- Session
- MessageConsumer
- MessageProducer

支援的訊息類型

- ByteMessage
- ObjectMessage
- TextMessage

支援的訊息認可模式

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

Note

`UNORDERED_ACKNOWLEDGE` 模式並不屬於 JMS 1.1 規格。此模式可協助 Amazon SQS 允許 JMS 用戶端明確認可訊息。

JMS 定義標頭和預訂屬性

進行傳送訊息

傳送訊息時，您可以設定每個訊息的以下標頭和屬性：

- `JMSXGroupID` (FIFO 佇列為必要，不允許使用於標準佇列)
- `JMS_SQS_DeduplicationId` (FIFO 佇列為選用，不允許使用於標準佇列)

傳送訊息後，Amazon SQS 會設定每個訊息的以下標頭和屬性：

- `JMSMessageID`
- `JMS_SQS_SequenceNumber` (僅限於 FIFO 佇列)

接收訊息

接收訊息後，Amazon SQS 會設定每個訊息的以下標頭和屬性：

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`
- `JMSXDeliveryCount`

- JMSXGroupID (僅限於 FIFO 佇列)
- JMS_SQS_DeduplicationId (僅限於 FIFO 佇列)
- JMS_SQS_SequenceNumber (僅限於 FIFO 佇列)

Amazon SQS 教學課程

本主題提供教學課程，協助您探索 Amazon SQS 功能。

教學課程

- [使用 建立 Amazon SQS 佇列 AWS CloudFormation](#)
- [教學課程：從 Amazon Virtual Private Cloud 傳送訊息至 Amazon SQS 佇列 Amazon Virtual Private Cloud](#)

使用 建立 Amazon SQS 佇列 AWS CloudFormation

使用 AWS CloudFormation 主控台搭配 JSON 或 YAML 範本來建立 Amazon SQS 佇列。如需詳細資訊，請參閱AWS CloudFormation 《使用者指南[AWS::SQS::Queue](#)》中的[使用 AWS CloudFormation 範本](#)和資源。

使用 AWS CloudFormation 建立 Amazon SQS 佇列。

1. 將下列 JSON 程式碼複製到名為 MyQueue.json 的檔案。若要建立標準佇列，請省略 FifoQueue 和 ContentBasedDeduplication 屬性。如需內容型重複資料刪除功能的相關資訊，請參閱[Amazon SQS 中的精確一次處理](#)。

Note

FIFO 佇列名稱結尾必須是 .fifo 尾碼。

```
{
  "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. 登入 [AWS CloudFormation 主控台](#)，然後選擇 Create Stack (建立堆疊)。
3. 在 Specify Template (指定範本) 面板中，選擇 Upload a template file (上傳範本檔案)、選擇您的 MyQueue.json 檔案，然後選擇 Next (下一步)。
4. 在 Specify Details (指定詳細資訊) 頁面，為 MyQueueStack Name (堆疊名稱) 輸入 `MyQueueStack`，然後選擇 Next (下一步)。
5. 在 Options (選項) 頁面上，選擇 Next (下一步)。
6. 在 Review (檢閱) 頁面上，選擇 Create (建立)。

AWS CloudFormation 開始建立 MyQueue 堆疊，並顯示 CREATE_IN_PROGRESS 狀態。程序完成後，AWS CloudFormation 會顯示 CREATE_COMPLETE 狀態。

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. (選用) 如果要顯示佇列的名稱、URL 和 ARN，請選擇堆疊的名稱，然後在下一頁展開 Outputs (輸出) 部分。

教學課程：從 Amazon Virtual Private Cloud 傳送訊息至 Amazon SQS 佇列 Amazon Virtual Private Cloud

本教學課程說明如何透過安全的私有網路將訊息傳送至 Amazon SQS 佇列。網路包括：

- 包含 Amazon EC2 執行個體的 VPC。
- 介面 VPC 端點，允許 Amazon EC2 執行個體連線至 Amazon SQS，而無需使用公有網際網路。

即使在全私有網路中，您也可以連線至 Amazon EC2 執行個體，並將訊息傳送至 Amazon SQS 佇列。如需詳細資訊，請參閱[適用於 Amazon SQS 的 Amazon Virtual Private Cloud 端點](#)。

Important

- Amazon Virtual Private Cloud 只能搭配 HTTPS Amazon SQS 端點使用。
- 當您設定 Amazon SQS 從 Amazon VPC 傳送訊息時，您必須啟用私有 DNS，並以格式 `sqs.us-east-2.amazonaws.com` 或 `sqs.us-east-2.api.aws` 雙堆疊端點指定端點。
- Amazon SQS 也使用端點服務透過 PrivateLink 支援 FIPS `com.amazonaws.region.sqs-fips` 端點。您可以連線至格式為 `sqs-fips.region.amazonaws.com` 的 FIPS 端點。
- 在 Amazon Virtual Private Cloud 中使用雙堆疊端點時，將使用 IPv4 和 IPv6 傳送請求。
- 私有 DNS 不支援 `queue.amazonaws.com` 或 `us-east-2.queue.amazonaws.com` 之類的延遲端點。

步驟 1：建立 Amazon EC2 金鑰對

金鑰對可讓您連接至 Amazon EC2 執行個體。它包含加密您登入資訊的公有金鑰，以及解密的私有金鑰。

1. 登入 [Amazon SNS 主控台](#)。
2. 在導覽功能表的 Network & Security (網路與安全性) 中，選擇 Key Pairs (金鑰對)。
3. 選擇 Create Key Pair (建立金鑰對)。
4. 在 Create Key Pair (建立金鑰對) 對話方塊中，在 Key pair name (金鑰對名稱) 輸入 SQS-VPCE-Tutorial-Key-Pair，然後選擇 Create (建立)。
5. 您的瀏覽器會自動下載私有金鑰檔案 SQS-VPCE-Tutorial-Key-Pair.pem。

Important

將此檔案存放在安全的地方。EC2 不會為相同的金鑰對產生兩次 .pem 檔案。

6. 若要允許 SSH 用戶端連接至 EC2 執行個體，請設定私有金鑰檔的許可，讓只有您的使用者才能夠擁有其讀取許可，例如：

```
chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
```

步驟 2：建立 AWS 資源

若要設定必要的基礎設施，您必須使用 an AWS CloudFormation template，這是建立包含 AWS 資源堆疊的藍圖，例如 Amazon EC2 執行個體和 Amazon SQS 佇列。

此教學的堆疊包含下列資源：

- 一個 VPC 和關聯的網路資源，包括子網路、安全群組、網際網路閘道和路由表。
- 在 VPC 子網路中啟動的 Amazon EC2 執行個體
- Amazon SQS 佇列

1. [SQS-VPCE-Tutorial-CloudFormation.yaml](#) 從 GitHub 下載名為 的 AWS CloudFormation 範本。
2. 登入 [AWS CloudFormation 主控台](#)。

3. 選擇 Create Stack (建立堆疊)。
4. 在 Select Template (選擇範本) 頁面中，選擇 Upload a template Amazon S3 (將範本上傳至 Amazon S3)、選擇 SQS-VPCE-SQS-Tutorial-CloudFormation.yaml 檔案，然後選擇 Next (下一步)。
5. 在 Specify Details (指定詳細資訊) 頁面上，執行下列作業：
 - a. 針對堆疊名稱 輸入 SQS-VPCE-Tutorial-Stack。
 - b. 對於 KeyName (KeyName)，選擇 SQS-VPCE-Tutorial-Key-Pair (SQS-VPCE-Tutorial-Key-Pair)。
 - c. 選擇下一步。
6. 在選項頁面上，選擇下一步。
7. 在檢閱頁面的功能區段中，選擇我確認 AWS CloudFormation 可能會使用自訂名稱建立 IAM 資源，然後選擇建立。

AWS CloudFormation 開始建立堆疊，並顯示 CREATE_IN_PROGRESS 狀態。程序完成後，AWS CloudFormation 會顯示 CREATE_COMPLETE 狀態。

步驟 3：確認 EC2 執行個體不是供公開存取

您的 AWS CloudFormation 範本會在 VPC SQS-VPCE-Tutorial-EC2-Instance 中啟動名為的 EC2 執行個體。此 EC2 執行個體不允許輸出流量，且無法將訊息傳送至 Amazon SQS。若要驗證是否如此，您必須連接至此執行個體、嘗試連接至公有端點，接著嘗試向 Amazon SQS 傳送訊息。

1. 登入 [Amazon SNS 主控台](#)。
2. 在導覽功能表的 Instances (執行個體) 下，選擇 Instances (執行個體)。
3. 選取 SQS-VPCE-Tutorial-EC2Instance (SQS-VPCE-Tutorial-EC2Instance)。
4. 在公有 DNS 下複製主機名稱，例如 ec2-203-0-113-0.us-west-2.compute.amazonaws.com。
5. 從包含您先前建立之金鑰對的 [目錄中](#)，使用以下命令連接至執行個體，例如：

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. 嘗試連接到任何公有端點，例如：

```
ping amazon.com
```

連接嘗試如預期失敗。

7. 請登入 [Amazon SQS 主控台](#)。
8. 從佇列清單中，選取 AWS CloudFormation 範本建立的佇列，例如 VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK。
9. 在詳細資訊表格上，複製 URL，例如 `https://sqs.us-east-2.amazonaws.com/123456789012/`。
10. 使用以下命令，從 EC2 執行個體，嘗試將訊息發佈至佇列，例如：

```
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."
```

傳送中嘗試如預期失敗。

Important

當您稍後為 Amazon SQS 建立 VPC 端點時，傳送中嘗試將會成功。

步驟 4：建立 Amazon SQS 的 Amazon VPC 端點

若要將您的 VPC 連接到 Amazon SQS，您必須定義介面 VPC 端點。在您新增端點後，您可以在 VPC 中使用來自 EC2 執行個體的 Amazon SQS API。這可讓您將訊息傳送到 AWS 網路中的佇列，而無需跨公有網際網路。

Note

EC2 執行個體仍然無法存取網際網路上的其他服務 AWS 和端點。

1. 請登入 [Amazon VPC 主控台](#)。
2. 在導覽功能表中，選擇 Endpoints (端點)。
3. 選擇建立端點。
4. 在建立端點頁面，針對服務名稱，選擇 Amazon SQS 的服務名稱。

 Note

服務名稱會根據目前的 AWS 區域而有所不同。例如，如果您位於美國東部（俄亥俄），則服務名稱為 `com.amazonaws.us-east-2.sqs`。

5. 針對 VPC，選擇 SQS-VPCE-Tutorial-VPC。
6. 針對 Subnets (子網路)，選擇其 Subnet ID (子網路 ID) 包含 SQS-VPCE-Tutorial-Subnet。
7. 針對 Security group (安全群組)，選擇 Select security groups (選取安全群組)，然後選擇其 Group Name (群組名稱) 中包含 SQS VPCE Tutorial Security Group 的安全群組。
8. 選擇建立端點。

界面 VPC 端點已建立，且其 ID 已顯示，例如，`vpce-0ab1cdef2ghi3j456k`。

9. 選擇關閉。

Amazon VPC 主控台開啟 Endpoints (端點) 頁面。

Amazon VPC 會開始建立端點並顯示等待中狀態。程序完成後，Amazon VPC 會顯示可用狀態。

步驟 5：傳送訊息到您的 Amazon SQS 佇列

VPC 現在已包含 Amazon SQS 的端點，您就可以連接至 EC2 執行個體並將訊息傳送至佇列。

1. 重新連接至 EC2 執行個體，例如：

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

2. 使用以下命令，嘗試將訊息發佈至佇列，例如：

```
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."
```

傳送中嘗試會成功，且會顯示訊息本文的 MD5 摘要和訊息 ID，例如：

```
{
  "MD5ofMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
  "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
```

```
}
```

如需有關從 AWS CloudFormation 範本建立的佇列接收和刪除訊息的資訊（例如，VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK），請參閱 [在 Amazon SQS 中接收和刪除訊息](#)。

如需有關刪除資源的資訊，請參閱下列各項：

- 《Amazon [VPC 使用者指南](#)》中的[刪除 VPC 端點](#)
- [刪除 Amazon SQS 佇列](#)
- 《Amazon EC2 使用者指南》中的[終止您的執行個體](#)
- 《Amazon [VPC 使用者指南](#)》中的[刪除您的 VPC](#)
- 《AWS CloudFormation 使用者指南》中的[刪除 AWS CloudFormation 主控台上的堆疊](#)
- 《Amazon EC2 使用者指南》中的[刪除您的金鑰對](#)

使用 AWS SDKs Amazon SQS 程式碼範例

下列程式碼範例示範如何搭配 AWS 軟體開發套件 (SDK) 使用 Amazon SQS。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含入門相關資訊和舊版 SDK 的詳細資訊。

開始使用

您好，Amazon SQS

下列程式碼範例示範如何開始使用 Amazon SQS。

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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();
        }
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

CMakeLists.txt CMake 檔案的程式碼。

```
# 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})

```

hello_sqs.cpp 來源檔案的程式碼。

```

#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;
}

```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的 [ListQueues](#)。

Go

SDK for Go V2

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
    }
}
}

```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [ListQueues](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

初始化 Amazon SQS 用戶端並列出佇列。

```

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"));
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
```

```
        .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
        .collect { queue ->
            println("The Queue URL is $queue")
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [ListQueues](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

Package.swift 檔。

```
import PackageDescription

let package = Package(
    name: "sqs-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/aws-labs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
```

```

        // Targets are the basic building blocks of a package, defining a module
        // or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sqs-basics",
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")
    ]
)

```

Swift 原始程式碼 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)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [ListQueues](#)。

程式碼範例

- [使用 AWS SDKs 的 Amazon SQS 基本範例](#)
 - [您好，Amazon SQS](#)
 - [使用 AWS SDKs 的 Amazon SQS 動作](#)
 - [搭配使用 AddPermission 與 CLI](#)
 - [ChangeMessageVisibility 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 ChangeMessageVisibilityBatch 與 CLI](#)
 - [CreateQueue 搭配 AWS SDK 或 CLI 使用](#)
 - [DeleteMessage 搭配 AWS SDK 或 CLI 使用](#)
 - [DeleteMessageBatch 搭配 AWS SDK 或 CLI 使用](#)
 - [DeleteQueue 搭配 AWS SDK 或 CLI 使用](#)
 - [GetQueueAttributes 搭配 AWS SDK 或 CLI 使用](#)
 - [GetQueueUrl 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 ListDeadLetterSourceQueues 與 CLI](#)
 - [ListQueues 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 PurgeQueue 與 CLI](#)
 - [ReceiveMessage 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 RemovePermission 與 CLI](#)
 - [SendMessage 搭配 AWS SDK 或 CLI 使用](#)
 - [SendMessageBatch 搭配 AWS SDK 或 CLI 使用](#)
 - [SetQueueAttributes 搭配 AWS SDK 或 CLI 使用](#)
- [使用 AWS SDKs Amazon SQS 案例](#)
 - [建立使用 Amazon SQS 傳送和擷取訊息的 Web 應用程式](#)
 - [使用 Step Functions 建立傳訊應用程式](#)
 - [建立 Amazon Textract Explorer 應用程式](#)
 - [使用 AWS SDK 建立和發佈至 FIFO Amazon SNS 主題](#)
 - [使用 AWS SDK 透過 Amazon Rekognition 偵測影片中的人物和物件](#)
 - [透過 SDK 使用 Amazon S3 管理大型 Amazon SQS 訊息 Amazon S3 AWS](#)
 - [使用 SDK 接收和處理 Amazon S3 事件通知 AWS](#)
 - [使用 SDK 將 Amazon SNS 訊息發佈至 Amazon SQS 佇列 AWS](#)

- [使用 AWS SDK 透過 Amazon SQS 傳送和接收批次訊息](#)
- [使用適用於 .NET AWS 的訊息處理架構來發佈和接收 Amazon SQS 訊息](#)
- [使用 Amazon SQS Java Messaging Library 來使用 Amazon SQS 的 Java Message Service \(JMS\) 界面](#)
- [使用 AWS SDK 處理佇列標籤和 Amazon SQS](#)
- [Amazon SQS 的無伺服器範例](#)
 - [使用 Amazon SQS 觸發條件調用 Lambda 函數](#)
 - [使用 Amazon SQS 觸發條件報告 Lambda 函數的批次項目失敗](#)

使用 AWS SDKs 的 Amazon SQS 基本範例

下列程式碼範例示範如何搭配 AWS SDKs 使用 Amazon Simple Queue Service 的基本概念。

範例

- [您好，Amazon SQS](#)
- [使用 AWS SDKs 的 Amazon SQS 動作](#)
 - [搭配使用 AddPermission 與 CLI](#)
 - [ChangeMessageVisibility 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 ChangeMessageVisibilityBatch 與 CLI](#)
 - [CreateQueue 搭配 AWS SDK 或 CLI 使用](#)
 - [DeleteMessage 搭配 AWS SDK 或 CLI 使用](#)
 - [DeleteMessageBatch 搭配 AWS SDK 或 CLI 使用](#)
 - [DeleteQueue 搭配 AWS SDK 或 CLI 使用](#)
 - [GetQueueAttributes 搭配 AWS SDK 或 CLI 使用](#)
 - [GetQueueUrl 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 ListDeadLetterSourceQueues 與 CLI](#)
 - [ListQueues 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 PurgeQueue 與 CLI](#)
 - [ReceiveMessage 搭配 AWS SDK 或 CLI 使用](#)
 - [搭配使用 RemovePermission 與 CLI](#)
 - [SendMessage 搭配 AWS SDK 或 CLI 使用](#)

- [SendMessageBatch 搭配 AWS SDK 或 CLI 使用](#)
- [SetQueueAttributes 搭配 AWS SDK 或 CLI 使用](#)

您好，Amazon SQS

下列程式碼範例示範如何開始使用 Amazon SQS。

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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($"{queue.QueueUrl}");
            Console.WriteLine();
        }
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

CMakeLists.txt CMake 檔案的程式碼。

```
# 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})

```

hello_sqs.cpp 來源檔案的程式碼。

```

#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;
}

```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Go

SDK for Go V2

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

package main

import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.

```

```
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    sqsClient := sqs.NewFromConfig(sdkConfig)
    fmt.Println("Let's list the queues for your account.")
    var queueUrls []string
    paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
    for paginator.HasMorePages() {
        output, err := paginator.NextPage(ctx)
        if err != nil {
            log.Printf("Couldn't get queues. Here's why: %v\n", err)
            break
        } else {
            queueUrls = append(queueUrls, output.QueueUrls...)
        }
    }
    if len(queueUrls) == 0 {
        fmt.Println("You don't have any queues!")
    } else {
        for _, queueUrl := range queueUrls {
            fmt.Printf("\t\t%v\n", queueUrl)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [ListQueues](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

初始化 Amazon SQS 用戶端並列出佇列。

```
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"));
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
            .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
            .collect { queue ->
                println("The Queue URL is $queue")
            }
    }
}
```

```
}  
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [ListQueues](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

Package.swift 檔。

```
import PackageDescription  
  
let package = Package(  
    name: "sqs-basics",  
    // Let Xcode know the minimum Apple platforms supported.  
    platforms: [  
        .macOS(.v13),  
        .iOS(.v15)  
    ],  
    dependencies: [  
        // Dependencies declare other packages that this package depends on.  
        .package(  
            url: "https://github.com/aws-labs/aws-sdk-swift",  
            from: "1.0.0"),  
        .package(  
            url: "https://github.com/apple/swift-argument-parser.git",  
            branch: "main"  
        )  
    ],  
    targets: [  
        // Targets are the basic building blocks of a package, defining a module  
        // or a test suite.  
        // Targets can depend on other targets in this package and products  
        // from dependencies.
```

```

        .executableTarget(
            name: "sqs-basics",
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")
    ]
)

```

Swift 原始程式碼 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)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [ListQueues](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDKs 的 Amazon SQS 動作

下列程式碼範例示範如何使用 AWS SDKs 執行個別 Amazon SQS 動作。每個範例均包含 GitHub 的連結，您可以在連結中找到設定和執行程式碼的相關說明。

這些摘錄會呼叫 Amazon SQS API，是必須在內容中執行之大型程式的程式碼摘錄。您可以在 [使用 AWS SDKs Amazon SQS 案例](#) 中查看內容中的動作。

下列範例僅包含最常使用的動作。如需完整清單，請參閱 [Amazon Simple Queue Service API 參考](#)。

範例

- [搭配使用 AddPermission 與 CLI](#)
- [ChangeMessageVisibility 搭配 AWS SDK 或 CLI 使用](#)
- [搭配使用 ChangeMessageVisibilityBatch 與 CLI](#)
- [CreateQueue 搭配 AWS SDK 或 CLI 使用](#)
- [DeleteMessage 搭配 AWS SDK 或 CLI 使用](#)
- [DeleteMessageBatch 搭配 AWS SDK 或 CLI 使用](#)
- [DeleteQueue 搭配 AWS SDK 或 CLI 使用](#)
- [GetQueueAttributes 搭配 AWS SDK 或 CLI 使用](#)
- [GetQueueUrl 搭配 AWS SDK 或 CLI 使用](#)
- [搭配使用 ListDeadLetterSourceQueues 與 CLI](#)
- [ListQueues 搭配 AWS SDK 或 CLI 使用](#)
- [搭配使用 PurgeQueue 與 CLI](#)
- [ReceiveMessage 搭配 AWS SDK 或 CLI 使用](#)
- [搭配使用 RemovePermission 與 CLI](#)
- [SendMessage 搭配 AWS SDK 或 CLI 使用](#)
- [SendMessageBatch 搭配 AWS SDK 或 CLI 使用](#)
- [SetQueueAttributes 搭配 AWS SDK 或 CLI 使用](#)

搭配使用 **AddPermission** 與 CLI

下列程式碼範例示範如何使用 AddPermission。

CLI

AWS CLI

將許可新增至佇列

此範例可讓指定的 AWS 帳戶傳送訊息至指定的佇列。

命令：

```
aws sqs add-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessageFromMyQueue --aws-account-ids 12345EXAMPLE --actions SendMessage
```

輸出：

```
None.
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [AddPermission](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例允許指定的 從指定的佇列 AWS 帳戶 傳送訊息。

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE  
-Label SendMessageFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [AddPermission](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

ChangeMessageVisibility 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 ChangeMessageVisibility。

C++

SDK for C++

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

    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();
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [ChangeMessageVisibility](#)。

CLI

AWS CLI

變更訊息的逾時可見性

此範例會將指定訊息的逾時可見性變更為 10 小時 (10 小時 * 60 分鐘 * 60 秒)。

命令：

```
aws sqs change-message-visibility --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --visibility-timeout 36000
```

輸出：

```
None.
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ChangeMessageVisibility](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

接收 Amazon SQS 訊息並變更其逾時可見性。

```
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;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ChangeMessageVisibility](#)。

SDK for JavaScript (v2)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

接收 Amazon SQS 訊息並變更其逾時可見性。

```
// 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");  
  }  
}  
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-managing-visibility-timeout.html#sqs-examples-managing-visibility-timeout-setting>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ChangeMessageVisibility](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會將指定佇列中具有指定接收控點的訊息可見性逾時變更為 10 小時 (10 小時 * 60 分鐘 * 60 秒 = 36000 秒)。

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -  
VisibilityTimeout 36000
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell Cmdlet Reference (V4) 中的 [ChangeMessageVisibility](#)。

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱適用於 Ruby 的 AWS SDK 《API 參考》中的 [ChangeMessageVisibility](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配使用 **ChangeMessageVisibilityBatch** 與 CLI

下列程式碼範例示範如何使用 `ChangeMessageVisibilityBatch`。

CLI

AWS CLI

以批次形式變更多則訊息的逾時可見性

此範例會將 2 個指定訊息的逾時可見性變更為 10 小時 (10 小時 * 60 分鐘 * 60 秒)。

命令：

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

輸入檔案 (change-message-visibility-batch.json)：

```
[
  {
    "Id": "FirstMessage",
    "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
    "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
    "ReceiptHandle": "AQEBkTUH...HifSnw==",
    "VisibilityTimeout": 36000
  }
]
```

輸出：

```
{
```

```

    "Successful": [
      {
        "Id": "SecondMessage"
      },
      {
        "Id": "FirstMessage"
      }
    ]
  }
}

```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ChangeMessageVisibilityBatch](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會變更 2 則訊息在指定佇列中具有指定接收控點的可見性逾時。第一個訊息的可見性逾時會變更為 10 小時 (10 小時 * 60 分鐘 * 60 秒 = 36000 秒)。第二個訊息的可見性逾時變更為 5 小時 (5 小時 * 60 分鐘 * 60 秒 = 18000 秒)。

```

$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

```

輸出：

```

Failed      Successful
-----

```



```
{ } {Request2, Request1}
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell Cmdlet Reference (V4) 中的 [ChangeMessageVisibilityBatch](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

CreateQueue 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 CreateQueue。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [將訊息發佈至佇列](#)
- [傳送和接收批次的訊息](#)
- [使用 Amazon SQS Java Messaging Library 來使用 JMS 介面](#)

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立具有特定名稱的佇列。

```
/// <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;
}
```

建立 Amazon SQS 佇列並傳送訊息給該佇列。

```
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;
}
}

```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [CreateQueue](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

    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();
}

```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [CreateQueue](#)。

CLI

AWS CLI

建立佇列

此範例會建立具有指定名稱的佇列，將訊息保留期間設定為 3 天 (3 天 * 24 小時 * 60 分鐘 * 60 秒)，並將佇列的無效字母佇列設定為接收計數上限為 1,000 則訊息的指定佇列。

命令：

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

輸入檔案 (create-queue.json)：

```

{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}

```

輸出：

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [CreateQueue](#)。

Go

SDK for Go V2

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [CreateQueue](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
}
}
}

```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [CreateQueue](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立 Amazon SQS 標準佇列。

```

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;
};
```

建立具有長輪詢的 Amazon SQS 佇列。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-create-queue>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [CreateQueue](#)。

SDK for JavaScript (v2)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立 Amazon SQS 標準佇列。

```
// 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);
  }
});
```

建立等待訊息送達的 Amazon SQS 佇列。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-create-queue>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [CreateQueue](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
suspend fun createQueue(queueNameVal: String): String {
    println("Create Queue")
    val createQueueRequest =
        CreateQueueRequest {
            queueName = queueNameVal
        }
}
```



```
    }

    SqsClient { 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()
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [CreateQueue](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會建立具有指定名稱的佇列。

```
New-SQSQueue -QueueName MyQueue
```

輸出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell Cmdlet Reference (V4) 中的 [CreateQueue](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [CreateQueue](#)。

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
# 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
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__

```

- 如需 API 詳細資訊，請參閱適用於 Ruby 的 AWS SDK 《API 參考》中的 [CreateQueue](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立 Amazon SQS 標準佇列。

```

TRY.
    oo_result = lo_sqs->createqueue( iv_queue_name = iv_queue_name ).
    "
    oo_result is returned for testing purposes.
    MESSAGE 'SQS queue created.' TYPE 'I'.
    CATCH /aws1/cx_sqsqueuedeletedrecently.
    MESSAGE 'After deleting a queue, wait 60 seconds before creating another
    queue with the same name.' TYPE 'E'.
    CATCH /aws1/cx_sqsqueuealreadyexists.
    MESSAGE 'A queue with this name already exists.' TYPE 'E'.
ENDTRY.

```

建立等待訊息送達的 Amazon SQS 佇列。

```

TRY.
    DATA lt_attributes TYPE /aws1/cl_sqsqueueattribmap_w=>tt_queueattributemap.
    DATA ls_attribute TYPE /aws1/
    cl_sqsqueueattribmap_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_sqsqueuedeletedrecently.
        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.

```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 [CreateQueue](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [CreateQueue](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

DeleteMessage 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 DeleteMessage。

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

從 Amazon SQS 佇列接收訊息，然後刪除訊息。

```
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;
}
}

```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [DeleteMessage](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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();
}

```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [DeleteMessage](#)。

CLI

AWS CLI

刪除訊息

此範例會刪除指定的訊息。

命令：

```
aws sqs delete-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

輸出：

None.

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [DeleteMessage](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
}
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [DeleteMessage](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

接收和刪除 Amazon SQS 訊息。

```
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,  
            })),  
        })),  
    );  
}  
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [DeleteMessage](#)。

SDK for JavaScript (v2)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

接收和刪除 Amazon SQS 訊息。

```
// 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);
      }
    });
  }
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-send-receive-messages.html#sqs-examples-send-receive-messages-receiving>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [DeleteMessage](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [DeleteMessage](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會從指定的佇列刪除具有指定接收控點的訊息。

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [DeleteMessage](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [DeleteMessage](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

DeleteMessageBatch 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 DeleteMessageBatch。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [處理 S3 事件通知](#)
- [將訊息發佈至佇列](#)
- [傳送和接收批次的訊息](#)

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/// <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();
    }

```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [DeleteMessageBatch](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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;
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [DeleteMessageBatch](#)。

CLI

AWS CLI

刪除批次形式的多個訊息

此範例會刪除指定的訊息。

命令：

```
aws sqs delete-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-batch.json
```

輸入檔案 (delete-message-batch.json) :

```
[
  {
    "Id": "FirstMessage",
    "ReceiptHandle": "AQEB1mg1...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
    "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

輸出 :

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [DeleteMessageBatch](#)。

Go

SDK for Go V2

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [DeleteMessageBatch](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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,
            })),
        }),
    );
}
};

```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [DeleteMessageBatch](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會從指定的佇列刪除具有指定接收控點的 2 則訊息。

```

$deleteMessageRequest1 = New-Object
    Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
    Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBq0VY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
    $deleteMessageRequest2

```

輸出：

```
Failed    Successful
-----
{}        {Request1, Request2}
```

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [DeleteMessageBatch](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [DeleteMessageBatch](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```



```
        )
        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.")
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [DeleteMessageBatch](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

DeleteQueue 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 DeleteQueue。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [將訊息發佈至佇列](#)
- [傳送和接收批次的訊息](#)
- [使用 Amazon SQS Java Messaging Library 來使用 JMS 介面](#)

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

使用佇列的 URL 刪除佇列。

```

/// <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;
}

```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [DeleteQueue](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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();
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [DeleteQueue](#)。

CLI

AWS CLI

刪除佇列

此範例會刪除指定的佇列。

命令：

```
aws sqs delete-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

輸出：

```
None.
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [DeleteQueue](#)。

Go

SDK for Go V2

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [DeleteQueue](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
    }
}
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [DeleteQueue](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

刪除 Amazon SQS 佇列。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-delete-queue>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [DeleteQueue](#)。

SDK for JavaScript (v2)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

刪除 Amazon SQS 佇列。

```
// 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);
    }
  });
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-delete-queue>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [DeleteQueue](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [DeleteQueue](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會刪除指定的佇列。

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- 如需 API 詳細資訊，請參閱《Cmdlet 參考》中的 [DeleteQueue](#) (V4)。AWS Tools for PowerShell

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [DeleteQueue](#)。

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
```

- 如需 API 詳細資訊，請參閱適用於 Ruby 的 AWS SDK 《API 參考》中的 [DeleteQueue](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
TRY.  
    lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).  
    MESSAGE 'SQS queue deleted' TYPE 'I'.  
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 [DeleteQueue](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [DeleteQueue](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

GetQueueAttributes 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 GetQueueAttributes。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [處理 S3 事件通知](#)
- [將訊息發佈至佇列](#)

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
/// <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;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [GetQueueAttributes](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [GetQueueAttributes](#)。

CLI

AWS CLI

取得佇列的屬性

此範例會取得所有指定的佇列屬性。

命令：

```
aws sqs get-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

輸出：

```
{
```

```
"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"
}
```

此範例只會取得指定佇列的訊息大小上限和可見性逾時屬性。

命令：

```
aws sqs get-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-names MaximumMessageSize VisibilityTimeout
```

輸出：

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [GetQueueAttributes](#)。

Go

SDK for Go V2

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [GetQueueAttributes](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetQueueAttributes](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例列出指定佇列的所有屬性。

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

輸出：

```
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":

  "SendMessageFromMyQueue", "Effect": "Allow", "Principal":
{"AWS": "80398EXAMPLE"}, "Action": "SQS:SendMessage", "Resource": "
arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"]}]
Attributes
: [[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
[ApproximateNumberOfMessagesNotVisible,
0], [ApproximateNumberOfMessagesDelayed, 0]...]

```

範例 2：此範例僅單獨列出指定佇列的指定屬性。

```

Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue

```

輸出：

```

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":

  "SendMessageFromMyQueue", "Effect": "Allow", "Principal":
{"AWS": "80398EXAMPLE"}, "Action": "SQS:SendMessage", "Resource": "

```

```
arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue"]]]}
Attributes                                : {[MaximumMessageSize, 262144],
[VisibilityTimeout, 30]}
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [GetQueueAttributes](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [GetQueueAttributes](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

GetQueueUrl 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 GetQueueUrl。

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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.");
    }
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [GetQueueUrl](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

        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();
}

```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [GetQueueUrl](#)。

CLI

AWS CLI

取得佇列 URL

此範例會取得指定佇列的 URL。

命令：

```
aws sqs get-queue-url --queue-name MyQueue
```

輸出：

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [GetQueueUrl](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
GetQueueUrlResponse getQueueUrlResponse = sqsClient
    .getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
    return getQueueUrlResponse.queueUrl();
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [GetQueueUrl](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

取得 Amazon SQS 佇列的 URL。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-get-queue-url>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetQueueUrl](#)。

SDK for JavaScript (v2)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

取得 Amazon SQS 佇列的 URL。

```
// 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);
    }
  });
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-get-queue-url>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [GetQueueUrl](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例列出具有指定名稱的佇列 URL。

```
Get-SQSQueueUrl -QueueName MyQueue
```

輸出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [GetQueueUrl](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [GetQueueUrl](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
TRY.  
    oo_result = lo_sqs->getqueueurl( iv_queue_name = iv_queue_name ).  
oo_result is returned for testing purposes. "  
    MESSAGE 'Queue URL retrieved.' TYPE 'I'.  
CATCH /aws1/cx_sqsqueue_does_not_exist.  
    MESSAGE 'The requested queue does not exist.' TYPE 'E'.  
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 [GetQueueUrl](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配使用 **ListDeadLetterSourceQueues** 與 CLI

下列程式碼範例示範如何使用 `ListDeadLetterSourceQueues`。

CLI

AWS CLI

列出無效字母來源佇列

此範例列出與指定的無效字母來源佇列相關聯的佇列。

命令：

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

輸出：

```
{  
  "queueUrls": [  
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",  
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"  
  ]  
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ListDeadLetterSourceQueues](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例列出依賴指定佇列做為無效字母佇列的任何佇列 URLs。

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

輸出：

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [ListDeadLetterSourceQueues](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

ListQueues 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 ListQueues。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
}

```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

CLI

AWS CLI

列出佇列

此範例列出所有佇列。

命令：

```
aws sqs list-queues
```

輸出：

```
{
  "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"
  ]
}
```

此範例只會列出以「我的」開頭的佇列。

命令：

```
aws sqs list-queues --queue-name-prefix My
```

輸出：

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
  ]
}
```



```
"https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
]
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ListQueues](#)。

Go

SDK for Go V2

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    sqsClient := sqs.NewFromConfig(sdkConfig)
```

```
fmt.Println("Let's list the queues for your account.")
var queueUrls []string
paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
for paginator.HasMorePages() {
    output, err := paginator.NextPage(ctx)
    if err != nil {
        log.Printf("Couldn't get queues. Here's why: %v\n", err)
        break
    } else {
        queueUrls = append(queueUrls, output.QueueUrls...)
    }
}
if len(queueUrls) == 0 {
    fmt.Println("You don't have any queues!")
} else {
    for _, queueUrl := range queueUrls {
        fmt.Printf("\t\t%v\n", queueUrl)
    }
}
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
    }
}
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [ListQueues](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

列出您的 Amazon SQS 佇列。

```
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;
}
```

```
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-listing-queues>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

SDK for JavaScript (v2)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

列出您的 Amazon SQS 佇列。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html#sqs-examples-using-queues-listing-queues>。

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
suspend fun listQueues() {
    println("\nList Queues")

    val prefix = "que"
    val listQueuesRequest =
        ListQueuesRequest {
            queueNamePrefix = prefix
        }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.listQueues(listQueuesRequest)
        response.queueUrls?.forEach { url ->
            println(url)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [ListQueues](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例列出所有佇列。

```
Get-SQSQueue
```

輸出：

```
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
```

範例 2：此範例列出以指定名稱開頭的任何佇列。

```
Get-SQSQueue -QueueNamePrefix My
```

輸出：

```
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
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell Cmdlet 參考 (V4) 中的 [ListQueues](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ListQueues](#)。

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```


- 如需 API 詳細資訊，請參閱適用於 Ruby 的 AWS SDK 《API 參考》中的 [ListQueues](#)。

Rust

SDK for Rust

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

擷取 區域中列出的第一個 Amazon SQS 佇列。

```
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())
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Rust 的 SDK API 參考》中的 [ListQueues](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
TRY.  
    oo_result = lo_sqs->listqueues( ).          " oo_result is returned for  
testing purposes. "  
    MESSAGE 'Retrieved list of queues.' TYPE 'I'.  
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 [ListQueues](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)  
        }  
    }
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [ListQueues](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配使用 **PurgeQueue** 與 CLI

下列程式碼範例示範如何使用 PurgeQueue。

CLI

AWS CLI

清除佇列

此範例會刪除指定佇列中的所有訊息。

命令：

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue
```

輸出：

```
None.
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [PurgeQueue](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會從指定的佇列刪除所有訊息。

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [PurgeQueue](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

ReceiveMessage 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 ReceiveMessage。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 S3 管理大型訊息](#)
- [處理 S3 事件通知](#)
- [將訊息發佈至佇列](#)
- [傳送和接收批次的訊息](#)

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

使用佇列的 URL 接收來自佇列的訊息。

```
/// <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;
    }

```

從 Amazon SQS 佇列接收訊息，然後刪除訊息。

```

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;
    }
}

```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [ReceiveMessage](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

    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();
}

```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [ReceiveMessage](#)。

CLI

AWS CLI

接收訊息

此範例最多可接收 10 則可用的訊息，並傳回所有可用的屬性。

命令：


```
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
```

輸出：

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEBzbVv...fqNzFw==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "9424c491...26bc3ae7",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "ApproximateFirstReceiveTimestamp": "1442428276921",
        "SenderId": "AIDAIAZKMSNQ7EXAMPLE",
        "ApproximateReceiveCount": "5",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        },
        "City": {
          "DataType": "String",
          "StringValue": "Any City"
        }
      }
    }
  ]
}
```

此範例會接收下一個可用的訊息，僅傳回 `SenderId` 和 `SentTimestamp` 屬性，以及 `PostalCode` 訊息屬性。

命令：

```
aws sqs receive-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names SenderId SentTimestamp --message-attribute-names PostalCode
```

輸出：

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

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [ReceiveMessage](#)。

Go

SDK for Go V2

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
}
```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [ReceiveMessage](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
try {
    ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
        .queueUrl(queueUrl)
        .numberOfMessages(5)
        .build();
    return sqsClient.receiveMessage(receiveMessageRequest).messages();
} catch (SqsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
return null;
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [ReceiveMessage](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

從 Amazon SQS 佇列接收訊息。

```
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,  
        }))),  
    );  
  }  
}
```

```
    }},  
  );  
}  
};
```

使用長輪詢支援從 Amazon SQS 佇列接收訊息。

```
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;  
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ReceiveMessage](#)。

SDK for JavaScript (v2)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

使用長輪詢支援從 Amazon SQS 佇列接收訊息。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-enable-long-polling.html#sqs-examples-enable-long-polling-on-receive-message>。

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [ReceiveMessage](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
suspend fun receiveMessages(queueUrlVal: String?) {
    println("Retrieving messages from $queueUrlVal")

    val receiveMessageRequest =
        ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.receiveMessage(receiveMessageRequest)
        response.messages?.forEach { message ->
            println(message.body)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [ReceiveMessage](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例列出指定佇列最多可收到接下來 10 則訊息的資訊。如果指定訊息屬性存在，該資訊將包含這些屬性的值。


```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
  StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
  east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

輸出：

```
Attributes          : {[SenderId, AIDAIAZKMSNQ7EXAMPLE], [SentTimestamp,
  1451495923744]}
Body                : Information about John Doe's grade.
MD5ofBody           : ea572796e3c231f974fe75d89EXAMPLE
MD5ofMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
MessageAttributes   : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
  [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId           : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle       : AQEBpfGp...20Q5cg==
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell Cmdlet Reference (V4) 中的 [ReceiveMessage](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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

```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [ReceiveMessage](#)。

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

require 'aws-sdk-sqs'
require 'aws-sdk-sts'

```

```

# Receives messages in a queue in Amazon Simple Queue Service (Amazon SQS).
#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param max_number_of_messages [Integer] The maximum number of messages
#   to receive. This number must be 10 or less. The default is 10.
# @example
#   receive_messages(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     10
#   )
def receive_messages(sqs_client, queue_url, max_number_of_messages = 10)
  if max_number_of_messages > 10
    puts 'Maximum number of messages to receive must be 10 or less. ' \
      'Stopping program.'
    return
  end

  response = sqs_client.receive_message(
    queue_url: queue_url,
    max_number_of_messages: max_number_of_messages
  )

  if response.messages.count.zero?
    puts 'No messages to receive, or all messages have already ' \
      'been previously received.'
    return
  end

  response.messages.each do |message|
    puts '-' * 20
    puts "Message body: #{message.body}"
    puts "Message ID:   #{message.message_id}"
  end
end

rescue StandardError => e
  puts "Error receiving messages: #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'

```

```

max_number_of_messages = 10

sts_client = Aws::STS::Client.new(region: region)

# For example:
# 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

sqs_client = Aws::SQS::Client.new(region: region)

puts "Receiving messages from queue '#{queue_name}'..."

receive_messages(sqs_client, queue_url, max_number_of_messages)
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__

```

- 如需 API 詳細資訊，請參閱適用於 Ruby 的 AWS SDK 《API 參考》中的 [ReceiveMessage](#)。

Rust

SDK for Rust

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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(())
}

```

- 如需 API 詳細資訊，請參閱《適用於 AWS Rust 的 SDK API 參考》中的 [ReceiveMessage](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

從 Amazon SQS 佇列接收訊息。

```

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.

```

使用長輪詢支援從 Amazon SQS 佇列接收訊息。

```

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.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 [ReceiveMessage](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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("---")  
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [ReceiveMessage](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配使用 **RemovePermission** 與 CLI

下列程式碼範例示範如何使用 RemovePermission。

CLI

AWS CLI

移除許可

此範例會從指定的佇列中移除具有指定標籤的許可。

命令：

```
aws sqs remove-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessageFromMyQueue
```

輸出：

```
None.
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [RemovePermission](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會從指定的佇列移除具有指定標籤的許可設定。

```
Remove-SQSPermission -Label SendMessageFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [RemovePermission](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

SendMessage 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 SendMessage。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [使用 S3 管理大型訊息](#)

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立 Amazon SQS 佇列並傳送訊息給該佇列。

```
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;
}
}

```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [SendMessage](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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();
}
```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [SendMessage](#)。

CLI

AWS CLI

傳送訊息

此範例會將具有指定訊息內文、延遲期間和訊息屬性的訊息傳送至指定的佇列。

命令：

```
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
```

輸入檔案 (send-message.json)：

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

輸出：

```
{
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [SendMessage](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

SendMessage 操作的兩個範例如下：

- 傳送包含內文和延遲的訊息
- 使用內文和訊息屬性傳送訊息

傳送包含內文和延遲的訊息。

```
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);
    }
}
}

```

傳送包含內文和訊息屬性的訊息。

```

/**
 * <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(thumbnailPath)))
                .dataType("Binary.jpg").build()
            );
        } catch (IOException e) {
            LOGGER.error("An I/O exception occurred reading thumbnail image: {}",
                e.getMessage(), e);
            throw new RuntimeException(e);
        }

        SendMessageRequest request = SendMessageRequest.builder()
            .queueUrl(queueUrl)
            .messageBody("Hello SQS")
            .messageAttributes(messageAttributeMap)
            .build();

        try {
            SendMessageResponse sendMessageResponse =
                SQS_CLIENT.sendMessage(request);
            LOGGER.info("Message ID: {}", sendMessageResponse.messageId());
        } catch (SqsException e) {
            LOGGER.error("Exception occurred sending message: {}",
                e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [SendMessage](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

傳送訊息至 Amazon SQS 佇列。

```
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;
};
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/sqs-examples-send-receive-messages.html#sqs-examples-send-receive-messages-sending>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [SendMessage](#)。

SDK for JavaScript (v2)

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

傳送訊息至 Amazon SQS 佇列。

```
// 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);
  }
});
```

- 如需詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK 開發人員指南》<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-send-receive-messages.html#sqs-examples-send-receive-messages-sending>。
- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [SendMessage](#)。

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
suspend fun sendMessages(
    queueUrlVal: String,
    message: String,
) {
    println("Sending multiple messages")
    println("\nSend message")
    val sendRequest =
        SendMessageRequest {
            queueUrl = queueUrlVal
            messageBody = message
            delaySeconds = 10
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->  
        sqsClient.sendMessageBatch(sendMessageBatchRequest)  
        println("Batch message were successfully sent.")  
    }  
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Kotlin 的 SDK API 參考》中的 [SendMessage](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會將具有指定屬性和訊息內文的訊息傳送至指定的佇列，且訊息傳遞延遲 10 秒。

```
$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
```

輸出：

MD5ofMessageAttributes	MD5ofMessageBody	MessageId
-----	-----	-----
1d3e51347bc042efbdf6dda31EXAMPLE c739-4d0c-818b-1820eEXAMPLE	51b0a3256d59467f973009b73EXAMPLE	c35fed8f-

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [SendMessage](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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

```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [SendMessage](#)。

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

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__
```

- 如需 API 詳細資訊，請參閱適用於 Ruby 的 AWS SDK 《API 參考》中的 [SendMessage](#)。

Rust

SDK for Rust

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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(())
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Rust 的 SDK API 參考》中的 [SendMessage](#)。

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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.
```

- 如需 API 詳細資訊，請參閱《適用於 AWS SAP ABAP 的 SDK API 參考》中的 [SendMessage](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

SendMessageBatch 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 SendMessageBatch。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [傳送和接收批次的訊息](#)

CLI

AWS CLI

以批次形式傳送多則訊息

此範例會將具有指定訊息內文、延遲期間和訊息屬性的 2 則訊息傳送至指定的佇列。

命令：

```
aws sqs send-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-batch.json
```

輸入檔案 (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"
      }
    }
  }
]

```

輸出：

```

{
  "Successful": [
    {
      "MD5OfMessageBody": "203c4a38...7943237e",
      "MD5OfMessageAttributes": "10809b55...baf283ef",
      "Id": "FuelReport-0001-2015-09-16T140731Z",
      "MessageId": "d175070c-d6b8-4101-861d-adeb3EXAMPLE"
    },
    {

```

```
"MD5fMessageBody": "2cf0159a...c1980595",
"MD5fMessageAttributes": "55623928...ae354a25",
"Id": "FuelReport-0002-2015-09-16T140930Z",
"MessageId": "f9b7d55d-0570-413e-b9c5-a9264EXAMPLE"
}
]
}
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [SendMessageBatch](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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);
```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [SendMessageBatch](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例會將 2 則具有指定屬性和訊息內文的訊息傳送至指定的佇列。第一個訊息的交付延遲 15 秒，第二個訊息的交付延遲 10 秒。

```
$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
```

輸出：

Failed	Successful
-----	-----
{}	{FirstMessage, SecondMessage}

- 如需 API 詳細資訊，請參閱AWS Tools for PowerShell 《Cmdlet Reference (V4)》中的 [SendMessageBatch](#)。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Python (Boto3) 的 SDK API 參考》中的 [SendMessageBatch](#)。

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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
```

- 如需 API 詳細資訊，請參閱適用於 Ruby 的 AWS SDK 《API 參考》中的 [SendMessageBatch](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

SetQueueAttributes 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 SetQueueAttributes。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- [將訊息發佈至佇列](#)

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

設定主題佇列的政策屬性。

```

/// <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;
}

```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 [SetQueueAttributes](#)。

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```

    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();
}

```

設定無效信件佇列。

```

    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();
}

```

設定 Amazon SQS 佇列以使用長輪詢。

```

    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();
}

```

- 如需 API 詳細資訊，請參閱適用於 C++ 的 AWS SDK 《API 參考》中的 [SetQueueAttributes](#)。

CLI

AWS CLI

設定佇列屬性

此範例會將指定的佇列設定為 10 秒的交付延遲、128 KB (128 KB * 1,024 位元組) 的訊息大小上限、3 天 (3 天 * 24 小時 * 60 分鐘 * 60 秒) 的訊息保留期間、20 秒的接收訊息等待時間，以及 60 秒的預設可見性逾時。此範例也會將指定的無效字母佇列與 1,000 則訊息的最大接收計數建立關聯。

命令：

```
aws sqs set-queue-attributes --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-attributes.json
```

輸入檔案 (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"
```

```
}
```

輸出：

```
None.
```

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 [SetQueueAttributes](#)。

Go

SDK for Go V2

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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

```

- 如需 API 詳細資訊，請參閱適用於 Go 的 AWS SDK 《API 參考》中的 [SetQueueAttributes](#)。

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

設定 Amazon SQS 以使用自訂 KMS 金鑰來使用伺服器端加密 (SSE)。

```

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();
        }
    }
}

```

- 如需 API 詳細資訊，請參閱AWS SDK for Java 2.x 《API 參考》中的 [SetQueueAttributes](#)。

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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;
};
```

設定 Amazon SQS 佇列以使用長輪詢。

```
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;
};
```

設定無效信件佇列。

```
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;
};
```

- 如需 API 詳細資訊，請參閱適用於 JavaScript 的 AWS SDK 《API 參考》中的 [SetQueueAttributes](#)。

PowerShell

PowerShell V4 的工具

範例 1：此範例示範如何設定將佇列訂閱至 SNS 主題的政策。當訊息發佈至主題時，訊息會傳送至訂閱的佇列。

```
# 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 }
```

範例 2：此範例會設定指定佇列的指定屬性。

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
"131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 如需 API 詳細資訊，請參閱 AWS Tools for PowerShell Cmdlet Reference (V4) 中的 [SetQueueAttributes](#)。

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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.")
}
```

- 如需 API 詳細資訊，請參閱《適用於 AWS Swift 的 SDK API 參考》中的 [SetQueueAttributes](#)。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDKs Amazon SQS 案例

下列程式碼範例示範如何在 Amazon SQS AWS SDKs 中實作常見案例。這些案例說明如何透過在 Amazon SQS 中呼叫多個函數或與其他函數結合，來完成特定任務 AWS 服務。每個案例均包含完整原始碼的連結，您可在連結中找到如何設定和執行程式碼的相關指示。

案例的目標是獲得中等水平的經驗，協助您了解內容中的服務動作。

範例

- [建立使用 Amazon SQS 傳送和擷取訊息的 Web 應用程式](#)
- [使用 Step Functions 建立傳訊應用程式](#)
- [建立 Amazon Textract Explorer 應用程式](#)
- [使用 AWS SDK 建立和發佈至 FIFO Amazon SNS 主題](#)
- [使用 AWS SDK 透過 Amazon Rekognition 偵測影片中的人物和物件](#)
- [透過 SDK 使用 Amazon S3 管理大型 Amazon SQS 訊息 Amazon S3 AWS](#)
- [使用 SDK 接收和處理 Amazon S3 事件通知 AWS](#)
- [使用 SDK 將 Amazon SNS 訊息發佈至 Amazon SQS 佇列 AWS](#)
- [使用 AWS SDK 透過 Amazon SQS 傳送和接收批次訊息](#)
- [使用適用於 .NET AWS 的訊息處理架構來發佈和接收 Amazon SQS 訊息](#)
- [使用 Amazon SQS Java Messaging Library 來使用 Amazon SQS 的 Java Message Service \(JMS\) 界面](#)
- [使用 AWS SDK 處理佇列標籤和 Amazon SQS](#)

建立使用 Amazon SQS 傳送和擷取訊息的 Web 應用程式

下列程式碼範例示範如何使用 Amazon SQS 建立簡訊應用程式。

Java

適用於 Java 2.x 的 SDK

示範如何使用 Amazon SQS API 來開發傳送和擷取訊息的 Spring REST API。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK for Kotlin

示範如何使用 Amazon SQS API 來開發傳送和擷取訊息的 Spring REST API。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Comprehend
- Amazon SQS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 Step Functions 建立傳訊應用程式

下列程式碼範例示範如何建立 AWS Step Functions 訊息應用程式，從資料庫資料表擷取訊息記錄。

Python

SDK for Python (Boto3)

示範如何使用 適用於 Python (Boto3) 的 AWS SDK 搭配 AWS Step Functions 來建立訊息應用程式，從 Amazon DynamoDB 資料表擷取訊息記錄，並使用 Amazon Simple Queue Service (Amazon SQS) 傳送它們。狀態機器會與 AWS Lambda 函數整合，以掃描資料庫是否有未傳送的訊息。

- 建立從 Amazon DynamoDB 資料表擷取和更新訊息記錄的狀態機器。
- 更新狀態機器定義，以便也向 Amazon Simple Queue Service (Amazon SQS) 傳送訊息。
- 開始和停用狀態機器執行。
- 使用服務整合從狀態機器連接至 Lambda、DynamoDB 和 Amazon SQS。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- DynamoDB
- Lambda

- Amazon SQS
- Step Functions

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

建立 Amazon Textract Explorer 應用程式

下列程式碼範例示範如何透過互動式應用程式探索 Amazon Textract 輸出。

JavaScript

適用於 JavaScript (v3) 的 SDK

示範如何使用 適用於 JavaScript 的 AWS SDK 建置 React 應用程式，該應用程式使用 Amazon Textract 從文件映像擷取資料，並將其顯示在互動式網頁中。此範例會在 Web 瀏覽器中執行，且登入資料需要經過驗證的 Amazon Cognito 身分。它使用 Amazon Simple Storage Service (Amazon S3 進行儲存，對於通知，它會輪詢訂閱 Amazon Simple Notification Service (Amazon SNS)) 主題的 Amazon Simple Queue Service (Amazon SQS) 佇列。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python (Boto3)

顯示如何使用 適用於 Python (Boto3) 的 AWS SDK 搭配 Amazon Textract 來偵測文件映像中的文字、表單和資料表元素。輸入影像和 Amazon Textract 輸出會顯示在 Tkinter 應用程式中，可讓您探索偵測到的元素。

- 將文件影像提交到 Amazon Textract，並探索偵測到元素的輸出。

- 將影像直接傳送至 Amazon Textract 或透過 Amazon Simple Storage Service (Amazon S3) 儲存貯體。
- 使用非同步 API 可以在任務完成時啟動將通知發布到 Amazon Simple Notification Service (Amazon SNS) 主題的任務。
- 輪詢 Amazon Simple Queue Service (Amazon SQS) 佇列以取得任務完成訊息並顯示結果。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 建立和發佈至 FIFO Amazon SNS 主題

下列程式碼範例示範如何建立並發布到 FIFO Amazon SNS 主題。

Java

適用於 Java 2.x 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

此範例

- 建立一個 Amazon SNS FIFO 主題、兩個 Amazon SQS FIFO 佇列和一個標準佇列。
- 訂閱佇列到主題並向該主題發布訊息。

[測試](#)可驗證每個佇列的訊息接收狀況。[完整範例](#)也會顯示存取政策的新增，並在最後刪除資源。

```
public class PriceUpdateExample {
    public final static SnsClient snsClient = SnsClient.create();
    public final static SqsClient sqsClient = SqsClient.create();

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName> <analyticsQueueName>\n\n" +
            "Where:\n" +
            "    fifoTopicName - The name of the FIFO topic that you want to create. \n\n" +
            "    wholesaleQueueARN - The name of a SQS FIFO queue that will be created for the wholesale consumer. \n\n" +
            "    retailQueueARN - The name of a SQS FIFO queue that will be created for the retail consumer. \n\n" +
            "    analyticsQueueARN - The name of a SQS standard queue that will be created for the analytics consumer. \n\n";
        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        final String fifoTopicName = args[0];
        final String wholeSaleQueueName = args[1];
        final String retailQueueName = args[2];
        final String analyticsQueueName = args[3];

        // For convenience, the QueueData class holds metadata about a queue: ARN, URL,
        // name and type.
        List<QueueData> queues = List.of(
            new QueueData(wholeSaleQueueName, QueueType.FIFO),
            new QueueData(retailQueueName, QueueType.FIFO),
            new QueueData(analyticsQueueName, QueueType.Standard));

        // Create queues.
        createQueues(queues);

        // Create a topic.
```

```
String topicARN = createFIFOTopic(fifoTopicName);

// Subscribe each queue to the topic.
subscribeQueues(queues, topicARN);

// Allow the newly created topic to send messages to the queues.
addAccessPolicyToQueuesFINAL(queues, topicARN);

// Publish a sample price update message with payload.
publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}",
"Consumables");

// Clean up resources.
deleteSubscriptions(queues);
deleteQueues(queues);
deleteTopic(topicARN);
}

public static String createFIFOTopic(String topicName) {
    try {
        // Create a FIFO topic by using the SNS service client.
        Map<String, String> topicAttributes = Map.of(
            "FifoTopic", "true",
            "ContentBasedDeduplication", "false",
            "FifoThroughputScope", "MessageGroup");

        CreateTopicRequest topicRequest = CreateTopicRequest.builder()
            .name(topicName)
            .attributes(topicAttributes)
            .build();

        CreateTopicResponse response = snsClient.createTopic(topicRequest);
        String topicArn = response.topicArn();
        System.out.println("The topic ARN is" + topicArn);

        return topicArn;
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

```

    public static void subscribeQueues(List<QueueData> queues, String topicARN) {
        queues.forEach(queue -> {
            SubscribeRequest subscribeRequest = SubscribeRequest.builder()
                .topicArn(topicARN)
                .endpoint(queue.queueARN)
                .protocol("sqs")
                .build();

            // Subscribe to the endpoint by using the SNS service client.
            // Only Amazon SQS queues can receive notifications from an Amazon
SNS FIFO
            // topic.
            SubscribeResponse subscribeResponse =
snsClient.subscribe(subscribeRequest);
            System.out.println("The queue [" + queue.queueARN + "] subscribed to
the topic [" + topicARN + "]);
            queue.subscriptionARN = subscribeResponse.subscriptionArn();
        });
    }

    public static void publishPriceUpdate(String topicArn, String payload, String
groupId) {

        try {
            // Create and publish a message that updates the wholesale price.
            String subject = "Price Update";
            String dedupId = UUID.randomUUID().toString();
            String attributeName = "business";
            String attributeValue = "wholesale";

            MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
                .dataType("String")
                .stringValue(attributeValue)
                .build();

            Map<String, MessageAttributeValue> attributes = new HashMap<>();
            attributes.put(attributeName, msgAttValue);
            PublishRequest pubRequest = PublishRequest.builder()
                .topicArn(topicArn)
                .subject(subject)
                .message(payload)
                .messageGroupId(groupId)
                .messageDeduplicationId(dedupId)
                .messageAttributes(attributes)

```

```
        .build();

        final PublishResponse response = snsClient.publish(pubRequest);
        System.out.println(response.messageId());
        System.out.println(response.sequenceNumber());
        System.out.println("Message was published to " + topicArn);

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
 - [CreateTopic](#)
 - [發布](#)
 - [Subscribe](#)

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立 Amazon SNS FIFO 主題，訂閱 Amazon SQS FIFO 和標準佇列到該主題，並向該主題發布訊息。

```
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

```

- 如需 API 的詳細資訊，請參閱 AWS SDK for Python (Boto3) API Reference 中的下列主題。
- [CreateTopic](#)
- [發布](#)
- [Subscribe](#)

SAP ABAP

適用於 SAP ABAP 的開發套件

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立 FIFO 主題、將 Amazon SQS FIFO 佇列訂閱至主題，然後將訊息發佈至 Amazon SNS 主題。

```
" Creates a FIFO topic. "
DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsmmap_w=>tt_topicattributesmap.
DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsmmap_w=>ts_topicattributesmap_maprow.
ls_tpc_attributes-key = 'FifoTopic'.
ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsmmap_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_snsstopiclimitexcdex.
        MESSAGE 'Unable to create more topics. You have reached the maximum
number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
to an SNS FIFO topic. "
    TRY.
        DATA(lo_subscribe_result) = lo_sns->subscribe(
            iv_topicarn = lv_topic_arn
            iv_protocol = 'sqs'
            iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.
        "
        ov_subscription_arn is returned for testing purposes. "
        MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
    CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    CATCH /aws1/cx_snssubscriptionlmt00.
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
'String'

iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.

        DATA(lo_result) = lo_sns->publish(
            iv_topicarn = lv_topic_arn
            iv_message = 'The price of your mobile plan has been increased from
$19 to $23'
            iv_subject = 'Changes to mobile plan'

```

```
        iv_messagegroupid = 'Update-2'
        iv_messagededuplicationid = 'Update-2.1'
        it_messageattributes = lt_msg_attributes ).
    ov_message_id = lo_result->get_messageid( ).
ov_message_id is returned for testing purposes. "
    MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的下列主題。
 - [CreateTopic](#)
 - [發布](#)
 - [Subscribe](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 透過 Amazon Rekognition 偵測影片中的人物和物件

下列程式碼範例示範如何使用 Amazon Rekognition 偵測映像中的人物和物件。

Java

適用於 Java 2.x 的 SDK

示範如何使用 Amazon Rekognition Java API 來建立應用程式，以偵測位於 Amazon Simple Storage Service (Amazon S3) 儲存貯體的映像中的人臉和物件。此應用程式可使用 Amazon Simple Email Service (Amazon SES) 向管理員傳送包含結果的電子郵件通知。

如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS

- Amazon SQS

Python

適用於 Python (Boto3) 的 SDK

使用 Amazon Rekognition 透過啟動非同步偵測任務來偵測映像中的人臉、物件和人物。此範例也會設定 Amazon Rekognition 以在任務完成時通知 Amazon Simple Notification Service (Amazon SNS) 主題，並訂閱 Amazon Simple Queue Service (Amazon SQS) 佇列到該主題。當佇列收到有關任務的訊息時，會擷取任務並輸出結果。

這個範例在 GitHub 上的檢視效果最佳。如需完整的原始碼和如何設定及執行的指示，請參閱 [GitHub](#) 上的完整範例。

此範例中使用的服務

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

透過 SDK 使用 Amazon S3 管理大型 Amazon SQS 訊息 Amazon S3 AWS

下列程式碼範例示範如何使用 Amazon SQS 延伸用戶端程式庫來處理大型 Amazon SQS 訊息。

Java

適用於 Java 2.x 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。


```
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;
    }
}
```

- 如需詳細資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-s3-messages.html>。
- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
 - [CreateBucket](#)
 - [PutBucketLifecycleConfiguration](#)
 - [ReceiveMessage](#)
 - [SendMessage](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 SDK 接收和處理 Amazon S3 事件通知 AWS

下列程式碼範例示範如何以物件導向的方式使用 S3 事件通知。

Java

適用於 Java 2.x 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

此範例示範如何使用 Amazon SQS 處理 S3 通知事件。

```
/**
```

```

    * 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);
}
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
 - [DeleteMessageBatch](#)
 - [GetQueueAttributes](#)
 - [PutBucketNotificationConfiguration](#)
 - [ReceiveMessage](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 SDK 將 Amazon SNS 訊息發佈至 Amazon SQS 佇列 AWS

下列程式碼範例示範如何：

- 建立主題 (FIFO 或非 FIFO)。
- 為主題訂閱多個佇列，並提供套用篩選條件的選擇。
- 發佈訊息至主題。
- 輪詢佇列以獲取收到的訊息。

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

在命令提示中執行互動式案例。

```
/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
"sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
```



```

{
    // Set up dependency injection for Amazon EventBridge.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonSQS>()
                .AddAWSService<IAmazonSimpleNotificationService>()
                .AddTransient<SNSWrapper>()
                .AddTransient<SQSWrapper>()
        )
        .Build();

    ServicesSetup(host);
    PrintDescription();

    await RunScenario();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
    SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
}

/// <summary>
/// Run the scenario for working with topics and queues.
/// </summary>
/// <returns>True if successful.</returns>
public static async Task<bool> RunScenario()
{
    try
    {
        await SetupTopic();
    }
}

```

```

        await SetupQueues();

        await PublishMessages();

        foreach (var queueUrl in _queueUrls)
        {
            var messages = await PollForMessages(queueUrl);
            if (messages.Any())
            {
                await DeleteMessages(queueUrl, messages);
            }
        }
        await CleanupResources();

        Console.WriteLine("Messaging with topics and queues scenario is
complete.");
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await CleanupResources();
        Console.WriteLine(new string('-', 80));
        return false;
    }
}

/// <summary>
/// Print a description for the tasks in the scenario.
/// </summary>
/// <returns>Async task.</returns>
private static void PrintDescription()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Welcome to messaging with topics and queues.");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"In this scenario, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
        $"{r\nYou can select from several options for
configuring the topic and the subscriptions for the 2 queues." +

```

```

        $"\\r\\nYou can then post to the topic and see the
results in the queues.\\r\\n");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Set up the SNS topic to be used with the queues.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<string> SetupTopic()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-
Out)." +
            $"\\r\\nFIFO topics deliver messages in order and support
deduplication and message filtering." +
            $"\\r\\nYou can then post to the topic and see the
results in the queues.\\r\\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO
topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ",
"example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended
to the topic name.\\r\\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic,
deduplication is supported." +
                $"\\r\\nDeduplication IDs are either set in the
message or automatically generated " +
                $"\\r\\nfrom content using a hash function.\\r\\n" +
                $"\\r\\nIf a message is successfully published to an
SNS FIFO topic, any message " +
                $"\\r\\npublished and determined to have the same
deduplication ID, " +
                $"\\r\\nwithin the five-minute deduplication
interval, is accepted but not delivered.\\r\\n" +

```

```

        $"\\r\\nFor more information about deduplication, " +
        $"\\r\\nsee https://docs.aws.amazon.com/sns/latest/
dg/fifo-message-dedup.html.");

        _useContentBasedDeduplication = GetYesNoResponse("Use content-based
deduplication instead of entering a deduplication ID?");
        Console.WriteLine(new string('-', 80));
    }

    _topicArn = await SnsWrapper.CreateTopicWithName(_topicName,
_useFifoTopic, _useContentBasedDeduplication);

    Console.WriteLine($"Your new topic with the name {_topicName}" +
        $"\\r\\nand Amazon Resource Name (ARN) {_topicArn}" +
        $"\\r\\nhas been created.\\r\\n");

    Console.WriteLine(new string('-', 80));
    return _topicArn;
}

/// <summary>
/// Set up the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task SetupQueues()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

    // Repeat this section for each queue.
    for (int i = 0; i < _queueCount; i++)
    {
        var queueName = GetUserResponse("Enter a name for an Amazon SQS
queue: ", $"example-queue-{i}");
        if (_useFifoTopic)
        {
            // Only explain this once.
            if (i == 0)
            {
                Console.WriteLine(
                    "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
            }
        }
    }
}

```

```

        var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
            _useFifoTopic);

        _queueUrls[i] = queueUrl;

        Console.WriteLine($"Your new queue with the name {queueName}" +
            $"\r\nand queue URL {queueUrl}" +
            $"\r\nhas been created.\r\n");

        if (i == 0)
        {
            Console.WriteLine(
                $"The queue URL is used to retrieve the queue ARN,\r\n" +
                $"which is used to create a subscription.");
            Console.WriteLine(new string('-', 80));
        }

        var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

        if (i == 0)
        {
            Console.WriteLine(
                $"An AWS Identity and Access Management (IAM) policy must" +
                $"be attached to an SQS queue, enabling it to receive\r\n" +
                $"messages from an SNS topic");
        }

        await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
            queueUrl);

        await SetupFilters(i, queueArn, queueName);
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up filters with user options for a queue.
/// </summary>
/// <param name="queueCount">The number of this queue.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="queueName">The name of the queue.</param>

```

```
/// <returns>Async Task.</returns>
public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
{
    if (_useFifoTopic)
    {
        Console.WriteLine(new string('-', 80));
        // Only explain this once.
        if (queueCount == 0)
        {
            Console.WriteLine(
                "Subscriptions to a FIFO topic can have filters." +
                "If you add a filter to this subscription, then only the
filtered messages " +
                "will be received in the queue.");

            Console.WriteLine(
                "For information about message filtering, " +
                "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

            Console.WriteLine(
                "For this example, you can filter messages by a " +
                "TONE attribute.");
        }

        var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
subscription to the topic?");

        string? filterPolicy = null;
        if (useFilter)
        {
            filterPolicy = CreateFilterPolicy();
        }
        var subscriptionArn = await
SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
queueArn);
        _subscriptionArns[queueCount] = subscriptionArn;

        Console.WriteLine(
            $"The queue {queueName} has been subscribed to the topic
{_topicName} " +
            $"with the subscription ARN {subscriptionArn}");
        Console.WriteLine(new string('-', 80));
    }
}
```

```

    }
}

/// <summary>
/// Use user input to create a filter policy for a subscription.
/// </summary>
/// <returns>The serialized filter policy.</returns>
public static string CreateFilterPolicy()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        $"You can filter messages by one or more of the following" +
        $"TONE attributes.");

    List<string> filterSelections = new List<string>();

    var selectionNumber = 0;
    do
    {
        Console.WriteLine(
            $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"\\t{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", filterSelections.Any() ? "0" :
"1");
        int.TryParse(selection, out selectionNumber);
        if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
        {
            filterSelections.Add(_tones[selectionNumber - 1]);
        }
    } while (selectionNumber != 0);

    var filters = new Dictionary<string, List<string>>
    {
        { "tone", filterSelections }
    };
    string filterPolicy = JsonSerializer.Serialize(filters);
    return filterPolicy;
}

```

```
/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                             "\r\nAll messages within the same group will be
received in the order " +
                             "they were published.");

            Console.WriteLine();
            var messageId = GetUserResponse("Enter a message group ID
for this message:", "1");

            if (!_useContentBasedDeduplication)
            {
                Console.WriteLine("Because you are not using content-based
deduplication, " +
                                 "you must enter a deduplication ID.");

                Console.WriteLine("Enter a deduplication ID for this
message.");
                deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
            }

            if (GetYesNoResponse("Add an attribute to this message?"))
            {
```



```

        Console.WriteLine("Enter a number for an attribute.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageID = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?",
false);
}
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
        "\r\nPress any key to continue.");
    if (UseConsole)
    {
        Console.ReadLine();
    }

    var moreMessages = true;
    var messages = new List<Message>();

```

```
        while (moreMessages)
        {
            var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl,
10);

            moreMessages = newMessages.Any();
            if (moreMessages)
            {
                messages.AddRange(newMessages);
            }
        }

        Console.WriteLine($"{messages.Count} message(s) were received by the
queue at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                               $"{"\n\t{message.Body}"}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message>
messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
```

```
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Clean up resources.");

    try
    {
        foreach (var queueUrl in _queueUrls)
        {
            if (!string.IsNullOrEmpty(queueUrl))
            {
                var deleteQueue =
                    GetYesNoResponse($"Delete queue with url {queueUrl}?");
                if (deleteQueue)
                {
                    await SqsWrapper.DeleteQueueByUrl(queueUrl);
                }
            }

            foreach (var subscriptionArn in _subscriptionArns)
            {
                if (!string.IsNullOrEmpty(subscriptionArn))
                {
                    await SnsWrapper.UnsubscribeByArn(subscriptionArn);
                }
            }

            var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
            if (deleteTopic)
            {
                await SnsWrapper.DeleteTopicByArn(_topicArn);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
```

```

    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question, bool defaultAnswer =
true)
    {
        if (UseConsole)
        {
            Console.WriteLine(question);
            var ynResponse = Console.ReadLine();
            var response = ynResponse != null &&
                ynResponse.Equals("y",
                    StringComparison.InvariantCultureIgnoreCase);

            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }

    /// <summary>
    /// Helper method to get a string response from the user through the console.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <param name="defaultAnswer">Optional default answer to use.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static string GetUserResponse(string question, string defaultAnswer)
    {
        if (UseConsole)
        {
            var response = "";
            while (string.IsNullOrEmpty(response))
            {
                Console.WriteLine(question);
                response = Console.ReadLine();
            }
            return response;
        }
        // If not using the console, use the default.
        return defaultAnswer;
    }
}

```

建立包裝 Amazon SQS 操作的類別。

```
/// <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;
    }
}

```

建立包裝 Amazon SNS 操作的類別。

```

    /// <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="amazonSNS">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;  
}  
}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。

- [CreateQueue](#)
- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [發布](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

C++

SDK for C++

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
Aws::Client::ClientConfiguration clientConfig;  
// Optional: Set to the AWS Region (overrides config file).  
// clientConfig.region = "us-east-1";
```

```

//! Workflow for messaging with topics and queues using Amazon SNS and Amazon
SQS.
/*!
\param clientConfig Aws client configuration.
\return bool: Successful completion.
*/
bool AwsDoc::TopicsAndQueues::messagingWithTopicsAndQueues(
    const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << "Welcome to messaging with topics and queues." << std::endl;
    printAsterisksLine();
    std::cout << "In this workflow, you will create an SNS topic and subscribe "
        << NUMBER_OF_QUEUES <<
        " SQS queues to the topic." << std::endl;
    std::cout
        << "You can select from several options for configuring the topic and
the subscriptions for the "
        << NUMBER_OF_QUEUES << " queues." << std::endl;
    std::cout << "You can then post to the topic and see the results in the
queues."
        << std::endl;

    Aws::SNS::SNSClient snsClient(clientConfiguration);

    printAsterisksLine();

    std::cout << "SNS topics can be configured as FIFO (First-In-First-Out)."
        << std::endl;
    std::cout
        << "FIFO topics deliver messages in order and support deduplication
and message filtering."
        << std::endl;
    bool isFifoTopic = askYesNoQuestion(
        "Would you like to work with FIFO topics? (y/n) ");

    bool contentBasedDeduplication = false;
    Aws::String topicName;
    if (isFifoTopic) {
        printAsterisksLine();
        std::cout << "Because you have chosen a FIFO topic, deduplication is
supported."
            << std::endl;
        std::cout
            << "Deduplication IDs are either set in the message or
automatically generated "

```

```

        << "from content using a hash function." << std::endl;
    std::cout
        << "If a message is successfully published to an SNS FIFO topic,
any message "
        << "published and determined to have the same deduplication ID, "
        << std::endl;
    std::cout
        << "within the five-minute deduplication interval, is accepted
but not delivered."
        << std::endl;
    std::cout
        << "For more information about deduplication, "
        << "see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html."
        << std::endl;
    contentBasedDeduplication = askYesNoQuestion(
        "Use content-based deduplication instead of entering a
deduplication ID? (y/n) ");
}

printAsterisksLine();

Aws::SQS::SQSClient sqsClient(clientConfiguration);
Aws::Vector<Aws::String> queueURLS;
Aws::Vector<Aws::String> subscriptionARNs;

Aws::String topicARN;
{
    topicName = askQuestion("Enter a name for your SNS topic. ");

    // 1. Create an Amazon SNS topic, either FIFO or non-FIFO.
    Aws::SNS::Model::CreateTopicRequest request;

    if (isFifoTopic) {
        request.AddAttributes("FifoTopic", "true");
        if (contentBasedDeduplication) {
            request.AddAttributes("ContentBasedDeduplication", "true");
        }
        topicName = topicName + FIFO_SUFFIX;

        std::cout
            << "Because you have selected a FIFO topic, '.fifo' must be
appended to the topic name."
            << std::endl;
    }
}

```

```

    }

    request.SetName(topicName);

    Aws::SNS::Model::CreateTopicOutcome outcome =
snsClient.CreateTopic(request);

    if (outcome.IsSuccess()) {
        topicARN = outcome.GetResult().GetTopicArn();
        std::cout << "Your new topic with the name '" << topicName
                    << "' and the topic Amazon Resource Name (ARN) " <<
std::endl;
        std::cout << "'" << topicARN << "' has been created." << std::endl;
    }
    else {
        std::cerr << "Error with TopicsAndQueues::CreateTopic. "
                    << outcome.GetError().GetMessage()
                    << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }
}

printAsterisksLine();

std::cout << "Now you will create " << NUMBER_OF_QUEUES
          << " SQS queues to subscribe to the topic." << std::endl;
Aws::Vector<Aws::String> queueNames;
bool filteringMessages = false;
bool first = true;
for (int i = 1; i <= NUMBER_OF_QUEUES; ++i) {
    Aws::String queueURL;
    Aws::String queueName;
    {
        printAsterisksLine();
        std::ostringstream ostream;
        ostream << "Enter a name for " << (first ? "an" : "the next")

```



```

        << " SQS queue. ";
        queueName = askQuestion(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 ostream;
        ostream << "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();  
}
```

- 如需 API 詳細資訊，請參閱《適用於 C++ 的 AWS SDK API 參考》中的下列主題。

- [CreateQueue](#)
- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [發布](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Go

SDK for Go V2

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

在命令提示中執行互動式案例。

```
import (  
    "context"  
    "encoding/json"  
    "fmt"  
    "log"  
    "strings"
```

```

"topics_and_queues/actions"

"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/service/sns"
"github.com/aws/aws-sdk-go-v2/service/sqs"
"github.com/aws/aws-sdk-go-v2/service/sqs/types"
"github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
    Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions
// so that
// they are simpler to read and understand.
type ScenarioRunner struct {
    questioner demotools.IQuestioner
    snsActor   *actions.SnsActions
    sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string,
bool, bool) {
    log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or
standard.\n" +
        "FIFO topics deliver messages in order and support deduplication and message
filtering.")
    isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
topics? (y/n) ", "y")

    contentBasedDeduplication := false
    if isFifoTopic {
        log.Println(strings.Repeat("-", 88))
        log.Println("Because you have chosen a FIFO topic, deduplication is supported.
\n" +
            "Deduplication IDs are either set in the message or are automatically
generated\n" +

```

```

    "from content using a hash function. If a message is successfully published to
\n" +
    "an SNS FIFO topic, any message published and determined to have the same\n" +
    "deduplication ID, within the five-minute deduplication interval, is accepted
\n" +
    "but not delivered. For more information about deduplication, see:\n" +
    "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
    contentBasedDeduplication = runner.questioner.AskBool(
    "\nDo you want to use content-based deduplication instead of entering a
deduplication ID? (y/n) ", "y")
}
log.Println(strings.Repeat("-", 88))

topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
if isFifoTopic {
    topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
    log.Printf("Because you have selected a FIFO topic, '%v' must be appended to
\n"+
    "the topic name.", FIFO_SUFFIX)
}

topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
contentBasedDeduplication)
if err != nil {
    panic(err)
}
log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN)
\n"+
    "'%v' has been created.", topicName, topicArn)

return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
isFifoTopic bool) (string, string) {
    queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS
queue. ", ordinal))
    if isFifoTopic {
        queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
        if ordinal == "first" {
            log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
            "be appended to the queue name.\n", FIFO_SUFFIX)
        }
    }
}

```

```

queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
if err != nil {
    panic(err)
}
log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
    "'%v' has been created.", queueName, queueUrl)

return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
    ctx context.Context, queueName string, queueUrl string, topicName string,
    topicArn string, ordinal string,
    isFifoTopic bool) (string, bool) {

    queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
    if err != nil {
        panic(err)
    }
    log.Printf("The ARN of your queue is: %v.\n", queueArn)

    err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
    if err != nil {
        panic(err)
    }
    log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
        "messages to it.")
    log.Println(strings.Repeat("-", 88))

    var filterPolicy map[string][]string
    if isFifoTopic {
        if ordinal == "first" {
            log.Println("Subscriptions to a FIFO topic can have filters.\n" +
                "If you add a filter to this subscription, then only the filtered messages\n"
            +
                "will be received in the queue.\n" +
                "For information about message filtering, see\n" +
                "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
                "For this example, you can filter messages by a \"tone\" attribute.")
        }
    }

    wantFiltering := runner.questioner.AskBool(
        fmt.Sprintf("Do you want to filter messages that are sent to \"%v\" \n"+
            "from the %v topic? (y/n) ", queueName, topicName), "y")

```



```

    if wantFiltering {
        log.Println("You can filter messages by one or more of the following \"tone\"
attributes.")

        var toneSelections []string
        askAboutTones := true
        for askAboutTones {
            toneIndex := runner.questioner.AskChoice(
                "Enter the number of the tone you want to filter by:\n", ToneChoices)
            toneSelections = append(toneSelections, ToneChoices[toneIndex])
            askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
the filter? (y/n) ", "y")
        }
        log.Printf("Your subscription will be filtered to only pass the following
tones: %v\n", toneSelections)
        filterPolicy = map[string][]string{TONE_KEY: toneSelections}
    }
}

subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
filterPolicy)
if err != nil {
    panic(err)
}
log.Printf("The queue %v is now subscribed to the topic %v with the subscription
ARN %v.\n",
    queueName, topicName, subscriptionArn)

return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn
string, isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
    var message string
    var groupId string
    var dedupId string
    var toneSelection string
    publishMore := true
    for publishMore {
        groupId = ""
        dedupId = ""
        toneSelection = ""
        message = runner.questioner.Ask("Enter a message to publish: ")
        if isFifoTopic {

```

```

    log.Println("Because you are using a FIFO topic, you must set a message group
ID.\n" +
    "All messages within the same group will be received in the order they were
published.")
    groupId = runner.questioner.Ask("Enter a message group ID: ")
    if !contentBasedDeduplication {
        log.Println("Because you are not using content-based deduplication,\n" +
            "you must enter a deduplication ID.")
        dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
    }
}
if usingFilters {
    if runner.questioner.AskBool("Add a tone attribute so this message can be
filtered? (y/n) ", "y") {
        toneIndex := runner.questioner.AskChoice(
            "Enter the number of the tone you want to filter by:\n", ToneChoices)
        toneSelection = ToneChoices[toneIndex]
    }
}

err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId,
TONE_KEY, toneSelection)
if err != nil {
    panic(err)
}
log.Println(("Your message was published.))

publishMore = runner.questioner.AskBool("Do you want to publish another
message? (y/n) ", "y")
}
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
[]string) {
    log.Println("Polling queues for messages...")
    for _, queueUrl := range queueUrls {
        var messages []types.Message
        for {
            currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
            if err != nil {
                panic(err)
            }
            if len(currentMsgs) == 0 {
                break
            }
        }
    }
}

```

```

    }
    messages = append(messages, currentMsgs...)
}
if len(messages) == 0 {
    log.Printf("No messages were received by queue %v.\n", queueUrl)
} else if len(messages) == 1 {
    log.Printf("One message was received by queue %v:\n", queueUrl)

} else {
    log.Printf("%v messages were received by queue %v:\n", len(messages),
queueUrl)
}
for msgIndex, message := range messages {
    messageBody := MessageBody{}
    err := json.Unmarshal([]byte(*message.Body), &messageBody)
    if err != nil {
        panic(err)
    }
    log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
}

if len(messages) > 0 {
    log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
    err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
    if err != nil {
        panic(err)
    }
}
}
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//

```

```
// It uses a questioner from the `demotools` package to get input during the
// example.
// This package can be found in the ../../demotools folder of this repo.
func RunTopicsAndQueuesScenario(
    ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
    resources := Resources{}
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.\n" +
                "Cleaning up any resources that were created...")
            resources.Cleanup(ctx)
        }
    }()
    queueCount := 2

    log.Println(strings.Repeat("-", 88))
    log.Printf("Welcome to messaging with topics and queues.\n\n"+
        "In this scenario, you will create an SNS topic and subscribe %v SQS queues to\n"+
        "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\n"+
        "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))
}

```

定義包裝此範例中所用 Amazon SNS 動作的結構。

```

import (
    "context"
    "encoding/json"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/sns"
    "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type SnsActions struct {
    SnsClient *sns.Client
}

// CreateTopic creates an Amazon SNS topic with the specified name. You can
// optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
// based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
    isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
    var topicArn string
    topicAttributes := map[string]string{}
    if isFifoTopic {
        topicAttributes["FifoTopic"] = "true"
    }
    if contentBasedDeduplication {
        topicAttributes["ContentBasedDeduplication"] = "true"
    }
    topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
        Name:      aws.String(topicName),
        Attributes: topicAttributes,
    })
    if err != nil {
        log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
    } else {
        topicArn = *topic.TopicArn
    }

    return topicArn, err
}

// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
    _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
        TopicArn: aws.String(topicArn)})
}
```

```
    if err != nil {
        log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
    }
    return err
}

// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
// an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
// policy
// so that messages are only sent to the queue when the message has the specified
// attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
    queueArn string, filterMap map[string][]string) (string, error) {
    var subscriptionArn string
    var attributes map[string]string
    if filterMap != nil {
        filterBytes, err := json.Marshal(filterMap)
        if err != nil {
            log.Printf("Couldn't create filter policy, here's why: %v\n", err)
            return "", err
        }
        attributes = map[string]string{"FilterPolicy": string(filterBytes)}
    }
    output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
        Protocol:          aws.String("sqs"),
        TopicArn:          aws.String(topicArn),
        Attributes:        attributes,
        Endpoint:          aws.String(queueArn),
        ReturnSubscriptionArn: true,
    })
    if err != nil {
        log.Printf("Couldn't subscribe queue %v to topic %v. Here's why: %v\n",
            queueArn, topicArn, err)
    } else {
        subscriptionArn = *output.SubscriptionArn
    }

    return subscriptionArn, err
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent
// to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
// group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
// value
// filter attribute can be specified so that the message can be filtered
// according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
string, groupId string, dedupId string, filterKey string, filterValue string)
error {
    publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
aws.String(message)}
    if groupId != "" {
        publishInput.MessageGroupId = aws.String(groupId)
    }
    if dedupId != "" {
        publishInput.MessageDeduplicationId = aws.String(dedupId)
    }
    if filterKey != "" && filterValue != "" {
        publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
            filterKey: {DataType: aws.String("String"), StringValue:
aws.String(filterValue)},
        }
    }
    _, err := actor.SnsClient.Publish(ctx, &publishInput)
    if err != nil {
        log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
err)
    }
    return err
}
```

定義包裝此範例中所用 Amazon SQS 動作的結構。

```
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
}
```

清除資源。

```
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)  
    }  
}  
}
```

- 如需 API 詳細資訊，請參閱《適用於 Go 的 AWS SDK API 參考》中的下列主題。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [發布](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Java

SDK for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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)
            .numberOfMessages(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.
            .numberOfMessages(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 "";
}
}

```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
- [CreateQueue](#)

- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [發布](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

JavaScript

SDK for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

這是此案例的進入點。

```
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();
}
```



```
};
```

上述程式碼提供必要的相依性並啟動案例。下一節包含範例的大量內容。

```
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();
    }
}
}
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的下列主題。

- [CreateQueue](#)
- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [發布](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Kotlin

SDK for Kotlin

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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\n\"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
    }
}
```



```
}
```

- 如需 API 詳細資訊，請參閱 AWS SDK for Kotlin API reference 中的下列主題。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [發布](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

Swift

SDK for Swift

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

```
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\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)
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 Swift 的 AWS SDK API 參考》中的下列主題。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [發布](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 透過 Amazon SQS 傳送和接收批次訊息

以下程式碼範例顯示做法：

- 建立 Amazon SQS 佇列。
- 將批次訊息傳送至佇列。
- 從佇列接收批次的訊息。
- 從佇列刪除訊息批次。

Python

SDK for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

建立 函數以包裝 Amazon SQS 訊息函數。

```
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

```

使用包裝函式以批次傳送和接收訊息。

```

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)
```

- 如需 API 的詳細資訊，請參閱 AWS SDK for Python (Boto3) API Reference 中的下列主題。
 - [CreateQueue](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [ReceiveMessage](#)
 - [SendMessageBatch](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用適用於 .NET AWS 的訊息處理架構來發佈和接收 Amazon SQS 訊息

下列程式碼範例示範如何使用適用於 .NET 的訊息處理架構，建立發佈和接收 Amazon SQS AWS 訊息的應用程式。

.NET

適用於 .NET 的 SDK

提供適用於 .NET AWS 的訊息處理架構教學課程。本教學課程會建立 Web 應用程式，允許使用者發佈 Amazon SQS 訊息和接收訊息的命令列應用程式。

如需有關如何設定和執行的完整原始程式碼和指示，請參閱《適用於 .NET 的 AWS SDK 開發人員指南》中的[完整教學](#)課程和 [GitHub](#) 上的範例。

此範例中使用的服務

- Amazon SQS

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 Amazon SQS Java Messaging Library 來使用 Amazon SQS 的 Java Message Service (JMS) 介面

下列程式碼範例示範如何使用 Amazon SQS Java Messaging Library 來使用 JMS 介面。

Java

適用於 Java 2.x 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

下列範例適用於標準 Amazon SQS 佇列，包括：

- 傳送文字訊息。
- 同步接收訊息。
- 以非同步方式接收訊息。
- 使用 CLIENT_ACKNOWLEDGE 模式接收訊息。
- 使用 UNORDERED_ACKNOWLEDGE 模式接收訊息。
- 使用 Spring 注入相依性。
- 公用程式類別，提供其他範例使用的常用方法。

如需搭配 Amazon SQS 使用 JMS 的詳細資訊，請參閱《[Amazon SQS 開發人員指南](#)》。

傳送文字訊息。

```
/**
 * This method establishes a connection to a standard Amazon SQS queue using
 the Amazon SQS
 * Java Messaging Library and sends text messages to it. It uses JMS (Java
 Message Service) API
 * with automatic acknowledgment mode to ensure reliable message delivery,
 and automatically
 * manages all messaging resources.
```

```

    *
    * @throws JMSEException If there is a problem connecting to or sending
    messages to the queue
    */
    public static void doSendTextMessage() throws JMSEException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            SqsClient.create()
        );

        // Create the connection in a try-with-resources statement so that it's
        closed automatically.
        try (SQSConnection connection = connectionFactory.createConnection()) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
            SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session that uses the JMS auto-acknowledge mode.
            Session session = connection.createSession(false,
            Session.AUTO_ACKNOWLEDGE);
            MessageProducer producer =
            session.createProducer(session.createQueue(QUEUE_NAME));

            createAndSendMessages(session, producer);
        } // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed");
    }

    /**
     * This method reads text input from the keyboard and sends each line as a
     separate message
     * to a standard Amazon SQS queue using the Amazon SQS Java Messaging
     Library. It continues
     * to accept input until the user enters an empty line, using JMS (Java
     Message Service) API to
     * handle the message delivery.
     *
     * @param session The JMS session used to create messages
     * @param producer The JMS message producer used to send messages to the
     queue
     */

```

```

private static void createAndSendMessages(Session session, MessageProducer
producer) {
    BufferedReader inputReader = new BufferedReader(
        new InputStreamReader(System.in, Charset.defaultCharset()));

    try {
        String input;
        while (true) {
            LOGGER.info("Enter message to send (leave empty to exit): ");
            input = inputReader.readLine();
            if (input == null || input.isEmpty()) break;

            TextMessage message = session.createTextMessage(input);
            producer.send(message);
            LOGGER.info("Send message {}", message.getJMSMessageID());
        }
    } catch (EOFException e) {
        // Just return on EOF
    } catch (IOException e) {
        LOGGER.error("Failed reading input: {}", e.getMessage(), e);
    } catch (JMSException e) {
        LOGGER.error("Failed sending message: {}", e.getMessage(), e);
    }
}

```

同步接收訊息。

```

/**
 * 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 (JMSEException e) {
        LOGGER.error("Error receiving from SQS: {}", e.getMessage(), e);
    }
}

```

以非同步方式接收訊息。

```

/**
 * This method sets up automatic message handling for a standard Amazon SQS
queue using the
 * Amazon SQS Java Messaging Library. It creates a listener that processes
messages as soon
 * as they arrive using JMS (Java Message Service), runs for 5 seconds, then
cleans up all
 * messaging resources.
 *
 * @throws JMSEException If there is a problem connecting to or receiving
messages from the queue
 */
public static void doReceiveMessageAsync() throws JMSEException {
    // Create a connection factory.
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        SqsClient.create()
    );

    // Create a connection.
    try (SQSConnection connection = connectionFactory.createConnection() ) {

        // Create the queue if needed.
        SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

        // Create a session.

```

```

        Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

        try {
            // Create a consumer for the queue.
            MessageConsumer consumer =
session.createConsumer(session.createQueue(QueueName));
            // Provide an implementation of the MessageListener interface,
which has a single 'onMessage' method.
            // We use a lambda expression for the implementation.
            consumer.setMessageListener(message -> {
                try {
                    SqsJmsExampleUtils.handleMessage(message);
                    message.acknowledge();
                } catch (JMSEException e) {
                    LOGGER.error("Error processing message: {}",
e.getMessage());
                }
            });
            // Start receiving incoming messages.
            connection.start();
            LOGGER.info("Waiting for messages...");
        } catch (JMSEException e) {
            throw new RuntimeException(e);
        }
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    } // The connection closes automatically. This also closes the session.
    LOGGER.info( "Connection closed" );
}

```

使用 CLIENT_ACKNOWLEDGE 模式接收訊息。

```

/**
 * This method demonstrates how message acknowledgment affects message
processing in a standard
 * Amazon SQS queue using the Amazon SQS Java Messaging Library. It sends
messages to the queue,

```

```
* then shows how JMS (Java Message Service) client acknowledgment mode
handles both explicit
* and implicit message confirmations, including how acknowledging one
message can automatically
* acknowledge previous messages.
*
* @throws JMSEException If there is a problem with the messaging operations
*/
public static void doReceiveMessagesSyncClientAcknowledge() throws
JMSEException {
    // Create a connection factory.
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        SqsClient.create()
    );

    // Create the connection in a try-with-resources statement so that it's
    closed automatically.
    try (SQSConnection connection = connectionFactory.createConnection() ) {

        // Create the queue if needed.
        SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
        TIME_OUT_SECONDS);

        // Create a session with client acknowledge mode.
        Session session = connection.createSession(false,
        Session.CLIENT_ACKNOWLEDGE);

        // Create a producer and consumer.
        MessageProducer producer =
        session.createProducer(session.createQueue(QUEUE_NAME));
        MessageConsumer consumer =
        session.createConsumer(session.createQueue(QUEUE_NAME));

        // Open the connection.
        connection.start();

        // Send two text messages.
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it.
        receiveMessage(consumer, false);
    }
}
```

```

        // Receive another message and acknowledge it.
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages
        reappear in the queue,
        LOGGER.info("Waiting for visibility timeout...");
        try {
            Thread.sleep(TIME_OUT_MILLIS);
        } catch (InterruptedException e) {
            LOGGER.error("Interrupted while waiting for visibility timeout",
e);

            Thread.currentThread().interrupt();
            throw new RuntimeException("Processing interrupted", e);
        }

        /* We will attempt to receive another message, but none will be
        available. This is because in
            CLIENT_ACKNOWLEDGE mode, when we acknowledged the second message,
        all previous messages were
            automatically acknowledged as well. Therefore, although we never
        directly acknowledged the first
            message, it was implicitly acknowledged when we confirmed the
        second one. */
        receiveMessage(consumer, true);
    } // The connection closes automatically. This also closes the session.
    LOGGER.info("Connection closed.");

}

/**
 * Sends a text message using the specified JMS MessageProducer and Session.
 *
 * @param producer    The JMS MessageProducer used to send the message
 * @param session     The JMS Session used to create the text message
 * @param messageText The text content to be sent in the message
 * @throws JMSEException If there is an error creating or sending the message
 */
private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSEException {
    // Create a text message and send it.
    producer.send(session.createTextMessage(messageText));
}

```



```

/**
 * Receives and processes a message from a JMS queue using the specified
 consumer.
 * The method waits for a message until the configured timeout period is
 reached.
 * If a message is received, it is logged and optionally acknowledged based
 on the
 * acknowledge parameter.
 *
 * @param consumer    The JMS MessageConsumer used to receive messages from
 the queue
 * @param acknowledge Boolean flag indicating whether to acknowledge the
 message.
 *
 *                If true, the message will be acknowledged after
 processing
 * @throws JMSEException If there is an error receiving, processing, or
 acknowledging the message
 */
private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSEException {
    // Receive a message.
    Message message = consumer.receive(TIME_OUT_MILLIS);

    if (message == null) {
        LOGGER.info("Queue is empty!");
    } else {
        // Since this queue has only text messages, cast the message object
 and print the text.
        LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
message).getText(), acknowledge);

        // Acknowledge the message if asked.
        if (acknowledge) message.acknowledge();
    }
}

```

使用 UNORDERED_ACKNOWLEDGE 模式接收訊息。

```

/**
 * Demonstrates message acknowledgment behavior in UNORDERED_ACKNOWLEDGE mode
 with Amazon SQS JMS.

```

```
* In this mode, each message must be explicitly acknowledged regardless of
receive order.
* Unacknowledged messages return to the queue after the visibility timeout
expires,
* unlike CLIENT_ACKNOWLEDGE mode where acknowledging one message
acknowledges all previous messages.
*
* @throws JMSEException      If a JMS-related error occurs during message
operations
*/
public static void doReceiveMessagesUnorderedAcknowledge() throws
JMSEException {
    // Create a connection factory.
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        SqsClient.create()
    );

    // Create the connection in a try-with-resources statement so that it's
    closed automatically.
    try( SQSConnection connection = connectionFactory.createConnection() ) {

        // Create the queue if needed.
        SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
        TIME_OUT_SECONDS);

        // Create a session with unordered acknowledge mode.
        Session session = connection.createSession(false,
        SQSSession.UNORDERED_ACKNOWLEDGE);

        // Create the producer and consumer.
        MessageProducer producer =
        session.createProducer(session.createQueue(QUEUE_NAME));
        MessageConsumer consumer =
        session.createConsumer(session.createQueue(QUEUE_NAME));

        // Open a connection.
        connection.start();

        // Send two text messages.
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it.
```

```

        receiveMessage(consumer, false);

        // Receive another message and acknowledge it.
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages
        reappear in the queue.
        LOGGER.info("Waiting for visibility timeout...");
        try {
            Thread.sleep(TIME_OUT_MILLIS);
        } catch (InterruptedException e) {
            LOGGER.error("Interrupted while waiting for visibility timeout",
e);

            Thread.currentThread().interrupt();
            throw new RuntimeException("Processing interrupted", e);
        }

        /* We will attempt to receive another message, and we'll get the
        first message again. This occurs
            because in UNORDERED_ACKNOWLEDGE mode, each message requires its
            own separate acknowledgment.
            Since we only acknowledged the second message, the first message
            remains in the queue for
            redelivery. */
        receiveMessage(consumer, true);

        LOGGER.info("Connection closed.");
    } // The connection closes automatically. This also closes the session.
}

/**
 * Sends a text message to an Amazon SQS queue using JMS.
 *
 * @param producer    The JMS MessageProducer for the queue
 * @param session      The JMS Session for message creation
 * @param messageText  The message content
 * @throws JMSEException If message creation or sending fails
 */
private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSEException {
    // Create a text message and send it.
    producer.send(session.createTextMessage(messageText));
}
/**

```

```

    * Synchronously receives a message from an Amazon SQS queue using the JMS
    API
    * with an acknowledgment parameter.
    *
    * @param consumer    The JMS MessageConsumer for the queue
    * @param acknowledge If true, acknowledges the message after receipt
    * @throws JMSEException If message reception or acknowledgment fails
    */
    private static void receiveMessage(MessageConsumer consumer, boolean
    acknowledge) throws JMSEException {
        // Receive a message.
        Message message = consumer.receive(TIME_OUT_MILLIS);

        if (message == null) {
            LOGGER.info("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object
            and print the text.
            LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
            message).getText(), acknowledge);

            // Acknowledge the message if asked.
            if (acknowledge) message.acknowledge();
        }
    }
}

```

使用 Spring 注入相依性。

```

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 (JMSEException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        }
    } // Spring context autocloses. Managed Spring beans that implement
    AutoClosable, such as the
    // 'connection' bean, are also closed.
    LOGGER.info("Context closed");
}

```

```

/**
 * Continuously checks for and processes messages from a standard Amazon SQS
message queue
 * using the Amazon SQS Java Messaging Library underlying the JMS API. This
method waits for incoming messages,
 * processes them when they arrive, and acknowledges their receipt using JMS
(Java Message
 * Service) operations. The method will stop checking for messages after 15
seconds of
 * inactivity.
 *
 * @param consumer The JMS message consumer used to receive messages from the
queue
 */
private static void receiveMessages(MessageConsumer consumer) {
    try {
        while (true) {
            LOGGER.info("Waiting for messages...");
            // Wait 15 seconds for a message.
            Message message =
consumer.receive(TimeUnit.SECONDS.toMillis(POLLING_SECONDS));
            if (message == null) {
                LOGGER.info("Shutting down after {} seconds of silence.",
POLLING_SECONDS);
                break;
            }
            SqsJmsExampleUtils.handleMessage(message);
            message.acknowledge();
            LOGGER.info("Message acknowledged.");
        }
    } catch (JMSEException e) {
        LOGGER.error("Error receiving from SQS.", e);
    }
}
}

```

Spring Bean 定義。

```

<?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>
```

公用程式類別，提供其他範例使用的常用方法。

```
package com.example.sqs.jms;

import com.amazon.sqs.javamessaging.AmazonSQSMessagingClientWrapper;
import com.amazon.sqs.javamessaging.ProviderConfiguration;
import com.amazon.sqs.javamessaging.SQSConnection;
import com.amazon.sqs.javamessaging.SQSConnectionFactory;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.exception.SdkException;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;

import java.time.Duration;
import java.util.Base64;
import java.util.Map;

/**
 * This utility class provides helper methods for working with Amazon Simple
 * Queue Service (Amazon SQS)
 * through the Java Message Service (JMS) interface. It contains common
 * operations for managing message
 * queues and handling message delivery.
 */
public class SqsJmsExampleUtils {
    private static final Logger LOGGER =
        LoggerFactory.getLogger(SqsJmsExampleUtils.class);
    public static final Long QUEUE_VISIBILITY_TIMEOUT = 5L;

    /**
     * This method verifies that a message queue exists and creates it if
     * necessary. The method checks for
```

```

    * an existing queue first to optimize performance.
    *
    * @param connection The active connection to the messaging service
    * @param queueName The name of the queue to verify or create
    * @param visibilityTimeout The duration in seconds that messages will be
hidden after being received
    * @throws JMSEException If there is an error accessing or creating the queue
    */
    public static void ensureQueueExists(SQSConnection connection, String
queueName, Long visibilityTimeout) throws JMSEException {
        AmazonSQSMessagingClientWrapper client =
connection.getWrappedAmazonSQSClient();

        /* In most cases, you can do this with just a 'createQueue' call, but
'getQueueUrl'
        (called by 'queueExists') is a faster operation for the common case where
the queue
        already exists. Also, many users and roles have permission to call
'getQueueUrl'
        but don't have permission to call 'createQueue'.
        */
        if( !client.queueExists(queueName) ) {
            CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
                .queueName(queueName)
                .attributes(Map.of(QueueAttributeName.VISIBILITY_TIMEOUT,
String.valueOf(visibilityTimeout)))
                .build();
            client.createQueue( createQueueRequest );
        }
    }

    /**
    * This method sends a simple text message to a specified message queue. It
handles all necessary
    * setup for the message delivery process.
    *
    * @param session The active messaging session used to create and send the
message
    * @param queueName The name of the queue where the message will be sent
    */
    public static void sendTextMessage(Session session, String queueName) {
        // Rest of implementation...

        try {

```

```

        MessageProducer producer =
session.createProducer( session.createQueue( queueName) );
        Message message = session.createTextMessage("Hello world!");
        producer.send(message);
    } catch (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 (JMSEException e) {
                    LOGGER.error("Error processing message: {}", e.getMessage());
                }
            });
            // Start receiving incoming messages.
            connection.start();
        } catch (JMSEException e) {
            throw new RuntimeException(e);
        }
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * This method performs cleanup operations after message processing is
     complete. It receives
     * any messages in the specified queue, removes the message queue and closes
     all
     * active connections to prevent resource leaks.
     *
     * @param queueName The name of the queue to be removed
     * @param visibilityTimeout The duration in seconds that messages are hidden
     after being received
     * @throws JMSEException If there is an error during the cleanup process
    
```

```

    */
    public static void cleanUpExample(String queueName, Long visibilityTimeout)
    throws JMSEException {
        LOGGER.info("Performing cleanup.");

        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            SqsClient.create()
        );

        try (SQSConnection connection = connectionFactory.createConnection() ) {
            ensureQueueExists(connection, queueName, visibilityTimeout);
            Session session = connection.createSession(false,
                Session.AUTO_ACKNOWLEDGE);

            receiveMessagesAsync(session, queueName, connection);

            SqsClient sqsClient =
            connection.getWrappedAmazonSQSClient().getAmazonSQSClient();
            try {
                String queueUrl = sqsClient.getQueueUrl(b ->
                b.queueName(queueName)).queueUrl();
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue deleted: {}", queueUrl);
            } catch (SdkException e) {
                LOGGER.error("Error during SQS operations: ", e);
            }
        }
        LOGGER.info("Clean up: Connection closed");
    }

    /**
     * This method creates a background task that sends multiple messages to a
     * specified queue
     * after waiting for a set time period. The task operates independently to
     * ensure efficient
     * message processing without interrupting other operations.
     *
     * @param queueName The name of the queue where messages will be sent
     * @param secondsToWait The number of seconds to wait before sending messages
     * @param numMessages The number of messages to send
     * @param visibilityTimeout The duration in seconds that messages remain
     * hidden after being received
     * @return A task that can be executed to send the messages
    
```

```

    */
    public static Runnable sendAMessageAsync(String queueName, Long
secondsToWait, Integer numMessages, Long visibilityTimeout) {
        return () -> {
            try {
                Thread.sleep(Duration.ofSeconds(secondsToWait).toMillis());
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException(e);
            }
            try {
                SQSConnectionFactory connectionFactory = new
SQSConnectionFactory(
                    new ProviderConfiguration(),
                    SqsClient.create()
                );
                try (SQSConnection connection =
connectionFactory.createConnection()) {
                    ensureQueueExists(connection, queueName, visibilityTimeout);
                    Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
                    for (int i = 1; i <= numMessages; i++) {
                        MessageProducer producer =
session.createProducer(session.createQueue(queueName));
                        producer.send(session.createTextMessage("Hello World " +
i + "!!"));
                    }
                } catch (JMSEException e) {
                    LOGGER.error(e.getMessage(), e);
                    throw new RuntimeException(e);
                }
            };
        }
    }
}

```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
 - [CreateQueue](#)
 - [DeleteQueue](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 AWS SDK 處理佇列標籤和 Amazon SQS

下列程式碼範例示範如何使用 Amazon SQS 執行標記操作。

Java

適用於 Java 2.x 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在 [AWS 程式碼範例儲存庫](#) 中設定和執行。

下列範例會建立佇列的標籤、列出標籤，以及移除標籤。

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.ListQueueTagsResponse;
import software.amazon.awssdk.services.sqs.model.QueueDoesNotExistException;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.util.Map;
import java.util.UUID;

/**
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials. For more
 * information, see the <a href="https://docs.aws.amazon.com/sdk-for-java/latest/
 * developer-guide/get-started.html">AWS
 * SDK for Java Developer Guide</a>.
 */
public class TagExamples {
    static final SqsClient sqsClient = SqsClient.create();
    static final String queueName = "TagExamples-queue-" +
        UUID.randomUUID().toString().replace("-", "").substring(0, 20);
```

```

    private static final Logger LOGGER =
        LoggerFactory.getLogger(TagExamples.class);

    public static void main(String[] args) {
        final String queueUrl;
        try {
            queueUrl = sqsClient.createQueue(b ->
b.queueName(queueName)).queueUrl();
            LOGGER.info("Queue created. The URL is: {}", queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because queue was not created.");
            throw new RuntimeException(e);
        }
        try {
            addTags(queueUrl);
            listTags(queueUrl);
            removeTags(queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because of an error in a method.");
        } finally {
            try {
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue successfully deleted. Program ending.");
                sqsClient.close();
            } catch (RuntimeException e) {
                LOGGER.error("Program ending.");
            } finally {
                sqsClient.close();
            }
        }
    }

    /** This method demonstrates how to use a Java Map to tag a queue.
     * @param queueUrl The URL of the queue to tag.
     */
    public static void addTags(String queueUrl) {
        // Build a map of the tags.
        final Map<String, String> tagsToAdd = Map.of(
            "Team", "Development",
            "Priority", "Beta",
            "Accounting ID", "456def");

        try {

```



```

        // Add tags to the queue using a Consumer<TagQueueRequest.Builder>
parameter.
        sqsClient.tagQueue(b -> b
            .queueUrl(queueUrl)
            .tags(tagsToAdd)
        );
    } catch (QueueDoesNotExistException e) {
        LOGGER.error("Queue does not exist: {}", e.getMessage(), e);
        throw new RuntimeException(e);
    }
}

/** This method demonstrates how to view the tags for a queue.
 * @param queueUrl The URL of the queue whose tags you want to list.
 */
public static void listTags(String queueUrl) {
    ListQueueTagsResponse response;
    try {
        // Call the listQueueTags method with a
Consumer<ListQueueTagsRequest.Builder> parameter that creates a
ListQueueTagsRequest.
        response = sqsClient.listQueueTags(b -> b
            .queueUrl(queueUrl));
    } catch (SqsException e) {
        LOGGER.error("Exception thrown: {}", e.getMessage(), e);
        throw new RuntimeException(e);
    }

    // Log the tags.
    response.tags()
        .forEach((k, v) ->
            LOGGER.info("Key: {} -> Value: {}", k, v));
}

/**
 * This method demonstrates how to remove tags from a queue.
 * @param queueUrl The URL of the queue whose tags you want to remove.
 */
public static void removeTags(String queueUrl) {
    try {
        // Call the untagQueue method with a
Consumer<UntagQueueRequest.Builder> parameter.
        sqsClient.untagQueue(b -> b
            .queueUrl(queueUrl)

```

```
        .tagKeys("Accounting ID") // Remove a single tag.
    );
} catch (SqsException e) {
    LOGGER.error("Exception thrown: {}", e.getMessage(), e);
    throw new RuntimeException(e);
}
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的下列主題。
- [ListQueueTags](#)
- [TagQueue](#)
- [UntagQueue](#)

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Amazon SQS 的無伺服器範例

下列程式碼範例示範如何使用 Amazon SQS 搭配 AWS SDKs。

範例

- [使用 Amazon SQS 觸發條件調用 Lambda 函數](#)
- [使用 Amazon SQS 觸發條件報告 Lambda 函數的批次項目失敗](#)

使用 Amazon SQS 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，以便接收在收到來自 SQS 佇列的訊息時觸發的事件。函數會從事件參數擷取訊息，並記錄每一則訊息的內容。

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 .NET 搭配 Lambda 來使用 SQS 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
        }
        catch { }
    }
}
```

```
        await Task.CompletedTask;
    }
    catch (Exception e)
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
        Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
```

Go

SDK for Go V2

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Go 搭配 Lambda 來使用 SQS 事件。

```
// 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 for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Java 搭配 Lambda 來使用 SQS 事件。

```
// 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 for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 JavaScript 搭配 Lambda 來使用 SQS 事件。

```
// 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;
  }
}
```

使用 TypeScript 搭配 Lambda 來使用 SQS 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK for PHP

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 PHP 搭配 Lambda 來使用 SQS 事件。

```
// 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 for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Python 搭配 Lambda 來使用 SQS 事件。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
def lambda_handler(event, context):  
    for message in event['Records']:  
        process_message(message)  
    print("done")  
  
def process_message(message):  
    try:  
        print(f"Processed message {message['body']}")  
        # TODO: Do interesting work based on the new message  
    except Exception as err:  
        print("An error occurred")  
        raise err
```

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Ruby 搭配 Lambda 來使用 SQS 事件。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

SDK for Rust

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 SQS 事件。

```
// 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
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 Amazon SQS 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何針對接收來自 SQS 佇列之事件的 Lambda 函數，實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

.NET

適用於 .NET 的 SDK

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 .NET 搭配 Lambda 報告 SQS 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
                SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
    }
}
```

```
    }
    return new SQSBatchResponse(batchItemFailures);
}

private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context)
{
    if (String.IsNullOrEmpty(message.Body))
    {
        throw new Exception("No Body in SQS Message.");
    }
    context.Logger.LogInformation($"Processed message {message.Body}");
    // TODO: Do interesting work based on the new message
    await Task.CompletedTask;
}
}
```

Go

SDK for Go V2

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Go 搭配 Lambda 報告 SQS 批次項目失敗。

```
// 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 for Java 2.x

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Java 搭配 Lambda 報告 SQS 批次項目失敗。

```
// 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 for JavaScript (v3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 JavaScript 搭配 Lambda 報告 SQS 批次項目失敗。

```
// 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}`);
}
```

使用 TypeScript 搭配 Lambda 報告 SQS 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
    from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
    Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
```


PHP

SDK for PHP

 Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 PHP 搭配 Lambda 報告 SQS 批次項目失敗。

```
// 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 for Python (Boto3)

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Python 搭配 Lambda 報告 SQS 批次項目失敗。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                # process message
            except Exception as e:
```

```
        batch_item_failures.append({"itemIdentifier":  
record['messageId']})  
  
    sqs_batch_response["batchItemFailures"] = batch_item_failures  
    return sqs_batch_response
```

Ruby

SDK for Ruby

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Ruby 搭配 Lambda 報告 SQS 批次項目失敗。

```
# 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  
end
```

Rust

SDK for Rust

Note

GitHub 上提供更多範例。尋找完整範例，並了解如何在[無伺服器範例](#)儲存庫中設定和執行。

使用 Rust 搭配 Lambda 報告 SQS 批次項目失敗。

```
// 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  
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 [搭配 AWS SDK 使用 Amazon SQS](#)。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

對 Amazon SQS 中的問題進行故障診斷

本主題針對使用 Amazon SQS 主控台、Amazon SQS API 或其他工具搭配 Amazon SQS 時可能遇到的常見錯誤和問題，提供疑難排解建議。如果您發現未列在此處的問題，您可以使用此頁面上的 Feedback (意見回饋) 按鈕來報告。

如需更多故障診斷建議和常見支援問題的解答，請瀏覽 [AWS 知識中心](#)。

主題

- [對 Amazon SQS 中的存取遭拒錯誤進行故障診斷](#)
- [對 Amazon SQS API 錯誤進行故障診斷](#)
- [對 Amazon SQS 無效字母佇列和 DLQ 重新驅動問題進行故障診斷](#)
- [對 Amazon SQS 中的 FIFO 限流問題進行故障診斷](#)
- [針對 Amazon SQS ReceiveMessage API 呼叫未傳回的訊息進行故障診斷](#)
- [對 Amazon SQS 網路錯誤進行故障診斷](#)
- [使用 對 Amazon Simple Queue Service 佇列進行故障診斷 AWS X-Ray](#)

對 Amazon SQS 中的存取遭拒錯誤進行故障診斷

下列主題涵蓋 Amazon SQS API 呼叫最常見的原因 `AccessDenied` 或 `AccessDeniedException` 錯誤。如需如何對這些錯誤進行故障診斷的詳細資訊，請參閱 AWS 知識中心指南中的 [如何對 Amazon SQS API 呼叫上的「AccessDenied」或「AccessDeniedException」錯誤進行故障診斷？](#)。

錯誤訊息範例：

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

- 或 -

```
An error occurred (KMS.AccessDeniedException) when calling the SendMessage operation: User: arn:aws:iam::xxxxx:user/xxxx is not authorized to perform: kms:GenerateDataKey on resource: arn:aws:kms:us-east-1:xxxx:key/xxxx with an explicit deny.
```

Amazon SQS 佇列政策和 IAM 政策

若要驗證請求者是否具有執行 Amazon SQS 操作的適當許可，請執行下列動作：

- 識別進行 Amazon SQS API 呼叫的 IAM 主體。如果 IAM 主體來自相同的帳戶，則 Amazon SQS 佇列政策或 AWS Identity and Access Management (IAM) 政策必須包含明確允許存取動作的許可。
- 如果委託人是 IAM 實體：
 - 您可以檢查 的右上角或使用 [aws sts get-caller-identity](#) 命令來識別 IAM 使用者 AWS Management Console 或角色。
 - 檢查與 IAM 使用者或角色相關的 IAM 政策。您可以使用下列其中一種方法：
 - 使用 IAM 政策 [模擬器測試 IAM 政策](#)。
 - 檢閱不同的 [IAM 政策類型](#)。
 - 如有需要，請[編輯您的 IAM 使用者政策](#)。
 - 檢查佇列政策並視需要[編輯](#)。
- 如果委託人是 AWS 服務，則 Amazon SQS 佇列政策必須明確允許存取。
- 如果委託人是跨帳戶委託人，則 Amazon SQS 佇列政策和 IAM 政策都必須明確允許存取。
- 如果政策使用條件元素，請檢查條件是否限制存取。

Important

任一政策中的明確拒絕都會覆寫明確允許。以下是 [Amazon SQS 政策](#) 的一些基本範例。

AWS Key Management Service 許可

如果您的 Amazon SQS 佇列已開啟[伺服器端加密 \(SSE\)](#) 並受客戶管理 AWS KMS key，則必須同時將許可授予生產者和消費者。若要確認佇列是否已加密，您可以使用 [GetQueueAttributes](#) API `KmsMasterKeyId` 屬性，或從加密下的佇列主控台進行。

- [生產者的必要許可](#)：

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
```

```
],  
"Resource": "<Key ARN>"  
}
```

- [消費者所需的許可](#)：

```
{  
  "Effect": "Allow",  
  "Action": [  
    "kms:Decrypt"  
  ],  
  "Resource": "<Key ARN>"  
}
```

- [跨帳戶存取](#)的必要許可：

```
{  
  "Effect": "Allow",  
  "Action": [  
    "kms:DescribeKey",  
    "kms:Decrypt",  
    "kms:ReEncrypt",  
    "kms:GenerateDataKey"  
  ],  
  "Resource": "<Key ARN>"  
}
```

選擇下列其中一個選項來啟用 Amazon SQS 佇列的加密：

- [SSE-Amazon SQS](#) (Amazon SQS 服務建立和管理的加密金鑰)。
- [AWS 受管預設金鑰](#) (alias/aws/sqs)
- [客戶受管金鑰](#)

不過，如果您使用的是 AWS 受管 [KMS 金鑰](#)，則無法修改預設金鑰政策。因此，若要提供其他服務和跨帳戶的存取權，請使用客戶受管金鑰。這樣做可讓您編輯金鑰政策。

VPC 端點政策

如果您透過 [Amazon Virtual Private Cloud \(Amazon VPC\) 端點存取 Amazon SQS Amazon Virtual Private Cloud](#)，Amazon SQS VPC 端點政策必須允許存取。您可以為 Amazon SQS 的 Amazon VPC 端點建立政策，您可以在其中指定以下內容：

1. 可執行動作的主體。
2. 可執行的動作。
3. 可供執行動作的資源。

在下列範例中，VPC 端點政策指定允許 IAM 使用者 *MyUser* 傳送訊息至 Amazon SQS 佇列 *MyQueue*。其他動作、IAM 使用者和 Amazon SQS 資源會被拒絕透過 VPC 端點存取。

```
{
  "Statement": [{
    "Action": ["sqs:SendMessage"],
    "Effect": "Allow",
    "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

組織服務控制政策

如果您的 AWS 帳戶 屬於 組織，AWS Organizations 政策可能會封鎖您存取 Amazon SQS 佇列。根據預設，AWS Organizations 政策不會封鎖對 Amazon SQS 的任何請求。不過，請確定您的 AWS Organizations 政策尚未設定為封鎖對 Amazon SQS 佇列的存取。如需如何檢查 AWS Organizations 政策的說明，請參閱AWS Organizations 《使用者指南》中的[列出所有政策](#)。

對 Amazon SQS API 錯誤進行故障診斷

下列主題涵蓋 Amazon SQS API 呼叫時最常傳回的錯誤，以及如何進行故障診斷。

QueueDoesNotExist 錯誤

當 Amazon SQS 服務找不到 Amazon SQS 動作的所述佇列時，將會傳回此錯誤。

可能的原因和緩解措施：

- 區域不正確：檢閱 Amazon SQS 用戶端組態，以確認您在用戶端上設定了正確的區域。當您未在用戶端上設定區域時，SDK 或從[組態檔案](#)或環境變數 AWS CLI 中選擇區域。如果 SDK 在組態檔案中找不到區域，則 SDK 預設會將區域設定為 us-east-1。
- 佇列可能最近遭到刪除：如果佇列在 API 呼叫之前遭到刪除，則 API 呼叫將傳回此錯誤。在發生錯誤之前，檢查 CloudTrail 是否有任何 [DeleteQueue](#) 操作。
- 許可問題：如果請求 AWS Identity and Access Management (IAM) 使用者或角色沒有必要的許可，則您可能會收到下列錯誤：

```
The specified queue does not exist or you do not have access to it.
```

檢查許可，並使用正確的許可進行 API 呼叫。

如需有關對 `QueueDoesNotExist` 錯誤進行故障診斷的詳細資訊，請參閱 AWS 知識中心指南中的[如何在對 Amazon SQS 佇列進行 API 呼叫時對 QueueDoesNotExist 錯誤進行故障診斷](#)。

InvalidAttributeValue 錯誤

更新 Amazon SQS 佇列資源政策，或具有不正確政策或主體的屬性時，將會傳回此錯誤。

可能的原因和緩解措施：

- 無效的資源政策：檢查資源政策是否具有所有必要欄位。如需詳細資訊，請參閱 [IAM JSON 政策元素參考](#)和[驗證 IAM 政策](#)。您也可以使用 [IAM 政策產生器](#)來建立和測試 Amazon SQS 資源政策。確定政策為 JSON 格式。
- 無效的主體：確保 Principal 元素存在於資源政策中，且該值有效。如果您的 Amazon SQS 資源政策 Principal 元素包含 IAM 實體，請在使用該政策之前確定實體存在。Amazon SQS 會驗證 IAM 實體的資源政策和檢查。如果 IAM 實體不存在，您將會收到錯誤。若要確認 IAM 實體，請使用 [GetRole](#)和 [GetUser](#) APIs。

如需如何對 `InvalidAttributeValue` 錯誤進行故障診斷的詳細資訊，請參閱 AWS 知識中心指南中的[如何在對 Amazon SQS 佇列進行 API 呼叫時對 QueueDoesNotExist 錯誤進行故障診斷](#)。

ReceiptHandle 錯誤

發出 [DeleteMessage](#) API 呼叫時，如果接收控點不正確 `ReceiptHandleIsInvalid` 或過期，`InvalidParameterValue` 可能會傳回錯誤或。

- `ReceiptHandleIsInvalid` 錯誤：如果接收控點不正確，您會收到類似此範例的錯誤：

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:  
The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- `InvalidParameterValue` 錯誤：如果接收控點已過期，您將收到類似此範例的錯誤：

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:  
Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The  
receipt handle has expired.
```

可能的原因和緩解措施：

系統會為每個收到的訊息建立接收處理常式，並且僅在可見性逾時期間有效。當可見性逾時期間過期時，訊息會在佇列上顯示供取用者使用。當您再次收到消費者的訊息時，會收到新的接收控點。為了防止不正確或過期的接收處理常式錯誤，請使用正確的接收處理常式，在 Amazon SQS 佇列可見性逾時期間內刪除訊息。

如需如何對 `ReceiptHandle` 錯誤進行故障診斷的詳細資訊，請參閱 AWS 知識中心指南中的 [如何在 Amazon SQS DeleteMessage API 呼叫時對「ReceiptHandleIsInvalid」和「InvalidParameterValue」錯誤進行故障診斷。](#)

對 Amazon SQS 無效字母佇列和 DLQ 重新驅動問題進行故障診斷

下列主題涵蓋 Amazon SQS DLQ 和 DLQ 重新驅動問題的最常見原因，以及如何進行故障診斷。如需詳細資訊，請參閱 AWS 知識中心指南中的 [如何疑難排解 Amazon SQS DLQ 重新驅動問題？](#)。

DLQ 問題

了解常見的 DLQ 問題以及如何解決這些問題。

使用主控台檢視訊息可能會導致訊息被移動至無效字母佇列

Amazon SQS 根據相對應佇列的再驅動政策來計算在主控台中檢視訊息的次數。因此，如果您在主控台中檢視對應佇列的重新驅動政策中指定的訊息次數，訊息會移至對應佇列的無效字母佇列。

若要調整此行為，可執行以下其中一項：

- 提高相應佇列的再驅動政策的最大接收量設定。
- 避免在主控台中檢視相對應佇列的訊息。

無效字母佇列的 `NumberOfMessagesSent` 和 `NumberOfMessagesReceived` 不相符

若您手動將訊息傳送至無效字母佇列，其會由 [NumberOfMessagesSent](#) 指標擷取。不過，若訊息是因為處理訊息的嘗試失敗而送到無效字母佇列，就不會由此指標擷取。因此，`NumberOfMessagesSent` 和 `NumberOfMessagesReceived` 的值可能不同。

建立和設定無效字母佇列重新驅動

無效字母佇列重新驅動需要您設定適當的[許可](#)，讓 Amazon SQS 從無效字母佇列接收訊息，並將訊息傳送到目的地佇列。如果您沒有正確的許可，無效字母佇列重新驅動任務可能會失敗。您可以檢視訊息重新驅動任務的狀態以修復問題，然後再試一次。

標準和 FIFO 佇列訊息失敗處理

[標準佇列](#)會保留處理訊息，直到[保留期間](#)過期為止。此持續處理可將佇列被未使用的訊息封鎖的機率降至最低。消費者重複刪除大量訊息可能會增加成本，並對硬體額外負載。若要降低成本，請將失敗的訊息移至無效字母佇列。

標準佇列也允許大量傳輸中訊息。如果您大部分的訊息都無法使用，而且沒有傳送到無效字母佇列，處理訊息的速度可能會變慢。若要維持佇列的效率，請確定您的應用程式正確處理訊息處理。

[FIFO 佇列](#)可依序消耗訊息群組內的訊息，藉此確保僅會經過恰好一次的處理。因此，雖然消費者可以繼續從另一個訊息群組擷取排序訊息，但第一個訊息群組仍然無法使用，直到成功處理封鎖佇列的訊息或移至無效字母佇列為止。

此外，FIFO 佇列允許較少數量的傳輸中訊息。為了防止 FIFO 佇列被訊息封鎖，請確定您的應用程式正確處理訊息。

如需詳細資訊，請參閱 [Amazon SQS 訊息配額](#) 和 [Amazon SQS 最佳實務](#)。

DLQ 重新驅動問題

了解常見的 DLQ 重新驅動問題以及如何解決這些問題。

AccessDenied 許可問題

當 DLQ 重新驅動失敗，因為 AWS Identity and Access Management (IAM) 實體沒有所需的許可時，就會發生此 `AccessDenied` 錯誤。

錯誤訊息範例：

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

發出 DLQ 重新驅動請求需要下列 API 許可：

若要啟動訊息重新驅動：

- 無效字母佇列許可：
 - `sqs:StartMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `kms:Decrypt` – 當無效字母佇列或原始來源佇列加密時。
- 目的地佇列許可：
 - `sqs:SendMessage`
 - `kms:GenerateDataKey` – 當目的地佇列加密時。
 - `kms:Decrypt` – 當目的地佇列加密時。

若要取消進行中的訊息重新驅動：

- 無效字母佇列許可：
 - `sqs:CancelMessageMoveTask`
 - `sqs:ReceiveMessage`
 - `sqs:DeleteMessage`
 - `sqs:GetQueueAttributes`
 - `kms:Decrypt` – 當無效字母佇列或原始來源佇列加密時。

若要顯示訊息移動狀態：

- 無效字母佇列許可：
 - `sqs:ListMessageMoveTasks`
 - `sqs:GetQueueAttributes`

NonExistentQueue 錯誤

當 Amazon SQS 來源佇列不存在或已刪除時，NonExistentQueue 就會發生錯誤。檢查並重新驅動到存在的 Amazon SQS 佇列。

錯誤訊息範例：

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource 錯誤

當您嘗試在下列情況下啟動 DLQ 重新驅動時，會發生此 `CouldNotDetermineMessageSource` 錯誤：

- 使用 [SendMessage](#) API 直接傳送至 DLQ 的 Amazon SQS 訊息。
- Amazon Simple Notification Service (Amazon SNS) 主題或 AWS Lambda 函數的訊息，其中已設定 DLQ。

若要解決此錯誤，請在啟動重新驅動時選擇重新驅動至自訂目的地。然後，輸入 Amazon SQS 佇列 ARN，將所有訊息從 DLQ 移至目的地佇列。

錯誤訊息範例：

```
Failed: CouldNotDetermineMessageSource
```

對 Amazon SQS 中的 FIFO 限流問題進行故障診斷

根據預設，FIFO 佇列支援 [SendMessage](#)、[ReceiveMessage](#) 和 [DeleteMessage](#) 的每個 API 動作每秒 300 個交易。即使佇列中的訊息可用，超過 300 TPS 的請求也會收到 `ThrottlingException` 錯誤。若要緩解這種情況，您可以使用下列方法：

- [在 Amazon SQS 中啟用 FIFO 佇列的高輸送量](#)。
- 使用 Amazon SQS API 批次動作 `SendMessageBatch`、`DeleteMessageBatch` 和 `ChangeMessageVisibilityBatch` 來提高每個 API 動作每秒高達 3,000 則訊息的 TPS 限制，並降低成本。針對 `ReceiveMessage` API，將 `MaxNumberOfMessages` 參數設定為每次交易最多接收十則訊息。如需詳細資訊，請參閱 [Amazon SQS 批次動作](#)。
- 對於高輸送量的 FIFO 佇列，請遵循建議來 [最佳化分割區使用率](#)。以批次傳送具有相同訊息群組 IDs 的訊息。刪除訊息，或變更來自相同 `ReceiveMessage` API 請求的接收控點批次中的訊息可見性逾時值。
- 增加唯一 [MessageGroupId](#) 值的數量。這允許跨 FIFO 佇列分割區均勻分佈。如需詳細資訊，請參閱使用 Amazon SQS 訊息群組 ID。

如需詳細資訊，請參閱 AWS 知識中心指南中的 [為什麼我的 Amazon SQS FIFO 佇列不會傳回其他訊息群組中的所有訊息或訊息？](#)。

針對 Amazon SQS ReceiveMessage API 呼叫未傳回的訊息進行故障診斷

下列主題涵蓋無法將 Amazon SQS 訊息傳回給消費者的最常見原因，以及如何進行疑難排解。如需詳細資訊，請參閱 AWS 知識中心指南中的 [為什麼我無法接收來自 Amazon SQS 佇列的訊息？](#)。

空佇列

若要判斷佇列是否為空，請使用長輪詢來呼叫 [ReceiveMessage](#) API。您也可以使用 `ApproximateNumberOfMessagesVisible`、`ApproximateNumberOfMessagesNotVisible` 和 `ApproximateNumberOfMessagesDelayed` CloudWatch 指標。如果所有指標值都設定為 0 幾分鐘，則佇列會被視為空白。

已達到飛行中限制

如果您使用 [長輪詢](#)，且佇列的飛行中限制 (FIFO 為 20000，標準為 120000) 遭到違反，Amazon SQS 將不會傳回 [超過配額限制](#) 的錯誤訊息。

訊息延遲

如果 Amazon SQS 佇列設定為 [延遲佇列](#)，或使用訊息 [計時器](#) 傳送訊息，則在延遲時間結束之前不會顯示訊息。若要驗證佇列是否設定為延遲佇列，請使用 `GetQueueAttributes` API `DelaySeconds` 屬性，或在交付延遲下從佇列主控台進行。檢查 [ApproximateNumberOfMessagesDelayed](#) CloudWatch 指標以了解是否有任何訊息延遲。

訊息正在處理中

如果不同的取用者輪詢訊息，則訊息在 [可見性逾時](#) 期間會處於進行中或隱藏狀態。其他輪詢可能會傳回空的接收。檢查 [ApproximateNumberOfMessagesVisible](#) CloudWatch 指標，以了解可接收的訊息數量。在 FIFO 佇列中，如果訊息群組 ID 為進行中的訊息，則除非您刪除訊息，否則不會再傳回任何訊息，否則會顯示訊息。這是因為 [訊息排序](#) 會維持在 FIFO 佇列中的訊息群組層級。

輪詢方法

如果您使用 [短輪詢](#)，(`WaitTimeSeconds` 為 0) Amazon SQS 會取樣其伺服器子集，並僅傳回來自這些伺服器的訊息。因此，即使訊息可供接收，您可能無法取得訊息。後續輪詢請求將傳回訊息。

如果您使用的是[長輪詢](#)，Amazon SQS 會輪詢所有伺服器，並在收集至少一個可用訊息後傳送回應，最多達指定的數量上限。如果 ReceiveMessage [WaitTimeSeconds](#) 的值太低，您可能不會收到所有可用的訊息。

對 Amazon SQS 網路錯誤進行故障診斷

下列主題涵蓋 Amazon SQS 中網路問題的最常見原因，以及如何進行故障診斷。

ETIMEDOUT error

當用戶端無法建立與 Amazon SQS 端點的 TCP 連線時，ETIMEDOUT 就會發生錯誤。

故障診斷：

- 檢查網路連線

透過執行 等命令來測試 Amazon SQS 的網路連線telnet。

Example: telnet sqs.us-east-1.amazonaws.com 443

- 檢查網路設定
 - 請確定您的本機防火牆規則、路由和存取控制清單 (ACLs) 允許您使用的連接埠流量。
 - 安全群組傳出（傳出）規則必須允許流向連接埠 80 或 443 的流量。
 - 網路 ACL 傳出（傳出）規則必須允許 TCP 連接埠 80 或 443 的流量。
 - 網路 ACL 傳入（傳入）規則必須允許 TCP 連接埠 1024-65535 上的流量。
 - 連線至公有網際網路的 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體必須具有[網際網路連線](#)。
- Amazon Virtual Private Cloud (Amazon VPC) 端點

如果您透過 Amazon VPC 端點存取 Amazon SQS，則端點安全群組必須允許傳入流量到連接埠 443 上的用戶端安全群組。與 VPC 端點子網路相關聯的網路 ACL 必須具有此組態：

- 網路 ACL 傳出（傳出）規則必須允許 TCP 連接埠 1024-65535（暫時性連接埠）上的流量。
- 網路 ACL 傳入（傳入）規則必須允許連接埠 443 上的流量。

此外，Amazon SQS VPC endpoint AWS Identity and Access Management (IAM) 政策必須允許存取。下列範例 VPC 端點政策指定允許 IAM 使用者 *MyUser* 傳送訊息至 Amazon SQS 佇列 *MyQueue*。其他動作、IAM 使用者和 Amazon SQS 資源會被拒絕透過 VPC 端點存取。


```
{
  "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

當無法判斷主機 IP 地址時，就會UnknownHostException發生錯誤。

故障診斷：

使用 nslookup 公用程式來傳回與主機名稱相關聯的 IP 地址：

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI 或適用於 Python 舊版端點的 SDK：

```
nslookup <region>.queue.amazonaws.com
```

如果您收到失敗的輸出，請遵循 AWS 知識中心指南中 [DNS 的運作方式和如何疑難排解部分或間歇性 DNS 失敗？](#) 中的指示。

如果您收到有效的輸出，則可能是應用程式層級的問題。若要解決應用程式層級的問題，請嘗試下列方法：

- 重新啟動您的應用程式。
- 確認您的 Java 應用程式沒有錯誤的 DNS 快取。如果可能，請設定您的應用程式以遵守 DNS TTL。如需詳細資訊，請參閱[設定 DNS 名稱查詢的 JVM TTL](#)。

如需有關如何疑難排解網路錯誤的詳細資訊，請參閱 AWS 知識中心指南中的[如何疑難排解 Amazon SQS “ETIMEOUT” 和 “UnknownHostException” 連線錯誤？](#)。

使用 對 Amazon Simple Queue Service 佇列進行故障診斷 AWS X-Ray

AWS X-Ray 會收集應用程式所提供服務請求的資料，並可讓您檢視和篩選資料，以識別潛在的問題和最佳化的機會。對於任何追蹤到您應用程式的要求，您可以看到有關請求、回應以及應用程式對下游 AWS 資源、微服務、資料庫和 HTTP Web APIs 進行呼叫的詳細資訊。

若要透過 Amazon SQS 傳送 AWS X-Ray 追蹤標頭，您可以執行下列其中一項操作：

- 使用 X-Amzn-Trace-Id [追蹤標頭](#)。
- 使用 AWSTraceHeader [訊息系統屬性](#)。

若要收集錯誤和延遲的資料，您必須使用 [AWS X-Ray SDK](#) 檢測 [AmazonSQS](#) 用戶端。

您可以使用 AWS X-Ray 主控台來檢視 Amazon SQS 與應用程式使用之其他服務之間的連線映射。您也可以使用主控台來檢視指標，例如平均延遲和失敗率。如需詳細資訊，請參閱《AWS X-Ray 開發人員指南》中的 [Amazon SQS 和 AWS X-Ray](#)。

Amazon SQS 中的安全性

本節提供 Amazon SQS 安全性、身分驗證和存取控制以及 Amazon SQS 存取政策語言的相關資訊。

主題

- [Amazon SQS 中的資料保護](#)
- [Amazon SQS 中的身分和存取管理](#)
- [在 Amazon SQS 中記錄和監控](#)
- [Amazon SQS 的合規驗證](#)
- [Amazon SQS 的恢復能力](#)
- [Amazon SQS 的基礎設施安全性](#)
- [Amazon SQS 的安全最佳實務](#)

Amazon SQS 中的資料保護

AWS [共同責任模型](#)適用於 Amazon Simple Queue Service 中的資料保護。如此模型所述，AWS 負責保護執行所有的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱[資料隱私權常見問答集](#)。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 [AWS 共同的責任模型和 GDPR](#) 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《使用者指南》中的[使用 CloudTrail 追蹤](#)。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱[聯邦資訊處理標準 \(FIPS\) 140-3](#)。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 Amazon SQS 或使用 AWS 服務 主控台、API AWS CLI 或其他 AWS SDKs 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

以下各節提供有關 Amazon SQS 中資料保護的資訊。

Amazon SQS 中的資料加密

資料保護是指保護往返 Amazon SQS 的傳輸中資料，以及存放在 Amazon SQS 資料中心內磁碟的靜態資料。您可以透過 Secure Sockets Layer (SSL) 或用戶端加密來保護傳輸中的資料。根據預設，Amazon SQS 會使用磁碟加密來存放訊息和檔案。您可以請求 Amazon SQS 先加密您的訊息，再將訊息儲存至其資料中心的加密檔案系統，以保護靜態資料。Amazon SQS 建議使用 SSE 來最佳化資料加密。

主題

- [Amazon SQS 中的靜態加密](#)
- [Amazon SQS 金鑰管理](#)

Amazon SQS 中的靜態加密

伺服器端加密 (SSE) 可讓您在加密佇列中傳輸敏感資料。SSE 會使用 SQS 受管加密金鑰 (SSE-SQS) 或 AWS Key Management Service (SSE-KMS) 中管理的金鑰來保護佇列中的訊息內容。如需使用 管理 SSE 的資訊 AWS Management Console，請參閱下列內容：

- [為佇列 \(主控台\) 設定 SSE-SQS](#)
- [為佇列 \(主控台\) 設定 SSE-KMS](#)

如需使用 適用於 Java 的 AWS SDK (以及 [CreateQueue](#)、和 [GetQueueAttributes](#) 動作) 管理 SSE 的資訊 [SetQueueAttributes](#)，請參閱下列範例：

- [搭配 Amazon SQS 佇列使用伺服器端加密](#)
- [設定的 KMS 許可 AWS 服務](#)

Amazon SQS 一收到訊息，SSE 就會將其加密。這些訊息以加密形式存放，並且 Amazon SQS 僅在將訊息傳送給授權的消費者時才會解密訊息。

Important

所有對啟用 SSE 之佇列的請求都必須使用 HTTPS 和[簽章版本 4](#)。

使用預設金鑰 (Amazon SQS 的 AWS 受管 KMS 金鑰) 的[加密佇列](#)無法在不同的 中叫用 Lambda 函數 AWS 帳戶。

可使用 AWS Security Token Service [AssumeRole](#) 動作將通知傳送至 Amazon SQS AWS 的服務部分功能與 SSE 相容，但僅適用於標準佇列：

- [Auto Scaling Lifecycle Hooks](#)
- [AWS Lambda 無效字母佇列](#)

如需有關其他服務與加密主題相容的資訊，請參閱 [設定 AWS 服務的 KMS 許可](#) 與您的服務文件。

AWS KMS 結合了安全、高可用性的硬體和軟體，以提供針對雲端擴展的金鑰管理系統。當您搭配 使用 Amazon SQS 時 AWS KMS，加密訊息資料的[資料金鑰](#)也會加密，並使用其保護的資料存放。

以下為使用 AWS KMS 的優點：

- 您可以自行建立和管理 [AWS KMS keys](#)。
- 您也可以使用 Amazon SQS 的 AWS 受管 KMS 金鑰，這對於每個帳戶和區域都是唯一的。
- AWS KMS 安全標準可協助您符合加密相關的合規要求。

如需詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的[什麼是 AWS Key Management Service ?](#)。

加密範圍

SSE 會加密 Amazon SQS 佇列中的訊息內文。

SSE 不會加密下列項目：

- 佇列中繼資料 (佇列名稱和屬性)
- 訊息中繼資料 (訊息 ID、時間戳記和屬性)
- 每個佇列指標

加密訊息可讓未授權或匿名使用者無法使用其內容。啟用 SSE 後，對加密佇列的匿名 `SendMessage` 和 `ReceiveMessage` 請求將被拒絕。Amazon SQS 安全性最佳實務建議您不要使用匿名請求。如果您希望將匿名請求傳送到 Amazon SQS 佇列，請務必停用 SSE。這不會影響 Amazon SQS 的正常運作：

- 只有在啟用加密佇列後傳送的訊息，才會對訊息進行加密。Amazon SQS 不會加密待處理訊息。
- 即使已停用其佇列的加密，已加密訊息仍會維持加密。

將訊息移到[無效信件佇列](#)並不會影響其加密：

- 當 Amazon SQS 將訊息從已加密來源佇列移到未加密的無效字母佇列，訊息會保持加密。
- 當 Amazon SQS 將訊息從未加密來源佇列移到已加密的無效字母佇列，訊息仍會保持未加密。

重要用語

以下重要術語有助於您更加了解 SSE 的功能。如需描述的詳細資訊，請參閱 [Amazon Simple Queue Service API 參考](#)。

資料金鑰

金鑰 (DEK) 負責加密 Amazon SNS 訊息的內容。

如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》和《AWS Encryption SDK 開發人員指南》中的[資料金鑰](#)。

資料金鑰重複使用期間

Amazon SQS 可以在 AWS KMS 再次呼叫之前重複使用資料金鑰來加密或解密訊息的時間長度，以秒為單位。代表秒數的整數，介於 60 秒 (1 分鐘) 和 86,400 秒 (24 小時) 之間。預設值為 300 (5 分鐘)。如需詳細資訊，請參閱[了解資料金鑰重複使用期間](#)。

Note

萬一無法連線 AWS KMS，Amazon SQS 會繼續使用快取的資料金鑰，直到重新建立連線為止。

KMS 金鑰 ID

在您的帳戶或其他帳戶中，AWS 受管 KMS 金鑰或自訂 KMS 金鑰的別名、別名 ARN、金鑰 ID 或金鑰 ARN。雖然 Amazon SQS 的 AWS 受管 KMS 金鑰別名一律為 `alias/aws/sqs`，但自訂 KMS 金鑰的別名可以是 `alias/MyAlias`。您可以使用這些 KMS 金鑰來保護 Amazon SQS 佇列中的訊息。

Note

請謹記以下幾點：

- 如果您未指定自訂 KMS 金鑰，Amazon SQS 會使用 Amazon SQS 的 AWS 受管 KMS 金鑰。
- 第一次使用 AWS Management Console 為佇列指定 Amazon SQS 的 AWS 受管 KMS 金鑰時，AWS KMS 會為 Amazon SQS 建立 AWS 受管 KMS 金鑰。
- 或者，當您第一次在啟用 SSE 的佇列上使用 `SendMessage` 或 `SendMessageBatch` 動作時，會為 Amazon SQS AWS KMS 建立 AWS 受管 KMS 金鑰。

您可以建立 KMS 金鑰、定義控制如何使用 KMS 金鑰的政策，以及使用主控台的客戶受管金鑰區段或動作來稽核 KMS 金鑰用量。AWS KMS [CreateKey](#) AWS KMS 如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的 [KMS 金鑰](#) 和 [建立金鑰](#)。如需 KMS 金鑰識別碼的更多範例，請參閱 AWS Key Management Service API 參考 API 中的 [KeyId](#)。如需有關尋找 KMS 金鑰識別碼的資訊，請參閱《AWS Key Management Service 開發人員指南》中的 [尋找金鑰 ID 和 ARN](#)。

Important

使用需支付額外費用 AWS KMS。如需詳細資訊，請參閱 [估算 AWS KMS 成本](#) 和 [AWS Key Management Service 定價](#)。

信封加密

加密資料的安全性有一部分取決於保護能夠解密資料的資料金鑰。Amazon SQS 會使用 KMS 金鑰為資料金鑰加密，然後使用加密的訊息儲存加密的資料金鑰。這種使用 KMS 金鑰來加密資料金鑰的做法，就是所謂的信封加密。

如需詳細資訊，請參閱《AWS Encryption SDK 開發人員指南》中的 [信封加密](#)。

Amazon SQS 金鑰管理

Amazon SQS 與 AWS Key Management Service (KMS) 整合，以管理伺服器端加密 (SSE) 的 [KMS 金鑰](#)。如需 SSE 資訊和金鑰管理定義，請參閱 [Amazon SQS 中的靜態加密](#)。Amazon SQS 使用 KMS 金鑰驗證和保護用於加密和解密訊息的資料金鑰。以下各節提供有關在 Amazon SQS 服務中使用 KMS 金鑰和資料金鑰的資訊。

設定 AWS KMS 許可

每個 KMS 金鑰都必須有一個金鑰政策。請注意，您無法修改 Amazon SQS AWS 受管 KMS 金鑰的金鑰政策。此 KMS 金鑰的政策包括帳戶中的所有主體 (獲授權可使用 Amazon SQS) 使用加密佇列的許可。

對於客戶受管 KMS 金鑰，您必須設定金鑰政策，以新增每個佇列生產者和消費者的許可。若要執行這項操作，您可以將生產者和消費者命名為 KMS 金鑰政策中的使用者。如需 AWS KMS 許可的詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[AWS KMS 資源和操作](#)或 [AWS KMS API 許可參考](#)。

或者，您也可以指派給主體 (這些主體會產生和消費加密訊息) 的 IAM 政策中，指定必要的許可。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的 [在 AWS KMS 中使用 IAM 政策](#)。

Note

雖然您可以設定全域許可來傳送和接收 Amazon SQS，但 AWS KMS 需要明確命名 IAM 政策 Resource 區段中特定區域中 KMS 金鑰的完整 ARN。

設定 AWS 服務的 KMS 許可

數個 AWS 服務可做為事件來源，將事件傳送至 Amazon SQS 佇列。若要允許這些事件來源使用加密佇列，您必須建立客戶受管 KMS 金鑰，並在金鑰政策中新增許可，服務才能使用所需的 AWS KMS API 方法。執行下列步驟來設定許可。

Warning

變用於加密 Amazon SQS 訊息的 KMS 金鑰時，請注意，使用舊 KMS 金鑰加密的現有訊息將保持使用該金鑰加密。若要解密這些訊息，您必須保留舊的 KMS 金鑰，並確保其金鑰政策授予 Amazon SQS `kms:Decrypt` 和 `kms:GenerateDataKey` 的許可。更新至用於加密新訊

息的新 KMS 金鑰後，請確保在刪除或停用舊 KMS 金鑰之前，處理所有以舊 KMS 金鑰加密的現有訊息並從佇列中移除。

1. 建立客戶管理的 KMS 金鑰。如需詳細資訊，請參閱《AWS Key Management Service 開發人員指南》中的[建立索引鍵](#)。
2. 若要允許 AWS 服務事件來源使用 kms:Decrypt 和 kms:GenerateDataKey API 方法，請將下列陳述式新增至 KMS 金鑰政策。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "service.amazonaws.com"
    },
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": "*"
  }]
}
```

將上述範例中的 "service" 取代為事件來源的服務名稱。事件來源包括下列服務。

事件來源	服務名稱
Amazon CloudWatch Events	events.amazonaws.com
Amazon S3 事件通知	s3.amazonaws.com
Amazon SNS 主題訂閱	sns.amazonaws.com

3. 使用 KMS 金鑰的 ARN [設定現有的 SSE 佇列](#)。
4. 提供加密佇列的 ARN 至事件來源。

設定生產者的 AWS KMS 許可

當[資料金鑰重複使用期間](#)到期時，生產者下次呼叫 `SendMessage` 或 `SendMessageBatch` 也會觸發呼叫 `kms:Decrypt` 和 `kms:GenerateDataKey`。呼叫 `kms:Decrypt` 是為了在使用新資料金鑰之前，先驗證其完整性。因此，生產者對於 KMS 金鑰必須擁有 `kms:Decrypt` 和 `kms:GenerateDataKey` 許可。

將下列陳述式新增至生產者的 IAM 政策。請記得對金鑰資源和佇列資源使用正確的 ARN 值。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }, {
    "Effect": "Allow",
    "Action": [
      "sqs:SendMessage"
    ],
    "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
  }]
}
```

設定取用者的 AWS KMS 許可

當資料金鑰重複使用期間到期時，消費者下一次呼叫 `ReceiveMessage` 也會觸發呼叫 `kms:Decrypt`，以在使用新資料金鑰之前，先驗證其完整性。因此，消費者對於用來加密指定佇列中訊息的 KMS 金鑰，必須擁有 `kms:Decrypt` 許可。如果佇列做為[無效字母佇列](#)使用，則消費者對於用來加密來源佇列中訊息的 KMS 金鑰，還必須擁有 `kms:Decrypt` 許可。將下列陳述式新增至消費者的 IAM 政策。請記得對金鑰資源和佇列資源使用正確的 ARN 值。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt"
    ]
  }]
}
```

```

    ],
    "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }, {
    "Effect": "Allow",
    "Action": [
      "sqs:ReceiveMessage"
    ],
    "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
  }]
}

```

使用混淆代理人保護設定 AWS KMS 許可

當金鑰政策陳述句中的主體為 [AWS 服務主體](#)，則可使用 [aws:SourceArn](#) 或 [aws:SourceAccount](#) 全域條件金鑰來防止[混淆代理人問題](#)。若要使用這些條件金鑰，請將值設定為要加密之資源的 Amazon Resource Name (ARN)。如果您不知道資源的 ARN，請改為使用 `aws:SourceAccount`。

在此 KMS 金鑰政策中，允許帳戶 111122223333 擁有的服務的特定資源為 Decrypt 和 GenerateDataKey 動作呼叫 KMS，這些都會在 SSE 使用 Amazon SQS 期間發生。

```

{
  "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:service::111122223333:resource"
        ]
      }
    }
  }]
}

```

使用啟用 SSE 的 Amazon SQS 佇列時，下列服務支援 `aws:SourceArn`：

- Amazon SNS
- Amazon S3
- CloudWatch Events
- AWS Lambda
- CodeBuild
- Amazon Connect Customer Profiles
- AWS Auto Scaling
- Amazon Chime

了解資料金鑰重複使用期間

[資料金鑰重複使用期間](#)定義 Amazon SQS 可重複使用相同資料金鑰的最長持續時間。當資料金鑰重複使用期間結束時，Amazon SQS 會產生新的資料金鑰。請注意下列有關重複使用期間的準則。

- 較短的重複使用期間可提供更好的安全性，但會導致更多呼叫 AWS KMS，這可能會產生超出 免費 方案的費用。
- 雖然加密和解密的資料金鑰是分別快取，但重複使用期間同時套用到這兩個資料金鑰的副本。
- 當資料金鑰重複使用期間結束時，對 `SendMessage` 的下一個呼叫或 `SendMessageBatch` 通常會觸發對 `GenerateDataKey` 的 AWS KMS 呼叫，以取得新的資料金鑰。此外，下一個呼叫 `SendMessage` 和 `ReceiveMessage` 將觸發呼叫 AWS KMS `Decrypt`，以在使用前驗證資料金鑰的完整性。
- [委託人](#) (AWS 帳戶 或使用者) 不會共用資料金鑰 (由唯一委託人傳送的訊息一律會取得唯一資料金鑰)。因此，對 的呼叫量 AWS KMS 是資料金鑰重複使用期間使用的唯一主體數量的倍數。

估算 AWS KMS 成本

為了預測成本並更好地了解您的 AWS 帳單，您可能想知道 Amazon SQS 使用您的 KMS 金鑰的頻率。

 Note

下列公式可以提供預期成本的良好概念，但由於 Amazon SQS 的分佈特性，實際成本可能會比較高。

若要計算 R 每個佇列的 API 請求數量 ()，請使用下列公式：

$$R = (B / D) * (2 * P + C)$$

B 是帳單週期 (以秒為單位)。

D 是[資料金鑰重複使用期間](#) (以秒為單位)。

P 是傳送至 Amazon SQS 佇列的生產[主體](#)數目。

C 是從 Amazon SQS 佇列接收的消費主體數目。

⚠ Important

一般來說，生產主體會導致消耗主體的雙倍成本。如需詳細資訊，請參閱 [了解資料金鑰重複使用期間](#)。

如果生產者和消費者有不同的使用者，成本會增加。

以下為計算範例。如需確切的定價資訊，請參閱 [AWS Key Management Service 定價](#)。

範例 1：計算 2 個委託人和 1 個佇列的 AWS KMS API 呼叫數量

此範例假設如下：

- 帳單週期是 1 月 1-31 日 (2,678,400 秒)。
- 資料金鑰重複使用期間設為 5 分鐘 (300 秒)。
- 有 1 個佇列。
- 有 1 個生產主體和 1 個消費主體。

$$(2,678,400 / 300) * (2 * 1 + 1) = 26,784$$

範例 2：計算多個生產者和消費者以及 2 個佇列的 AWS KMS API 呼叫數量

此範例假設如下：

- 帳單週期是 2 月 1-28 日 (2,419,200 秒)。
- 資料金鑰重複使用期間設為 24 小時 (86,400 秒)。
- 有 2 個佇列。

- 第 1 個佇列有 3 個生產主體和 1 個消費主體。
- 第 2 個佇列有 5 個生產主體和 2 個消費主體。

$$(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532$$

AWS KMS 錯誤

當您使用 Amazon SQS 和 時 AWS KMS，您可能會遇到錯誤。下列參考說明錯誤和可能的疑難排解解決方案。

- [常見的 AWS KMS 錯誤](#)
- [AWS KMS 解密錯誤](#)
- [AWS KMS GenerateDataKey 錯誤](#)

Amazon SQS 中的網際網路流量隱私權

適用於 Amazon SQS 的 Amazon Virtual Private Cloud (Amazon VPC) 端點是 VPC 中的邏輯實體，僅允許連線到 Amazon SQS。VPC 會將請求路由至 Amazon SQS，並將回應路由回 VPC。以下各節提供使用 VPC 端點以及建立 VPC 端點政策的相關資訊。

適用於 Amazon SQS 的 Amazon Virtual Private Cloud 端點

如果您使用 Amazon VPC 託管 AWS 資源，您可以在 VPC 與 Amazon SQS 之間建立連線。您可以使用此連線來將訊息傳送至 Amazon SQS 佇列，而不超過公有網際網路。

Amazon VPC 可讓您在自訂虛擬網路中啟動 AWS 資源。您可利用 VPC 來控制您的網路設定，例如 IP 地址範圍、子網路、路由表和網路閘道。如需有關 Amazon VPC 的詳細資訊，請參閱《[Amazon VPC 使用者指南](#)》。

若要將 VPC 連接至 Amazon SQS，您必須先定義介面 VPC 端點，其可讓您將 VPC 連接至其他 AWS 服務。端點能為 Amazon SQS 提供可靠、可擴展性的連線，無須使用網際網路閘道、網路位址轉譯 (NAT) 執行個體或 VPN 連接。如需詳細資訊，請參閱本指南中的 [教學課程：從 Amazon Virtual Private Cloud 傳送訊息至 Amazon SQS 佇列](#) [Amazon Virtual Private Cloud](#) 和 [範例 5：如果不是來自 VPC 端點，則拒絕存取](#)，以及《Amazon VPC 使用者指南》中的 [介面 VPC 端點 \(AWS PrivateLink\)](#)。

Important

- Amazon 虛擬私有雲端只能搭配 HTTPS Amazon SQS 端點使用。

- 當您設定 Amazon SQS 從 Amazon VPC 傳送訊息時，您必須啟用私有 DNS，並指定格式為 `sqs.us-east-2.amazonaws.com` 或雙堆疊端點 `sqs.us-east-2.api.aws` 的端點。
- Amazon SQS 也使用端點服務透過 PrivateLink 支援 FIPS `com.amazonaws.region.sqs-fips` 端點。您可以連線至格式為 `sqs-fips.region.amazonaws.com` 的 FIPS 端點。
- 在 Amazon Virtual Private Cloud 中使用雙堆疊端點時，將使用 IPv4 和 IPv6 傳送請求。
- 私有 DNS 不支援 `queue.amazonaws.com` 或 `us-east-2.queue.amazonaws.com` 之類的延遲端點

為 Amazon SQS 建立 VPC 端點政策

您可以為 Amazon SQS 的 Amazon VPC 端點建立政策，在其中您可以指定以下內容：

- 可執行動作的主體。
- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 [使用 VPC 端點控制服務的存取](#)。

以下範例 VPC 端點政策指定允許使用者 MyUser 傳送訊息到 Amazon SQS 佇列 MyQueue。

```
{
  "Statement": [{
    "Action": ["sqs:SendMessage"],
    "Effect": "Allow",
    "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

拒絕以下各項：

- 其他 Amazon SQS API 動作，例如 `sqs:CreateQueue` 和 `sqs>DeleteQueue`。
- 其他嘗試使用此 VPC 端點的 使用者和規則。
- MyUser 傳送訊息至不同的 Amazon SQS 佇列。

Note

IAM 使用者仍然可以從外部 VPC 使用其他 Amazon SQS API 動作。如需詳細資訊，請參閱 [範例 5：如果不是來自 VPC 端點，則拒絕存取](#)。

Amazon SQS 中的身分和存取管理

AWS Identity and Access Management (IAM) 是一種 AWS 服務，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制哪些人員可進行身分驗證 (登入) 並獲得授權 (具有許可) 以使用 Amazon SQS 資源。IAM 是您可以免費使用 AWS 服務的。

目標對象

使用方式 AWS Identity and Access Management (IAM) 會有所不同，取決於您在 Amazon SQS 中執行的工作。

服務使用者 – 如果您使用 Amazon SQS 執行任務，您的管理員會為您提供您需要的憑證和許可。隨著您為了執行作業而使用的 Amazon SQS 功能數量變多，您可能會需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 Amazon SQS 中的功能，請參閱 [對 Amazon Simple Queue Service 身分和存取進行故障診斷](#)。

服務管理員 – 若您在公司負責管理 Amazon SQS 資源，您應該擁有 Amazon SQS 的完整存取權。您的任務是判斷服務使用者應存取的 Amazon SQS 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司搭配 Amazon SQS 使用 IAM 的方式，請參閱 [如何搭配 IAM 使用 Amazon Simple Queue Service](#)。

IAM 管理員 - 如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 Amazon SQS 存取權的詳細資訊。若要檢視您可以在 IAM 中使用的基 Amazon SQS 身分型政策範例，請參閱 [政策最佳實務](#)。

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入的方式。您必須以 AWS 帳戶根使用者身分、IAM 使用者身分或擔任 IAM 角色來驗證 (登入 AWS)。

您可以使用透過身分來源提供的登入資料，以聯合身分 AWS 身分身分登入。AWS IAM Identity Center (IAM Identity Center) 使用者、您的單一登入身分驗證，以及您的 Google 或 Facebook 登

入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱AWS 登入 《使用者指南》中的[如何登入您的 AWS 帳戶](#)。

如果您以 AWS 程式設計方式存取，AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的憑證以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的[適用於 API 請求的AWS Signature 第 4 版](#)。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如，AWS 建議您使用多重要素驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的[多重要素驗證](#)和《IAM 使用者指南》中的[IAM 中的AWS 多重要素驗證](#)。

AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取帳戶中的所有 AWS 服務和資源。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的[需要根使用者憑證的任務](#)。

聯合身分

最佳實務是要求人類使用者，包括需要管理員存取權的使用者，使用身分提供者的聯合身分來 AWS 服務使用臨時憑證來存取。

聯合身分是來自您的企業使用者目錄、Web 身分提供者、AWS Directory Service、Identity Center 目錄，或使用透過身分來源提供的 AWS 服務登入資料存取的任何使用者。當聯合身分存取時 AWS 帳戶，它們會擔任角色，而角色會提供臨時登入資料。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連接並同步到您自己的身分來源中的一組使用者 AWS 帳戶和群組，以便在所有和應用程式中使用。如需 IAM Identity Center 的詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的[什麼是 IAM Identity Center ?](#)。

IAM 使用者和群組

[IAM 使用者](#)是 中的身分 AWS 帳戶，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有

長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 [IAM 使用者指南](#) 中的為需要長期憑證的使用案例定期輪換存取金鑰。

[IAM 群組](#) 是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供臨時憑證。如需更多資訊，請參閱《IAM 使用者指南》中的 [IAM 使用者的使用案例](#)。

IAM 角色

[IAM 角色](#) 是 中具有特定許可 AWS 帳戶 的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在 中擔任 IAM 角色 AWS Management Console，您可以從[使用者切換至 IAM 角色（主控台）](#)。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的[擔任角色的方法](#)。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱《[IAM 使用者指南](#)》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的[許可集](#)。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以直接將政策連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 [IAM 中的跨帳戶資源存取](#)。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
- 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有當服務收到需要與

其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱 [《轉發存取工作階段》](#)。

- 服務角色 – 服務角色是服務擔任的 [IAM 角色](#)，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的 [建立角色以委派許可權給 AWS 服務](#)。
- 服務連結角色 – 服務連結角色是一種連結至的服務角色類型 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的 [使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式](#)。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件，當與身分或資源相關聯時，AWS 會定義其許可。當委託人（使用者、根使用者或角色工作階段）發出請求時，會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 [JSON 政策概觀](#)。

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console AWS CLI、或 API AWS 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的 [透過客戶管理政策定義自訂 IAM 許可](#)。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策之間選擇，請參閱《IAM 使用者指南》中的[在受管政策和內嵌政策間選擇](#)。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中[指定主體](#)。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF 和 Amazon VPC 是支援 ACLs 的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的[存取控制清單 \(ACL\) 概觀](#)。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 Principal 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的[IAM 實體許可界限](#)。
- 服務控制政策 (SCPs) – SCPs 是 JSON 政策，可指定中組織或組織單位 (OU) 的最大許可 AWS Organizations。AWS Organizations 是一種服務，用於分組和集中管理您企業擁有 AWS 帳戶的多個。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱《AWS Organizations 使用者指南》中的[服務控制政策](#)。
- 資源控制政策 (RCP) - RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的

許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需 Organizations 和 RCPs 的詳細資訊，包括支援 RCPs AWS 服務的清單，請參閱 AWS Organizations 《使用者指南》中的[資源控制政策 \(RCPs\)](#)。

- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的[工作階段政策](#)。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的[政策評估邏輯](#)。

在 Amazon SQS 中管理存取的概觀

每個 AWS 資源都由擁有 AWS 帳戶，而建立或存取資源的許可是由許可政策所管理。帳戶管理員可以將許可政策連接到 IAM 身分 (使用者、群組和角色)，有些服務 (例如 Amazon SQS) 也支援將許可政策連接到資源。

Note

帳戶管理員 (或管理員使用者) 是由具有管理權限的使用者。如需詳細資訊，請參《[IAM 使用者指南](#)》中的 IAM 最佳實務。

授與許可時，您指定哪些使用者取得許可、他們獲得許可的資源，以及可以對這些資源進行的特定動作。

Amazon Simple Queue Service 資源和操作

在 Amazon SQS 中，唯一的資源是佇列。在政策中，您可以使用 Amazon Resource Name (ARN) 來識別要套用政策的資源。以下資源都有與其關聯的唯一 ARN：

資源類型	ARN 格式
佇列	<code>arn:aws:sqs: <i>region</i>:<i>account_id</i> :<i>queue_name</i></code>

以下是佇列的 ARN 格式範例：

- my_queue 美國東部（俄亥俄）區域中名為 my_queue 之佇列的 ARN，屬於 AWS 帳戶 123456789012：

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- 在 Amazon SQS 支援的每個不同區域中，名為 my_queue 之佇列的 ARN：

```
arn:aws:sqs:*:123456789012:my_queue
```

- 在佇列名稱中使用 * 或 ? 做為萬用字元的 ARN。在下列範例中，ARN 符合字首為 my_prefix_ 的所有佇列：

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

您可以透過呼叫 [GetQueueAttributes](#) 動作，取得現有佇列的 ARN 值。QueueArn 屬性的值是佇列的 ARN。如需關於 ARN 的詳細資訊，請參閱《IAM 使用者指南》中的 [IAM ARN](#)。

Amazon SQS 提供可用於佇列資源的一組動作。如需詳細資訊，請參閱 [Amazon SQS API 許可：動作和資源參考](#)。

了解資源所有權

無論誰建立資源，都會 AWS 帳戶 擁有在帳戶中建立的資源。具體而言，資源擁有者就是對資源建立請求進行身分驗證的主體實體 (即根帳戶、使用者或 IAM 角色) 的 AWS 帳戶。下列範例說明其如何運作：

- 如果您使用的根帳戶登入 AWS 帳戶 資料來建立 Amazon SQS 佇列，則您的 AWS 帳戶 是資源的擁有者（在 Amazon SQS 中，資源是 Amazon SQS 佇列）。
- 如果您在 中建立使用者，AWS 帳戶 並將建立佇列的許可授予使用者，則使用者可以建立佇列。但是您的 AWS 帳戶 (也是該使用者所屬的帳戶) 擁有該佇列資源。
- 如果您在 中建立 AWS 帳戶 具有建立 Amazon SQS 佇列許可的 IAM 角色，則任何可以擔任該角色的人都可以建立佇列。您的 AWS 帳戶 (角色所屬的) 擁有佇列資源。

管理 資源的存取

許可政策說明授予帳戶的許可權。下一節則說明用於建立許可政策的可用選項。

 Note

本節著重討論如何在 Amazon SQS 的環境中使用 IAM，它不提供 IAM 服務的詳細資訊。如需完整的 IAM 文件，請參閱《IAM 使用者指南》中的[什麼是 IAM？](#)。如需有關 IAM 政策語法和說明的資訊，請參閱IAM 使用者指南中的 [AWS IAM 政策參考](#)。

連接到 IAM 身分的政策稱為身分類型政策 (IAM 政策)，而連接到資源的政策參考資源類型政策。

身分型政策

有兩種方式可讓您授與 Amazon SQS 佇列的許可給使用者：使用 Amazon SQS 政策系統，以及使用 IAM 政策系統。您可以使用任一種系統或兩者都使用，將政策連接至使用者或角色。在大多數情況下，使用任一系統都能達到相同的結果。例如，您可以執行下列動作：

- 將許可政策連接至您帳戶中的使用者或群組 - 若要授與使用者建立 Amazon SQS 佇列的許可，您可以將許可政策連接至使用者或使用者所屬的群組。
- 將許可政策連接到另一個 中的使用者 AWS 帳戶 – 您可以將許可政策連接到另一個 中的使用者 AWS 帳戶，以允許他們與 Amazon SQS 佇列互動。不過，跨帳戶許可不適用於下列動作：

跨帳戶許可權不會套用至下列動作：

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

若要授予這些動作的存取權，使用者必須屬於 AWS 帳戶 擁有 Amazon SQS 佇列的相同。

- 將許可政策連接至角色（授予跨帳戶許可） – 若要將跨帳戶許可授予 SQS 佇列，您必須同時結合 IAM 和資源型政策：
 1. 在帳戶 A 中（擁有佇列）：
 - 將資源型政策連接至 SQS 佇列。此政策必須明確授予必要的許可（例如 [SendMessage](#)、[ReceiveMessage](#)）給帳戶 B 中的委託人（例如 IAM 角色）。
 2. 在帳戶 A 中，建立 IAM 角色：
 - 允許帳戶 B 或 AWS 服務擔任角色的信任政策。

 Note

如果您希望 AWS 服務（例如 Lambda 或 EventBridge）擔任該角色，請在信任政策中指定服務委託人（例如 `lambda.amazonaws.com`）。

- 以身分為基礎的政策，授予擔任角色與佇列互動的許可。
3. 在帳戶 B 中，授予許可以擔任帳戶 A 中的角色。

您必須設定佇列的存取政策，以允許跨帳戶主體。僅 IAM 身分型政策不足以跨帳戶存取 SQS 佇列。

如需使用 IAM 來委派許可的相關資訊，請參閱《IAM 使用者指南》中的[存取管理](#)。

雖然 Amazon SQS 可與 IAM 政策搭配使用，但它有自己的政策基礎設施。您可以將 Amazon SQS 政策與佇列搭配使用，以指定哪些 AWS 帳戶可以存取佇列。您可以指定存取權類型和條件（例如，如果請求在 2010 年 12 月 31 日之前提出則授與使用 `SendMessage`、`ReceiveMessage` 之許可的條件）。您可以授予許可的特定動作是整體 Amazon SQS 動作清單的一部分。當您撰寫 Amazon SQS 政策並將 * 指定為「允許所有 Amazon SQS 動作」，這表示使用者可以執行這個子集中的所有動作。

下圖說明其中一個涵蓋動作子集的基本 Amazon SQS 政策概念。此政策適用於 `queue_xyz`，並授予 AWS 帳戶 1 和 AWS 帳戶 2 許可，以將任何允許的動作用於指定的佇列。

 Note

政策中的資源指定為 `123456789012/queue_xyz`，其中 123456789012 是擁有佇列的帳戶的帳戶 AWS ID。

SQS Policy on queue_xyz

Allow who:

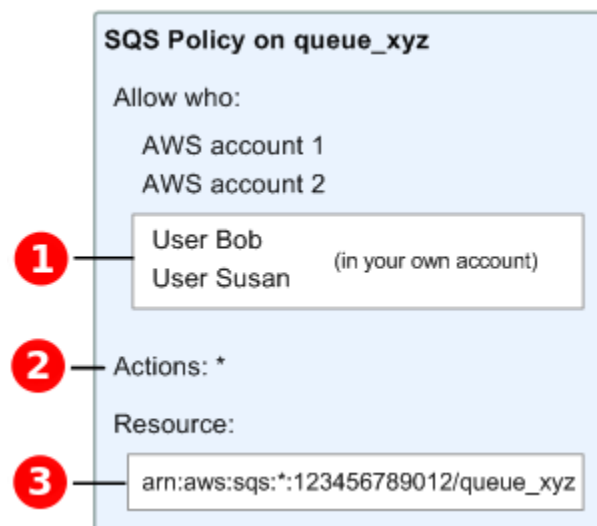
AWS account 1
AWS account 2

Actions: *

Resource:

123456789012/queue_xyz

隨著引進 IAM 以及使用者和 Amazon Resource Name (ARN) 的概念，SQS 政策產生了一些改變。以下圖表說明這些改變。



1 如需為不同帳戶中的使用者授予許可的詳細資訊，請參閱《IAM 使用者指南》中的[教學課程：使用 IAM 角色委派跨 AWS 帳戶存取](#)。

2 * 中包含的動作子集已擴展。如需允許動作的清單，請參閱[Amazon SQS API 許可：動作和資源參考](#)。

3 您可以使用 Amazon Resource Name (ARN) 來指定資源，這是在 IAM 政策中指定資源的標準方法。如需有關 Amazon SQS 佇列的 ARN 格式的資訊，請參閱 [Amazon Simple Queue Service 資源和操作](#)。

您

例如，根據上圖中的 Amazon SQS 政策，擁有 AWS 帳戶 1 或 AWS 帳戶 2 安全登入資料的任何人都可以存取 queue_xyz。此外，在您自己 AWS 帳戶中的使用者 Bob 和 Susan (具有 ID 123456789012) 都能存取佇列。

在引進 IAM 前，Amazon SQS 會自動給予佇列建立者對於佇列的完全控制權 (也就是該佇列上所有可能 Amazon SQS 動作的存取權)。但現在已不是這樣，除非建立者使用 AWS 安全憑證。具有建立佇列許可的任何使用者還必須具有使用其他 Amazon SQS 動作的許可，才能對他建立的佇列進行任意操作。

以下是範例政策，允許使用者使用所有 Amazon SQS 動作，但僅限於名稱字首為常值字串 bob_queue_ 的佇列。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
  }]
}
```

如需詳細資訊，請參閱 [搭配 Amazon SQS 使用政策](#) 和《IAM 使用者指南》中的 [身分 \(使用者、群組和角色\)](#)。

指定政策元素：動作、效果、資源和主體

對於每一個 [Amazon Simple Queue Service 資源](#)，該服務會定義一組 [動作](#)。為了授予這些動作的許可，Amazon SQS 定義了一組可在政策中指定的動作。

Note

執行動作可能需要不只一個動作的許可。在授與特定動作的許可時，您也同時識別允許或拒絕對其執行動作的資源。

以下是最基本的政策元素：

- 資源 – 在政策中，您可以使用 Amazon Resource Name (ARN) 來識別要套用政策的資源。
- 動作 – 您可以使用動作關鍵字來識別您要允許或拒絕的資源動作。例如，sqs:CreateQueue 許可權允許使用者執行 Amazon Simple Queue Service CreateQueue 動作。

- 效果 - 您可以指定使用者要求特定動作時會有什麼效果；可為允許或拒絕。如果您不明確授與資源的存取權，將會隱含拒絕存取。您也可以明確拒絕資源的存取權，這樣做可以確保使用者無法存取資源，即使另有其他政策授與存取。
- 主體：在身分型政策 (IAM 政策) 中，政策所連接的使用者就是隱含主體。對於資源型政策，您可以指定想要收到許可的使用者、帳戶、服務或其他實體 (僅適用於資源型政策)。

如需進一步了解 Amazon SQS 政策語法和描述的詳細資訊，請參閱《IAM 使用者指南》中的 [AWS IAM 政策參考](#)。

如需所有 Amazon Simple Queue Service 動作及其所套用之資源的資料表，請參閱 [Amazon SQS API 許可：動作和資源參考](#)。

如何搭配 IAM 使用 Amazon Simple Queue Service

在您使用 IAM 管理對 Amazon SQS 的存取權之前，請瞭解哪些 IAM 功能可以與 Amazon SQS 搭配使用。

您可以搭配 Amazon Simple Queue Service 使用的 IAM 功能

IAM 功能	Amazon SQS 支援
身分型政策	是
資源型政策	是
政策動作	是
政策資源	是
政策條件索引鍵 (服務特定)	是
ACL	否
ABAC(政策中的標籤)	部分
臨時憑證	是
轉送存取工作階段 (FAS)	是

IAM 功能	Amazon SQS 支援
服務角色	是
服務連結角色	否

若要全面了解 Amazon SQS 和其他 AWS 服務如何與大多數 IAM 功能搭配使用，請參閱《[AWS IAM 使用者指南](#)》中的與 IAM 搭配使用的 [服務](#)。

存取控制

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF 和 Amazon VPC 是支援 ACLs 的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的 [存取控制清單 \(ACL\) 概觀](#)。

Note

請務必了解，所有 AWS 帳戶 都可以將其許可委派給其帳戶下的使用者。跨帳戶存取可讓您共用 AWS 資源的存取權，而不必管理其他使用者。如需使用跨帳戶存取的相關資訊，請參閱 IAM 使用者指南中的 [啟用跨帳戶存取權](#)。

如需 Amazon SQS 自訂政策中跨內容許可權和條件金鑰的進一步詳細資訊，請參閱 [Amazon SQS 自訂政策的限制](#)。

Amazon SQS 身分型政策

支援身分型政策：是

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的 [透過客戶管理政策定義自訂 IAM 許可](#)。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。若要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 [IAM JSON 政策元素參考](#)。

Amazon SQS 的身分型政策範例

若要檢視 Amazon SQS 身分型政策的範例，請參閱 [政策最佳實務](#)。

Amazon SQS 中的資源型政策

支援資源型政策：是

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中[指定主體](#)。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

如需啟用跨帳戶存取權，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，做為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當主體和資源位於不同的時 AWS 帳戶，信任帳戶中的 IAM 管理員也必須授予主體實體（使用者或角色）存取資源的許可。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。如需詳細資訊，請參閱《IAM 使用者指南》中的 [IAM 中的快帳戶資源存取](#)。

Amazon SQS 的政策動作

支援政策動作：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 Action 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

若要查看 Amazon SNS 動作的清單，請參閱服務授權參考中的 [Amazon Simple Queue Service 定義的資源](#)。

Amazon SQS 中的政策動作會在動作之前使用下列字首：

sqs

若要在單一陳述式中指定多個動作，請用逗號分隔。

```
"Action": [  
    "sqs:action1",  
    "sqs:action2"  
]
```

若要檢視 Amazon SQS 身分型政策的範例，請參閱 [政策最佳實務](#)。

Amazon SQS 的政策資源

支援政策資源：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 [Amazon Resource Name \(ARN\)](#) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*" 
```

若要查看 Amazon SQS 資源類型及其 ARN 的清單，請參閱服務授權參考中的 [Amazon Simple Queue Service 定義的動作](#)。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 [Amazon Simple Queue Service 定義的資源](#)。

若要檢視 Amazon SQS 身分型政策的範例，請參閱 [政策最佳實務](#)。

Amazon SQS 的政策條件金鑰

支援服務特定政策條件金鑰：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用 [條件運算子](#) 的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵，AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值，會使用邏輯 OR 操作 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需更多資訊，請參閱 IAM 使用者指南中的 [IAM 政策元素：變數和標籤](#)。

AWS 支援全域條件金鑰和服務特定的條件金鑰。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的 [AWS 全域條件內容索引鍵](#)。

若要查看 Amazon SQS 條件索引鍵的清單，請參閱服務授權參考中的 [Amazon Simple Queue Service 的條件索引鍵](#)。若要了解您可以搭配哪些動作和資源使用條件金鑰，請參閱 [Amazon Simple Queue Service 定義的資源](#)。

若要檢視 Amazon SQS 身分型政策的範例，請參閱 [政策最佳實務](#)。

Amazon SQS 的 ACL

支援 ACL：否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

搭配使用 ABAC 與 Amazon SQS

支援 ABAC (政策中的標籤)：部分

屬性型存取控制 (ABAC) 是一種授權策略，可根據屬性來定義許可。在 AWS 中，這些屬性稱為標籤。您可以將標籤連接至 IAM 實體 (使用者或角色) 和許多 AWS 資源。為實體和資源加上標籤是 ABAC 的第一步。您接著要設計 ABAC 政策，允許在主體的標籤與其嘗試存取的資源標籤相符時操作。

ABAC 在成長快速的環境中相當有幫助，並能在政策管理變得繁瑣時提供協助。

如需根據標籤控制存取，請使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的 [條件元素](#) 中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的 [使用 ABAC 授權定義許可](#)。如要查看含有設定 ABAC 步驟的教學課程，請參閱 IAM 使用者指南中的 [使用屬性型存取控制 \(ABAC\)](#)。

將暫時憑證與 Amazon SQS 搭配使用

支援臨時憑證：是

當您使用臨時登入資料登入時，有些 AWS 服務 無法運作。如需詳細資訊，包括哪些 AWS 服務 使用臨時登入資料，請參閱 [《AWS 服務 IAM 使用者指南》](#) 中的使用 IAM 的。

如果您 AWS Management Console 使用使用者名稱和密碼以外的任何方法登入，則使用臨時登入資料。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取時，該程序會自動建立臨時登入資料。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的[從使用者切換至 IAM 角色 \(主控台\)](#)。

您可以使用 AWS CLI 或 AWS API 手動建立臨時登入資料。然後，您可以使用這些臨時登入資料來存取 AWS。AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱 [IAM 中的暫時性安全憑證](#)。

轉送 Amazon SQS 的存取工作階段

支援轉寄存取工作階段 (FAS)：是

當您使用 IAM 使用者或角色在 中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有當服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱[轉發存取工作階段](#)。

Amazon SQS 的服務角色

支援服務角色：是

服務角色是服務擔任的 [IAM 角色](#)，可代您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的[建立角色以委派許可權給 AWS 服務](#)。

Warning

變更服務角色的許可可能會中斷 Amazon SQS 功能。只有在 Amazon SQS 提供指引時，才能編輯服務角色。

Amazon SQS 的服務連結角色

支援服務連結角色：否

服務連結角色是連結至的一種服務角色 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服务連結角色的詳細資訊，請參閱[可搭配 IAM 運作的AWS 服務](#)。在表格中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

AWS 受管政策的 Amazon SQS 更新

若要新增許可給使用者、群組和角色，使用 AWS 受管政策比自己撰寫政策更容易。[建立 IAM 客戶受管政策](#)需要時間和專業知識，而受管政策可為您的團隊提供其所需的許可。若要快速開始使用，您可以使用 AWS 受管政策。這些政策涵蓋常見的使用案例，並可在您的 AWS 帳戶中可用。如需 AWS 受管政策的詳細資訊，請參閱《IAM 使用者指南》中的 [AWS 受管政策](#)。

AWS 服務會維護和更新 AWS 受管政策。您無法變更 AWS 受管政策中的許可。服務偶爾會將其他許可新增至 AWS 受管政策，以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群組和角色)。當新功能啟動或新操作可用時，服務最有可能更新 AWS 受管政策。服務不會從 AWS 受管政策中移除許可，因此政策更新不會破壞您現有的許可。

此外，AWS 支援跨多個 服務之任務函數的受管政策。例如，ReadOnlyAccess 這項 AWS 受管政策提供針對所有 AWS 服務和資源的唯讀存取權限。當服務啟動新功能時，會為新操作和資源 AWS 新增唯讀許可。如需任務職能政策的清單和說明，請參閱 IAM 使用者指南中[有關任務職能的AWS 受管政策](#)。

AWS 受管政策：AmazonSQSFullAccess

您可將 AmazonSQSFullAccess 政策連接到 Amazon SQS 身分。此政策授與允許 Amazon SQS 完整存取的許可。

若要檢視此政策的許可，請參閱 AWS 受管政策參考中的 [AmazonSQSFullAccess](#)。

AWS 受管政策：AmazonSQSReadOnlyAccess

您可將 AmazonSQSReadOnlyAccess 政策連接到 Amazon SQS 身分。此政策授與允許 Amazon SQS 唯讀存取的許可。

若要檢視此政策的許可，請參閱 AWS 受管政策參考中的 [AmazonSQSReadOnlyAccess](#)。

AWS 受管政策：SQSUnlockQueuePolicy

如果您為成員帳戶設定不正確的佇列政策，以拒絕所有使用者存取您的 Amazon SQS 佇列，您可以使用 SQSUnlockQueuePolicy AWS 受管政策來解鎖佇列。

如需如何移除設定錯誤佇列政策以拒絕所有主體存取 Amazon SQS 佇列的詳細資訊，請參閱《IAM 使用者指南》中的[對 AWS Organizations 成員帳戶執行特權任務](#)。

AWS 受管政策的 Amazon SQS 更新

檢視自此服務開始追蹤 Amazon SQS AWS 受管政策更新以來的詳細資訊。如需有關此頁面變更的自動提醒，請訂閱 Amazon SQS [文件歷程記錄](#)頁面上的 RSS 摘要。

變更	描述	日期
SQSUnlockQueuePolicy	Amazon SQS 新增了名為 AWS 的新受管政策 SQSUnlockQueuePolicy，以解鎖佇列，並移除設定錯誤的佇列政策，拒絕所有主體存取 Amazon SQS 佇列。	2024 年 11 月 15 日
AmazonSQSReadOnlyAccess	Amazon SQS 新增 ListQueueTags 動作，這會擷取與指定 Amazon SQS 佇列相關聯的所有標籤。它可讓您檢視已指派給佇列的鍵/值對，用於組織或中繼資料目的。此動作會與 ListQueueTags API 操作相關聯。	2024 年 6 月 20 日
AmazonSQSReadOnlyAccess	Amazon SQS 新增了一個新動作，可讓您列出特定來源佇列下最新的訊息移動任務 (最多 10 個)。此動作會與	2023 年 6 月 9 日

變更	描述	日期
	ListMessageMoveTasks API 操作相關聯。	

對 Amazon Simple Queue Service 身分和存取進行故障診斷

請使用以下資訊來協助您診斷和修正使用 Amazon SQS 和 IAM 時可能遇到的常見問題。

我未獲授權，不得在 Amazon SQS 中執行動作

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在 mateojackson 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 `sqs:GetWidget` 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
sqs:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 Mateo 政策，允許他使用 `sqs:GetWidget` 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 `iam:PassRole` 動作，您的政策必須更新，允許您將角色傳遞給 Amazon SQS。

有些 AWS 服務可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 marymajor 的 IAM 使用者嘗試使用主控台在 Amazon SQS 中執行動作時，發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 Amazon SQS 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 Amazon SQS 是否支援這些功能，請參閱 [如何搭配 IAM 使用 Amazon Simple Queue Service](#)。
- 若要了解如何 AWS 帳戶 在您擁有的 資源間提供存取權，請參閱《[IAM 使用者指南](#)》中的在[您擁有 AWS 帳戶 的另一個 IAM 使用者中提供存取權](#)。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的[將存取權提供給第三方 AWS 帳戶 擁有](#)。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的[將存取權提供給在外部進行身分驗證的使用者 \(聯合身分\)](#)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 [IAM 中的跨帳戶資源存取](#)。

我想要解除鎖定我的佇列

如果您的 AWS 帳戶 屬於組織，AWS Organizations 政策可能會封鎖您存取 Amazon SQS 資源。根據預設，AWS Organizations 政策不會封鎖對 Amazon SQS 的任何請求。不過，請確定您的 AWS Organizations 政策尚未設定為封鎖對 Amazon SQS 佇列的存取。如需如何檢查 AWS Organizations 政策的說明，請參閱AWS Organizations 《使用者指南》中的[列出所有政策](#)。

此外，如果您錯誤地設定成員帳戶的佇列政策以拒絕所有使用者存取您的 Amazon SQS 佇列，您可以透過在 IAM 中啟動成員帳戶的特權工作階段來解鎖佇列。啟動特權工作階段後，您可以刪除設定錯誤的佇列政策，以重新取得佇列的存取權。如需詳細資訊，請參閱《IAM 使用者指南》中的[在 AWS Organizations 成員帳戶上執行特殊權限任務](#)。

搭配 Amazon SQS 使用政策

這個主題提供身分型政策範例，在該政策中帳戶管理員可以將許可政策連接至 IAM 身分 (使用者、群組和角色)。

⚠ Important

建議您先檢閱簡介主題，其中說明基本概念及有何選項可供您管理對 Amazon Simple Queue Service 資源的存取權。如需詳細資訊，請參閱 [在 Amazon SQS 中管理存取的概觀](#)。

在 ListQueues 例外情況下，所有 Amazon SQS 動作皆會支援資源層級許可權。如需詳細資訊，請參閱 [Amazon SQS API 許可：動作和資源參考](#)。

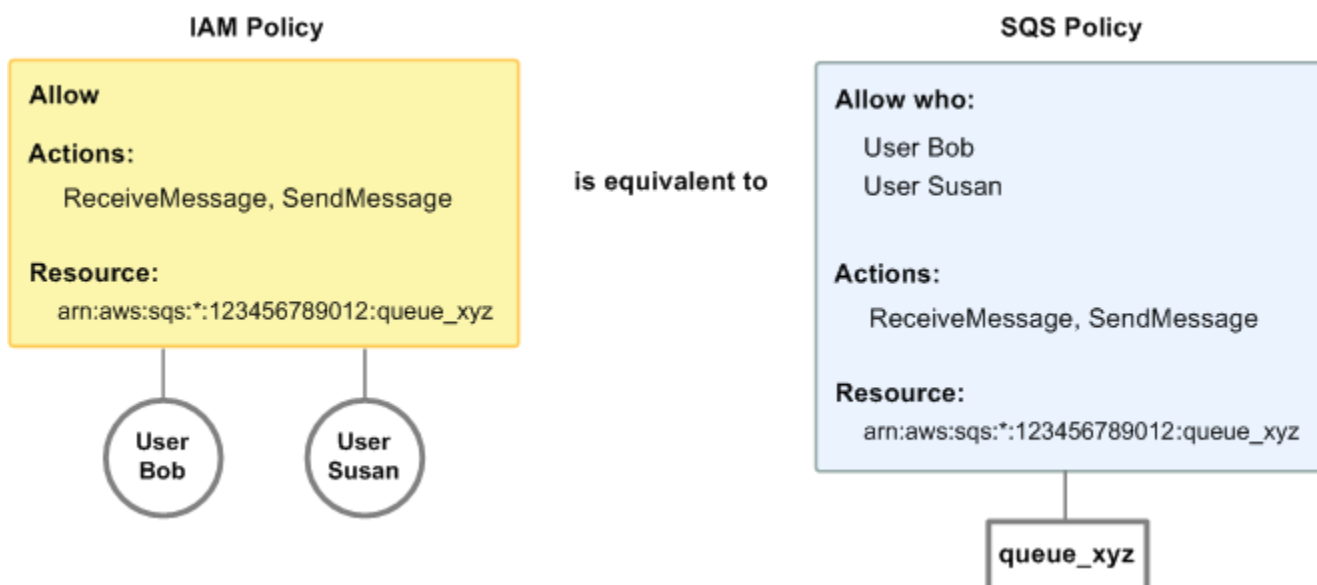
使用 Amazon SQS 和 IAM 政策

有兩種方式可授予使用者許可給 Amazon SQS 資源：使用 Amazon SQS 政策系統（以資源為基礎的政策）和使用 IAM 政策系統（以身分為基礎的政策）。您可以使用一種或兩種方法，但 ListQueues 動作除外，這是只能在 IAM 政策中設定的區域許可。

例如，下圖顯示 IAM 政策和相當的 Amazon SQS 政策。IAM 政策授予 queue_xyz 您 AWS 帳戶中呼叫之佇列的 Amazon SQS ReceiveMessage 和 SendMessage 動作權利，且政策會連接到名為 Bob 和 Susan 的使用者 (Bob 和 Susan 具有政策中所述的許可)。此 Amazon SQS 政策也提供 Bob 和 Susan 相同佇列的 ReceiveMessage 和 SendMessage 動作權利。

📌 Note

下列範例顯示沒有條件的簡單政策。您可以在任一政策中指定特定條件，並取得相同結果。



IAM 和 Amazon SQS 政策之間存在一個主要差異：Amazon SQS 政策系統可讓您將許可授予其他 AWS 帳戶，而 IAM 則不會。

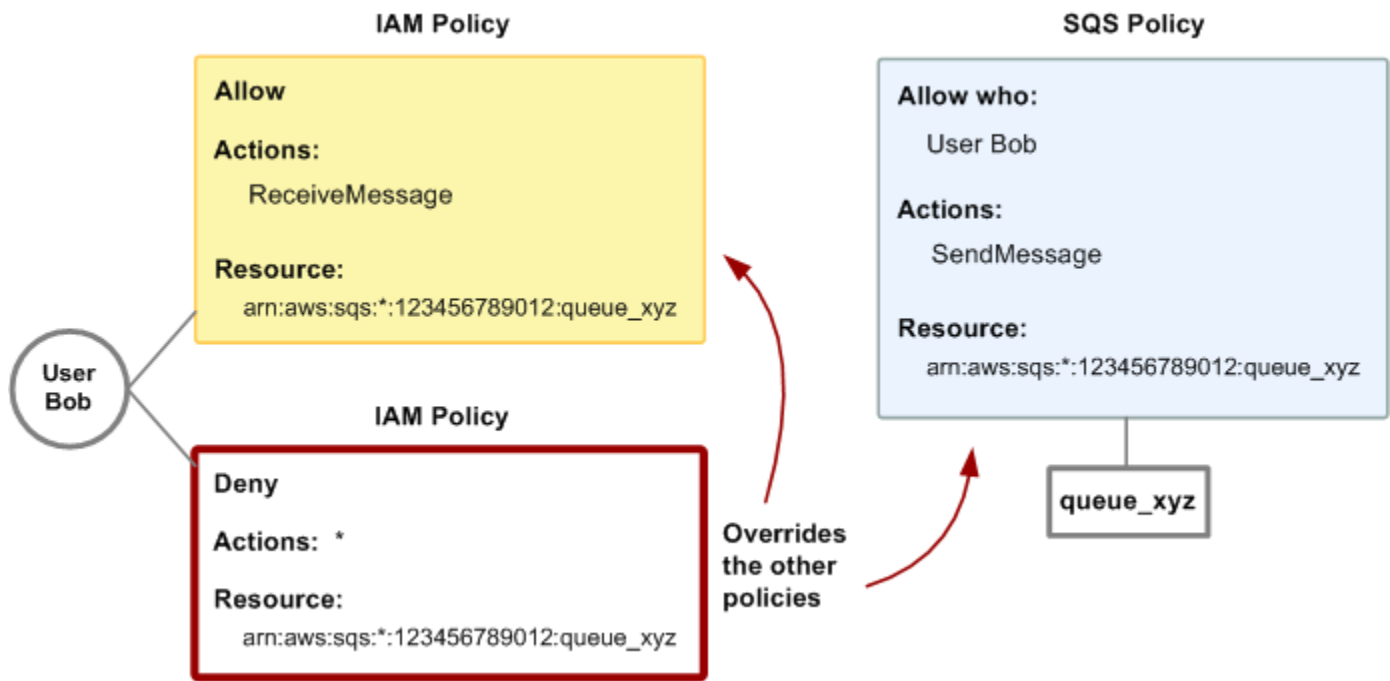
由您決定如何一起使用兩個系統來管理您的許可。以下範例顯示兩個政策系統如何搭配運作。

- 在第一個範例中，Bob 同時有 IAM 政策和 Amazon SQS 政策套用到他的帳戶。IAM 政策授予他的帳戶在 queue_xyz 上的 ReceiveMessage 動作許可，而 Amazon SQS 政策提供他的帳戶在相同佇列上的 SendMessage 動作許可。此圖說明了此概念。



如果 Bob 傳送 ReceiveMessage 請求至 queue_xyz，IAM 政策會允許此動作。如果 Bob 傳送 SendMessage 請求至 queue_xyz，Amazon SQS 政策會允許此動作。

- 在第二個範例中，Bob 濫用他對於 queue_xyz 的存取權，所以必須移除他對於佇列的整個存取權。最簡單的方式是新增一個拒絕他存取該佇列所有動作的政策。此政策會覆寫其他兩個政策，因為明確deny一律覆寫allow。如需政策評估邏輯的相關資訊，請參閱[搭配 Amazon SQS 存取政策語言使用自訂政策](#)。此圖說明了此概念。



您也可以將其他陳述式新增到拒絕 Bob 任何佇列存取類型的 Amazon SQS 政策。這和新增 IAM 政策來拒絕 Bob 存取佇列擁有相同的效果。如需涵蓋 Amazon SQS 動作和資源之政策的詳細資訊，請參閱 [Amazon SQS 政策的基本範例](#)。如需撰寫 Amazon SQS 政策的詳細資訊，請參閱 [搭配 Amazon SQS 存取政策語言使用自訂政策](#)。

使用 Amazon SQS 主控台所需的許可

想要使用 Amazon SQS 主控台的使用者，必須擁有一組最基本的許可，才能使用其 AWS 帳戶中的 Amazon SQS 佇列。例如，使用者必須具備呼叫 `ListQueues` 動作的許可才能列出佇列，必須具備呼叫 `CreateQueue` 動作的許可才能建立佇列。除了 Amazon SQS 許可權限，若要將 Amazon SQS 佇列訂閱至 Amazon SNS 主題，主控台另外還需要 Amazon SNS 動作的許可。

如果您建立了比最基本的必要許可更嚴格的 IAM 政策，對於採取該 IAM 政策的使用者，主控台可能無法如預期運作。

對於僅呼叫 AWS CLI 或 Amazon SQS 動作的使用者，您不需要允許最低主控台許可。

Amazon SQS 的身分型政策範例

根據預設，IAM 使用者和角色不具備建立或修改 Amazon SQS 資源的許可。他們也無法使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 AWS API 來執行任務。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

如需了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的[建立 IAM 政策 \(主控台\)](#)。

如需 Amazon SQS 所定義之動作和資源類型的詳細資訊，包括每種資源類型的 ARN 格式，請參閱《服務授權參考》中的[適用 Amazon Simple Queue Service 的動作、資源和條件索引鍵](#)。

Note

當您設定 Amazon EC2 Auto Scaling 的生命週期關聯時，您無須撰寫將訊息傳送至 Amazon SQS 佇列的政策。如需詳細資訊，請參閱[《Amazon EC2 使用者指南》中的 Amazon EC2 Auto Scaling 生命週期掛鉤](#)。Amazon EC2

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 Amazon SQS 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱 IAM 使用者指南中的[AWS 受管政策](#)或[任務職能的 AWS 受管政策](#)。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的[IAM 中的政策和許可](#)。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 等使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南中的[IAM JSON 政策元素：條件](#)。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的[使用 IAM Access Analyzer 驗證政策](#)。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 [IAM 安全最佳實務](#)。

使用 Amazon SQS 主控台

若要存取 Amazon Simple Queue Service 主控台，您必須擁有最基本的一組許可。這些許可必須允許您列出和檢視中 Amazon SQS 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為了確保使用者和角色仍然可以使用 Amazon SQS 主控台，也請將 Amazon SQS `AmazonSQSReadOnlyAccess` AWS 受管政策連接到實體。如需詳細資訊，請參閱《IAM 使用者指南》中的 [新增許可到使用者](#)。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視連接到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{
  "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": "*"
}
]
}
```

允許使用者建立佇列

在下列範例政策中，我們為 Bob 建立一個政策，允許他存取所有 Amazon SQS 動作，但僅能用於名稱字首為常值字串 `alice_queue_` 的佇列。

Amazon SQS 不會自動授予佇列建立者使用該佇列的許可。因此，除了 IAM 政策中的 `CreateQueue` 動作，我們還必須明確授與 Bob 使用所有 Amazon SQS 動作的許可。

```

{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
  }]
}
```

允許開發人員將訊息寫入共用佇列

在下列範例中，我們為開發人員建立群組，並連接政策，讓群組使用 Amazon SQS `SendMessage` 動作，但僅適用於屬於指定 AWS 帳戶 且名為 `MyCompanyQueue` 的佇列。

```

{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
  }]
}
```

您可以使用 `*` 代替 `SendMessage` 授與主體對共享佇列進行以下動

作：`ChangeMessageVisibility`、`DeleteMessage`、`GetQueueAttributes`、`GetQueueUrl`、`ReceiveMessage` 和 `SendMessage`。

Note

儘管 `*` 包含由其他許可類型提供的存取權，Amazon SQS 會個別考量各項許可。例如，可以將 `*` 和 `SendMessage` 許可同時授與一名使用者，即便 `*` 包含由 `SendMessage` 提供的存取權。移除許可時也適用此概念。若主體僅有 `*` 許可，提出移除 `SendMessage` 許可的請求並不會使主體只獨缺該項許可。反之，由於主體並未擁有明確的 `SendMessage` 許可，此請求將不會起作用。若想要讓主體僅有 `ReceiveMessage` 許可，請先新增 `ReceiveMessage` 許可再移除 `*` 許可。

允許管理員取得佇列的一般大小

在下列範例中，我們為管理員建立群組，並連接政策，讓群組將 Amazon SQS `GetQueueAttributes` 動作與屬於指定 AWS 帳戶的所有佇列搭配使用。

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

允許合作夥伴傳送訊息至特定佇列

您可以使用 Amazon SQS 政策或 IAM 政策來完成這項工作。如果您的合作夥伴有 AWS 帳戶，則使用 Amazon SQS 政策可能會比較容易。不過，擁有 AWS 安全登入資料之合作夥伴公司中的任何使用者都可以傳送訊息到佇列。如果您想將存取權限制至特定的使用者或應用程式，您必須將合作夥伴視同您自己公司中的使用者，並改為使用 IAM 政策，而非 Amazon SQS 政策。

此範例執行下列動作：

1. 建立稱為 `WidgetCo` 的群組，來代表合作夥伴公司。
2. 為合作夥伴公司中需要存取權的特定使用者或應用程式，建立使用者。
3. 將使用者新增至群組。

4. 連接政策，提供僅能對名為 WidgetPartnerQueue 的佇列，僅限存取名為 SendMessage 之動作的存取權給群組。

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
  }]
}
```

Amazon SQS 政策的基本範例

本節說明常見 Amazon SQS 使用案例的範例政策。

當您將政策連接到使用者時，可以使用主控台來驗證每個政策的效果。起初，使用者沒有許可且無法在主控台進行任何操作。隨著您將政策連接到使用者，便可以驗證使用者在主控台上能夠執行各種動作。

Note

我們建議您使用兩個瀏覽器視窗：一個用於授予許可，另一個 AWS Management Console 用於使用使用者的登入資料登入，以便在將許可授予使用者時驗證許可。

範例 1：授予一個許可給一個 AWS 帳戶

下列範例政策授予美國東部（俄亥俄）444455556666/queue1 區域中名為 之佇列的 111122223333 SendMessage 許可 AWS 帳戶 編號。

```
{
  "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"
  ]
}
```

範例 2：將兩個許可授予一個 AWS 帳戶

下列範例政策會授予名為 之佇列111122223333的 SendMessage和 ReceiveMessage許可 AWS 帳戶 編號444455556666/queue1。

```
{
  "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"
  ]
}
```

範例 3：將所有許可授予兩個 AWS 帳戶

下列範例政策授予兩個不同的 AWS 帳戶 數字 (111122223333 和 444455556666) 許可，以使用 Amazon SQS 允許123456789012/queue1在美國東部（俄亥俄）區域中對名為 的佇列共用存取的所有動作。

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

範例 4：授與跨帳戶許可給角色和使用者名稱

下列範例政策授予 role1 和 AWS 帳戶 數字 username1 下的 111122223333 跨帳戶許可，以使用 Amazon SQS 允許 123456789012/queue1 在美國東部（俄亥俄）區域中對名為 的佇列共用存取的所有動作。

跨帳戶許可權不會套用至下列動作：

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

```
{
  "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"
  ]
}

```

範例 5：授與許可給所有使用者

以下範例政策授與所有使用者 (匿名使用者) 對名為 111122223333/queue1 之佇列的 ReceiveMessage 許可。

```

{
  "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"
  }]
}

```

範例 6：授與有限時間的許可給所有使用者

以下範例政策授與所有使用者 (匿名使用者) 對名為 111122223333/queue1 之佇列的 ReceiveMessage 許可，但僅限於 2009 年 1 月 31 日下午 12:00 (中午) 到下午 3:00 的期間。

```

{
  "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"
    }
  }
}]
}

```

範例 7：授與所有許可給 CIDR 範圍的所有使用者

以下範例政策授予所有使用者 (匿名使用者) 許可，對名為 111122223333/queue1 的佇列使用所有可共用的 Amazon SQS 動作，但僅限來自 192.0.2.0/24 CIDR 範圍的請求。

```

{
  "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"
      }
    }
  }]
}

```

範例 8：不同 CIDR 範圍內使用者的允許清單和封鎖清單許可

以下範例政策有兩個陳述式：

- 第一個陳述式授與 192.0.2.0/24 CIDR 範圍內 (192.0.2.188 除外) 的所有使用者 (匿名使用者) 許可，對名為 111122223333/queue1 的佇列使用 SendMessage 動作。
- 第二個陳述式封鎖 12.148.72.0/23 CIDR 範圍內的所有使用者 (匿名使用者)，不允許使用佇列。

```

{

```



```

"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"
    }
  }
}]
}

```

搭配 Amazon SQS 存取政策語言使用自訂政策

若要僅根據 AWS 帳戶 ID 授予基本許可 (例如 [SendMessage](#) 或 [ReceiveMessage](#))，您不需要撰寫自訂政策。請改用 Amazon SQS [AddPermission](#) 動作。

若要根據請求時間或請求者的 IP 地址等特定條件允許或拒絕存取，您必須建立自訂 Amazon SQS 政策，並使用 [SetQueueAttributes](#) 動作將其上傳。

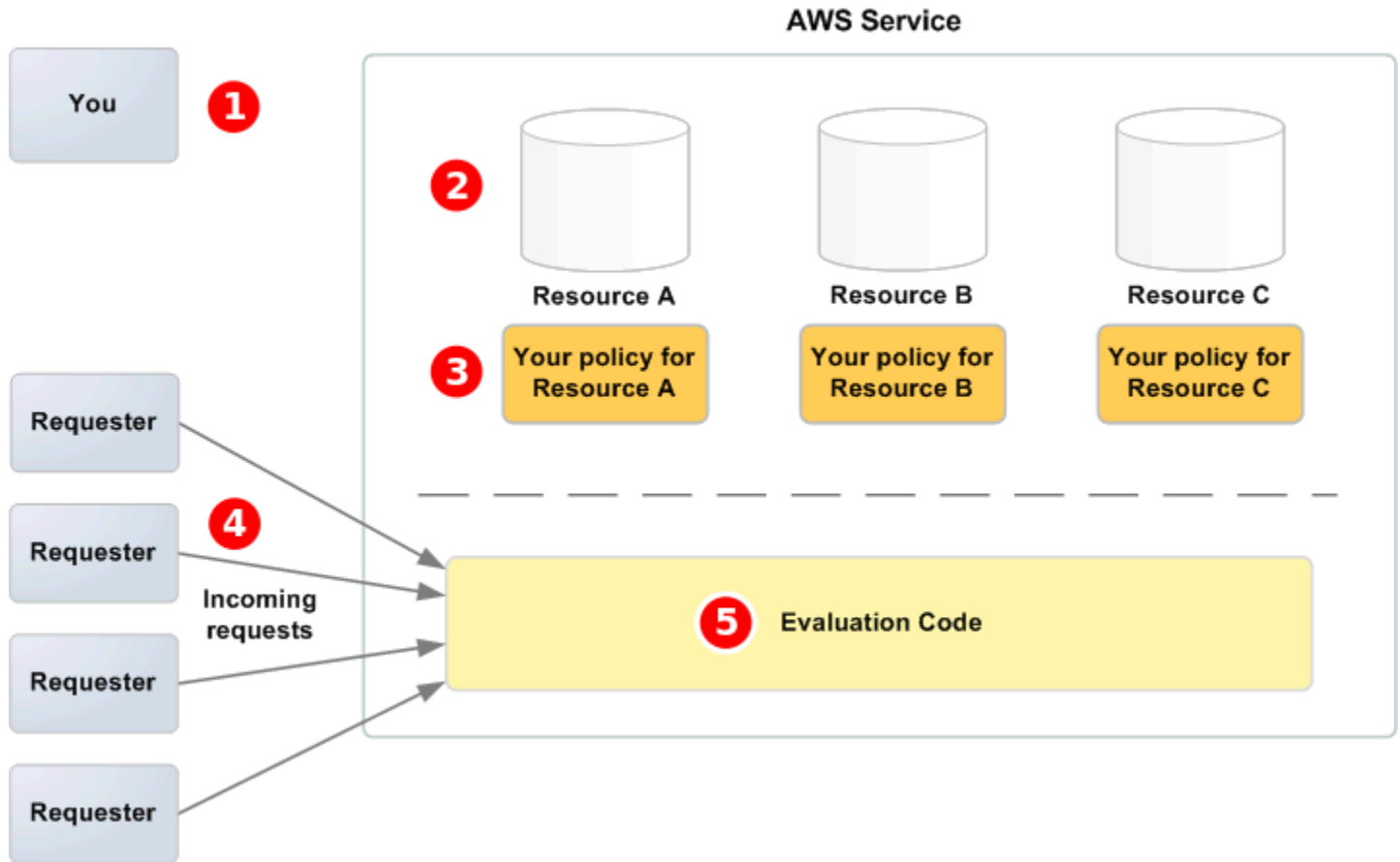
主題

- [Amazon SQS 存取控制架構](#)
- [Amazon SQS 存取控制處理工作流程](#)
- [Amazon SQS 存取政策語言重要概念](#)

- [Amazon SQS 存取政策語言評估邏輯](#)
- [Amazon SQS 存取政策語言中明確拒絕和預設拒絕之間的關係](#)
- [Amazon SQS 自訂政策的限制](#)
- [自訂 Amazon SQS 存取政策語言範例](#)

Amazon SQS 存取控制架構

下圖說明 Amazon SQS 資源的存取控制。



1
您是資源擁有者。

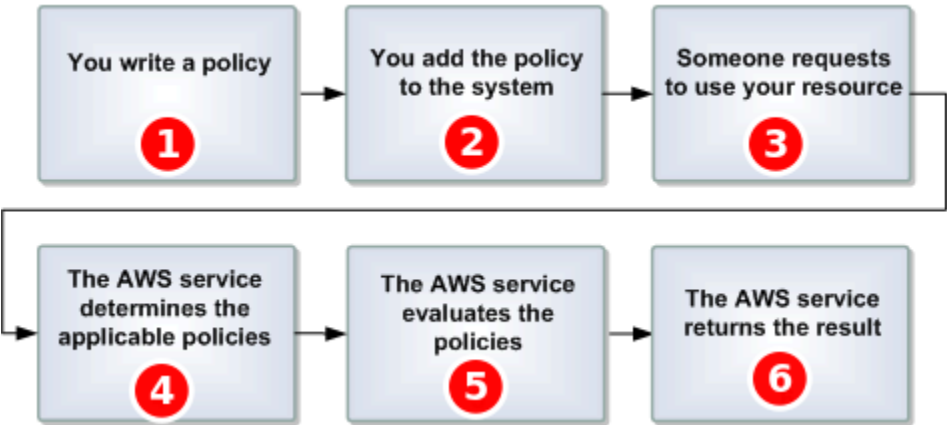
2
服務中包含的資源 AWS（例如 Amazon SQS 佇列）。

3
您的政策。每個資源擁有一個政策是理想的做法。AWS 此服務提供您用來上傳和管理政策的 API。

- 4
- 申請者及其傳入的 AWS 服務請求。
- 5
- 存取政策語言評估代碼。這是 AWS 服務內的一組程式碼，可針對適用的政策評估傳入請求，並判斷是否允許請求者存取資源。

Amazon SQS 存取控制處理工作流程

下圖說明使用 Amazon SQS 存取政策語言的一般控制存取流程。



- 1
- 為您的佇列撰寫 Amazon SQS 政策。
- 2
- 您可以將政策上傳至 AWS。AWS 此服務提供您用來上傳政策的 API。例如，您使用 Amazon SQS SetQueueAttributes 動作，為特定 Amazon SQS 佇列上傳政策。
- 3
- 某人傳送請求，要求使用您的 Amazon SQS 佇列。
- 4
- Amazon SQS 會檢查所有可用Amazon SQS 政策，並判斷何者適用。
- 5
- SQS 評估政策並判定是否允許申請者使用您的佇列。

6

根

據政策評估結果，Amazon SQS 傳回 Access denied 錯誤給申請者或繼續處理請求。

Amazon SQS 存取政策語言重要概念

若要撰寫自己的政策，您必須熟悉 [JSON](#) 和數個重要概念。

Allow

將[效果](#)設為allow之[Statement](#)的結果。

Action

[Principal](#)獲許可執行的活動，通常是對 AWS 提出請求。

預設拒絕

沒有[Allow](#)或[明確拒絕](#)設定的[Statement](#)結果。

Condition

有關[許可](#)的任何限制或詳細資訊。典型條件與日期和時間以及 IP 地址相關。

效果

您想要[政策](#)的[Statement](#)在評估時間傳回的結果。您在撰寫政策陳述式時指定deny或allow。政策評估時間有三個可能結果：[預設拒絕](#)、[Allow](#)和[明確拒絕](#)。

明確拒絕

將[效果](#)設為deny之[Statement](#)的結果。

評估

Amazon SQS 根據 [政策](#) 用來判斷應拒絕或允許傳入請求的程序。

發行者

撰寫[政策](#)以授與資源許可的使用者。根據定義，發行者一律是資源擁有者。AWS 不允許 Amazon SQS 使用者為他們未擁有的資源建立政策。

索引鍵

特定特性，即存取限制的基礎。

許可

使用[Condition](#)和[索引鍵](#)來允許或不允許存取資源的概念。

政策

做為一個或多個[陳述式](#)之容器的文件。



Amazon SQS 使用政策來判斷是否將資源的存取權授與使用者。

Principal

收到[政策](#)中[許可](#)的使用者。

Resource

[Principal](#)請求存取的物件。

Statement

以存取政策語言撰寫的單一許可正式描述，屬於更廣泛 [政策](#) 文件的一部分。

要求者

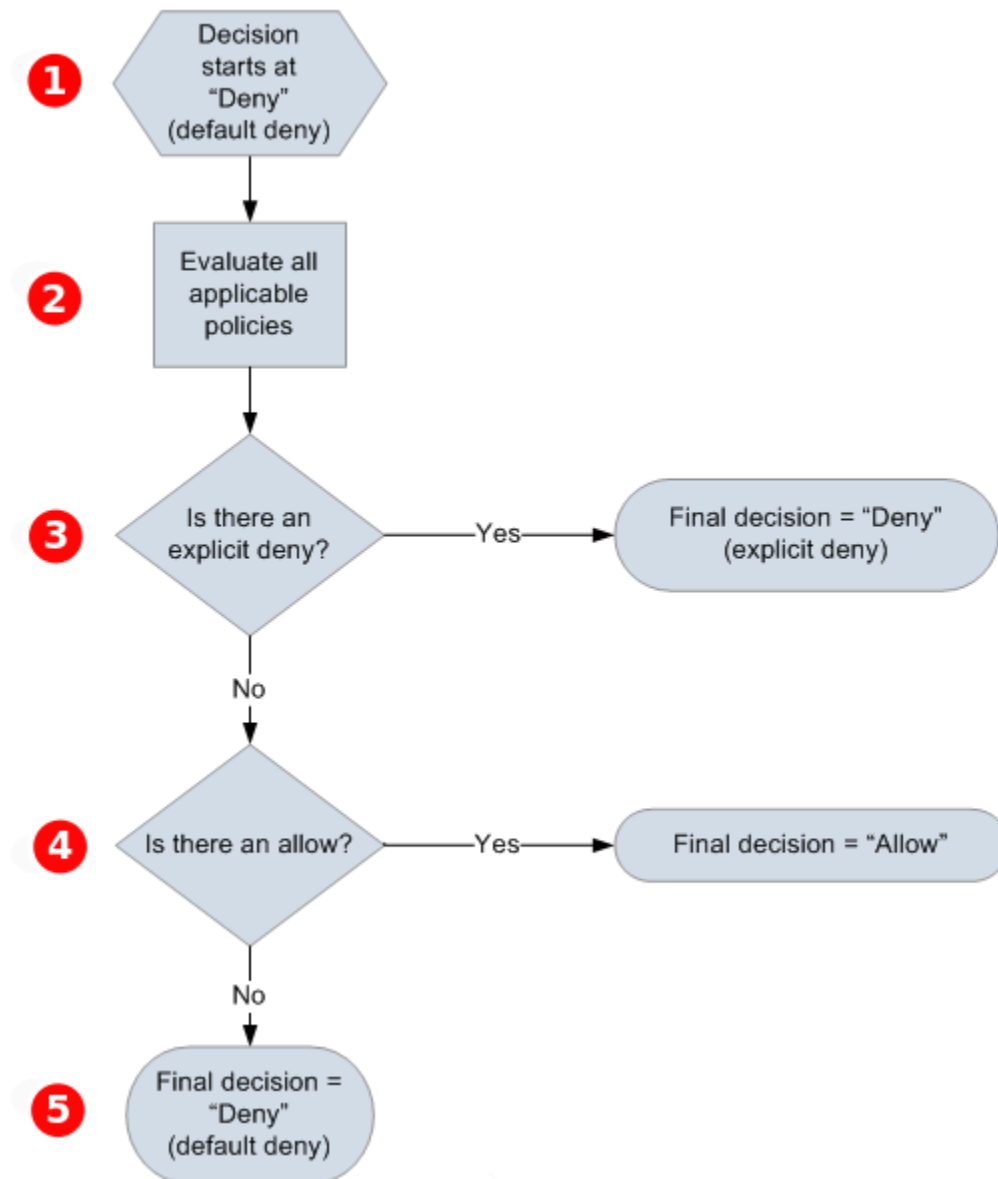
傳送存取[Resource](#)請求的使用者。

Amazon SQS 存取政策語言評估邏輯

在評估時間，Amazon SQS 會判斷應該允許或是拒絕來自資源擁有者以外其他人的請求。評估邏輯遵循幾個基本規則：

- 根據預設，會拒絕來自您以外任何人對使用您資源的所有請求。
- [Allow](#) 會覆寫任何 [預設拒絕](#)。
- [明確拒絕](#) 會覆寫任何 allow。
- 評估政策的順序並不重要。

下圖詳細說明 Amazon SQS 如何評估存取許可的決定。



1
決定以預設拒絕開始。

2
強制執行程式碼會評估所有適用於請求的政策 (根據資源、主體、動作和條件)。強制執行程式碼評估政策的順序並不重要。

3
強制執行程式碼會尋找可以套用到請求的明確拒絕指示。如果找到一個，強制執行程式碼會傳回拒絕的決定，然後完成程序。

4

如果找不到明確拒絕指示，強制執行程式碼會尋找適用於請求的任何允許指示。如果找到一個，強制執行程式碼會傳回允許的決定，然後完成程序 (服務繼續處理請求)。

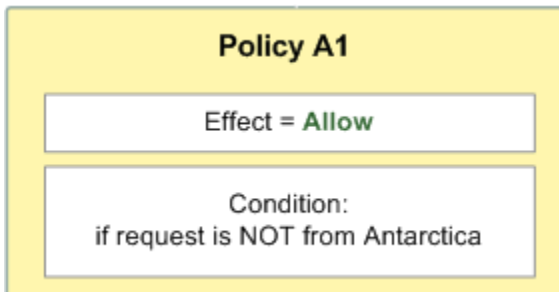
5

如果找不到允許指示，則最後決定是拒絕 (因為沒有明確拒絕或允許，這會被視為預設拒絕)。

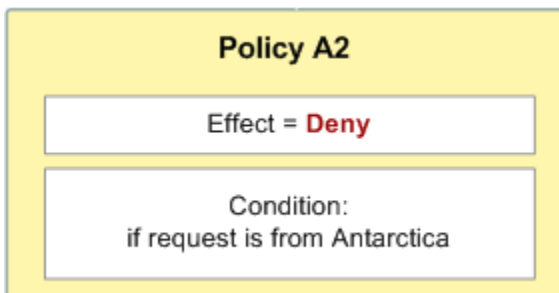
Amazon SQS 存取政策語言中明確拒絕和預設拒絕之間的關係

如果 Amazon SQS 政策無法直接套用至請求，請求會產生 [預設拒絕](#)。例如，如果使用者請求使用 Amazon SQS 的許可，但套用至該使用者的唯一政策是可以使用 DynamoDB，則請求會產生預設拒絕。

如果不符合陳述式中的條件，請求會產生預設拒絕。如果符合陳述式中的所有條件，則請求會根據政策中的元素值，產生 [Allow](#) 或 [明確拒絕](#)。[效果](#) 政策未指定不符合條件時要做什麼，所以該狀況下的預設結果是預設拒絕。例如，您想要阻止來自南極的請求。您撰寫政策 A1，僅允許非來自南極的請求。下圖說明 Amazon SQS 政策。

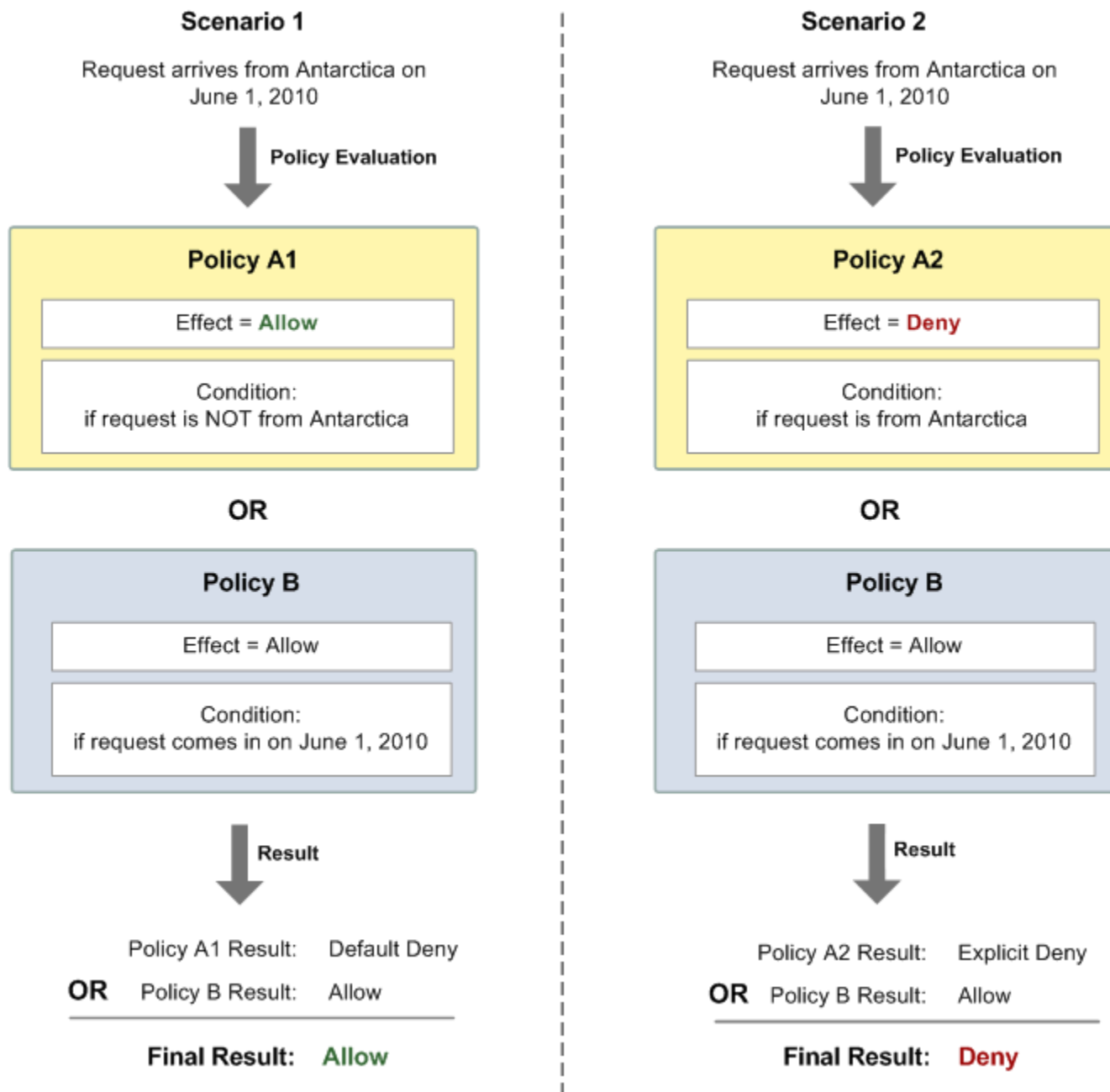


如果使用者從美國傳送請求，則符合條件 (請求非來自南極)，請求會產生允許。不過，如果使用者從南極傳送請求，便不符合條件，請求預設為預設拒絕。您可以撰寫政策 A2 以明確拒絕來自南極的請求，將結果變更為明確拒絕。下圖說明該政策。



如果使用者從南極傳送請求，便符合條件，請求會產生明確拒絕。

預設拒絕和明確拒絕之間的差異很重要，因為允許可以覆寫前者，但不能覆寫後者。例如，政策 B 允許在 2010 年 6 月 1 日到達的請求。下圖比較結合此政策與政策 A1 和政策 A2。



在案例 1，政策 A1 產生預設拒絕，而政策 B 產生允許，因為政策允許在 2010 年 6 月 1 日傳入的請求。政策 B 的允許會覆寫政策 A1 的預設拒絕，請求因此而被允許。

在案例 2，政策 B2 會產生明確拒絕，而政策 B 會產生允許。政策 A2 的明確拒絕會覆寫政策 B 的允許，請求因此而被拒絕。

Amazon SQS 自訂政策的限制

跨帳戶存取權

跨帳戶許可權不會套用至下列動作：

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTask](#)
- [ListQueues](#)
- [ListQueueTags](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

條件金鑰

目前，Amazon SQS 僅支援 [IAM 提供的條件金鑰](#) 的有限子集。如需詳細資訊，請參閱 [Amazon SQS API 許可：動作和資源參考](#)。

自訂 Amazon SQS 存取政策語言範例

以下是典型 Amazon SQS 存取政策的範例。

範例 1：提供許可給一個帳戶

以下範例 Amazon SQS 政策提供允許對 AWS 帳戶 444455556666 擁有的 queue2 傳送及接收的 AWS 帳戶 111122223333 許可權。

```
{
  "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"
  ]
}

```

範例 2：提供許可給一或多個帳戶

下列範例 Amazon SQS 政策提供一或多個 AWS 帳戶 在特定期間內存取您帳戶擁有的佇列。有需要使用 [SetQueueAttributes](#) 動作來撰寫此政策並上傳到 Amazon SQS，因為 [AddPermission](#) 動作在授與佇列存取權時不允許指定時間限制。

```

{
  "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"
      }
    }
  ]
}

```

```
}
```

範例 3：提供許可給來自 Amazon EC2 執行個體的請求

以下範例 Amazon SQS 政策提供許可給來自 Amazon SQS 執行個體的請求。此範例是根據「[範例 2：提供許可給一或多個帳戶](#)」範例：限制對 2009 年 6 月 30 日中午 12 點 (UTC) 之前的存取權，它限制對 IP 範圍 203.0.113.0/24 的存取權。有需要使用 [SetQueueAttributes](#) 動作來撰寫此政策並上傳到 Amazon SQS，因為 [AddPermission](#) 動作在授與佇列存取權時不允許指定 IP 地址限制。

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

範例 4：拒絕特定帳戶的存取權

下列範例 Amazon SQS 政策拒絕對佇列的特定 AWS 帳戶 存取。此範例以「[範例 1：提供許可給一個帳戶](#)」範例為基礎：拒絕存取指定的 AWS 帳戶。有需要使用 [SetQueueAttributes](#) 動作來撰寫此政策並上傳到 Amazon SQS，因為 [AddPermission](#) 動作在授與佇列存取權時不允許拒絕佇列的存取權 (僅允許授與佇列的存取權)。

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

範例 5：如果不是來自 VPC 端點，則拒絕存取

下列範例 Amazon SQS 政策會限制對 queue1 的存取：111122223333 只能從 VPC 端點 ID vpce-1a2b3c4d (使用 `aws:sourceVpce` 條件指定) 執行 [SendMessage](#) 和 [ReceiveMessage](#) 動作。如需詳細資訊，請參閱 [適用於 Amazon SQS 的 Amazon Virtual Private Cloud 端點](#)。

Note

- `aws:sourceVpce` 條件不需要 VPC 端點資源的 ARN，其只需要 VPC 端點 ID。
- 您可以透過在第二個陳述式中拒絕所有 Amazon SQS 動作 (`sqs:*`)，來修改下列範例以限制對特定 VPC 端點的所有動作。然而，這類政策陳述式規定必須透過此政策中定義的特定 VPC 端點來做出所有動作 (包含修改佇列許可所需的管理動作)，可能可避免使用者在未來修改佇列許可。

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

使用臨時安全憑證搭配 Amazon SQS

除了使用自己的安全登入資料建立使用者之外，IAM 也可讓您將臨時安全登入資料授予任何使用者，讓使用者存取您的 AWS 服務和資源。您可以管理擁有 AWS 帳戶的使用者。您也可以為沒有 AWS 帳戶（聯合身分使用者）的系統管理使用者。此外，您為存取 AWS 資源而建立的應用程式也可以被視為「使用者」。

您可以使用這些臨時安全憑證，對 Amazon SQS 提出請求。API 程式庫會使用這些憑證來運算出必要的簽章值，以驗證您的請求。若使用過期的憑證傳送請求，Amazon SQS 會拒絕該請求。

Note

您不能根據臨時憑證來設定政策。

先決條件

1. 使用 IAM 建立臨時安全憑證：
 - 安全權杖
 - 存取金鑰 ID
 - 私密存取金鑰
2. 準備您的登入字串搭配臨時存取金鑰 ID 和安全權杖。
3. 請使用臨時私密存取金鑰，而不是您自己的私密存取金鑰來簽署查詢 API 請求。

Note

當您提交已簽署的查詢 API 請求，請使用臨時存取金鑰 ID 而不是自己的存取金鑰 ID，並包含安全權杖。如需暫時安全登入資料之 IAM 支援的詳細資訊，請參閱《IAM 使用者指南》中的[授予 AWS 資源的暫時存取權](#)。

若要使用臨時安全憑證呼叫 Amazon SQS 查詢 API 動作

1. 使用 請求臨時安全字串 AWS Identity and Access Management。如需詳細資訊，請參閱《[IAM 使用者指南](#)》中的建立臨時安全登入資料以啟用 IAM 使用者的存取權。

IAM 會傳回安全權杖、存取金鑰 ID 和私密存取金鑰。

2. 請使用臨時存取金鑰 ID (而不是自己的存取金鑰 ID) 來準備查詢，並包含安全權杖。使用臨時私密存取金鑰 (而不是您自己的金鑰) 簽署您的請求。
3. 使用臨時存取金鑰 ID 和安全權杖來提交已簽署的查詢字串。

以下範例示範如何使用臨時安全憑證，來驗證 Amazon SQS 請求。**AUTHPARAMS** 的結構取決於 API 請求的簽署。如需詳細資訊，請參閱《Amazon Web Services 一般參考》中的[簽署 AWS API 請求](#)。

```
https://sqs.us-east-2.amazonaws.com/  
?Action=CreateQueue  
&DefaultVisibilityTimeout=40  
&QueueName=MyQueue  
&Attribute.1.Name=VisibilityTimeout  
&Attribute.1.Value=40  
&Expires=2020-12-18T22%3A52%3A43PST
```

```
&SecurityToken=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

以下範例使用臨時安全憑證，使用 SendMessageBatch 動作傳送兩則訊息。

```
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
```

具有最低權限政策的加密 Amazon SQS 佇列的存取管理

您可以使用 Amazon SQS，並透過使用與 [AWS Key Management Service \(KMS\)](#) 整合的伺服器端加密 (SSE) 來交換應用程式之間的敏感資料。透過整合 Amazon SQS 和 AWS KMS，您可以集中管理保護 Amazon SQS 的金鑰，以及保護其他 AWS 資源的金鑰。

多個 AWS 服務可以充當將事件傳送至 Amazon SQS 的事件來源。若要讓事件來源存取加密的 Amazon SQS 佇列，您需要使用 [客戶管理](#) AWS KMS 的金鑰來設定佇列。然後，使用 金鑰政策允許服務使用所需的 AWS KMS API 方法。此服務也需要驗證存取權才能讓佇列傳送事件。您可以使用 Amazon SQS 政策來達成此目的，這是一項資源型政策，可用來控制對 Amazon SQS 佇列及其資料的存取。

下列各節提供有關如何透過 Amazon SQS 政策和 AWS KMS 金鑰政策控制對加密 Amazon SQS 佇列的存取的資訊。本指南中的政策將會協助您獲得[最低權限](#)。

本指南也說明資源型政策如何使用 [aws:SourceArn](#)、[aws:SourceAccount](#) 和 [aws:PrincipalOrgID](#) 全域 IAM 條件內容索引鍵來解決[混淆代理人問題](#)。

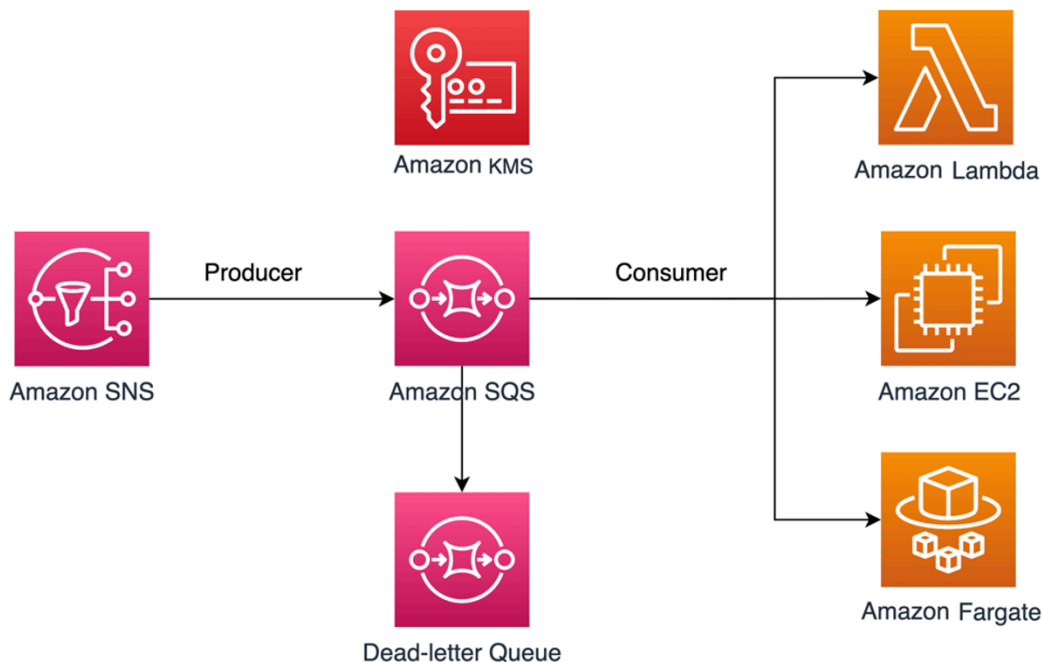
主題

- [概觀](#)

- [Amazon SQS 的最低權限金鑰政策](#)
- [適用於無效字母佇列的 Amazon SQS 政策陳述式](#)
- [防止跨服務混淆代理人問題](#)
- [使用 IAM Access Analyzer 檢閱跨帳戶存取權](#)

概觀

在本主題中，我們會引導您完成一個常見使用案例，說明如何建立金鑰政策和 Amazon SQS 佇列政策。這個使用案例如下圖所示。



在此範例中，訊息產生者是 [Amazon Simple Notification Service \(SNS\)](#) 主題，其設定為將訊息散發到加密的 Amazon SQS 佇列。訊息取用者是一種運算服務，例如 [AWS Lambda](#) 函數、[Amazon Elastic Compute Cloud \(EC2\)](#) 執行個體或 [AWS Fargate](#) 容器。然後，您的 Amazon SQS 佇列會設定為將失敗訊息傳送到[無效字母佇列 \(DLQ\)](#)。這很適合用來為應用程式或傳訊系統偵錯，因為 DLQ 可讓您隔離未取用的訊息以判斷無法成功處理的原因。在本主題定義的解決方案中，使用運算服務 (例如 Lambda 函數) 來處理存放在 Amazon SQS 佇列中的訊息。如果訊息取用者位於虛擬私有雲端 (VPC) 中，則本指南中包含的 [DenyReceivingIfNotThroughVPC](#) 政策陳述式可讓您將訊息接收限制為該特定 VPC。

Note

本指南僅包含政策聲明形式的必要 IAM 許可。若要建構政策，您需要將陳述式新增至 Amazon SQS 政策或 AWS KMS 金鑰政策。本指南不提供如何建立 Amazon SQS 佇列或 AWS KMS 金鑰的指示。如需有關如何建立這些資源的指示，請參閱[建立 Amazon SQS 佇列](#)和[建立金鑰](#)。

本指南中定義的 Amazon SQS 政策不支援將訊息直接重新推動到相同或不同的 Amazon SQS 佇列。

Amazon SQS 的最低權限金鑰政策

在本節中，我們會說明 AWS KMS 中您用來加密 Amazon SQS 佇列的客戶受管金鑰所需的最低權限許可。透過這些許可權，您可以在實作最低權限時限制只有預定實體的存取權。金鑰政策必須包含以下政策陳述式，詳述如下：

- [授予 AWS KMS 金鑰的管理員許可](#)
- [授予金鑰中繼資料的唯讀存取權](#)
- [將 Amazon SNS KMS 許可權授予 Amazon SNS，以將訊息發佈到佇列](#)
- [允許取用者解密佇列中的訊息](#)

授予 AWS KMS 金鑰的管理員許可

若要建立 AWS KMS 金鑰，您需要提供 AWS KMS 管理員許可給用來部署 AWS KMS 金鑰的 IAM 角色。這些管理員權定義於下列 AllowKeyAdminPermissions 政策陳述式中。當您將此陳述式新增至 AWS KMS 金鑰政策時，請務必以用於部署 AWS KMS 金鑰、管理 AWS KMS 金鑰或兩者的 IAM 角色 Amazon Resource Name (ARN) 取代 *<admin-role ARN>*。這可以是部署管道的 IAM 角色，也可以是 [AWS Organizations](#) 中的[組織管理員角色](#)。

```
{
  "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

在 AWS KMS 金鑰政策中，Resource 元素的值必須是 *，這表示「此 AWS KMS 金鑰」。星號 (*) 識別 AWS KMS 附加金鑰政策的金鑰。

授予金鑰中繼資料的唯讀存取權

若要授予其他 IAM 角色對金鑰中繼資料的唯讀存取權，請將 AllowReadAccessToKeyMetaData 陳述式新增至您的金鑰政策。例如，下列陳述式可讓您列出帳戶中的所有 AWS KMS 金鑰，以供稽核之用。此陳述式會授予 AWS 根使用者對金鑰中繼資料的唯讀存取權。因此，當帳戶中的任何 IAM 主體在身分型政策具有下列陳述式中列出的許可權時，都可以存取金鑰中繼資料：kms:Describe*、kms:Get* 和 kms:List*。確實將 *<account-ID>* 取代為您自己的資訊。

```

{
  "Sid": "AllowReadAccesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",

```

```

    "kms:List*"
  ],
  "Resource": "*"
}
```

將 Amazon SNS KMS 許可權授予 Amazon SNS，以將訊息發佈到佇列

若要允許 Amazon SNS 主題將訊息發佈到加密的 Amazon SQS 佇列，請將 AllowSNSToSendToSQS 政策聲明新增至您的金鑰政策。此陳述式授予 Amazon SNS 許可，以使用 AWS KMS 金鑰發佈至您的 Amazon SQS 佇列。確實將 *<account-ID>* 取代為您自己的資訊。

Note

陳述Condition式中的 只會限制對相同 AWS 帳戶中 Amazon SNS 服務的存取。

```

{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
}
```

允許取用者解密佇列中的訊息

下列 AllowConsumersToReceiveFromTheQueue 陳述式授予 Amazon SQS 訊息取用者將從加密之 Amazon SQS 佇列接收到的訊息解密的必要許可。附加政策陳述式時，請將 *<##### ARN>* 取代為訊息取用者的 IAM 執行期角色 ARN。

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

最低權限 Amazon SQS 政策

本節引導您針對本指南所涵蓋的使用案例 (例如 Amazon SNS 轉換至 Amazon SQS) 的最低權限 Amazon SQS 佇列政策。定義的策略旨在透過使用 Deny 和 Allow 陳述式的混合來防止意外存取。這些 Allow 陳述式會授予對預定實體的存取權。這些 Deny 陳述式可防止其他非預定實體存取 Amazon SQS 佇列，同時將政策條件內的預定實體排除在外。

Amazon SQS 政策包含下列陳述式，詳述如下：

- [限制 Amazon SQS 管理許可權](#)
- [限制來自指定之組織的 Amazon SQS 佇列動作](#)
- [將 Amazon SQS 許可授予取用者](#)
- [傳輸中強制加密](#)
- [限制訊息傳輸至特定 Amazon SNS 主題](#)
- [\(選用\) 限制接收特定 VPC 端點的訊息](#)

限制 Amazon SQS 管理許可權

以下 RestrictAdminQueueActions 政策陳述式將 Amazon SQS 管理許可權限制為僅允許用於部署佇列、管理佇列或兩者的一或多個 IAM 角色。請務必使用您自己的資訊取代 `<#####>`。指定用於部署 Amazon SQS 佇列的 IAM 角色的 ARN，以及任何應具有 Amazon SQS 管理許可權之管理員角色的 ARN。

```
{
```

```

{
  "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>"
      ]
    }
  }
}

```

限制來自指定之組織的 Amazon SQS 佇列動作

若要協助保護 Amazon SQS 資源免於外部存取 (由您的 [AWS 組織](#) 外的實體存取)，請使用下列陳述式。此陳述式將 Amazon SQS 佇列存取權限制於您在 Condition 中指定的組織。請務必將 **<SQS queue ARN>** 取代為用於部署 Amazon SQS 佇列之 IAM 角色的 ARN 取代；以及將 **<org-id>** 取代為您的組織 ID。

```

{
  "Sid": "DenyQueueActionsOutsideOrg",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes",
    "sqs:ReceiveMessage"
  ],

```

```

"Resource": "<SQS queue ARN>",
"Condition": {
  "StringNotEquals": {
    "aws:PrincipalOrgID": [
      "<org-id>"
    ]
  }
}
}

```

將 Amazon SQS 許可授予取用者

若要從 Amazon SQS 佇列接收訊息，您需要向訊息取用者提供必要的許可。下列政策陳述式授予您指定之取用者取用 Amazon SQS 佇列訊息的必要許可。將陳述式新增到 Amazon SQS 政策時，請務必將 **<#### IAM ##### ARN>** 取代為取用者所使用之 IAM 執行期角色的 ARN；並將 **<SQS ## ARN>** 取代為用於部署 Amazon SQS 佇列的 IAM 角色的 ARN。

```

{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": "<consumer's IAM execution role ARN>"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:GetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>"
}

```

若要防止其他實體從 Amazon SQS 佇列接收訊息，請將該 `DenyOtherConsumersFromReceiving` 陳述式新增至 Amazon SQS 佇列政策。此陳述式會將訊息消取用限制在您指定的取用者，即使其他取用者的身分權限會授予他們存取權，也不允許其他取用者擁有存取權。確實將 **<SQS ## ARN>** 和 **<##### ARN>** 取代為您自己的資訊。

```

{
  "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>"
  }
}
}
```

傳輸中強制加密

下列 DenyUnsecureTransport 政策陳述式強制取用者和生產者使用安全通道 (TLS 連線) 從 Amazon SQS 佇列傳送和接收訊息。請務必將 **<SQS ## ARN>** 取代為用於部署 Amazon SQS 佇列的 IAM 角色的 ARN。

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}
```

限制訊息傳輸至特定 Amazon SNS 主題

以下 AllowSNSToSendToTheQueue 政策陳述式允許指定 Amazon SNS 主題將訊息傳送至 Amazon SQS 佇列。請務必將 **<SQS ## ARN>** 取代為用於部署 Amazon SQS 佇列的 IAM 角色的 ARN；並將 **<SNS ## ARN>** 取代為 Amazon SNS 主題 ARN。

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

下列 DenyAllProducersExceptSNSFromSending 政策陳述式可防止其他生產者傳送訊息至佇列。將 **<SQS ## ARN>** 和 **<SNS ## ARN>** 取代為您自己的資訊。

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```


(選用) 限制接收特定 VPC 端點的訊息

若要限制只接收特定 [VPC 端點](#) 的訊息，請將以下政策陳述式新增至 Amazon SQS 佇列政策。除非訊息來自所需的 VPC 端點，否則此陳述式可防止訊息取用者從佇列接收訊息。將 **<SQS ## ARN>** 取代為用於部署 Amazon SQS 佇列之 IAM 角色的 ARN，以及將 **<vpce_id>** 取代為 VPC 端點的 ID。

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

適用於無效字母佇列的 Amazon SQS 政策陳述式

將以下政策陳述式 (以其陳述式 ID 識別) 新增至您的 DLQ 存取政策：

- RestrictAdminQueueActions
- DenyQueueActionsOutsideOrg
- AllowConsumersToReceiveFromTheQueue
- DenyOtherConsumersFromReceiving
- DenyUnsecureTransport

除了將上述政策陳述式新增至 DLQ 存取政策之外，您也應該新增陳述式以限制傳輸至 Amazon SQS 佇列的訊息，如下節所述。

限制 Amazon SQS 佇列的訊息傳輸

若要限制只存取來自相同帳戶的 Amazon SQS 佇列，請將以下 DenyAnyProducersExceptSQS 政策陳述式新增至 DLQ 佇列政策。此陳述式不會將訊息傳輸限制到特定佇列，因為您需要在建立主要佇列之前部署 DLQ，因此在建立 DLQ 時不會知道 Amazon SQS ARN。如果您需要只限制對一個

Amazon SQS 佇列的存取，請在知道時使用 Amazon SQS 來源佇列的 ARN 來修改 Condition 中的 `aws:SourceArn`。

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

Important

本指南中定義的 Amazon SQS 佇列政策不會將 `sqs:PurgeQueue` 動作限制為一或多個特定 IAM 角色。`sqs:PurgeQueue` 動作可讓您刪除 Amazon SQS 佇列中的所有訊息。您也可以使用此動作在不取代 Amazon SQS 佇列的情況下變更訊息格式。為應用程式偵錯時，您可以清除 Amazon SQS 佇列移除可能的錯誤訊息。測試應用程式時，您可以透過 Amazon SQS 佇列驅動大量訊息，然後在進入生產之前清除佇列以重新開始。不將此動作限制為特定角色的原因是，在部署 Amazon SQS 佇列時可能不知道此角色。您必須將此許可權新增至角色的身分型政策，才能清除佇列。

防止跨服務混淆代理人問題

[混淆代理人問題](#)屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多權限的實體執行該動作。為了防止這種情況，如果您提供第三方（稱為跨帳戶）或其他 AWS 服務（稱為跨服務）存取您帳戶中的資源，AWS 會提供可協助您保護帳戶的工具。本節中的政策陳述式可幫助您預防跨服務混淆代理人問題。

在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為協助防範此問題，本文中定義的資源型政策會使用 [aws:SourceArn](#)、[aws:SourceAccount](#) 和 [aws:PrincipalOrgID](#) 全

域 IAM 條件內容索引鍵。這會限制服務對 AWS Organizations 中特定資源、特定帳戶或特定組織的許可。

使用 IAM Access Analyzer 檢閱跨帳戶存取權

您可以使用 [AWS IAM Access Analyzer](#) 檢閱 Amazon SQS 佇列政策和 AWS KMS 金鑰政策，並在 Amazon SQS 佇列或 AWS KMS 金鑰授予外部實體存取權時提醒您。IAM Access Analyzer 可協助識別組織與帳戶中，與信任區域外實體共用的[資源](#)。此信任區域可以是您在啟用 IAM Access Analyzer 時指定的 AWS Organizations 內的 AWS 帳戶或組織。

IAM Access Analyzer 使用邏輯式推理來分析 AWS 環境中的資源型政策，藉此識別與外部主體共用的資源。針對信任區域外共用資源的每一個執行個體，Access Analyzer 都會產生一份調查結果。[調查結果](#)包括存取權及其被授與存取的外部主體的資訊。請檢閱調查結果，判斷此為有意為之且安全的存取權，還是非預期且有安全風險的存取。對於任何非預定的存取，請檢閱受影響的政策並加以修正。如需 IAM Access Analyzer AWS 如何識別意外存取 AWS 資源的詳細資訊，請參閱此[部落格文章](#)。

如需 AWS IAM Access Analyzer 的詳細資訊，請參閱 [AWS IAM Access Analyzer 文件](#)。

Amazon SQS API 許可：動作和資源參考

當您設定 [存取控制](#) 並撰寫可連接到 IAM 身分的許可政策時，可以使用以下表格做為參考。包含每個 Amazon Simple Queue Service 動作、您可以授予執行動作許可的對應動作，以及您可以授予許可 AWS 的資源。

在政策的 Action 欄位中指定動作，然後在政策的 Resource 欄位中指定資源值。若要指定動作，請使用 sqs: 字首後接 動作名稱 (例如，sqs:CreateQueue)。

目前，Amazon SQS 支援 [IAM 中提供的全域條件內容金鑰](#)。

Amazon Simple Queue Service API 和動作所需的許可

[AddPermission](#)

動作：sqs:AddPermission

資源：arn:aws:sqs:*region*:*account_id*:*queue_name*

[ChangeMessageVisibility](#)

動作：sqs:ChangeMessageVisibility

資源：arn:aws:sqs:*region*:*account_id*:*queue_name*

[ChangeMessageVisibilityBatch](#)

動作 : sqs:ChangeMessageVisibilityBatch

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[CreateQueue](#)

動作 : sqs:CreateQueue

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[DeleteMessage](#)

動作 : sqs>DeleteMessage

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[DeleteMessageBatch](#)

動作 : sqs>DeleteMessageBatch

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[DeleteQueue](#)

動作 : sqs>DeleteQueue

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[GetQueueAttributes](#)

動作 : sqs:GetQueueAttributes

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[GetQueueUrl](#)

動作 : sqs:GetQueueUrl

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[ListDeadLetterSourceQueues](#)

動作 : sqs:ListDeadLetterSourceQueues

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[ListQueues](#)

動作 : sqs:ListQueues

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[ListQueueTags](#)

動作 : sqs:ListQueueTags

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[PurgeQueue](#)

動作 : sqs:PurgeQueue

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[ReceiveMessage](#)

動作 : sqs:ReceiveMessage

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[RemovePermission](#)

動作 : sqs:RemovePermission

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[SendMessage](#) 和 [SendMessageBatch](#)

動作 : sqs:SendMessage

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[SetQueueAttributes](#)

動作 : sqs:SetQueueAttributes

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[TagQueue](#)

動作 : sqs:TagQueue

資源 : arn:aws:sqs:*region*:*account_id*:*queue_name*

[UntagQueue](#)

動作 : sqs:UntagQueue

資源：`arn:aws:sqs:region:account_id:queue_name`

在 Amazon SQS 中記錄和監控

Amazon Simple Queue Service 已與整合 [AWS CloudTrail](#)，此服務提供使用者、角色或所採取動作的記錄 AWS 服務。CloudTrail 會將 Amazon SQS 的所有 API 呼叫擷取為事件。擷取的呼叫包括從 Amazon SQS 主控台呼叫，以及對 Amazon SQS API 操作的程式碼呼叫。使用 CloudTrail 收集的資訊，您可以判斷對 Amazon SQS 提出的請求、提出請求的 IP 地址、提出請求的時間，以及其他詳細資訊。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根使用者還是使用者憑證提出。
- 請求是否代表 IAM Identity Center 使用者提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務服務提出。

當您建立帳戶 AWS 帳戶時 CloudTrail 會在中處於作用中狀態，而且您會自動存取 CloudTrail 事件歷史記錄。CloudTrail 事件歷史記錄為 AWS 區域中過去 90 天記錄的管理事件，提供可檢視、可搜尋、可下載且不可變的記錄。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」中的 [使用 CloudTrail 事件歷史記錄](#)。檢視事件歷史記錄不會產生 CloudTrail 費用。

如需 AWS 帳戶過去 90 天內持續記錄的事件，請建立線索或 [CloudTrail Lake](#) 事件資料存放區。

Amazon CloudWatch 警示

在您指定的期間內監控單一指標，並根據指標值，在數個期間內相對於定義的閾值採取一或多個動作。例如，您可以設定 CloudWatch 警示來傳送通知至 Amazon SNS 主題，或觸發動作來傳送訊息至 Amazon SQS 佇列。CloudWatch 警示不會僅因為處於特定狀態而執行動作；狀態必須變更，並在定義的期間數內保持在該狀態。

如需詳細資訊，請參閱 [建立 Amazon SQS 指標的 CloudWatch 警示](#) 和 [使用 Amazon CloudWatch 建立無效字母佇列的警示](#)。

Amazon CloudWatch Logs

透過設定您的應用程式或 Lambda 函數，以處理訊息將日誌傳送至 CloudWatch Logs，來監控、存放和存取與 Amazon SQS 相關的日誌檔案。您可以使用這些日誌來分析訊息處理、偵錯問題，以及監控 Amazon SQS 工作流程的效能。

如需詳細資訊，請參閱[使用 記錄 Amazon Simple Queue Service API 呼叫 AWS CloudTrail](#)。

Amazon CloudWatch Events

使用 Amazon CloudWatch Events 偵測 AWS 環境中的變更或特定事件，並將其路由至 Amazon SQS 佇列。這可讓您擷取事件資料、觸發工作流程或存放事件以供稍後處理。

如需詳細資訊，請參閱本指南[使用 Amazon EventBridge 從 AWS 服務到 Amazon SQS 的通知 自動化 Amazon SQS](#) 中的 和 [EventBridge 是 Amazon EventBridge 使用者指南中的 Amazon CloudWatch Events 的演變](#)。 EventBridge

AWS CloudTrail 日誌

CloudTrail 會擷取使用者、角色或 在 Amazon SQS 上執行動作的詳細記錄 AWS 服務。這些日誌可讓您追蹤 API 呼叫，例如 [SendMessage](#)、[ReceiveMessage](#) 或 [DeleteQueue](#)，並提供金鑰詳細資訊，例如提出請求的人員、發生的時間，以及原始 IP 地址。

如需詳細資訊，請參閱[使用 記錄 Amazon Simple Queue Service API 呼叫 AWS CloudTrail](#)。

AWS Trusted Advisor

Trusted Advisor 使用從服務 AWS 客戶開發的最佳實務，協助最佳化 Amazon SQS 用量。它會檢閱您的 Amazon SQS 佇列，並提供可行的建議，以增強安全性、改善訊息處理可靠性，並降低成本。例如，它可能會建議啟用無效字母佇列或改善佇列存取政策，以確保安全操作。

如需詳細資訊，請參閱《支援 使用者指南》中的 [AWS Trusted Advisor](#)。

CloudTrail 追蹤

線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。使用 建立的所有線索 AWS Management Console 都是多區域。您可以使用 AWS CLI 建立單一或多區域追蹤。建議您建立多區域追蹤，因為您擷取 AWS 區域 帳戶中所有的活動。如果您建立單一區域追蹤，您只能檢視追蹤 AWS 區域中記錄的事件。如需追蹤的詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的[為您的 AWS 帳戶建立追蹤](#)和[為組織建立追蹤](#)。

您可以透過建立追蹤，免費將持續管理事件的一個複本從 CloudTrail 傳遞至您的 Amazon S3 儲存貯體，但這樣做會產生 Amazon S3 儲存費用。如需 CloudTrail 定價的詳細資訊，請參閱 [AWS CloudTrail 定價](#)。如需 Amazon S3 定價的相關資訊，請參閱 [Amazon S3 定價](#)。

CloudTrail Lake 事件資料存放區

CloudTrail Lake 讓您能夠對事件執行 SQL 型查詢。CloudTrail Lake 會將分列式 JSON 格式的現有事件轉換為 [Apache ORC](#) 格式。ORC 是一種單欄式儲存格式，針對快速擷取資料進行了最佳化。

系統會將事件彙總到事件資料存放區中，事件資料存放區是事件的不可變集合，其依據為您透過套用[進階事件選取器](#)選取的條件。套用於事件資料存放區的選取器控制哪些事件持續存在並可供您查詢。如需 CloudTrail Lake 的詳細資訊，請參閱AWS CloudTrail 《使用者指南》中的[使用 AWS CloudTrail Lake](#)。

CloudTrail Lake 事件資料存放區和查詢會產生費用。建立事件資料存放區時，您可以選擇要用於事件資料存放區的[定價選項](#)。此定價選項將決定擷取和儲存事件的成本，以及事件資料存放區的預設和最長保留期。如需 CloudTrail 定價的詳細資訊，請參閱 [AWS CloudTrail 定價](#)。

使用 記錄 Amazon Simple Queue Service API 呼叫 AWS CloudTrail

CloudTrail 可讓您使用兩種事件類型記錄和監控 Amazon SQS 操作：資料事件和管理事件。這可讓您輕鬆地追蹤和稽核帳戶中的 Amazon SQS 活動。

CloudTrail 中的 Amazon SQS 資料事件

[資料事件](#)提供有關在資源上執行或在資源中執行的資源操作的資訊（例如，傳送訊息至 Amazon SQS 物件）。這些也稱為資料平面操作。資料事件通常是大量資料的活動。根據預設，CloudTrail 不會記錄資料事件。CloudTrail 事件歷史記錄不會記錄資料事件。

資料事件需支付額外的費用。如需 CloudTrail 定價的詳細資訊，請參閱 [AWS CloudTrail 定價](#)。

您可以使用 CloudTrail 主控台或 CloudTrail API 操作 AWS CLI來記錄 Amazon SQS 資源類型的資料事件。如需如何記錄資料事件的詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的[使用 AWS Management Console記錄資料事件](#)和[使用 AWS Command Line Interface記錄資料事件](#)。

若要使用 CloudTrail 記錄 Amazon SQS 資料事件，您必須使用進階事件選擇器來設定您要記錄的特定 Amazon SQS 資源或動作。包含[AWS::SQS::Queue](#)擷取佇列相關動作的資源類型。您可以使用 eventName（例如[SendMessage](#)事件）等篩選條件，進一步精簡您的記錄偏好設定。如需詳細資訊，請參閱 CloudTrail API 參考中的 [AdvancedEventSelector](#)。

資料事件類型 (主控台)	resources.type 值	記錄到 CloudTrail 的資料 API
Amazon SQS 佇列	AWS::SQS::Queue	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch

資料事件類型 (主控台)	resources.type 值	記錄到 CloudTrail 的資料 API
		• GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

使用進階事件選擇器來篩選欄位，並僅記錄重要事件。如需這些欄位的詳細資訊，請參閱AWS CloudTrail API 參考中的[AdvancedFieldSelector](#)。

CloudTrail 中的 Amazon SQS 管理事件

[管理事件](#)提供在 資源上執行的管理操作的相關資訊 AWS 帳戶。這些也稱為控制平面操作。根據預設，CloudTrail 記錄管理事件。

Amazon SQS 會將下列控制平面操作記錄到 CloudTrail 做為管理事件。

- [AddPermission](#)
- [CancelMessageMoveTask](#)
- [CreateQueue](#)
- [DeleteQueue](#)
- [ListMessageMoveTasks](#)
- [PurgeQueue](#)
- [RemovePermission](#)
- [SetQueueAttributes](#)
- [StartMessageMoveTask](#)
- [TagQueue](#)
- [UntagQueue](#)

Amazon SQS 事件範例

一個事件代表任何來源提出的單一請求，並包含請求 API 操作的相關資訊、操作的日期和時間、請求參數等。CloudTrail 日誌檔案不是公有 API 呼叫的已排序堆疊追蹤，因此事件不會以任何特定順序顯示。

以下範例顯示的 CloudTrail 事件會示範 SendMessage 操作。

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EXAMPLE_PRINCIPAL_ID",
    "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/SessionName",
    "accountId": "123456789012",
    "accessKeyId": "ACCESS_KEY_ID",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed",
        "accountId": "123456789012",
        "userName": "RoleToBeAssumed"
      },
      "attributes": {
        "creationDate": "2023-11-07T22:13:06Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2023-11-07T23:59:11Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessage",
  "awsRegion": "ap-southeast-4",
  "sourceIPAddress": "10.0.118.80",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue",
    "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageDeduplicationId": "MsgDedupIdSdk1ae1958f2-bbe8-4442-83e7-4916e3b035aa",
    "messageGroupId": "MsgGroupIdSdk16"
```

```
},
"responseElements": {
  "mD50fMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
  "mD50fMessageSystemAttributes": "f88f0587f951b7f5551f18ae699c3a9d",
  "messageId": "93bb6e2d-1090-416c-81b0-31eb1faa8cd8",
  "sequenceNumber": "18881790870905840128"
},
"requestID": "c4584600-fe8a-5aa3-a5ba-1bc42f055fae",
"eventID": "98c735d8-70e0-4644-9432-b6ced4d791b1",
"readOnly": false,
"resources": [
  {
    "accountId": "123456789012",
    "type": "AWS::SQS::Queue",
    "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue"
  }
],
"eventType": "AwsApiCall",
"managementEvent": false,
"recipientAccountId": "123456789012",
"eventCategory": "Data",
"tlsDetails": {
  "tlsVersion": "TLSv1.2",
  "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
  "clientProvidedHostHeader": "sqs.ap-southeast-4.amazonaws.com"
}
```

Note

ListQueues 操作是唯一的案例，因為它不會對特定資源採取行動。因此，ARN 欄位不包含佇列名稱，而是改用萬用字元 (*)。

如需有關 CloudTrail 記錄內容的資訊，請參閱《AWS CloudTrail 使用者指南》中的 [CloudTrail record contents](#)。

使用 CloudWatch 監控 Amazon SQS 佇列

Amazon SQS 與 Amazon CloudWatch 會整合在一起，以便您可使用 CloudWatch 檢視及分析適用於 Amazon SQS 佇列的指標。您可以使用 [AWS CLI](#) 或使用 [CloudWatch API](#)，從 [Amazon SQS 主控台](#)、[CloudWatch 主控台](#) 檢視和分析佇列的指標。您也可以為 Amazon SQS 指標 [設定 CloudWatch 警示](#)。

自動收集 Amazon SQS 佇列的 CloudWatch 指標，並以一分鐘間隔將該指標推送到 CloudWatch。這些指標會針對所有符合 CloudWatch 作用中指導方針的佇列進行收集。如果佇列包含任何訊息，或有任何動作存取佇列，CloudWatch 會將佇列視為作用中長達六個小時。

當 Amazon SQS 佇列處於非作用中狀態超過六小時，會將 Amazon SQS 服務視為在休眠狀態，並停止向 CloudWatch 服務傳送指標。在 Amazon SQS 佇列處於非作用中狀態的期間內，無法在 Amazon SQS 的 CloudWatch 指標中將遺失的資料或代表零的資料視覺化。

Note

- 當未授權針對佇列呼叫 API 的使用者，且請求失敗時，可以啟用 Amazon SQS 佇列。
- 開啟佇列的頁面時，Amazon SQS 主控台會執行 [GetQueueAttributes](#) API 呼叫。GetQueueAttributes API 請求會啟用佇列。
- 從非作用中狀態啟動佇列時，CloudWatch 指標會延遲最多 15 分鐘。
- Amazon SQS 指標是免費的並在 CloudWatch 中回報。這是 Amazon SQS 服務的一部分。
- 標準和 FIFO 佇列均支援 CloudWatch 指標。

存取 Amazon SQS 的 CloudWatch 指標

Amazon SQS 與 Amazon CloudWatch 會整合在一起，以便您可使用 CloudWatch 檢視及分析適用於 Amazon SQS 佇列的指標。您可以使用 [AWS CLI](#) 或使用 [CloudWatch API](#)，從 [Amazon SQS 主控台](#)、[CloudWatch 主控台](#) 檢視和分析佇列的指標。您也可以為 Amazon SQS 指標[設定 CloudWatch 警示](#)。

使用 Amazon SQS 主控台

使用 Amazon SQS 主控台存取和分析最多 10 個 Amazon SQS 佇列的指標。

1. 請登入 [Amazon SQS 主控台](#)。
2. 在佇列清單中，選擇 (勾選) 您想存取其指標之佇列的方塊。您最多可以顯示 10 個佇列的指標。
3. 選擇 Monitoring (監控) 索引標籤。

SQS metrics (SQS 指標) 部分會顯示各種圖形。

4. 若要了解特定圖形代表的意義，請將滑鼠的游標移至所需圖形旁的



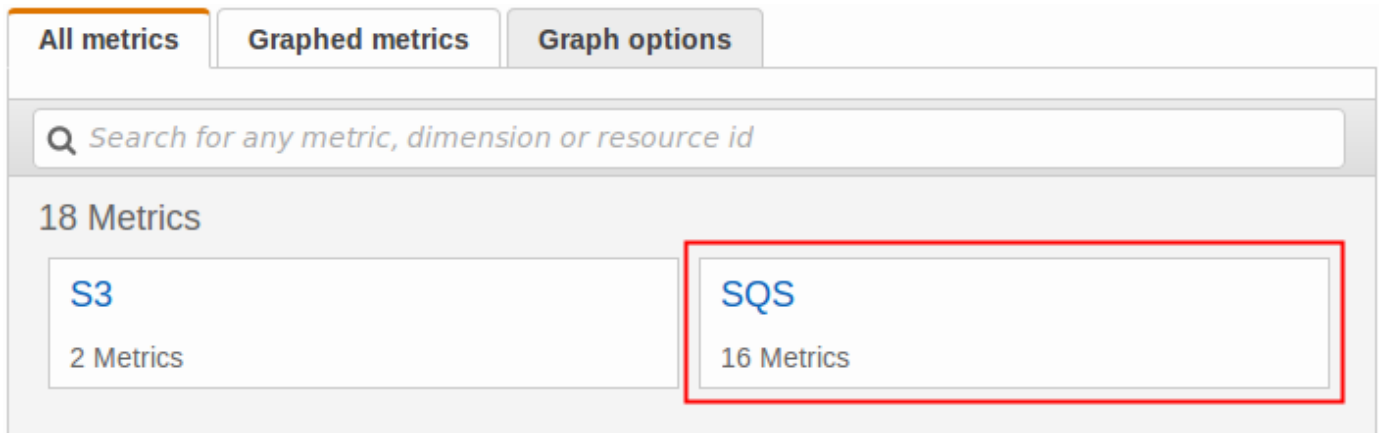
上，或請參閱 [適用於 Amazon SQS 的可用 CloudWatch 指標](#)。

- 若要同時變更所有圖表的時間範圍，請在 Time Range (時間範圍) 選擇所需的時間範圍 (例如 Last Hour (前一小時))。
- 若要查看個別圖表的其他統計資料，請選擇圖表。
- 在監控詳細資訊對話方塊中，選取統計資料 (例如總和)。如需支援的統計資料清單，請參閱[適用於 Amazon SQS 的可用 CloudWatch 指標](#)。
- 若要變更個別圖形顯示的時間範圍和時間間隔 (例如，若要顯示過去 24 小時的時間範圍而不是前 5 分鐘，或是顯示每小時的時間間隔而不是每 5 分鐘)，請在圖形對話方塊仍顯示時，在 Time Range (時間範圍) 中選擇所需的時間範圍 (例如 Last 24 Hours (過去 24 小時))。對於 Period (期間)，選擇特定時間範圍內所需的時間期間 (例如 1 Hour (1 小時))。當您完成查看圖表，請選擇 Close (關閉)。
- (選用) 如果要使用其他 CloudWatch 功能，請在監控索引標籤中，選擇檢視所有指標，然後按照[使用 Amazon CloudWatch 主控台](#) 程序的指示進行。

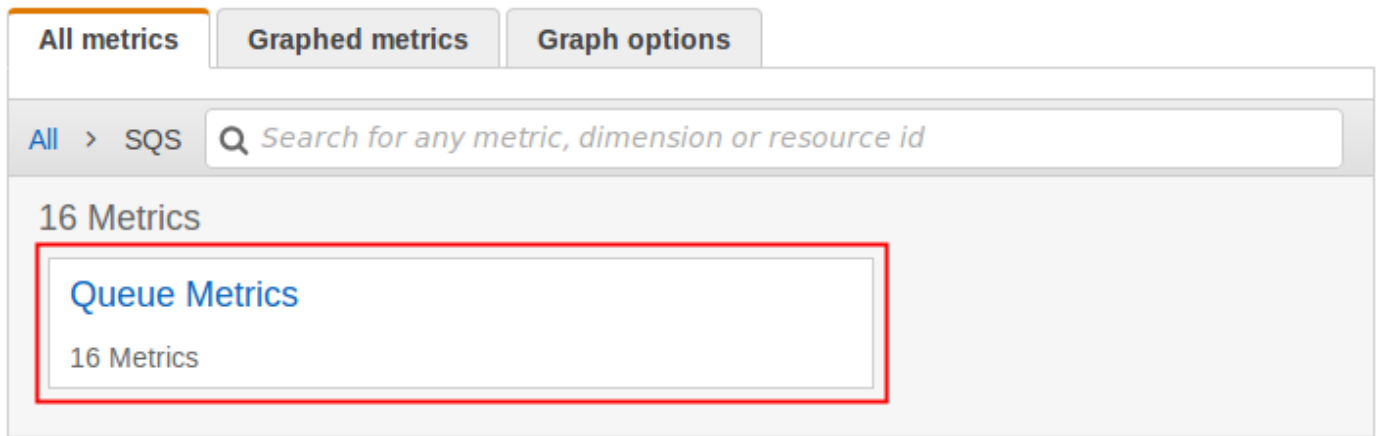
使用 Amazon CloudWatch 主控台

使用 CloudWatch 主控台來存取和分析 Amazon SQS 指標。

- 登入 [CloudWatch 主控台](#)。
- 在導覽面板上，選擇 Metrics (指標)。
- 選取 SQS 指標命名空間。

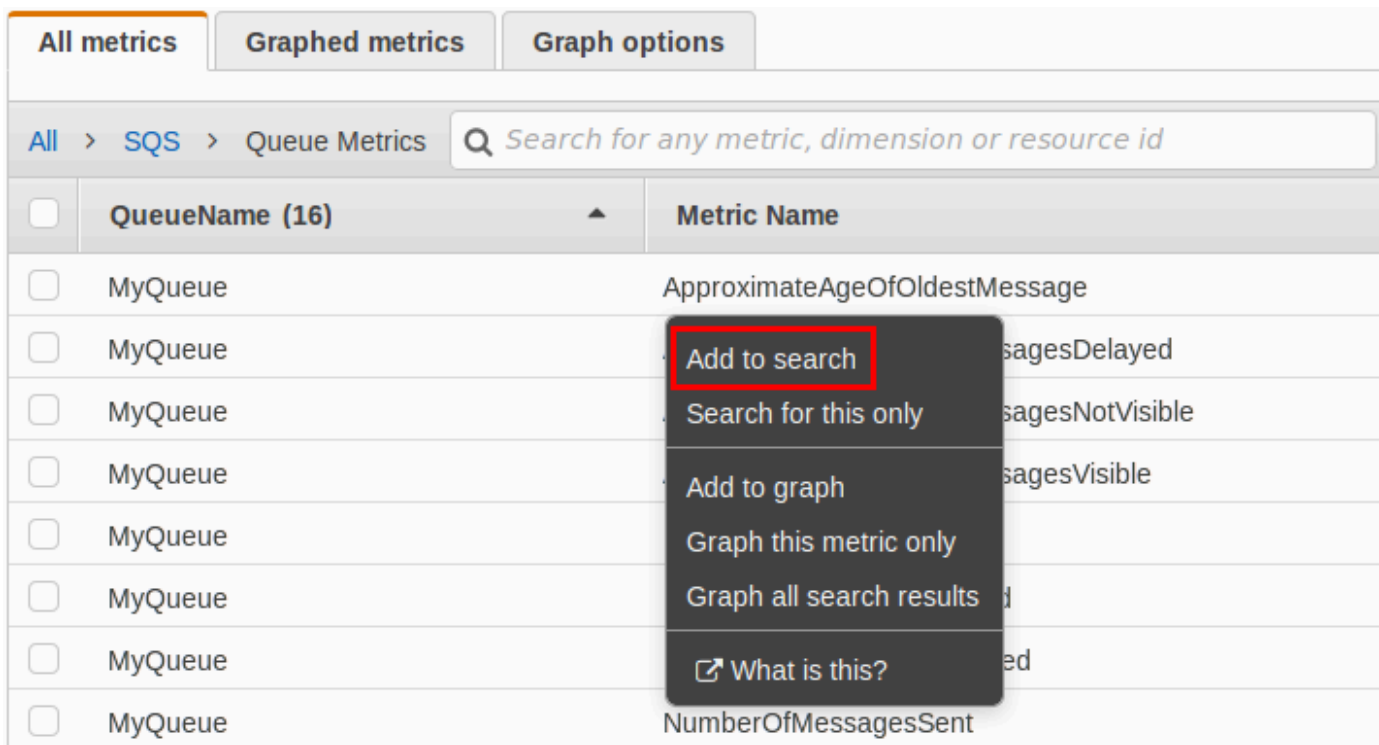


- 選取 Queue Metrics (佇列指標) 指標維度。



5. 您現在可以檢查 Amazon SQS 指標：

- 若要排序指標，請使用直欄標題。
- 若要將指標圖形化，請選取指標旁的核取方塊。
- 若要依指標篩選，請選擇指標名稱，然後選擇 Add to search (新增至搜尋)。



如需詳細資訊和其他選項，請參閱《Amazon CloudWatch 使用者指南》中的[圖形指標](#)和[使用 Amazon CloudWatch 儀表板](#)。

使用 AWS Command Line Interface

若要使用 存取 Amazon SQS 指標 AWS CLI，請執行 [get-metric-statistics](#) 命令。

如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的[取得指標的統計資料](#)。

使用 CloudWatch API

若要使用 CloudWatch API 存取 Amazon SQS 指標，請使用 [GetMetricStatistics](#) 動作。

如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的[取得指標的統計資料](#)。

建立 Amazon SQS 指標的 CloudWatch 警示

CloudWatch 可讓您在指標達到指定的閾值時觸發警示。例如，您可以建立警示的 `NumberOfMessagesSent` 指標。例如，如果超過 100 個訊息在 1 小時內傳送到 `MyQueue` 佇列，會傳送一封電子郵件通知。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的[建立 Amazon CloudWatch 警示](#)。

1. 登入 AWS Management Console，並在 <https://console.aws.amazon.com/cloudwatch/> 開啟 CloudWatch 主控台。
2. 選擇 Alarms (警示)，然後選擇 Create Alarm (建立警示)。
3. 在 Create Alarm (建立警示) 對話方塊中的 Select Metric (選取指標) 部分，選擇 Browse Metrics (瀏覽指標)、SQS (SQS)。
4. 對於 SQS > Queue Metrics (SQS > 佇列指標)，選擇 QueueName (佇列名稱) 和 Metric Name (指標名稱) 以設定警示，然後選擇 Next (下一步)。如需可用指標的清單，請參閱 [適用於 Amazon SQS 的可用 CloudWatch 指標](#)。

在下列範例中，選擇適用於警示的 `NumberOfMessagesSent` 佇列的 `MyQueue` 指標。送出訊息如超過 100 則，警示就會觸發。

5. 在 Create Alarm (建立警示) 對話方塊中 Define Alarm (定義警示) 部分，請執行下列動作：
 - a. 在 Alarm Threshold (警示閾值) 下，輸入警示的 Name (名稱) 和 Description (說明)。
 - b. 將 is (是) 設定為 > 100。
 - c. 設定 for (為) 至 1 out of 1 datapoints (1 傳出 1 資料點)。
 - d. 在 Alarm preview (警示預覽) 中，設定 Period (期間) 為 1 Hour (1 小時)。
 - e. 將 Statistic (統計資料) 設定為 Standard (標準)、Sum (總和)。
 - f. 在 Actions (動作) 下的 Whenever this alarm (每當此警示)，選擇 State is ALARM (狀態為警示)。

如果您希望 CloudWatch 在觸發警示時就會傳送通知，選取現有 Amazon SNS 主題或選擇新的清單，然後輸入電子郵件地址以逗號分隔。

 Note

如果您建立新的 Amazon SNS 主題，電子郵件地址必須先經過驗證才會接收通知。如果此警示狀態在驗證電子郵件地址之前變更，就不會收到通知。

6. 選擇建立警示。

警示已建立。

適用於 Amazon SQS 的可用 CloudWatch 指標

Amazon SQS 會傳送下列指標至 CloudWatch。

 Note

對於某些指標，結果是近似值，因為 Amazon SQS 的分散式架構。在大多數情況下，計數應接近佇列中的實際訊息數目。

Amazon SQS 指標

Amazon SQS 會自動將操作指標發佈至 AWS/SQS 命名空間下的 [Amazon CloudWatch](#)。這些指標可協助您監控佇列運作狀態和效能。由於 SQS 的分散式本質，許多值都是近似值，但對於大多數的操作決策來說都足夠準確。

 Note

- 只有在佇列作用中時，所有指標才會發出非負值。
- 在至少傳送一則訊息之前，不會發出某些指標（例如 `SentMessageSize`）。

指標	描述	個單位	報告行為	重要備註
ApproximateAgeOfOldestMessage	佇列中最舊未處理訊息的存留期。	秒鐘	如果佇列包含至少一個作用中訊息，則回報。	- 對於標準佇列，如果收到訊息三次以上但未刪除，SQS 會將訊息移至佇列的背面。指標接著會反映未超過接收閾值的下一個訊息的存留期。即使有再驅動政策，也會發生此重新排序。 - 毒害訊息（重複接收但從未刪除的訊息）會從此指標中排除，直到成功處理為止。 - 當訊息在超過 <code>maxReceiveCount</code> 之後移至 DLQ，存留期會重設。在這種情況下，DLQ 的指標會反映訊息移動的時間，而不是最初傳送的時間。 - FIFO 佇列不會重新排序訊息以保留順序。失敗的訊息會封鎖其訊息群組，直到刪除或過期為止。如果已設定 DLQ，則訊息會在達到接收閾值後傳送至該處。
ApproximateNumberOfGroupsWithInflightMessages	僅適用於 FIFO。具有一或多個傳輸中訊息的訊息群組數量。	計數	如果 FIFO 佇列處於作用中狀態，則回報。	- 消費者從佇列收到訊息後，但尚未刪除或過期，訊息會被視為進行中。 - 此指標可協助您疑難排解和最佳化 FIFO 佇列輸送量。高值通常表示強並行。 - 如果佇列有大型待處理項目，且此值仍低，請考慮擴展取用者或增加作用中訊息群組的數量。

指標	描述	個單位	報告行為	重要備註
				如需輸送量和傳輸中限制，請參閱 Amazon SQS 配額。
ApproximateNumberOfMessagesDelayed	佇列中延遲且無法立即擷取的訊息數量。	計數	如果佇列中存在延遲訊息，則回報。	- 適用於設定預設延遲的佇列，以及使用 DelaySeconds 參數傳送的個別訊息。 - 延遲訊息在延遲期間到期之前會保持隱藏狀態，這可能會影響感知的佇列待處理項目或輸送量。
ApproximateNumberOfMessagesNotVisible	已接收但尚未刪除或過期的傳輸中訊息數量。	計數	如果傳輸中的訊息存在，則回報。	- 透過 ReceiveMessage API 傳送給消費者後，訊息會進入傳輸中狀態。 - 在可見性逾時時段期間，其他取用者會暫時隱藏這些訊息。 - 使用此指標追蹤訊息處理延遲或停滯的消費者。
ApproximateNumberOfMessagesVisible	目前可供擷取和處理的訊息數量。	計數	如果佇列處於作用中狀態，則回報。	- 反映佇列中目前的處理待處理項目。 - 可以累積多少訊息沒有硬性限制，但它們受限於佇列設定的 保留期間。 - 持續較高的值可能表示佈建不足的消費者或處理邏輯停滯。

指標	描述	個單位	報告行為	重要備註
NumberOfEmptyReceives 1	未傳回訊息的 ReceiveMessage API 呼叫數量。	計數	在接收操作期間報告。	- 此指標可協助識別輪詢行為中的效率低落或未充分利用的取用者執行個體。 - 當佇列為空、取用者使用短輪詢，或訊息處理速度超過產生速度時，可能會發生高值。 - 這不是佇列狀態的精確指標。它反映服務端行為，可能包括重試。
NumberOfDuplicatedSentMessages	僅適用於 FIFO。已刪除但未新增至佇列的已傳送訊息數量。	計數	偵測到重複 MessageId 值或內容時回報。	- SQS 會根據 MessageId 或內容型雜湊（如果啟用）刪除重複訊息。 - 高值可能表示生產者在 5 分鐘的重複資料刪除時段內重複傳送相同的訊息。 - 使用此指標對備援生產者邏輯進行故障診斷，或確認重複資料刪除如預期般運作。

指標	描述	個單位	報告行為	重要備註
NumberOfMessagesDeleted 1	成功從佇列刪除的訊息數量。	計數	使用有效的接收控點報告每個刪除請求。	- 此指標會計算所有成功的刪除操作，即使刪除相同的訊息超過一次。 - higher-than-expected值的常見原因包括： - 在可見性逾時過期且再次收到訊息後，使用不同的接收控點多次刪除相同訊息。 - 使用相同的接收控點重複刪除，這仍會傳回成功狀態並增加指標。 - 使用此指標追蹤訊息處理成功，但不將其視為唯一已刪除訊息的確切計數。
NumberOfMessagesReceived 1	ReceiveMessage API 傳回的訊息數量。	計數	在接收操作期間報告。	- 這包括傳回給消費者的所有訊息，包括稍後因可見性逾時過期而傳回佇列的訊息。 - 如果未刪除單一訊息，則可以多次接收訊息，這可能會導致此指標超過傳送的訊息數量。 - 使用此項目來追蹤消費者活動，但不要將其視為處理的唯一訊息計數。

指標	描述	個單位	報告行為	重要備註
NumberOfMessagesSent 1	成功新增至佇列的訊息數量。	計數	每次手動傳送成功時回報。	- 手動呼叫 <code>SendMessage</code> 或 <code>SendMessageBatch</code> 會計入，包括直接以 DLQ 為目標的呼叫。 - 超過 <code>maxReceiveCount</code> 後自動移至 DLQ 的訊息 <code>maxReceiveCount</code> 不包含在此指標中。 - 因此，<code>NumberOfMessagesSent</code> 可能低於 <code>NumberOfMessagesReceived</code> - 特別是如果再驅動政策將許多訊息移至幕後 DLQs。
SentMessageSize 1	成功傳送至佇列的訊息大小。	位元組	在傳送至少一個訊息之前不會發出。	- 在佇列收到第一個訊息之前，此指標不會出現在 CloudWatch 主控台中。 - 使用此指標以位元組為單位追蹤每個訊息的大小。這有助於分析承載趨勢或估計輸送量成本。 - SQS 的訊息大小上限為 256 KB。

1 這些指標反映系統層級的活動，可能包括重試、重複或延遲訊息。請勿使用原始計數來估計即時佇列狀態，而不考慮訊息生命週期行為。

無效字母佇列 (DLQs) 和 CloudWatch 指標

使用 DLQs 時，請務必了解 Amazon SQS 指標的行為：

- **NumberOfMessagesSent** – 此指標對 DLQs 的行為不同：
 - 手動傳送 – 此指標會擷取手動傳送至 DLQ 的訊息。

- 自動重新驅動 – 此指標不會擷取因處理失敗而自動移至 DLQ 的訊息。因此，`NumberOfMessagesSent` 和 `NumberOfMessagesReceived` 指標可能會顯示 DLQs 的差異。
- DLQs 的建議指標 – 若要監控 DLQ 的狀態，請使用 `ApproximateNumberOfMessagesVisible` 指標。此指標表示目前可在 DLQ 中處理的訊息數量。

Amazon SQS 指標的維度

CloudWatch 中的 Amazon SQS 指標使用單一維度：**QueueName**。所有指標資料都會依佇列的名稱分組和篩選。

監控秘訣

使用關鍵指標和 CloudWatch 警示有效監控 SQS，以偵測佇列待處理項目、最佳化效能，並保持在服務限制內。

- 根據 設定 [CloudWatch 警示](#) `ApproximateNumberOfMessagesVisible`，以捕捉待辦項目成長。
- 監控 `NumberOfEmptyReceives` 以調整輪詢頻率並降低 API 成本。
- 在 FIFO 佇列 `ApproximateNumberOfGroupsWithInflightMessages` 中使用 來診斷輸送量限制。
- 檢閱 [SQS 配額](#) 以了解指標閾值和服務限制。

Amazon SQS 的合規驗證

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱 [AWS 服務 合規計劃範圍內](#) 然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 [AWS Compliance Programs](#)。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱 [在 中下載報告 AWS Artifact](#)。

使用 時的合規責任 AWS 服務 取決於資料的敏感度、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源以協助合規：

- [安全合規與治理](#) - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- [HIPAA 合格服務參考](#) – 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- [AWS 合規資源](#) – 此工作手冊和指南的集合可能適用於您的產業和位置。
- [AWS 客戶合規指南](#) – 透過合規的角度了解共同責任模型。本指南摘要說明保護 的最佳實務，AWS 服務 並將指南映射到跨多個架構的安全控制（包括國家標準和技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)）。

- AWS Config 開發人員指南中的[使用規則評估資源](#) – AWS Config 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- [AWS Security Hub](#) – 這 AWS 服務 可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱「[Security Hub 控制參考](#)」。
- [Amazon GuardDuty](#) – 這會監控您的環境是否有可疑和惡意活動，以 AWS 服務 偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- [AWS Audit Manager](#) – 這 AWS 服務 可協助您持續稽核 AWS 用量，以簡化您管理風險的方式，以及符合法規和產業標準的方式。

Amazon SQS 的恢復能力

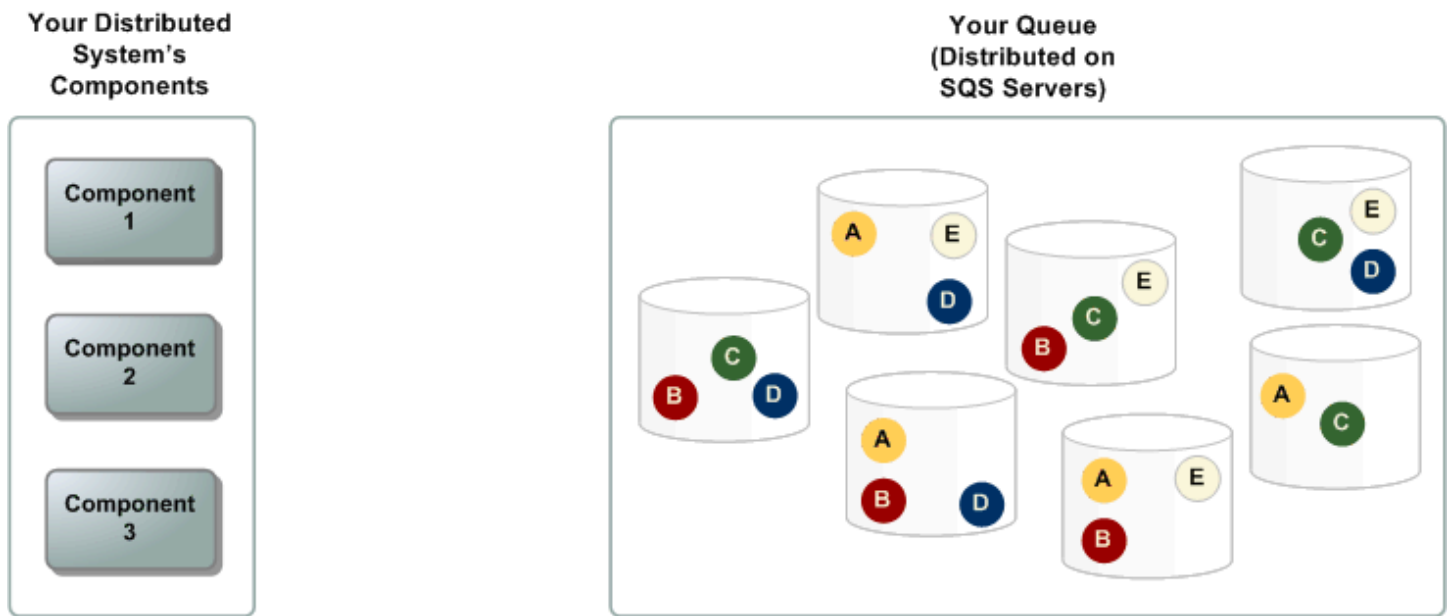
AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置。AWS 區域提供多個實體隔離和隔離的可用區域，這些可用區域以低延遲、高輸送量和高備援聯網連接。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。如需 AWS 區域和可用區域的詳細資訊，請參閱[AWS 全球基礎設施](#)。

除了 AWS 全球基礎設施之外，Amazon SQS 還提供分散式佇列。

分散式佇列

分散式傳訊系統中有三個主要部分：分散式系統的元件、佇列（在 Amazon SQS 伺服器上分佈），以及佇列中的訊息。

在以下情況中，您的系統有數個生產者（可傳送訊息到佇列的元件）以及消費者（從佇列接收訊息的元件）。佇列（存放訊息 A 到 E）將訊息以備援方式存放在多個 Amazon SQS 伺服器上。



Amazon SQS 的基礎設施安全性

作為受管服務，Amazon SQS 受到 [Amazon Web Services：安全程序概觀](#) 白皮書中所述 AWS 的全球網路安全程序保護。

您可以使用 AWS 發佈的 API 動作，透過網路存取 Amazon SQS。用戶端必須支援 Transport Layer Security (TLS) 1.2 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。

您必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署請求。或者，您可以使用 [AWS Security Token Service](#) (AWS STS) 來產生暫時安全登入資料以便簽署請求。

您可從任何聯網位置呼叫這些 API 動作，但 Amazon SQS 支援資源型存取政策，這類政策可以根據來源 IP 地址納入限制。您也可以使用 Amazon SQS 政策從特定 Amazon VPC 端點或特定 VPC 控制存取。這可有效隔離對指定 Amazon SQS 佇列的網路存取，使其僅與 AWS 網路中的特定 VPC 隔離。如需詳細資訊，請參閱[範例 5：如果不是來自 VPC 端點，則拒絕存取](#)。

Amazon SQS 的安全最佳實務

AWS 為 Amazon SQS 提供許多安全功能，您應該在自己的安全政策內容中檢閱這些功能。以下是 Amazon SQS 的預防性安全最佳實務。

 Note

針對常見使用案例和實作提供具體的實作指導。我們建議您在特定使用案例、架構和威脅模型的內容中檢視這些最佳實務。

確定佇列無法公開存取

除非您明確要求網際網路上的任何人能夠讀取或寫入您的 Amazon SQS 佇列，否則您應該確保您的佇列無法公開存取（全球所有人或任何已驗證 AWS 的使用者皆可存取）。

- 避免建立 Principal 設定為 "" 的政策。
- 避免使用萬用字元 (*)。請改為命名特定使用者或使用者。

實作最低權限存取

當您授與許可時，您可決定接收許可的人員、許可適用的佇列，以及您要對這些佇列允許的特定 API 動作。對於降低安全性風險及降低錯誤或惡意意圖的影響而言，實作最低權限非常重要。

遵循授與最低權限的標準安全建議。也就是說，只授與執行特定工作所需的許可。您可以使用安全政策的組合來實作此動作。

Amazon SQS 會使用生產者與消費者模型，並需要三種類型的使用者帳戶存取權：

- 管理員 – 建立、修改和刪除佇列的存取權。管理員也會控制佇列政策。
- 生產者 – 傳送訊息至佇列的存取權。
- 消費者 – 從佇列接收和刪除訊息的存取權。

如需詳細資訊，請參閱下列章節：

- [Amazon SQS 中的身分和存取管理](#)
- [Amazon SQS API 許可：動作和資源參考](#)
- [搭配 Amazon SQS 存取政策語言使用自訂政策](#)

針對需要 Amazon SQS 存取的應用程式 AWS 和服務使用 IAM 角色

對於 Amazon EC2 等應用程式 AWS 或服務來存取 Amazon SQS 佇列，他們必須在其 AWS API 請求中使用有效的 AWS 登入資料。由於這些登入資料不會自動輪換，因此您不應直接將 AWS 登入資料存放在應用程式或 EC2 執行個體中。

反之，您應使用 IAM 角色來為需存取 Amazon SQS 的應用程式和服務管理暫時性登入資料。使用角色時，您不需要將長期登入資料（例如使用者名稱、密碼和存取金鑰）分發給 EC2 執行個體 AWS 或服務，例如 AWS Lambda。相反地，該角色會提供暫時許可，供應用程式在呼叫其他 AWS 資源時使用。

如需詳細資訊，請參閱 [IAM 角色](#) 和 [常見的角色方案：使用者、應用程式和服務](#) 中的 IAM 使用者指南。

實作伺服器端加密

若要減輕資料外洩問題，請使用靜態加密，並利用與訊息儲存在不同位置的金鑰來加密訊息。伺服器端加密 (SSE) 會提供靜態資料加密。Amazon SQS 會在儲存資料時於訊息層級進行加密，並在您存取訊息時為您進行解密。SSE 使用中管理的金鑰 AWS Key Management Service。只要您有驗證請求並具備存取許可，存取加密資料與未加密資料的方式並無不同。

如需詳細資訊，請參閱 [Amazon SQS 中的靜態加密](#) 及 [Amazon SQS 金鑰管理](#)。

強制加密傳輸中的資料

如果沒有 HTTPS (TLS)，則網路型攻擊者可以利用中間人之類的攻擊，竊聽網路流量或操控它。使用佇列政策中的 [aws:SecureTransport](#) 條件，只允許透過 HTTPS (TLS) 加密的連線。

考慮使用 VPC 端點來存取 Amazon SQS

如果您的佇列必須能夠與之互動，但絕對不能公開於網際網路，請使用 VPC 端點，將佇列存取權限制為特定 VPC 內的主機。您可以使用佇列政策，從特定的 Amazon VPC 端點或特定的 VPC 控制對佇列的存取權。

Amazon SQS VPC 端點會提供兩種控制訊息存取的方式：

- 您可以控制經由特定 VPC 端點允許的請求、使用者或群組。
- 您可以使用佇列政策，控制哪些 VPC 或 VPC 端點可存取您的佇列。

如需詳細資訊，請參閱 [適用於 Amazon SQS 的 Amazon Virtual Private Cloud 端點](#) 及 [為 Amazon SQS 建立 VPC 端點政策](#)。

相關 Amazon SQS 資源

下表列出在您使用此服務時可能有用的相關資源。

資源	描述
Amazon Simple Queue Service API 參考資料	動作、參數和資料類型的說明，以及服務會傳回的錯誤清單。
AWS CLI 命令參考中的 Amazon SQS	您可以使用 來處理佇列的 AWS CLI 命令描述。
區域與終端節點	Amazon SQS 區域與端點的相關資訊
產品頁面	Amazon SQS 相關資訊的主要網頁。
開發論壇	以社群為基礎的論壇，供開發人員討論 Amazon SQS 相關技術問題。
AWS Premium Support 資訊	有關 AWS Premium Support 的資訊的主要網頁，這是 one-on-one、快速回應的支援管道，可協助您在 AWS 基礎設施服務上建置和執行應用程式。

文件歷史記錄

下表說明 2019 年 1 月後 Amazon Simple Queue Service 開發人員指南的重要變更。如需有關此文件更新的通知，您可以訂閱 [RSS 摘要](#)。

服務功能有時會逐步推出至提供服務 AWS 的區域。我們僅針對第一個版本更新此文件。我們不會提供有關區域可用性的資訊，也不會宣布後續區域的推展情況。如需服務功能的區域可用性和訂閱更新通知的相關資訊，請參閱[最新消息 AWS ?](#)。

變更	描述	日期
新增對雙堆疊 (IPv4 和 IPv6) 端點的支援	Amazon SQS 現在支援雙堆疊 (IPv4 和 IPv6) 端點，允許透過兩個 IP 通訊協定存取佇列。	2025 年 4 月 17 日
所有 Amazon SQS APIs CloudTrail 整合	針對所有 Amazon SQS APIsCloudTrail 整合。	2025 年 1 月 10 日
SQSUnlockQueuePolicy	Amazon SQS 新增了名為 AWS 的新受管政策SQSUnlock QueuePolicy，以解鎖佇列，並移除設定錯誤的佇列政策，拒絕所有主體存取 Amazon SQS 佇列。	2024 年 11 月 15 日
AWS kms:Decrypt	Amazon SQS 不再需要 SendMessage API 的 kms:Decrypt 許可。客戶現在只需要用於加密佇列的 KMS 金鑰kms:GenerateDataKey 許可，但仍需要呼叫的kms:Decrypt 許可ReceiveMessage。	2024 年 7 月 24 日
FIFO 指標更新	支援NumberOfDuplicatedSentMessages 並ApproximateNumberOfGroupsWi	2024 年 7 月 3 日

	thInflightMessages 新 增至 Amazon SQS FIFO 指 標。	
ListQueueTags AmazonSQSReadOnlyAccess 受管政策中支援的動作	AmazonSQSReadOnlyAccess 受管政策支援ListQueue Tags 擷取與指定 Amazon SQS 佇列相關聯的所有標籤。	2024 年 5 月 2 日
AWS JSON 通訊協定	使用 AWS JSON 通訊協定提 出 API 請求。	2023 年 7 月 27 日
新章節說明 Amazon SQS 的 AWS 受管政策，以及這些政策 的更新	Amazon SQS 新增了一個 新動作，可讓您列出特定來 源佇列下最新的訊息移動任 務 (最多 10 個)。此動作會與 ListMessageMoveTasks API 操作相關聯。	2023 年 6 月 7 日
使用 API 的無效字母佇列再驅 動	使用 Amazon SQS API 設定無 效字母佇列再驅動。	2023 年 6 月 7 日
Amazon SQS 的 ABAC	屬性型存取控制 (ABAC) 使用 佇列標籤提供靈活且可擴充的 存取許可。	2022 年 11 月 10 日
FIFO 高輸送量限制增加	增加商業區域 FIFO 高輸送量 模式的預設配額，再加上 FIFO 高輸送量文件最佳化。	2022 年 10 月 20 日
提供預設伺服器端加密 (SSE)	伺服器端加密 (SSE) 預設會使 用 SQL 擁有的加密 (SSE)。	2022 年 9 月 26 日
提供 Amazon SQS 混淆代理人 預防支援	混淆代理人預防可讓您在請求 中指定新標頭，這些標頭在 使用 Amazon SQS 受管 SSE 時，會根據 KMS 政策中的條件 進行檢查。	2021 年 12 月 29 日

現可支援受管 SSE	Amazon SQS 受管 SSE (SSE-SQS) 是受管的伺服器端加密，使用 Amazon SQL 擁有的加密金鑰來保護透過訊息佇列傳送的敏感資料。	2021 年 11 月 23 日
現已提供無效字母佇列再驅動	Amazon SQS 支援標準佇列的 無效字母佇列再驅動 。	2021 年 11 月 10 日
提供 FIFO 佇列中訊息的高輸送量	Amazon SQS FIFO 佇列的高輸送量可為 FIFO 佇列中的訊息提供更高的每秒交易數目 (TPS)。如需輸送量配額的相關資訊，請參閱 訊息相關的配額 。	2021 年 5 月 27 日
預覽版提供 FIFO 佇列中訊息的高輸送量	Amazon SQS FIFO 佇列的高輸送量為預覽版本，可能會有異動。此功能為 FIFO 佇列中的訊息提供更高的每秒交易數目 (TPS)。如需輸送量配額的相關資訊，請參閱 訊息相關的配額 。	2020 年 12 月 17 日
全新 Amazon SQS 主控台設計	為了簡化開發和生產工作流程，Amazon SQS 主控台提供 全新的使用者體驗 。	2020 年 7 月 8 日
Amazon SQS 支援 listQueues 和 listDeadLetterSourceQueues 的分頁	您可以指定要從 listQueues 或 listDeadLetterSourceQueues 請求傳回的結果數量上限。	2020 年 6 月 22 日
Amazon SQS 在所有 AWS 區域中支援 1 分鐘的 Amazon CloudWatch 指標，GovCloud (US) 區域除外 AWS	Amazon SQS 適用的 1 分鐘 Amazon CloudWatch 指標可用於所有區域，AWS GovCloud (US) 區域除外。	2020 年 1 月 9 日

Amazon SQS 支援 1 分鐘的 CloudWatch 指標	Amazon SQS 的一分鐘 CloudWatch 指標目前僅可在以下區域使用：美國東部 (俄亥俄)、歐洲 (愛爾蘭)、歐洲 (斯德哥爾摩) 和亞太區域 (東京)。	2019 年 11 月 25 日
AWS Lambda Amazon SQS FIFO 佇列的 觸發器可用	您可以設定送達 FIFO 佇列的訊息來觸發 Lambda 函數。	2019 年 11 月 25 日
Amazon SQS 的伺服器端加密 (SSE) 可在中國區域使用	適用於 Amazon SQS 的 SSE 在中國區域中可供使用。	2019 年 11 月 13 日
FIFO 佇列現已在中東 (巴林) 區域提供	FIFO 佇列現已在中東 (巴林) 區域提供。	2019 年 10 月 10 日
Amazon SQS 的 Amazon Virtual Private Cloud (Amazon VPC) 端點可在 AWS GovCloud (美國東部) 和 AWS GovCloud (美國西部) 區域使用	您可以從 AWS GovCloud (美國東部) 和 AWS GovCloud (美國西部) 區域中的 Amazon VPC 傳送訊息至 Amazon SQS 佇列。	2019 年 9 月 5 日
Amazon SQS 允許 AWS X-Ray 使用 訊息系統屬性對佇列進行疑難排解	您可以使用 X-Ray，對透過 Amazon SQS 佇列傳送的訊息進行疑難排解。這個版本會將 MessageSystemAttribute 請求參數新增至 SendMessage 和 SendMessageBatch API 操作 (可讓您透過 Amazon SQS 傳送 X-Ray 追蹤標頭)、 ReceiveMessage API 操作的 AWSTraceHeader 屬性，以及 MessageSystemAttributeValue 資料類型。	2019 年 8 月 28 日

[Amazon SQS 佇列建立時可進行標記](#)

您可以使用單一 Amazon SQS API 呼叫、AWS SDK 函數或 AWS Command Line Interface (AWS CLI) 命令，同時建立佇列並指定其標籤。此外，Amazon SQS 支援 `aws:TagKeys` 和 `aws:RequestTag` AWS Identity and Access Management (IAM) 金鑰。

2019 年 8 月 22 日

[Amazon SQS 的暫時佇列用戶端現已開放使用](#)

使用常見的訊息模式 (例如要求-回應) 時，暫存佇列可協助您節省開發時間和部署成本。您可以使用[暫存佇列用戶端](#)來建立高輸送量、符合成本效益的應用程式管理暫存佇列。

2019 年 7 月 25 日

[SSE for Amazon SQS 可在 AWS GovCloud \(美國東部\) 區域使用](#)

Amazon SQS 的伺服器端加密 (SSE) 可在 AWS GovCloud (美國東部) 區域取得。

2019 年 6 月 20 日

[FIFO 佇列可在亞太區域 \(香港\)、中國 \(北京\)、AWS GovCloud \(美國東部\) 和 AWS GovCloud \(美國西部\) 區域使用](#)

FIFO 佇列可在亞太區域 (香港)、中國 (北京)、AWS GovCloud (美國東部) 和 AWS GovCloud (美國西部) 區域使用。

2019 年 5 月 15 日

[Amazon VPC 端點政策可用於 Amazon SQS](#)

您可以為 Amazon SQS 建立 Amazon VPC 端點政策。

2019 年 4 月 4 日

[FIFO 佇列現已在歐洲 \(斯德哥爾摩\) 和中國 \(寧夏\) 區域提供](#)

FIFO 佇列現已在歐洲 (斯德哥爾摩) 和中國 (寧夏) 區域提供。

2019 年 3 月 14 日

[FIFO 佇列可在提供 Amazon SQS 的所有區域中使用](#)

FIFO 佇列可在美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (加利佛尼亞北部)、美國西部 (奧勒岡)、亞太區域 (孟買)、亞太區域 (首爾)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭)、歐洲 (倫敦)、歐洲 (巴黎) 和南美洲 (聖保羅) 區域使用。

2019 年 2 月 7 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。