aws

開發人員指南

# Amazon Simple Queue Service

# Amazon Simple Queue Service: 開發人員指南

# Table of Contents

# 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。提供通用的 Web 服務 API，您可透過 AWS SDK 支援的任何程式設計程語言加以存取。

主題



## 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at-least-once 訊息傳遞，而 FIFO 佇列僅支援一次訊息處理和高輸送量模式。
- 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。

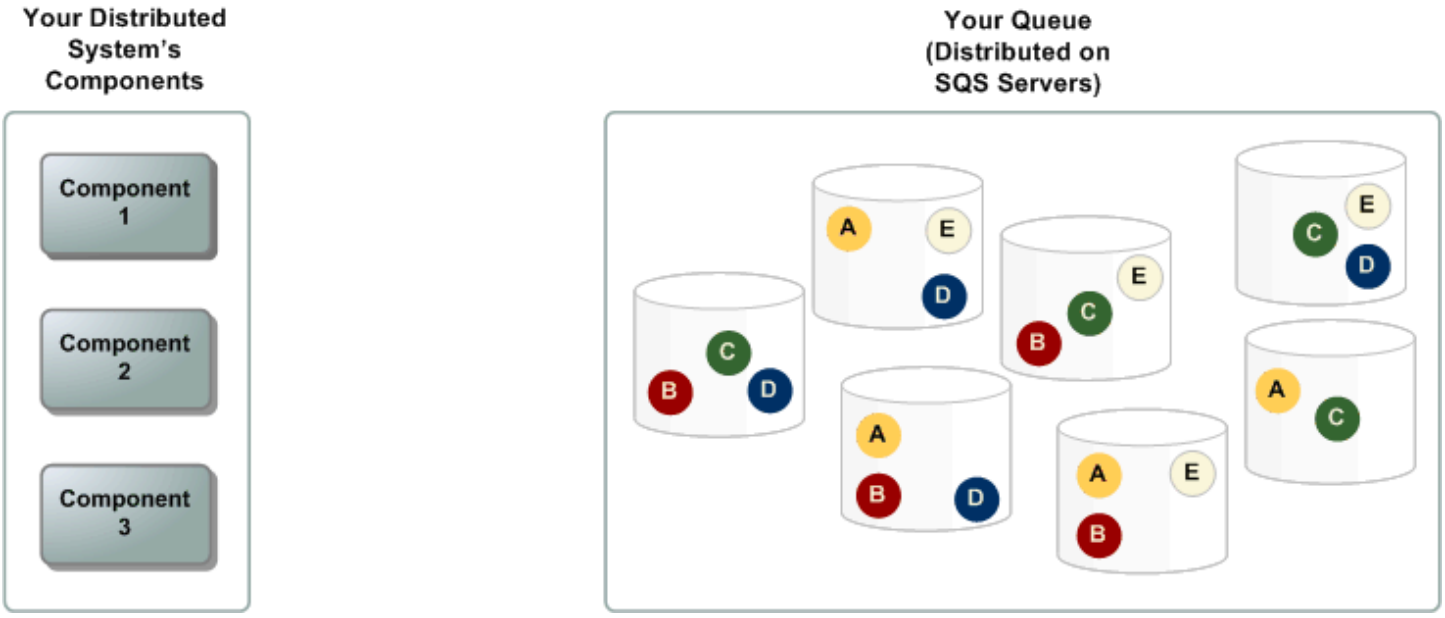
## 基本 Amazon SQS 架構

本節概述分散式傳訊系統的各個部分，並說明 Amazon SQS 訊息的生命週期。

# 分散式佇列

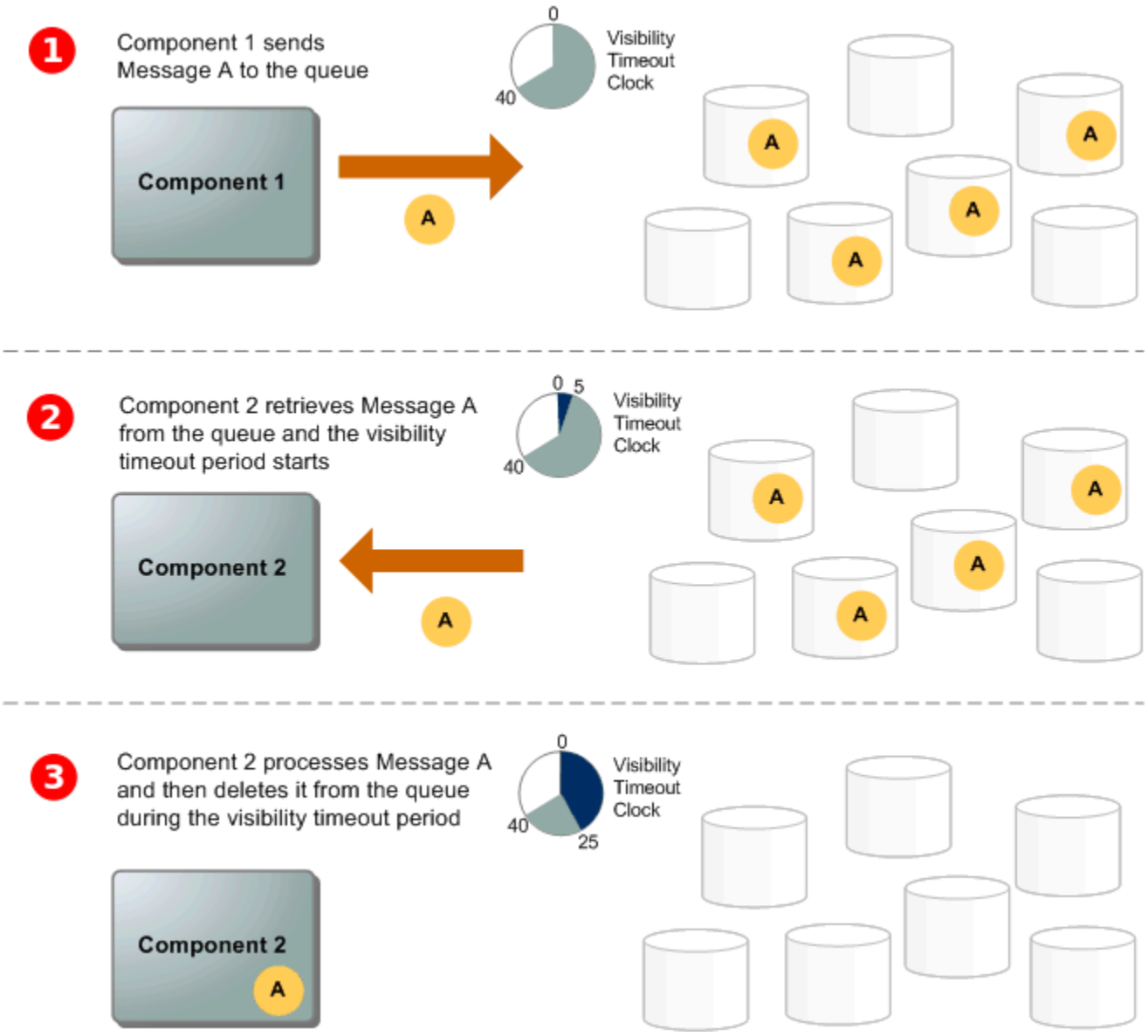
分散式傳訊系統有三個主要部分：分散式系統的元件、您的佇列 (分散在 Amazon SQS 伺服器上)，以及佇列中的訊息。

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) 將訊息 A 傳送至佇列，而訊息以備援方式分散到 Amazon SQS 伺服器上。

2

當消費者 (元件 2) 準備好要處理訊息，其會耗用佇列上訊息，因此傳回訊息 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

消費者 (元件 2) 從佇列刪除訊息 A，以免在可見性逾時到期後又再次接收和處理該則訊息。

> **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 是受管的簡訊服務，具有高度擴展性且易於使用。以下概述這些服務之間的差異：

Amazon SQS 提供整合與分離分散式軟體系統和元件的託管佇列。Amazon SQS 提供通用 Web 服務 API，您可以使用 AWS SDK 支援的任何程式語言存取該 API。佇列中的訊息通常由單一訂閱者處理。Amazon SQS 和 Amazon SNS 通常一起使用來建立扇出簡訊應用程式。

Amazon SNS 是一種發佈-訂閱服務，會提供從發佈者 (也稱為生產者) 到多個訂閱者端點 (也稱為消費者) 的訊息傳遞。發佈者透過製作並傳送訊息到主題 (其為邏輯存取點和通訊管道) 與訂閱者進行非同步的通訊。訂閱者可以訂閱 Amazon SNS 主題，並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，以及行動簡訊 (SMS)。Amazon SNS 充當訊息路由器，可即時將訊息傳遞給訂閱者。如果訂閱者在訊息發佈時無法使用，則不會儲存訊息以供日後擷取。

Amazon MQ 是一種受管訊息代理程式服務，提供與業界標準簡訊協定的相容性，例如 Advanced Message Queueing Protocol (AMQP) 和 Message Queuing Telemetry Transport (MQTT)。目前，Amazon MQ 支援 Apache ActiveMQ 和 RabbitMQ 引擎類型。

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 |

| 資源類型 | Amazon SNS | Amazon SQS | Amazon MQ |
| --- | --- | --- | --- |
| 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。我們建議使用 Amazon MQ 從現有訊息代理程式遷移應用程式，這些代理程式仰賴 API (例如 JMS) 或進階訊息佇列通訊協定 (AMQP)、MQTT 和簡單文字導向訊息通訊協定 (STOMP) 等協定的相容性。 OpenWire

# 設定 Amazon SQS

第一次使用 Amazon SQS 之前，您必須先完成以下步驟。

主題



## 步驟 1：建立 AWS 帳戶 和 IAM 使用者

如要存取任何 AWS 服務，首先您需要建立 AWS 帳戶，即可以使用 AWS 產品的 Amazon.com 帳戶。您可以使用您的 AWS 帳戶 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。

為了避免使用 AWS 帳戶 根使用者來進行 Amazon SQS 動作，最佳實務是為需對 Amazon SQS 進行管理存取的每個人建立 IAM 使用者。

### 註冊 AWS 帳戶

如果您還沒有 AWS 帳戶，請完成以下步驟建立新帳戶。

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。

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

   註冊 AWS 帳戶時，會建立 AWS 帳戶根使用者。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為最佳安全實務，將管理存取權指派給管理使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

註冊程序完成後，AWS 會傳送一封確認電子郵件給您。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。

## 建立管理使用者

當您註冊 AWS 帳戶 之後，請保護您的 AWS 帳戶根使用者，啟用 AWS IAM Identity Center，並建立管理使用者，讓您可以不使用根使用者處理日常作業。

保護您的 AWS 帳戶根使用者

1. 選擇 根使用者 並輸入您的 AWS 帳戶電子郵件地址，以帳戶擁有者身分登入 AWS Management Console。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)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶根使用者啟用虛擬 MFA 裝置 (主控台)。

建立管理使用者

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，將管理權限授予管理使用者。

   若要取得有關使用 IAM Identity Center 目錄 做為身分識別來源的教學課程，請參閱《使用 AWS IAM Identity Center 使用者指南中的以預設 IAM Identity Center 目錄 設定使用者存取權限。

以管理員的身分登入

- 若要使用您的 IAM 身分中心使用者登入，請使用建立 IAM 身分中心使用者時傳送至您電子郵件地址的登入 URL。

  如需有關如何使用 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。

# 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 和私密存取金鑰是專屬於 AWS Identity and Access Management。請勿將它們與其他 AWS 服務 (例如 Amazon EC2 金鑰對) 的憑證搞混。

若使用者想要與 AWS Management Console 之外的 AWS 互動，則需要程式設計存取權。授予程式設計存取權的方式取決於存取 AWS 的使用者類型。

若要授予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 | By |
| --- | --- | --- |
| 人力身分<br><br>(IAM Identity Center 中管理的使用者) | 使用臨時憑證簽署對 AWS CLI、AWS SDKs 或 AWS APIs 的程式設計請求。 | 請依照您要使用的介面所提供的指示操作。<br><br>• 關於 AWS CLI，請參閱 AWS Command Line Interface使用者指南 中的 設定 AWS CLI 來使用 AWS IAM Identity Center。<br>• 關於 AWS SDKs、工具和 AWS APIs，請參閱 AWSSDKs 和工具參考指南 中的 IAM Identity Center 驗證。 |
| IAM | 使用臨時憑證簽署對 AWS CLI、AWS SDKs 或 AWS APIs 的程式設計請求。 | 請遵循 IAM 使用者指南 中 使用臨時憑證搭配 AWS 資源 中的指示。 |
| IAM | (不建議使用)<br>使用長期憑證簽署 AWS CLI、AWS SDKs 或 AWS APIs 的程式設計要求。 | 請依照您要使用的介面所提供的指示操作。<br><br>• 關於 AWS CLI，請參閱 AWS Command Line Interface使用者指南 中的 使 |

| 哪個使用者需要程式設計存取權？ | 到 | By |
| --- | --- | --- |
| | | 用 IAM 使用者憑證進行驗證。<br>• 關於 AWS SDKs 和工具，請參閱 AWSSDKs 和工具參考指南 中的 使用長期憑證進行驗證。<br>• 關於 AWS API，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

本指南包括使用適用於 Java 的 AWS 開發套件範例。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。

您可以在其他編程語言，如圍棋JavaScript，Python 和 Ruby 開發AWS應用程序。如需詳細資訊，請參閱 AWS 上開發和管理應用程式的工具。

> **Note**
>
> 您可以瀏覽 Amazon SQS，而無需使用 AWS Command Line Interface (AWS CLI) 或 Windows PowerShell 等工具撰寫程式碼。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指令程式參考的 Amazon 簡單佇列服務區段中找到 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。

# Amazon SQS 入門

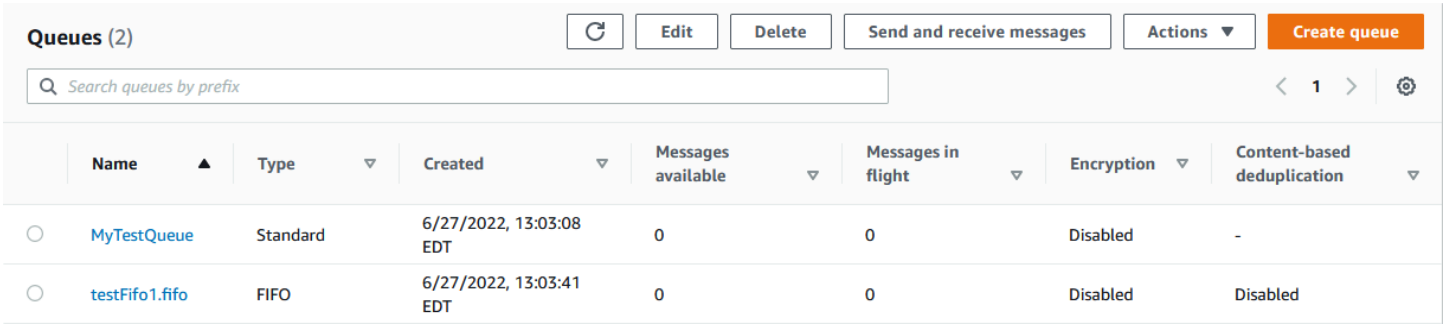
在本節中，您將學習如何使用 Amazon SQS 主控台建立標準佇列或 FIFO 佇列。

## 必要條件

開始之前，請完成 設定 Amazon SQS 中的步驟。

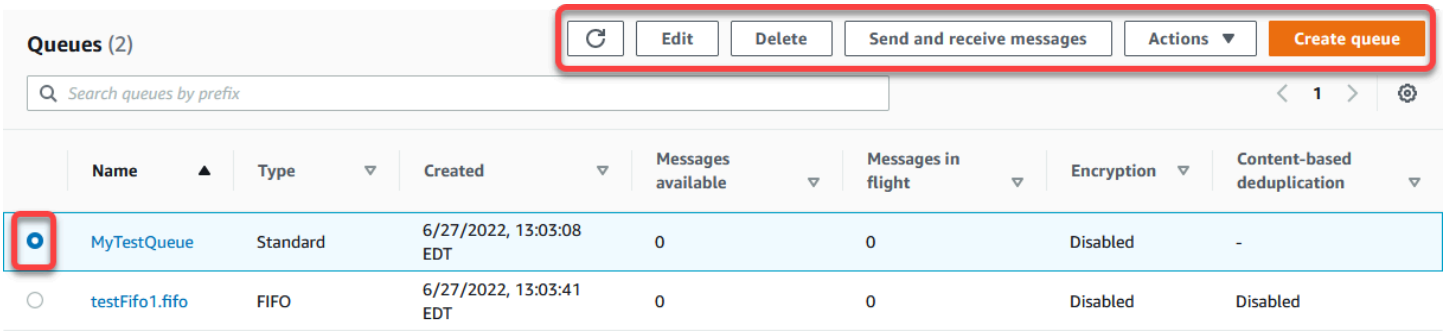
## 了解 Amazon SQS 主控台

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

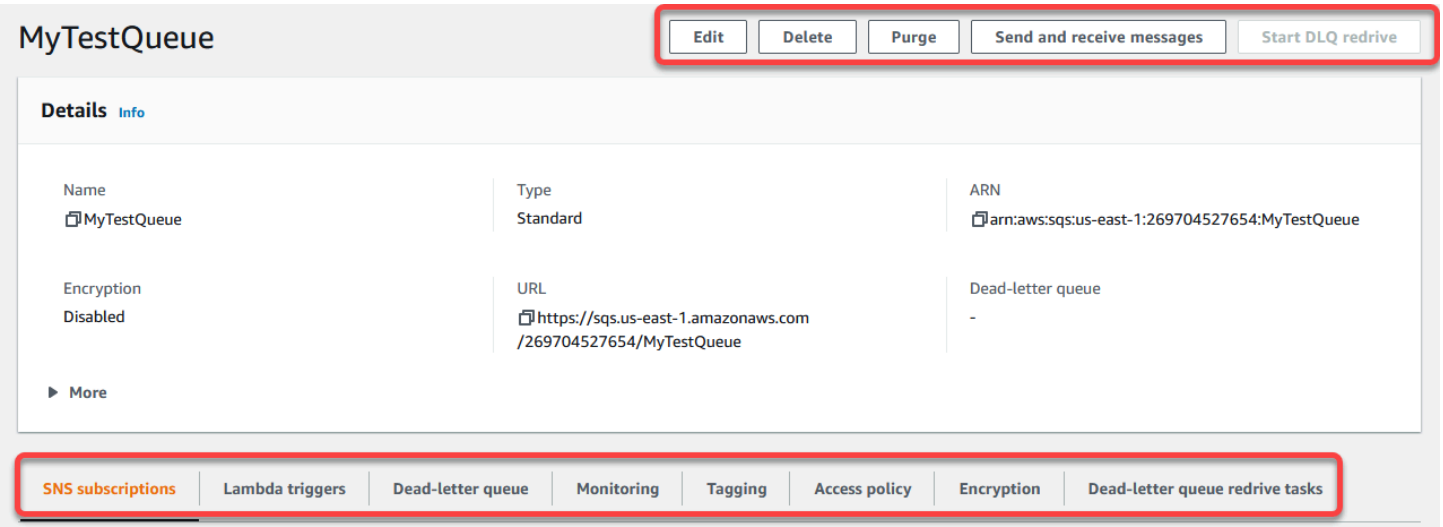



每個佇列的項目會顯示佇列類型以及佇列的其他相關資訊。類型欄可協助您一目了然地區分標準佇列和先進先出 (FIFO) 佇列。

在佇列頁面中，有兩種方式可對佇列執行動作。您可以選擇佇列名稱旁邊的選項，然後選擇要對佇列執行的動作。




您也可以選擇佇列名稱，以開啟佇列的詳細資訊頁面。詳細資訊頁面包含的動作與佇列頁面相同。此外，您可以選擇詳細資訊區段下的其中一個標籤來檢視其他組態詳細資訊和動作。




# Amazon SQS 佇列類型

Amazon SQS 支援兩種佇列類型 – 標準佇列和 FIFO 佇列。使用下表中的資訊來選擇適用於您情況的佇列。若要進一步了解 Amazon SQS 佇列，請參閱 Amazon SQS 標準佇列入門 和 Amazon SQS FIFO 佇列入門。

| 標準佇列 | FIFO 佇列 |
| --- | --- |
| 無限制的輸送量 – 標準佇列支援每秒 API 動作 (`SendMessage` 、`ReceiveMessage` 或 `DeleteMessage` ) 接近無限次數的 API 呼叫。<br><br>至少傳遞一次 – 一個訊息至少傳遞一次，但偶爾會傳遞一個以上的訊息副本。<br><br>盡力按順序傳遞 – 訊息偶爾不會按照傳送順序傳遞。 | 高輸送量 – 如果您使用批次處理，則每個 API 方法 (`SendMessageBatch` 、`ReceiveMessage` 或 `DeleteMessageBatch` ) 每秒最多可支援 3,000 則訊息。每秒 3,000 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。如需申請提高配額，請提交支援申請。沒有批次處理，FIFO 佇列支援每秒最多 300 個 API 呼叫，每個 API 方法 (`SendMessage` 、`ReceiveMessage` 、或 `DeleteMessage` )。<br><br>恰好處理一次 – 訊息只會傳遞一次並保持可用狀態，直到消費者處理訊息並將之刪除。此種佇列不會出現重複的情況。<br><br>先進先出傳遞 – 會嚴格遵守訊息發送和接收的順序。 |

| 標準佇列 | FIFO 佇列 |
| --- | --- |
|  |  |
| 當傳輸量很重要時，在應用程式之間傳送資料，例如：<br>• 將即時的使用者請求從密集的背景作業中分離出來：讓使用者在調整媒體規模或加以編碼時同時進行上傳。<br>• 將任務分配至多個工作者節點：處理大量的信用卡驗證請求。<br>• 批次處理訊息以供未來處理：為多個項目排程，以新增至資料庫。 | 當事件順序很重要時，在應用程式之間傳送資料，例如：<br>• 確保使用者輸入的命令以正確的順序執行。<br>• 以正確順序傳送價格修改來顯示正確的產品價格。<br>• 學生必須先註冊帳戶，否則無法註冊課程。 |

# 建立 Amazon SQS 標準佇列並傳送訊息

這是為 Amazon SQS 建立標準佇列的方法。

## 建立佇列 (主控台)

您可以使用 Amazon SQS 主控台來建立標準佇列。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 都可存取佇列名稱，包括帳單和 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. (選擇性) 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 SQL 受管加密金鑰，為佇列設定伺服器端加密 (SSE) (主控台) 和 為佇列設定伺服器端加密 (SSE) (主控台)。

> **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。

## 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

這是為 Amazon SQS 建立 FIFO 佇列的方法。

## 建立佇列

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 都可存取佇列名稱，包括帳單和 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 FIFO 佇列的高輸送量 及 訊息相關配額。

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 SQL 受管加密金鑰，為佇列設定伺服器端加密 (SSE) (主控台) 和 為佇列設定伺服器端加密 (SSE) (主控台)。

   > **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。

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。

## 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 FIFO 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 FIFO 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。

# 管理 Amazon SQS 佇列

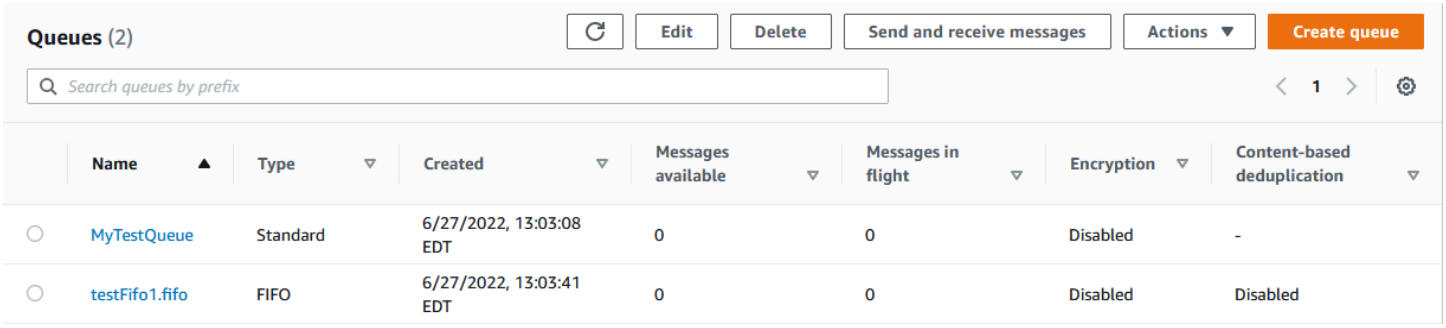
本節顯示如何使用 Amazon SQS 主控台管理佇列和訊息，有助於您更熟悉 Amazon SQS。

## 必要條件

開始之前，請完成 設定 Amazon SQS 中的步驟。

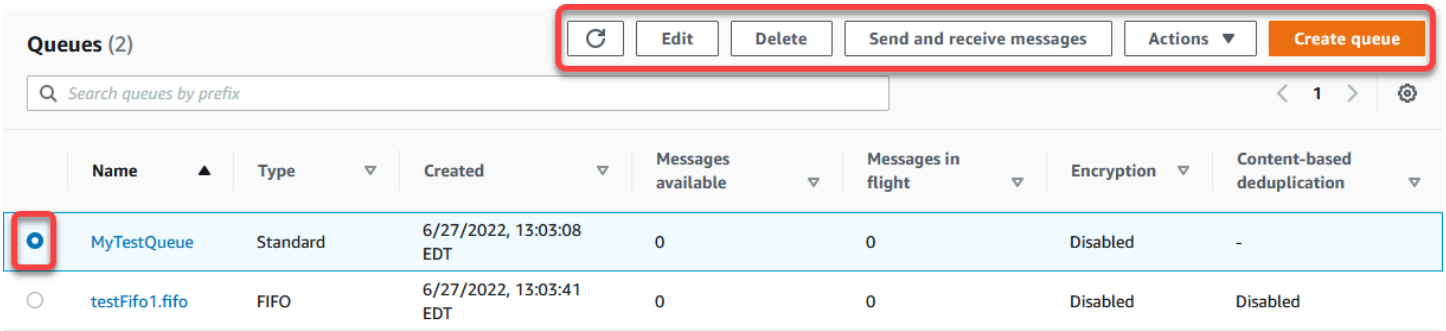
## 了解 Amazon SQS 主控台

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

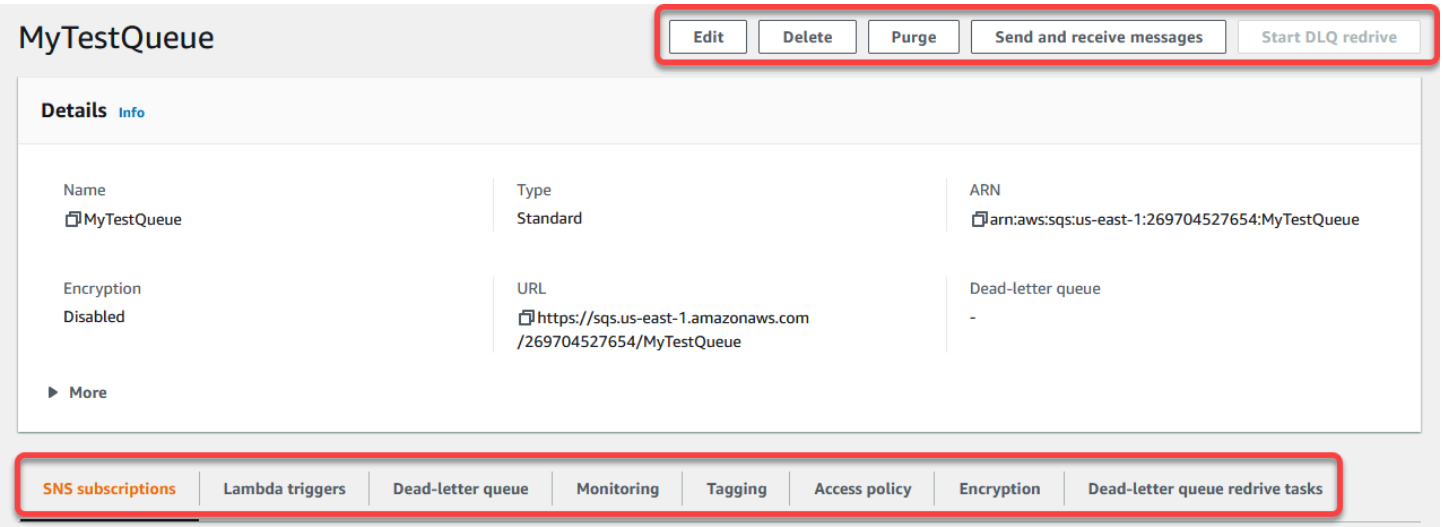



每個佇列的項目會顯示佇列類型以及佇列的其他相關資訊。類型欄可協助您一目了然地區分標準佇列和先進先出 (FIFO) 佇列。

在佇列頁面中，有兩種方式可對佇列執行動作。您可以選擇佇列名稱旁邊的選項，然後選擇要對佇列執行的動作。




您也可以選擇佇列名稱，以開啟佇列的詳細資訊頁面。詳細資訊頁面包含的動作與佇列頁面相同。此外，您可以選擇詳細資訊區段下的其中一個標籤來檢視其他組態詳細資訊和動作。




# 編輯佇列 (主控台)

您可以使用 Amazon SQS 主控台編輯任何佇列組態參數 (佇列類型除外)，以及新增或移除佇列功能。

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. 選擇儲存。

   主控台會顯示佇列的詳細資訊頁面。

# 接收和刪除訊息 (主控台)

將訊息傳送至佇列之後，您可以接收和刪除訊息。當您從佇列要求訊息時，您無法指定要擷取的訊息。但可以指定想要取得的訊息最大數量 (上限為 10)。

> **Note**
>
> 由於 Amazon SQS 是分散式系統，因此包含極少訊息的佇列可能會顯示對接收請求的空白回應。在這種情況下，請重新執行請求以取得您的訊息。視應用程式的需求而定，您可能必須使用短輪詢或長輪詢才能接收訊息。

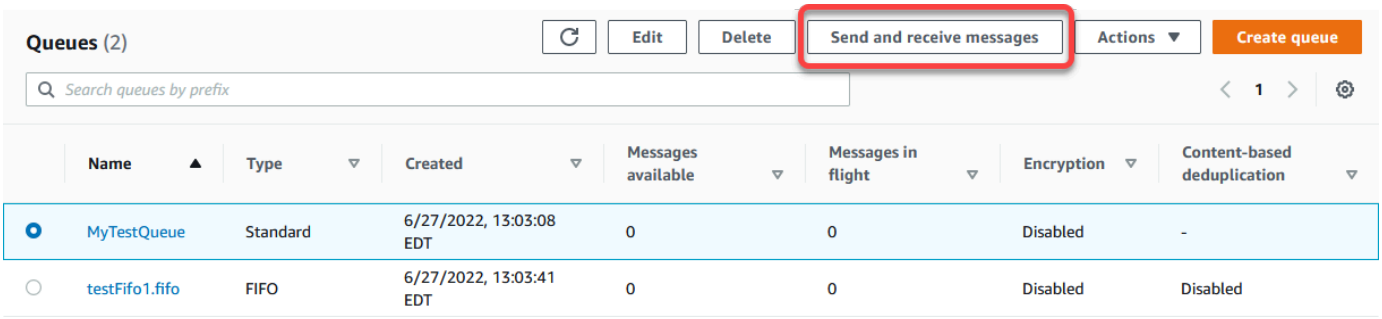
Amazon SQS 不會在接收訊息後自動刪除訊息，以防您沒有成功收到訊息 (例如消費者可能接收失敗或連線中斷)。若要刪除訊息，則必須傳送個別申請，表示已成功接收和處理訊息，而不再需要該訊息。請注意，您必須先收到訊息，才能刪除訊息。

> **Note**
>
> 從 Amazon SQS 主控台接收訊息後，主控台會立即將訊息設定回可見狀態，以便再次接收訊息。

如需有關接收和刪除訊息的 API 選項的詳細資訊，請參閱 Amazon SQS API 參考指南。

若要接收和刪除訊息 (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，選擇佇列。
4. 選擇傳送及接收訊息。




主控台會顯示傳送和接收訊息頁面。

5. 選擇訊息輪詢。

Amazon SQS 會開始輪詢佇列中的訊息。接收訊息區段右側的進度列會顯示輪詢的持續時間。

訊息區段會顯示已接收訊息的清單。對於每封訊息，清單會顯示訊息 ID、傳送日期、大小和接收計數。

6. 若要刪除訊息，請選擇您要刪除的訊息，然後選擇刪除。
7. 在刪除訊息對話方塊中，選擇刪除。

# 確認佇列是空的

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。

若要確認佇列是空的 (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，然後再將該佇列視為空。如需量度的詳細 CloudWatch 資訊，請參閱Amazon SQS 的可用 CloudWatch 指標。

# 刪除佇列

如果您不再使用 Amazon SQS 佇列並且預計在不久的將來不會使用它，我們建議將其刪除。

> **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。

若要刪除佇列 (AWS CLIAWS API)

您可以使用下列其中一項命令來刪除佇列：

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

## 清除來自 Amazon SQS 佇列的訊息 (主控台)

如果您不想刪除 Amazon SQS 佇列但想刪除其中的所有訊息，請清除佇列。訊息刪除過程最多約需 60 秒。建議您等候 60 秒的時間，無論佇列大小如何。

> **Important**
>
> 清除佇列後，您無法擷取任何已刪除的訊息。

若要清除佇列 (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，選擇要整個清除的佇列。
4. 從動作選擇清除。
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 入門常見任務

現在您已經建立佇列並已學會如何傳送、接收和刪除訊息，以及如何刪除佇列，您可能會希望試用下列項目：

- 若要觸發 Lambda 函數，請參閱 設定佇列以觸發 AWS Lambda 函數 (主控台)。
- 了解如何設定佇列，包括 SSE 和其他功能。
- 了解如何傳送具有屬性的訊息。
- 了解如何從 VPC 傳送訊息。
- 若要探索 Amazon SQS 的功能和架構，請參閱 Amazon SQS 佇列類型 和 基本 Amazon SQS 架構。

- 若要了解可協助您充分運用 Amazon SQS 的指導方針和注意事項，請參閱 Amazon SQS 的最佳實務。
- 探索其中一個開發 AWS 套件的 Amazon SQS 範例，例如AWS SDK for Java 2.x 開發人員指南。
- 若要進一步了解 Amazon SQS 命 AWS CLI 令，請參閱命AWS CLI 令參考。
- 若要了解 Amazon SQS 動作，請參閱《Amazon Simple Queue Service API 參考》。
- 進一步了解如何以程式設計的方式與 Amazon SQS 互動，請閱讀使用 API 並探索範本程式碼與程式庫以及開發人員中心：
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows 與 .NET

- 請至自動化和疑難排解 Amazon SQS 佇列章節了解如何追蹤成本和資源。
- 請至安全性章節了解如何保護您的資料及其存取權。
- 進一步了解 Amazon SQS 工作流程和程序：

# Amazon SQS 標準佇列入門

Amazon SQS 提供標準當做預設的佇列類型。標準佇列支援每秒 API 動作 (SendMessage、ReceiveMessage 或 DeleteMessage) 接近無限次數的 API 呼叫。標準佇列支援 at-least-once 訊息傳遞。但有時候一則訊息的多份副本可能不會按順序傳遞 (因為高度分散的架構所允許的輸送量近乎無限制)。標準佇列會盡力按次序傳遞，確保訊息大致上會依照發送的順序來傳遞。

Amazon SQS 在確認 SendMessage 之前，會以備援方式將訊息存放在多個可用區域 (AZ) 中。由於訊息複本儲存在多個 AZ 中，因此任何單一電腦、網路或 AZ 故障都不會導致訊息無法存取。

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 建立佇列 (主控台)。如需 Java 範例，請參閱 Amazon SQS Java 開發套件範例。

只要應用程式可以處理抵達超過一次且未照順序排列的訊息，您就可以在許多案例使用標準訊息佇列，例如：

- 將即時的使用者請求從密集的背景作業中分離出來 – 讓使用者在調整媒體規模或加以編碼時同時進行上傳。
- 將任務分配至多個工作節點 – 處理大量的信用卡驗證請求。
- 批次處理訊息以供未來處理 – 為多個項目排程，以新增至資料庫。

如需與標準佇列相關的配額，請參閱 配額。

關於使用標準佇列的最佳實務，請參閱 對於 Amazon SQS 標準和 FIFO 佇列的建議。

## 訊息排序

標準佇列會盡可能維持訊息的順序，但一則訊息的多個副本可能不會按照順序遞交。若您的系統需要維持順序，建議使用 FIFO (先進先出) 佇列，或是在每則訊息中加入定序資訊，如此便可在接收訊息時重新排列訊息的順序。

## 一個t-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。

若發生此種情況，則該訊息位於無法使用的伺服器上的副本並未刪除，而當您接收訊息時可能會再次收到該則訊息副本。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 佇列和訊息識別符

本節說明標準和 FIFO 佇列的識別符。這些識別符可以協助您尋找和操作特定的佇列和訊息。

主題



## Amazon SQS 標準佇列的識別符

如需下列識別符的詳細資訊，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符，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)。

以下為接收控點的範例 (分散在三行內)。

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

# 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 (_)。 |

| 配額 | 描述 |
| --- | --- |
| | **Note**<br>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。 |
| | 標記動作限制為每個 AWS 帳戶 30 TPS。如果您的應用程式需要更高的輸送量，請提交要求。 |

# Amazon SQS FIFO 佇列入門

FIFO (先進先出) 佇列不但擁有標準佇列的所有功能，在需要重視操作和事件的順序，或是不容許複本存在的情況下，其設計還可以強化應用程式之間的傳訊能力。

您可能使用 FIFO 佇列的情況範例如下：

- 訂單至關重要的電子商務訂單管理系統
- 與需要按順序處理事件的第三方系統整合
- 依照輸入的順序處理使用者輸入的內容
- 通訊和聯網 - 依相同的順序傳送和接收資料和資訊
- 電腦系統 - 確保使用者輸入的指令以正確的順序執行
- 教育機構 - 學生必須先註冊帳戶，否則無法註冊課程
- 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 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 “訊息相關配額”。

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 建立佇列 (主控台)。如需 Java 範例，請參閱 Amazon SQS Java 開發套件範例。

關於使用 FIFO 佇列的最佳實務，請參閱 對於 Amazon SQS FIFO 佇列的其他建議 及 對於 Amazon SQS 標準和 FIFO 佇列的建議。

## FIFO 交付邏輯

以下概念可協助您更佳了解從 FIFO 傳送和接收訊息。

傳送訊息

若連續將多則訊息傳送至 FIFO 佇列，每則訊息都有一個不同的訊息重複資料刪除 ID，Amazon SQS 會存放訊息並確認傳輸。接下來會按照訊息傳輸的正確順序來接收和處理每則訊息。

在 FIFO 佇列中，訊息是按照訊息群組 ID 來排序。若有多個主機 (或同一個主機上的不同執行緒) 將訊息群組 ID 相同的多則訊息傳送至 FIFO 佇列，Amazon SQS 會按訊息抵達接受處理的順序來存放訊息。為了確保 Amazon SQS 會維持訊息傳送和接收時的順序，每個訊息產生器都應使用專屬的訊息群組 ID 來傳送所有訊息。

FIFO 佇列邏輯僅會個別套用至各個訊息群組 ID。各訊息群組 ID 分別代表 Amazon SQS 佇列中不同順序的訊息群組 ID。每個訊息群組 ID 的所有訊息均會嚴格按照順序傳送和接收。但若訊息群組 ID 的值不同，便可能不會按照順序來傳送和接收訊息。請務必在訊息群組 ID 與訊息之間建立關聯性。若未提供訊息群組 ID，該動作會失敗。若需要訊息依序排列的單一群組，請為傳送至 FIFO 佇列的訊息提供相同的訊息群組 ID。

接收訊息

您無法提出請求接收具有特定群組 ID 的訊息。

從 FIFO 佇列接收具有複數訊息群組 ID 的訊息時，Amazon SQS 首先會嘗試盡可能傳回具有相同訊息群組 ID 的訊息。如此可讓其他消費者來處理訊息群組 ID 不同的訊息。當您收到含有訊息群組 ID 的訊息時，除非您刪除該訊息或訊息變成可見，否則不會再傳回相同訊息群組 ID 的訊息。

> **Note**
>
> 使用 `MaxNumberOfMessages` 動作的 `ReceiveMessage` 請求參數，即可透過一次呼叫接收多達 10 則訊息。這些訊息將保持其 FIFO 順序，而且可能具有相同的訊息群組 ID。因此，若同樣具有此訊息群組 ID 的訊息少於 10 則，您接收到的同一批 10 則訊息中可能就會有來自另一訊息群組 ID 的訊息，但仍然是按照 FIFO 順序。

多次重試

FIFO 佇列允許生產者或消費者嘗試多次重試：

- 如果生產者偵測到失敗的 `SendMessage` 動作，可以使用相同的重複資料刪除 ID，視需要多次重試傳送。假設生產者在重複資料刪除間隔到期之前至少收到一個確認，多次重試既不會影響訊息的排序，也不會引入重複項目。

- 如果消費者偵測到失敗的 ReceiveMessage 動作，它可以視需要使用相同的接收要求嘗試 ID 重試多次。假設消費者在可見性逾時到期前至少收到一個確認，則多次重試不會影響訊息的排序。
- 當您收到含有訊息群組 ID 的訊息時，除非您刪除該訊息或訊息變成可見，否則不會再傳回相同訊息群組 ID 的訊息。

## 訊息排序

FIFO 佇列是由標準佇列改良而來，與之相輔相成。此佇列類型的最重要功能是 FIFO (先進先出) 交付和僅處理一次：

- 系統會嚴格保留訊息傳送和接收的順序，並且會傳送一次訊息，而且在取用者處理並刪除訊息之前仍然無法使用。
- 此種佇列不會出現重複的情況。

此外，FIFO 佇列可支援訊息群組，允許單一佇列中出現多個按順序排列的訊息群組。FIFO 佇列中的訊息群組數目沒有配額。

## 恰好一次的處理

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。

# 從標準佇列移到 FIFO 佇列

若您既有的應用程式使用的是標準佇列，而您想利用 FIFO 佇列的排序或是恰好一次的處理功能，則必須正確設定您的應用程式。

> **Note**
>
> 您無法將既有的標準佇列轉換為 FIFO 佇列。若要轉移，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後，再將它重新建立為 FIFO 佇列。

若要確認您的應用程式是否能搭配 FIFO 佇列正確運作，請使用下列檢查清單：

- 使用建議的 FIFO 高輸送量模式來增加輸送量。若要進一步了解簡訊配額，請參閱 訊息相關配額。
- FIFO 佇列不支援各訊息的延遲，僅支援各佇列的延遲。若您的應用程式將每則訊息的 `DelaySeconds` 參數設為相同值，請務必修改應用程式，將各訊息的延遲移除，並改為將整個佇列設為 `DelaySeconds`。
- 訊息群組是一項獨特的 FIFO 功能，可讓客戶並行處理訊息，同時保持各自的順序。客戶會指定訊息群組 ID，將訊息組織成訊息群組。訊息群組通常以指定工作負載的業務維度為基礎。為了更好地擴展 FIFO 佇列，請使用更精細的業務維度作為訊息 ID。您分發訊息的訊息群組 ID 越多，FIFO 可供使用的訊息數就越多。
- 將訊息傳送至 FIFO 佇列前，請先確認以下項目：
  - 若您的應用程式可傳送內文相同的訊息，您可以修改應用程式，使之能夠為每則傳送出去的訊息提供一個專屬的訊息重複資料刪除 ID。
  - 若您的應用程式傳送的訊息內文不會重複，則可以啟用內容型的重複資料刪除功能。
- 您不用對消費者的程式碼進行任何變更。不過若是會花很長的時間在處理訊息，且您的可見性逾時的值設得相當高，請考慮為個別的 `ReceiveMessage` 動作新增請求嘗試 ID。無此可讓您在網路連線故障的時候重視重新嘗試接收，並避免佇列因為接收嘗試失敗而暫停。

如需詳細資訊，請參閱 Amazon Simple Queue Service API 參考。

# FIFO 佇列的高輸送量

FIFO 佇列的高輸送量支援每個 API 每秒的要求數目較高。若要增加 FIFO 佇列高輸送量的請求數目，您可以增加使用的訊息群組數目。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。有關具有高輸送量的 FIFO 配額的每個佇列配額的資訊，請參閱 訊息相關配額 和 SQS FIFO 佇列高輸送量的分割區和資料分配。

主題



# 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空間配置，會在 AWS 區域內的多個可用區之間自動複製。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：

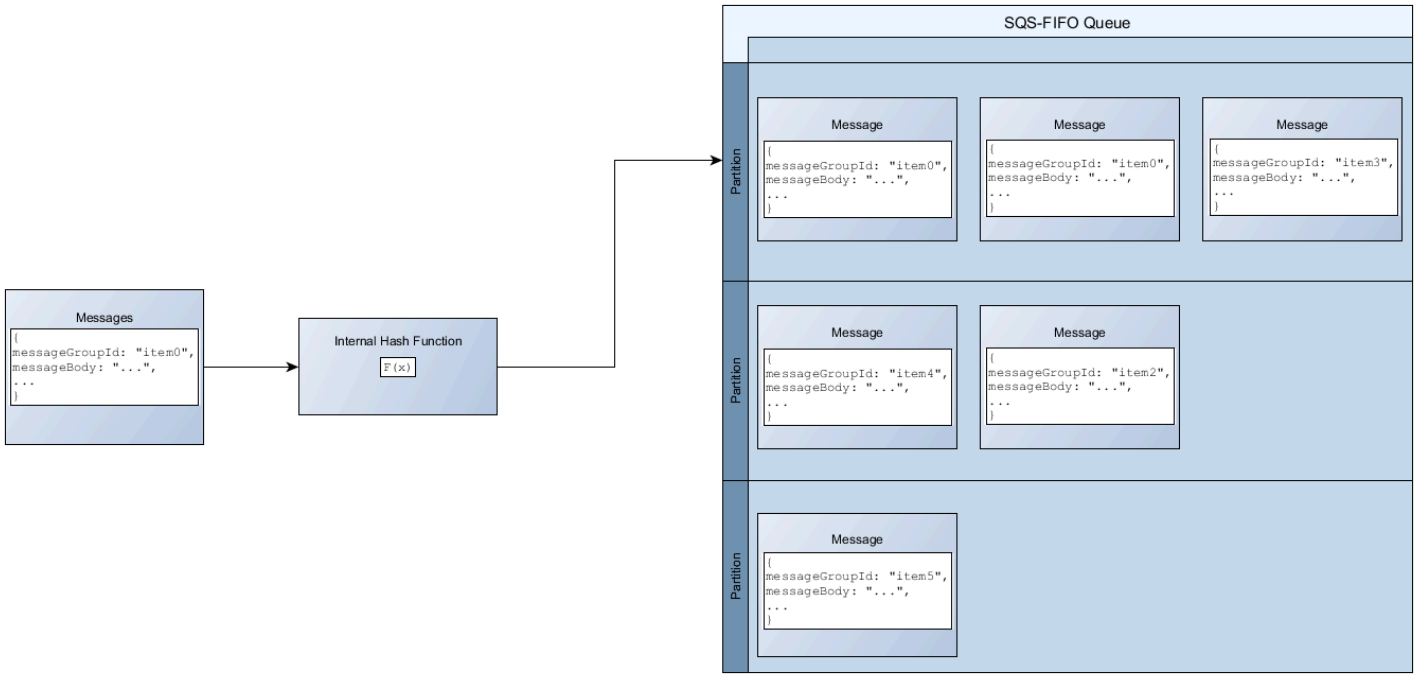
- 如果目前的要求率接近或超過現有分割區可支援的值，則會配置額外的分割區，直到佇列達到地區配額為止。如需配額的相關資訊，請參閱 訊息相關配額。
- 如果目前的分割區使用率低，則可能會減少分割區的數目。

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**
>
> Amazon SQS 已經過最佳化，可將項目一致分佈到 FIFO 佇列的分割區，而不論分割區數目為何。AWS 建議您使用可以有大量不同值的訊息群組 ID。

## 最佳化分割區利用率

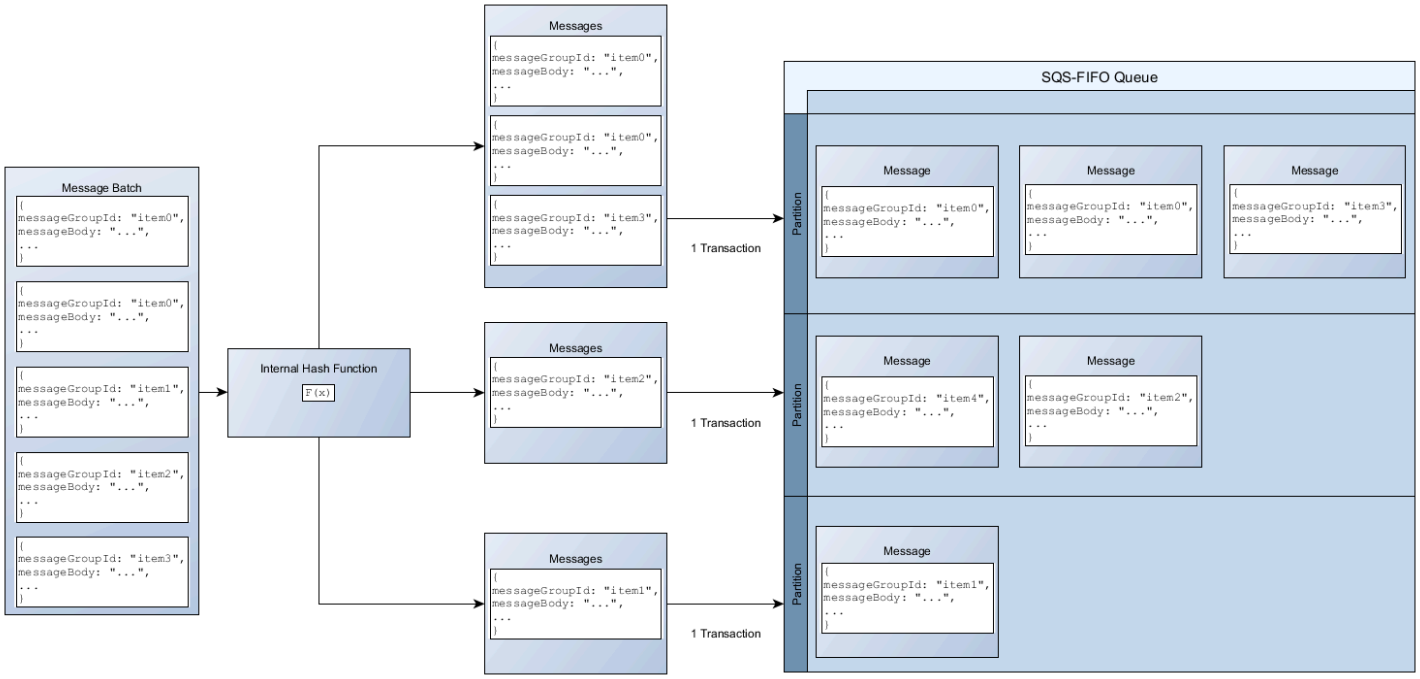
每個分割區最多支援每秒 3,000 則訊息進行批次處理，或支援的區域中傳送、接收和刪除作業，每秒最多可支援 300 則訊息。如需高輸送量訊息配額的詳細資訊，請參閱 Amazon Web Services 一般參考中的 Amazon SQS 服務配額。

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 ID 批次處理訊息。

若要最佳化 `DeleteMessageBatch` 和 `ChangeMessageVisibilityBatch` API 的分割區使用率，AWS 建議您使用 `MaxNumberOfMessages` 參數設為 10 的 `ReceiveMessage` 請求，並批次處理單一 `ReceiveMessage` 請求傳回的接收控點。

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。

## 為 FIFO 佇列啟用高輸送量

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 訊息相關配額 中的訊息輸送量。

# 重要用語

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。

訊息重複資料刪除 ID

用於已傳送訊息重複資料刪除的權杖。如果成功傳送具有特定訊息重複資料刪除 ID 的訊息，則使用相同訊息重複資料刪除 ID 傳送的任何訊息都會成功接受，但在 5 分鐘重複資料刪除間隔期間不會傳送。

> **Note**
>
> Amazon SQS 會繼續追蹤訊息重複資料刪除 ID，即使訊息已收到並刪除也一樣。

訊息群組 ID

標籤，指定訊息屬於特定訊息群組。屬於同一訊息群組的訊息總是按照相對於訊息群組的嚴格順序逐一處理 (不過，屬於不同訊息群組的訊息可能不會依序處理)。

接受請求嘗試 ID

用於 `ReceiveMessage` 呼叫重複資料刪除的權杖。

序號

Amazon SQS 指派給每則訊息的大型非連續數字。

# 相容性

用戶端

Amazon SQS 緩衝非同步用戶端目前不支援 FIFO 佇列。

服務

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

儘管允許您將 FIFO 佇列設定為目標，但某些 AWS 傳送通知給 Amazon SQS 的外部服務可能與 FIFO 佇列不相容。

下列 AWS 服務功能目前與 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 佇列和訊息識別碼

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。

主題



## Amazon SQS FIFO 佇列的識別碼

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。

以下是具MyQueue有 AWS 帳戶號碼123456789012的使用者所擁有的 FIFO 佇列的佇列 URL。

```
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+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+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 恰好一次的處理 和《Amazon Simple Queue Service API 參考》。

### 訊息重複資料刪除 ID

用於已傳送訊息重複資料刪除的權杖。如果成功傳送具有特定訊息重複資料刪除 ID 的訊息，則使用相同訊息重複資料刪除 ID 傳送的任何訊息都會成功接受，但在 5 分鐘重複資料刪除間隔期間不會傳送。

### 訊息群組 ID

標籤，指定訊息屬於特定訊息群組。屬於同一訊息群組的訊息總是按照相對於訊息群組的嚴格順序逐一處理 (不過，屬於不同訊息群組的訊息可能不會依序處理)。

### 序號

Amazon SQS 指派給每則訊息的大型非連續數字。

# 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) | 對於 FIFO 佇列，最多可以有 20,000 個傳輸中訊息 (消費者從佇列接收，但尚未從佇列中刪除)。如果達到此配額，Amazon SQS 不會傳回任何錯誤訊息。 |

| 配額 | 描述 |
| --- | --- |
| 佇列名稱 | 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。 |
| 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。 |
| | 標記動作限制為每個 AWS 帳戶 30 TPS。如果您的應用程式需要更高的輸送量，請提交要求。 |

# Amazon SQS 配額

本主題列出 Amazon Simple Queue Service (Amazon SQS) 內的配額。

主題



## 訊息相關配額

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<br><br>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。<br><br>MessageGroupId 對於 FIFO 佇列而言是必需的。您無法將它用於標準佇列。<br><br>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。 |

| 配額 | 描述 |
| --- | --- |
| | 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 | 標準佇列支援每秒 API 動作 (SendMessage 、ReceiveMessage 或 DeleteMessage ) 接近無限次數的 API 呼叫。 |
| | FIFO 佇列<br>• FIFO 佇列支援每個 API 動作 (SendMessage 、ReceiveMessage 和 DeleteMessage ) 每秒 300 筆交易的配額。<br>• 如果您使用批次處理，FIFO 佇列支援每個 API 動作 (SendMessage 、ReceiveMessage 和 DeleteMessage ) 每秒最多 3,000 則訊息。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。如需申請提高配額，請提交支援申請。 |

| 配額 | 描述 |
| --- | --- |
| | FIFO 佇列的高輸送量<br><br>• 在沒有批次處理 (SendMessage 、ReceiveMessage 和 DeleteMessage ) 的情況下，在美國東部 (維吉尼亞北部)、美國西部 (奧勒岡) 和歐洲 (愛爾蘭) 區域，用於 FIFO 佇列的高輸送量在每個 API 動作每秒最多可處理 70,000 筆交易。<br>• 對於美國東部 (俄亥俄) 和歐洲 (法蘭克福) 區域，預設輸送量為每個 API 動作每秒 18,000 筆交易。<br>• 對於亞太區域 (孟買)、亞太區域 (新加坡)、亞太區域 (雪梨) 和亞太區域 (東京) 區域，預設輸送量為每個 API 動作每秒 9,000 筆交易。<br>• 對於歐洲 (倫敦) 和南美洲 (聖保羅)，預設輸送量為每個 API 動作每秒 4,500 筆交易。<br>• 如需最大輸送量，請增加您用於不批次處理傳送之訊息的訊息群組 ID 數目。<br>• 您可透過在美國東部 (維吉尼亞北部)、美國西部 (奧勒岡) 及歐洲 (愛爾蘭) 區域使用批次處理 API (SendMessageBatch 和 DeleteMessageBatch ) 將輸送量增加到每秒最多 700,000 則訊息。每秒 700,000 則訊息代表每秒 70,000 筆交易，每一筆含一批 10 則訊息。<br><br>對於歐洲 (法蘭克福) 和美國東部 (俄亥俄) 區域，您可以透過使用批次處理 API 每秒達到 180,000 則訊息。每秒 180,000 則訊息代表每秒 18,000 筆交易，每一筆含一批 10 則訊息。<br><br>對於亞太區域 (孟買)、亞太區域 (新加坡)、亞太區域 (雪梨) 和亞太區域 (東京) 區域，批次處理時每秒最多可處理高達 90,000 則訊息。若要在使用 SendMessageBatch 和 DeleteMessageBatch 時達到最大輸送量，批次要求中的所有訊息都必須使用相同的訊息群組 ID。 |

| 配額 | 描述 |
| --- | --- |
| | • 對於歐洲 (倫敦) 和南美洲 (聖保羅) 區域，批次處理時每秒最多可處理高達 45,000 則訊息。若要在使用 `SendMessageBatch` 和 `DeleteMessageBatch` 時達到最大輸送量，批次要求中的所有訊息都必須使用相同的訊息群組 ID。<br>• 在所有其他 AWS 區域中，每個 API 動作的最大輸送量為每秒 2,400 (不含批次處理) 或 24,000 則 (使用批次處理) 訊息。<br>• 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。 |
| 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)。<br><br>若要傳送大於 256 KiB 的訊息，您可以使用適用於 Java 的 Amazon SQS 擴充用戶端程式庫和 Amazon SQS 擴充用戶端程式庫 (適用於 Python)。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。<br><br>**Note**<br>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。 |
| 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 政策相關配額。 |

# 政策相關配額

下表列出政策相關的配額。

| Name | 最大 |
| --- | --- |
| 位元組 | 8,192 |
| 條件 | 10 |
| 主體 | 50 |
| 陳述式 | 20 |
| 每個陳述式的動作 | 7 |

# Amazon SQS 特性和功能

Amazon SQS 提供下列特性與功能：

主題



## 訊息中繼資料

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。您可以使用訊息系統屬性，存放像是 AWS X-Ray 等其他 AWS 服務的中繼資料。

主題



### Amazon SQS 訊息屬性

Amazon SQS 可讓您將結構化中繼資料 (例如時間戳記、地理空間資料、簽章及識別符) 加入使用訊息屬性的訊息。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。如需有關使用 Amazon SQS 主控台傳送訊息屬性的相關資訊，請參閱傳送具有屬性的訊息 (主控台)。

> **Note**
>
> 請勿混淆訊息屬性與訊息系統屬性：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。

主題



## 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》中的 `SendMessage`)。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AWS SDK for Java，則可略過本節。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 MD5 訊息摘要的 AWS SDK，您必須使用以下準則來執行 MD5 訊息摘要計算。

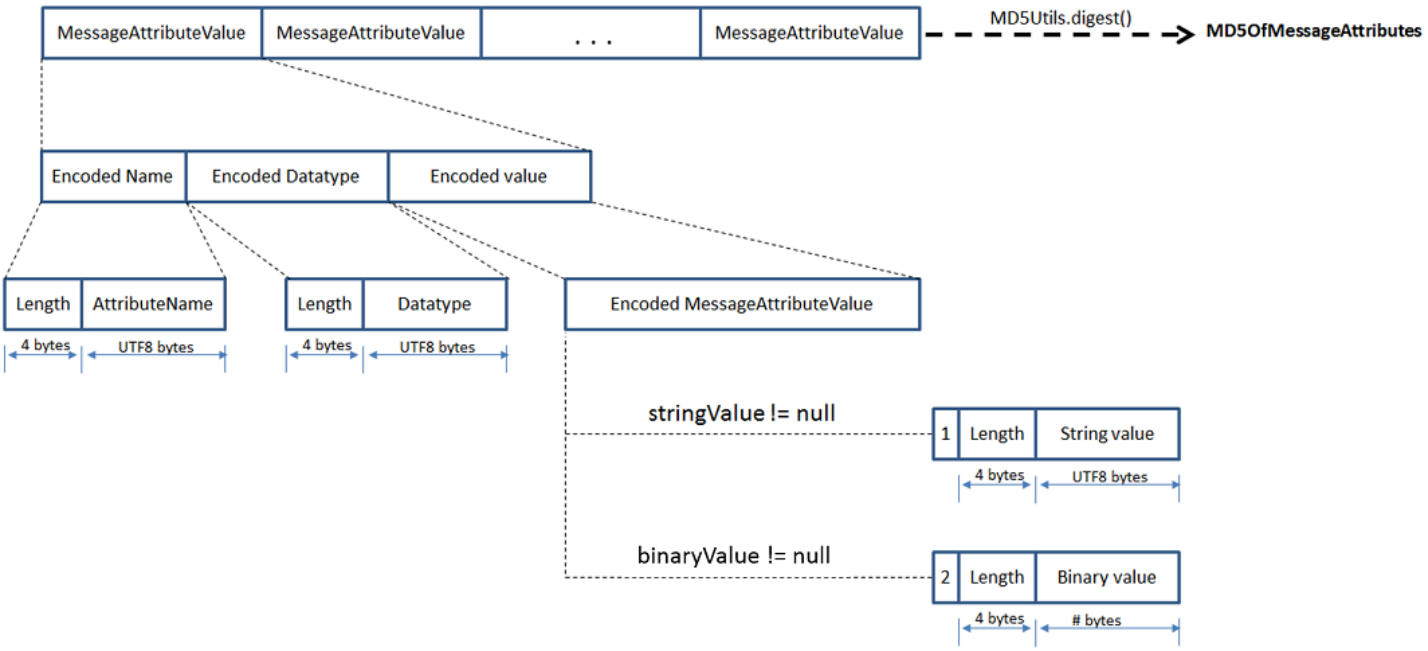
> **Note**
>
> 一律在 MD5 訊息摘要計算中包含自訂資料類型尾碼。

### 概要

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《Amazon Simple Storage Service API 參考》中的 SendMessage 和 SendMessageBatch API 動作的 MessageSystemAttribute 請求參數、ReceiveMessage API 動作的 AWSTraceHeader 屬性以及 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。其類型必須是 String，且其值必須是格式正確的 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

為了協助預估處理佇列訊息所需的資源，Amazon SQS 可以判斷佇列中延遲、可見、不可見的訊息的大致數量。如需可見性的詳細資訊，請參閱 Amazon SQS 可見性逾時。

> **Note**
>
> 對於標準佇列，由於 Amazon SQS 的分散式架構，結果是近似值。在大多數情況下，計數應接近佇列中的實際訊息數目。

對於 FIFO 佇列，結果是準確的。

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。 | ApproximateNumberOfMessages Visible |
| 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。 | ApproximateNumberOfMessages Delayed |
| 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 |

## 列出佇列分頁

listQueues 和 listDeadLetterQueues API 方法支援選用的分頁控制項。根據預設，這些 API 方法會在回應訊息中傳回多達 1000 個佇列。您可以將 MaxResults 參數設定為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。如果未設定 MaxResults，回應會包含最多 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您的 AWS 帳單，以反映您自己的成本結構。若要這樣做，請註冊以取得 AWS 帳戶 帳單，以納入標籤鍵和值。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 | 索引鍵 | 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。
> - 標記動作限制為每個 AWS 帳戶 30 TPS。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 配額。

# Amazon SQS 短期和長輪詢

Amazon SQS 提供短輪詢和長輪詢，以接收來自佇列的訊息。默認情況下，佇列使用短輪詢。

使用短輪詢時，`ReceiveMessage` 請求只查詢伺服器的子集 (根據加權隨機分佈)，以尋找可包含在回應中的訊息。即使查詢找不到任何訊息，Amazon SQS 也會立即傳送回應。

使用長輪詢時，`ReceiveMessage` 要求會查詢所有伺服器是否有訊息。Amazon SQS 會在收集至少一個可用訊息後傳送回應，最多為請求中指定的訊息數目上限。只有在輪詢等待時間到期時，Amazon SQS 才會傳送空白回應。

以下各節說明短輪詢和長輪詢的詳細資訊。

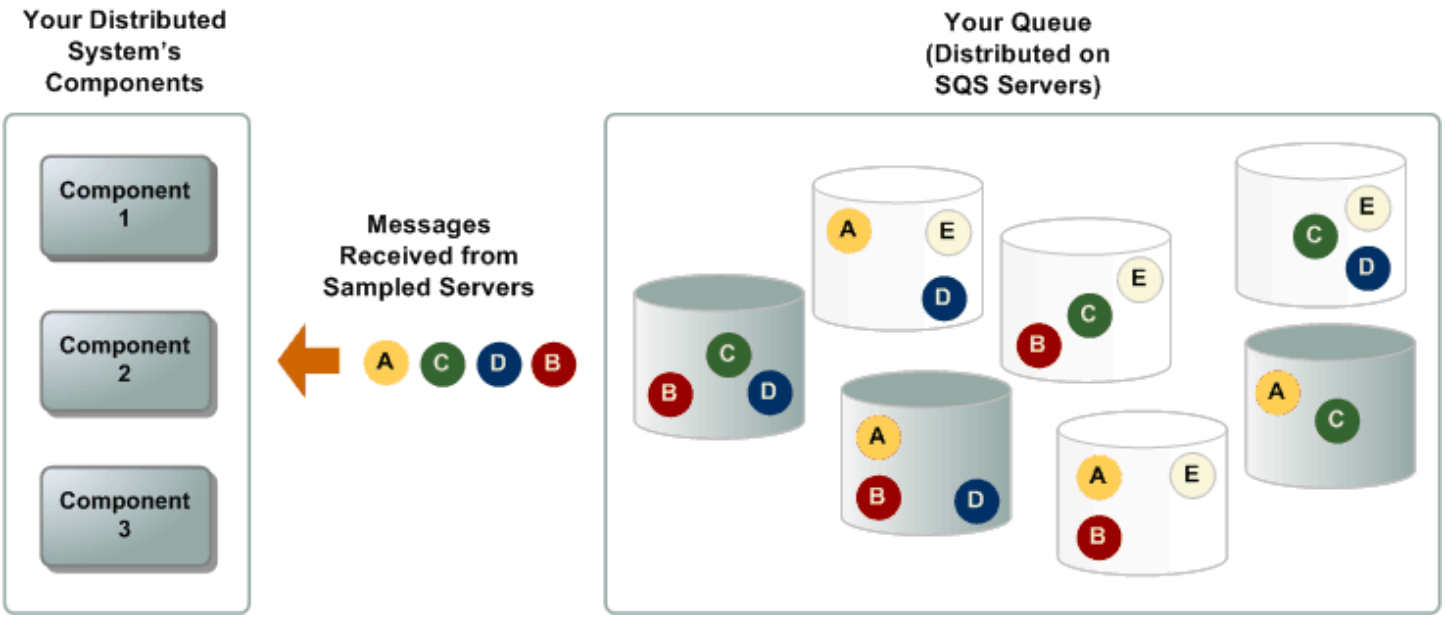
主題



## 以短輪詢消耗訊息

使用短輪詢自佇列消耗訊息時，Amazon SQS 會對其伺服器的子集進行採樣 (按加權隨機分佈)，並且僅從那些伺服器傳回訊息。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 主控台為新佇列

或現有佇列啟用長輪詢的相關資訊，請參閱 設定佇列參數 (主控台)。如需最佳實務做法，請參閱「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，您也可能會收到空白回應，尤其是當您為 ReceiveMessageWaitTimeSeconds 參數指定較低的值時。
- 透過查詢所有 Amazon SQS 伺服器 (而非子集) 來減少錯誤的空白回應。
- 訊息一旦可供使用時，立即傳回訊息。

關於如何確認佇列是否為空，詳細資訊請參閱 確認佇列是空的。

## 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 無效字母佇列

Amazon SQS 支援無效字母佇列 (DLQ)，當其他佇列 (來源佇列) 無法成功處理 (消費) 訊息時，可以將該訊息傳送至無效字母佇列。無效字母佇列很適合用來為應用程式或傳訊系統偵錯，因為它們可讓您隔離未取用的訊息以判斷無法成功處理的原因。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 設定無效字母佇列 (主控台)。偵錯消費者應用程式或消費者應用程式可用來使用訊息之後，您就可以使用無效字母佇列再驅動功能，將訊息移回來源佇列。

> **Important**
>
> 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。

主題

## 無效字母佇列的運作方式

有時候訊息會因各種可能的問題而無法處理，例如生產者或消費者的應用程式出現錯誤狀況，或是預料之外的狀態變更而導致應用程式的程式碼出現問題。舉例而言，若有個使用者在網路上下訂了某個產品 ID，但該產品 ID 已被刪除，網路商店的程式碼失敗並顯示錯誤，而帶有該次下訂請求的訊息就被送到無效字母佇列中。

有時生產者和消費者可能會錯誤解讀用來通訊的協定的某些方面，而導致訊息毀損或遺失。消費者的硬體錯誤也可能導致訊息承載毀損。

再驅動政策指定了來源佇列、無效字母佇列，還有在原始佇列的消費者在指定次數內無法處理訊息，此時 Amazon SQS 要將訊息從前者移到後者的情況。`maxReceiveCount` 是消費者在將訊息移至無效字母佇列之前，嘗試從佇列接收訊息而不將其刪除的次數。將 `maxReceiveCount` 設定為較低的值 (例如 1) 會導致無法接收訊息，導致訊息移至無效字母佇列。此類失敗包括網路錯誤和用戶端相依性錯誤。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。此政策適用於潛在的無效字母佇列。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。預設值是允許所有來源佇列使用無效字母佇列。如果您選擇允許特定佇列 (使用 `byQueue` 選項)，則可以使用來源佇列 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。

若要指定無效字母佇列，請使用主控台或 AWS 開發套件。每個傳送訊息至無效字母佇列的佇列皆必須如此。相同類型的多個佇列可以將目標設為同一個無效字母佇列。如需詳細資訊，請參閱 設定無效字母佇列 (主控台) 以及 `CreateQueue` 或 `SetQueueAttributes` 動作的 `RedrivePolicy` 和 `RedriveAllowPolicy` 屬性。

**Important**

FIFO 佇列的無效字母佇列必須也是 FIFO 佇列。同樣地，標準佇列的無效字母佇列必須也是標準佇列。

請以相同的 AWS 帳戶 建立無效字母佇列和其他會將訊息傳送至無效字母佇列的佇列。此外，無效字母佇列的所在區域必須與其他使用無效字母佇列的佇列相同。舉例而言，若您在美國東部 (俄亥俄) 區域中建立了一個佇列，而您希望讓該佇列使用無效字母佇列，則第二個佇列也必須位在美國東部 (俄亥俄) 區域內。

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。將訊息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，最佳作法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。

## 無效字母佇列有何優點？

無效字母佇列的主要任務是處理未取用訊息的生命週期。無效字母佇列可讓您擱置和隔離無法正確處理的訊息，以找出為何無法成功處理。設定無效字母佇列可完成以下項目：

- 為移至無效字母佇列的任何訊息設定警示。
- 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，以診斷軟體問題或生產者或消費者的硬體問題。
- 判定您是否有給予消費者儲存的訊息處理時間。

## 不同的佇列類型如何處理訊息故障？

### 標準佇列

標準佇列會繼續處理訊息，直到保留期間過期為止。這樣對訊息的持續處理，可將佇列被無法處理的訊息堵塞住的可能性降至最低。持續的訊息處理也可以加快佇列的復原速度。

在處理上千則訊息的系統中，若有大量消費者反覆無法接收和刪除的訊息，會增加成本並讓硬體受到額外的負擔。請不要試著一直處理失敗的訊息直到訊息到期，而最好在嘗試處理訊息幾次後，便將訊息移動到無效字母佇列。

> **Note**
>
> 標準佇列允許大量的傳輸中訊息出現。若大多數的訊息無法消耗又不送往無效字母佇列，則有效訊息的處理速率會變慢。因此，為了保持佇列的效率，請確保您的應用程式正確處理訊息。

### FIFO 佇列

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令一個訊息群組擷取排序好的訊息，第一個訊息群組會維持無法使用的狀態，直到阻塞該佇列的訊息成功處理完畢或移至無效字母佇列。

> **Note**
>
> FIFO 佇列允許傳輸中訊息的數量較少。因此，為了防止 FIFO 佇列被訊息阻塞，請確保您的應用程式正確處理訊息。
>
> 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。

## 何時該使用無效字母佇列？

✔ 請務必將無效字母佇列搭配標準佇列一起使用。若應用程式不需仰賴排序功能，請一律使用無效字母佇列。無效字母佇列可協助執行錯誤訊息傳輸操作的錯誤診斷作業。

> **Note**
>
> 即便使用的是無效字母佇列，也應該持續監控佇列，並重新嘗試發送因暫時性原因而傳送失敗的訊息。

✔ 請使用無效字母佇列來減少訊息的數量，並降低系統接觸到毒丸 (poison-pill) 訊息 (可接收到無法處理的訊息) 的可能性。

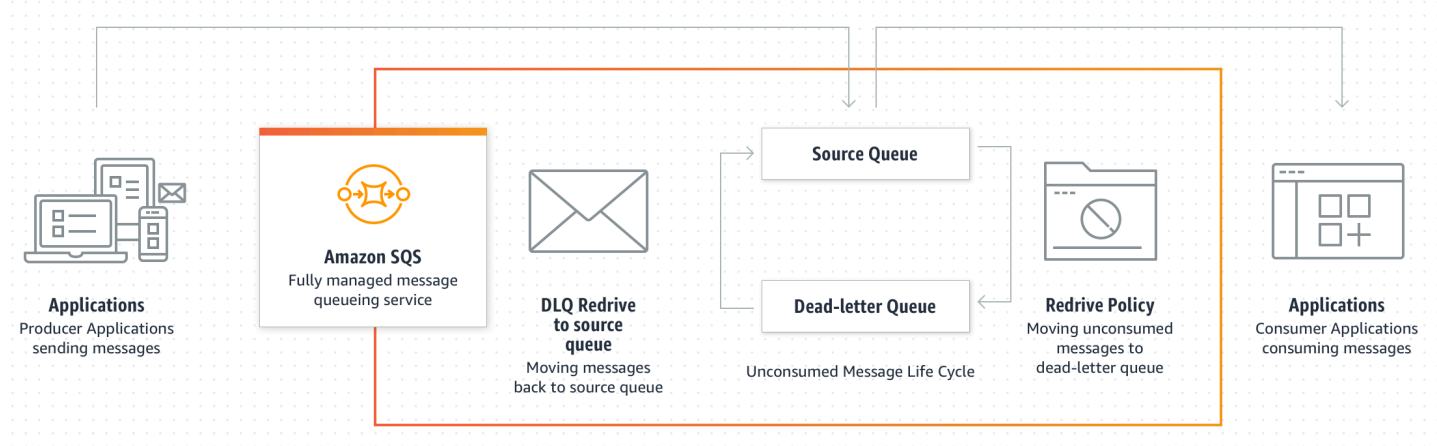
✘ 若希望能夠持續無限重試傳輸訊息，請勿將無效字母佇列搭配標準佇列使用。舉例而言，若您的程式必須等待相依的程序轉為使用中或可供使用，請不要使用無效字母佇列。

✘ 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。

## 將訊息從無效字母佇列中移出

您可以使用無效字母佇列再驅動來管理未取用訊息的生命週期。調查完標準或 FIFO 無效字母佇列中未取用訊息的可用屬性和相關中繼資料之後，您可以將訊息再驅動回其來源佇列。無效字母佇列再驅動會在移動訊息時批次處理訊息，以減少 API 呼叫計費。

再驅動任務代表使用者使用 Amazon SQS 的 `SendMessageBatch`、`ReceiveMessage` 和 `DeleteMessageBatch` API 來再度驅動訊息。因此，所有再驅動的訊息都會被視為具有新 `messageid`、`enqueueTime` 和保留期的新訊息。無效字母佇列再驅動的定價使用調用的 API 呼叫數量，並根據 Amazon SQS 定價開立帳單。




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。如需有關設定無效字母佇列再驅動的指示，請參閱 設定無效字母佇列再驅動。

> **Note**
>
> 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。
> 從 FIFO 無效字母佇列再驅動訊息時，訊息 groupID 和 deduplicationID 將保持不變，而訊息會收到新的 messageID。
> Amazon SQS 無效字母佇列會按照收到訊息的順序重新推動訊息，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，若生產者同時從無效字母佇列接收重新推動的訊息時將訊息傳送到來源 FIFO 佇列，則重新推動的訊息將與生產者的新訊息緊密結合。

## 無效字母佇列的疑難排解

在某些情況下，Amazon SQS 無效字母佇列可能無法總是按照預期行動。本節會提供常見問題的概觀，並說明如何解決問題。

### 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。因此若您在主控台檢視訊息的次數達到相對應佇列的再驅動政策所指定的次數，訊息便會移動至相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 的值有可能不同。

### 如需建立和設定無效字母佇列再驅動的相關資訊

請注意，無效字母佇列再驅動要求您為 Amazon SQS 設定適當的許可，以接收來自無效字母佇列的訊息，並將訊息傳送到目的地佇列。在許可不足的情況下，無效字母佇列再驅動到來源佇列不會起始訊息再驅動，而且任務可能會失敗。您可檢視訊息再驅動任務的狀態，以修復問題並重試。

主題

- 設定無效字母佇列 (主控台)
- 設定無效字母佇列再驅動
- Amazon SQS 無效字母佇列再驅動的 CloudTrail 更新和許可需求

## 設定無效字母佇列 (主控台)

無效字母佇列是一或多個來源佇列可用於未成功使用之訊息的佇列。如需詳細資訊，請參閱 Amazon SQS 無效字母佇列。

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。如需有關建立佇列做為無效字母佇列的指示，請參閱 建立佇列 (主控台)

FIFO 佇列的無效字母佇列必須也是 FIFO 佇列。同樣地，標準佇列的無效字母佇列必須也是標準佇列。

建立或編輯佇列時，可以設定無效字母佇列。

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. 選擇佇列，然後選擇編輯。
4. 捲動至無效字母佇列區段，然後選擇啟用。
5. 選擇您要與此來源佇列產生關聯的現有無效字母佇列的 Amazon Resource Name (ARN)。
6. 若要設定訊息的最大接收次數，超過此數後才可將訊息傳送到無效字母佇列，請將最大接收量設定為 1 到 1,000 之間的值。
7. 完成設定無效字母佇列後，請選擇儲存。

   儲存佇列之後，主控台會顯示佇列的詳細資料頁面。在詳細資訊頁面上，無效字母佇列索引標籤會在無效字母佇列中顯示最大接收量和無效字母佇列 ARN。

# 設定無效字母佇列再驅動

您可以設定無效字母佇列再驅動，將標準未使用的訊息從現有無效字母佇列移回其來源佇列。如需無效字母佇列的詳細資訊，請參閱 將訊息從無效字母佇列中移出。

## 為現有的標準佇列設定無效字母佇列再驅動 (API)

您可以使用以下 API 動作來設定無效字母佇列再驅動。

| 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 RUNNING 時，才能取消訊息移動。 |

## 為現有的標準佇列設定無效字母佇列再驅動 (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)。

> **Note**
>
> 自訂目的地佇列必須符合無效字母佇列的類型。例如，如果無效字母佇列是 FIFO 佇列，則自訂目的地佇列也必須是 FIFO 佇列。

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。
   > 再驅動任務會重設保留期間。新的 `messageID` 和 `enqueueTime` 被分配給再驅動的訊息。

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 `sqs:StartMessageMoveTask` 、`sqs:ReceiveMessage` 、`sqs:DeleteMessage` 和 `sqs:GetQueueAttributes` 。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 `kms:Decrypt` 需要用來加密訊息的任何 KMS 金鑰。<br>• 新增目的地佇列的 `sqs:SendMessage` 。如果目的地佇列已加密，則 `kms:GenerateDataKey` 和 `kms:Decrypt` 也是必要的。 |

| 最低許可 | 必要的 API 方法 |
| --- | --- |
| 若要取消進行中的訊息再驅動 | • 新增無效字母佇列的 sqs:CancelMessageMoveTask 、sqs:ReceiveMessage 、sqs:DeleteMessage 和 sqs:GetQueueAttributes 。如果無效字母佇列已加密 (也稱為 SSE 佇列)，kms:Decrypt 也需要。 |
| 若要顯示訊息移動狀態 | • 新增無效字母佇列的 sqs:ListMessageMoveTasks 和 sqs:GetQueueAttributes 。 |

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

請遵循下列步驟來設定無效字母佇列再驅動的最低許可：

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 建立具有下列許可的政策，並將其附加至您的登入 IAM 使用者或角色：

   - sqs:StartMessageMoveTask
   - sqs:CancelMessageMoveTask
   - sqs:ListMessageMoveTasks
   - sqs:ListDeadLetterSourceQueues
   - sqs:ReceiveMessage
   - sqs:DeleteMessage
   - sqs:GetQueueAttributes
   - 無效字母佇列的 Resource ARN (例如「arn:aws:sqs:*<DLQ_region>*:*<DLQ_accountId>*:*<DLQ_name>*」)。
   - sqs:SendMessage
   - 目的地佇列的 Resource ARN (例如，「arn: aws: 方格:*< DestQueue _ ## >: < _* 名稱 >」)。DestQueue DestQueue
   - kms:Decrypt - 允許解密動作。
   - kms:GenerateDataKey

- 已用於加密原始來源佇列中訊息的任何 KMS 加密金鑰的 Resource ARN (例如「arn:aws:kms:*<region>*:*<accountId>*:key/*<keyId_used to encrypt the message body>*」)。
- 用於再驅動目的地佇列的 KMS 加密金鑰的資源 ARN (例如「arn:aws:kms:*<region>*:*<accountId>*:key/*<keyId_used for the destination queue>*」)。

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"
      ],
      "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>"
    },
    {
"Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:<region>:<accountId>:key/<keyId>"
    }
  ]
}
```

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

使用下列步驟來設定標準未加密無效字母佇列的最低許可。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. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在導覽窗格中，選擇政策。
3. 建立具有下列許可的政策，並將其附加至您的登入 IAM 使用者或角色：

   - `sqs:StartMessageMoveTask`
   - `sqs:CancelMessageMoveTask`
   - `sqs:ListMessageMoveTasks`
   - `sqs:ReceiveMessage`
   - `sqs:DeleteMessage`
   - `sqs:GetQueueAttributes`
   - 無效字母佇列的 `Resource` ARN (例如「arn:aws:sqs:*<DLQ_region>*:*<DLQ_accountId>*:*<DLQ_name>*」)。
   - `sqs:SendMessage`
   - *###### Resource ARN (####arn: aws: ##:< DestQueue _ ## >: < _ ## >#)#DestQueue DestQueue*

   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"
      ],
```

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

# Amazon SQS 無效字母佇列再驅動的 CloudTrail 更新和許可需求

在 2023 年 6 月 8 日，Amazon SQS 推出了適用於 AWS SDK 和 AWS Command Line Interface (CLI) 的無效字母佇列 (DLQ) 再驅動。這項功能是 AWS 主控台已支援 DLQ 再驅動的補充功能。如果您先前曾使用 AWS 主控台再驅動無效字母佇列訊息，您可能會受到下列變更的影響：

- 無效字母佇列再驅動的 CloudTrail 事件重新命名
- 更新了無效字母佇列再驅動的許可

## 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」。「帳戶 A」用於在 2023 年 6 月 8 日之前在 us-east-1 中再驅動 AWS 主控台上的訊息，但不適用於 eu-west-1。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選項之一，則只能用於在 us-east-1 的 AWS 主控台上再驅動訊息，而不能用於 eu-west-1。

> **Important**
>
> 如果您的 DLQ 再驅動許可在 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。
> 但是，如果您在 2023 年 8 月期間在 AWS 主控台上使用了 DLQ 再驅動功能，則您可以延期至 2023 年 10 月 15 日，以便根據這些選項之一採用新許可。
> 如需更多詳細資訊，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

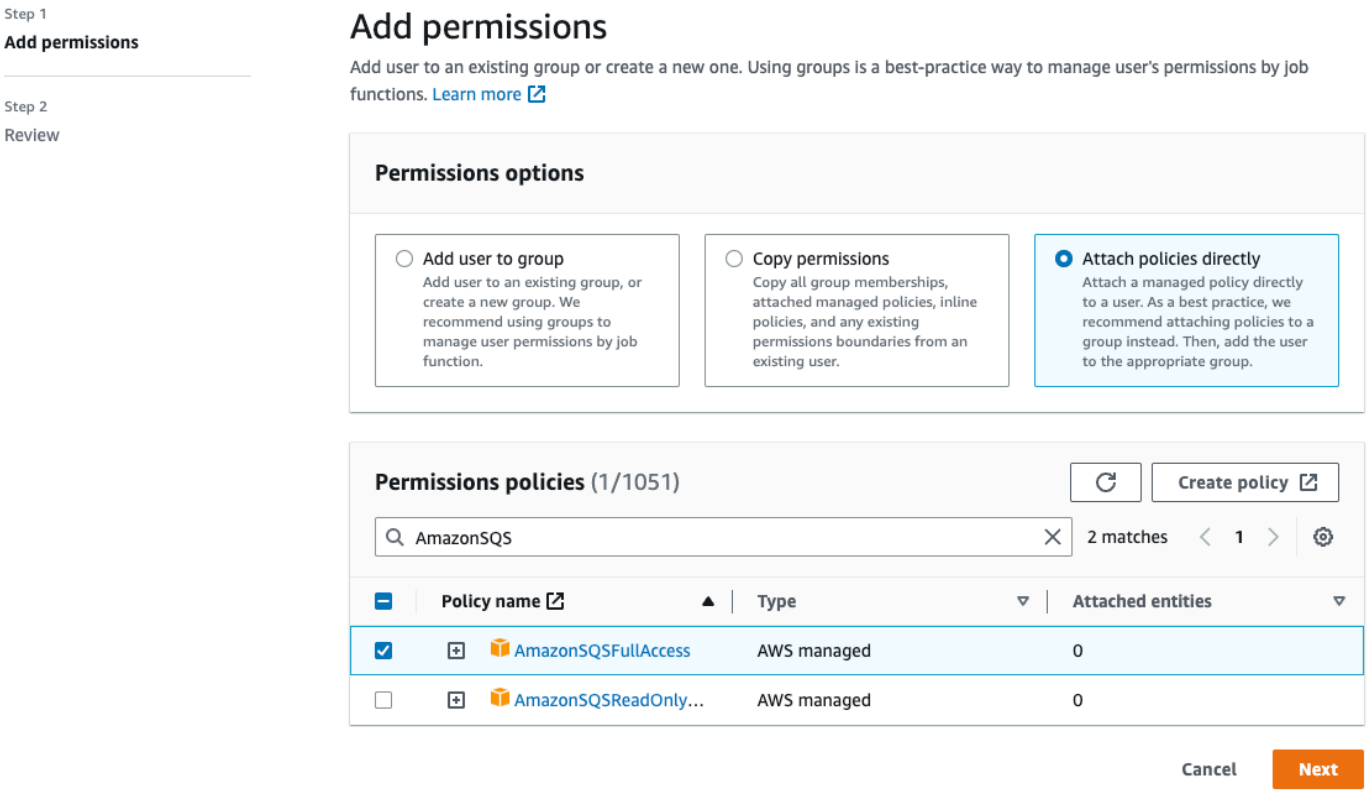
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage",
```

```
        "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。




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

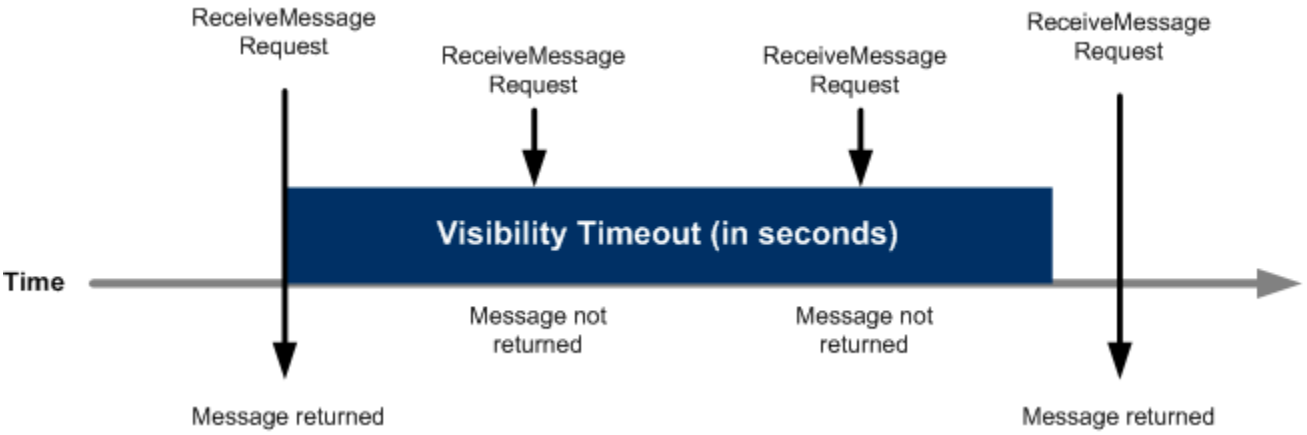
如果您使用的是客戶管理政策 (CMP)，則可以使用 AWS CloudTrail 和 IAM 來識別受佇列許可更新影響的政策。

> **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。

# Amazon SQS 可見性逾時

當消費者從佇列接收和處理訊息時，訊息會保留在佇列中。Amazon SQS 不會自動刪除訊息。由於 Amazon SQS 是分散式系統，不保證消費者會實際接收到訊息 (例如因為連線問題或消費者應用程式的問題而未接收到)。因此消費者必須在接收和處理訊息後，才可將訊息從佇列上刪除。




接收訊息後，訊息會立刻保留在佇列中。為了避免其他消費者再次處理該訊息，Amazon SQS 會設定可見性逾時，在這段期間中 Amazon SQS 會避免所有消費者接收或處理該訊息。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。如需有關使用主控台來設定佇列的可見性逾時的資訊，請參閱 設定佇列參數 (主控台)。

> **Note**
>
> 至於標準佇列，可見性逾時並不保證不會收到兩次訊息。如需更多詳細資訊，請參閱 一個t-least-once 交付。
> FIFO 佇列允許生產者或消費者嘗試多次重試：
>
> - 如果生產者偵測到失敗的 SendMessage 動作，可以使用相同的重複資料刪除 ID，視需要多次重試傳送。假設生產者在重複資料刪除間隔到期之前至少收到一個確認，多次重試既不會影響訊息的排序，也不會引入重複項目。
> - 如果消費者偵測到失敗的處理 ReceiveMessage 行動，它可以視需要使用相同的接收要求嘗試識別碼重試多次。假設消費者在可見性逾時到期前至少收到一個確認，則多次重試不會影響訊息的排序。
> - 當您收到含有訊息群組識別碼的訊息時，除非您刪除該訊息或訊息變成可見，否則不會再傳回相同訊息群組識別碼的訊息。

主題



## 傳送中訊息

Amazon SQS 訊息有三種基本狀態：

1. 由生產者傳送到佇列。
2. 消費者從佇列接收。
3. 已從佇列中刪除。

訊息被認為是在生產者傳送到佇列之後被儲存，但消費者尚未從佇列接收（也就是，在狀態 1 和 2 之間）。儲存的訊息數目沒有配額。消費者從佇列接收訊息後，即視為處於傳送中狀態，但尚未從佇列中刪除 (也就是狀態 2 和 3 之間)。傳送中訊息數量有配額。

> **Important**
>
> 套用至傳送中訊息的配額與無限數量的儲存訊息無關。

f對於大部分的標準佇列 (視佇列流量和未處理訊息而定)，傳送中訊息中最多可有約 120,000 則 (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。

對於 FIFO 佇列，最多可有 20,000 則傳送中訊息 (消費者從佇列接收，但尚未從佇列中刪除)。如果達到此配額，Amazon SQS 不會傳回任何錯誤訊息。

> **Important**
>
> 使用 FIFO 佇列時，如果在可見性逾時時段之外接收要求，`DeleteMessage` 操作將會失敗。如果可見性逾時為 0 秒，則必須在傳送的相同毫秒內刪除訊息，否則會視為放棄訊息。如果 `MaxNumberOfMessages` 參數大於 1，這可能會導致 Amazon SQS 在對 `ReceiveMessage` 操作的相同回應中包含重複訊息。如需其他詳細資訊，請參閱 Amazon SQS FIFO API 的運作方式。

## 設定可見性逾時

當 Amazon SQS 傳回訊息時，可見性逾時即會開始。在這段期間內，消費者會處理和刪除訊息。但是若消費者在刪除訊息前失敗，而您的系統未在可見性逾時到期前呼叫 `DeleteMessage` 動作，則該訊息會變成可由其他消費者看見，且會再度收到該訊息。若必須僅能接收訊息一次，您的消費者就需要在可見性逾時的期間內將之刪除。

每個 Amazon SQS 佇列都會預設 30 秒的可見性逾時設定。您可以變更整個佇列的設定。一般而言，可見性逾時應設定為您的應用程式處理及刪除佇列中的訊息所需的最長時間。收到訊息時，您可以為傳回的訊息特別設定可見性逾時，而無需變更整個佇列的逾時設定。如需詳細資訊，請參閱即時處理訊息章節中的最佳實務。

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 (例如 43,200 秒)，因為 ReceiveMessage 請求會啟動計時器。例如，如果您收到一條訊息，並透過傳送 VisibilityTimeout 等於 43,200 秒的 ChangeMessageVisibility 呼叫立即設定最大 12 小時，則可能會失敗。不過，除非透過 ReceiveMessage 請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。

## 變更訊息的可見性逾時

當您從佇列接收到訊息並開始處理時，該佇列的可見性逾時可能不夠充足 (例如您可能需要處理和刪除訊息)。可以使用 ChangeMessageVisibility 動作來指定逾時的新值，以縮短或延遲訊息的可見性。

舉例而言，若佇列的預設逾時值為 60 秒，而收到訊息後已經過 15 秒，則可以發送 ChangeMessageVisibility 呼叫，將 VisibilityTimeout 設為 10 秒，則從送出 ChangeMessageVisibility 呼叫開始會再計算 10 秒。因此若在最初變更可見性逾時 (共 25 秒) 的 10 秒後企圖變更可見性逾時獲式刪除訊息，就會導致錯誤。

> **Note**
>
> 從呼叫 ChangeMessageVisibility 動作開始，新的逾時期間便開始生效。此外，新的逾時期間僅適用於特定接收的訊息。ChangeMessageVisibility 不會影響後來接收的訊息或後續佇列的逾時。

## 結束訊息的可見性逾時

從佇列接收訊息時，您可能會發現其實並不想要處理或刪除該訊息。Amazon SQS 允許終止特訂訊息的可見性逾時。如此可讓系統中的其他元件立刻看到訊息，訊息也開放處理。

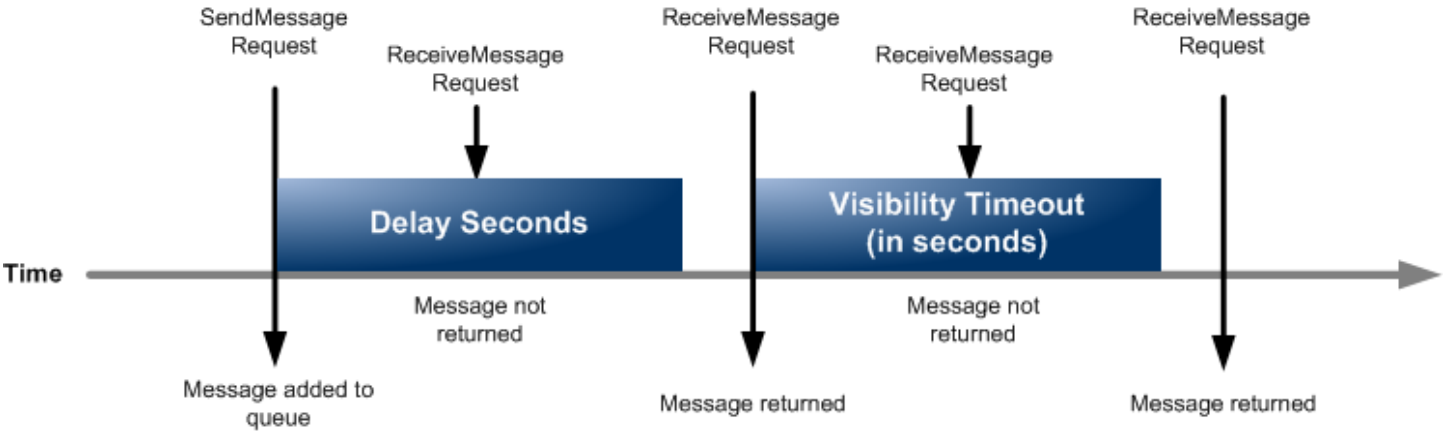
若要終止訊息在呼叫 ReceiveMessage 後的可見性逾時，請呼叫ChangeMessageVisibility並將 VisibilityTimeout 設為 0 秒。

# 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 設定佇列參數 (主控台)。

> **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。




若要設定個別訊息的延遲秒數，而不是針對整個佇列進行設定，請使用訊息計時器來允許 Amazon SQS 使用訊息佇列計時器的 `DelaySeconds` 值，而不是延遲佇列的 `DelaySeconds` 值。

# Amazon SQS 暫存佇列

使用常見的訊息模式 (例如請求-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。

用戶端會將多個暫存佇列 (即針對特定程序隨需建立的應用程式管理佇列) 自動對應到單一 Amazon SQS 佇列。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。

主題



## 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。如需最佳實務做法，請參閱「避免在虛擬佇列中重複使用相同的訊息群組 ID」。

> **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。如需更多詳細資訊，請參閱 訊息相關配額。

`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。

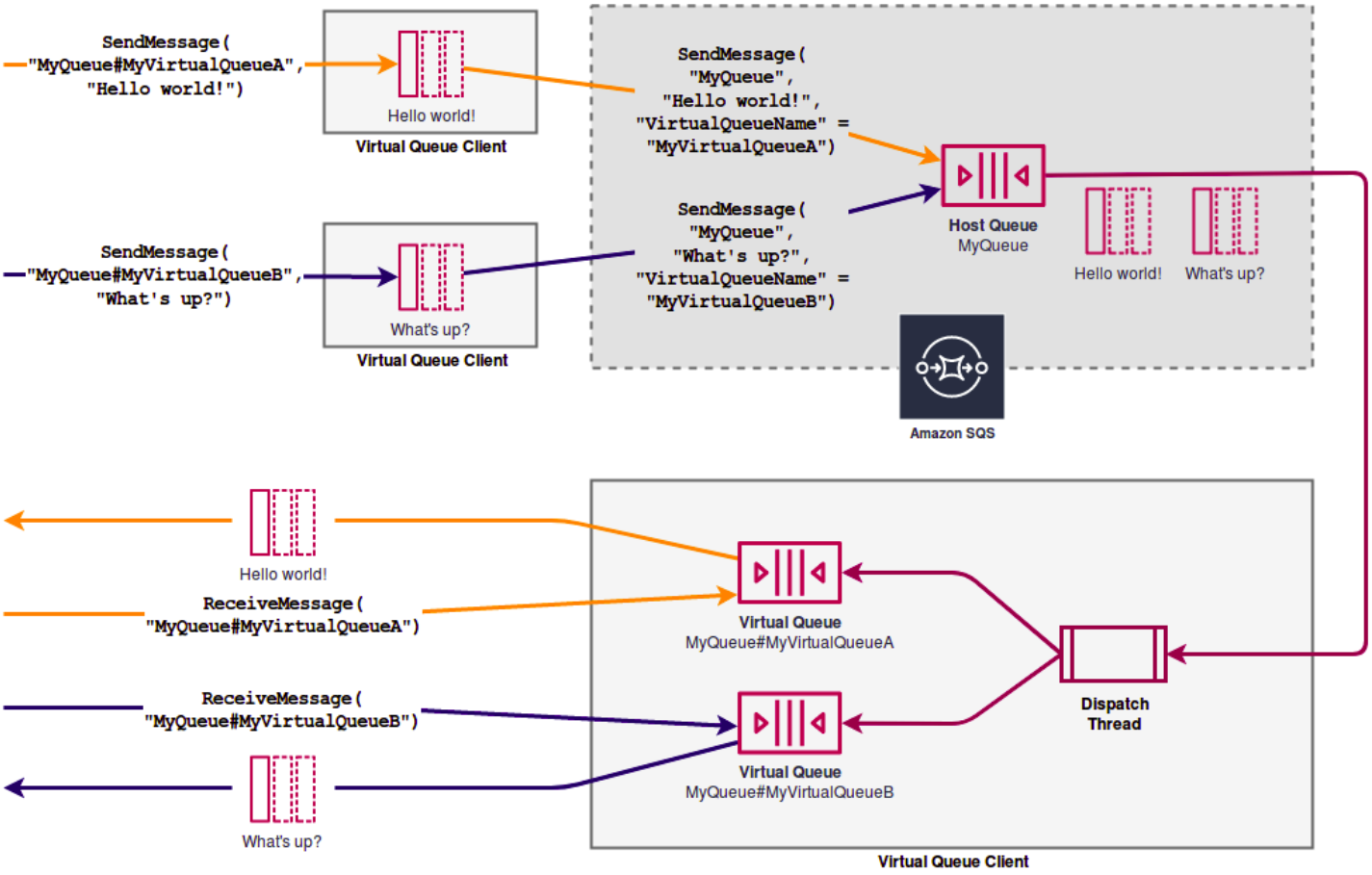
消費者呼叫虛擬佇列 URL 上的 ReceiveMessage API 動作時，暫時佇列用戶端便會在本機封鎖呼叫，直到背景執行緒傳送訊息至虛擬佇列。(此程序與緩衝處理過的非同步用戶端中的訊息預先提取相似：單一 API 動作最多可以提供訊息給 10 個虛擬佇列。) 刪除虛擬佇列時，會移除任何用戶端側資源，不會自行呼叫 Amazon SQS。

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。如需更多詳細資訊，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.sendResponseMessage(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中的佇列。

> **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

訊息計時器可為要加入佇列的訊息指定初始的隱藏期。舉例而言，若傳送一則附帶 45 秒計時器的訊息，就不會在佇列內的前 45 秒內對取用者顯示該訊息。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 傳送訊息。

> **Note**
>
> FIFO 佇列不支援個別訊息的計時器。

若要為整個佇列設定延遲期間，而非個別訊息，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。

# 透過 Amazon SQS 主控台存取亞馬遜 EventBridge 管道

Amazon EventBridge 管道將源連接到目標。管道旨在用於支持的源和目標之間的 point-to-point集成，並支持高級轉換和擴展。 EventBridge 管道提供可高度擴展的方式，將 Amazon SQS 佇列連接到諸如 Step Functions、Amazon SQS 和 API Gateway 等服AWS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 控制台以查看管道詳細信息。
- 啟動主 EventBridge 控台以建立新管道，並將佇列作為其來源。

如需將 Amazon SQS 佇列設定為管道來源的詳細資訊，請參閱 Amazon SQS 佇列做為來源，請參閱 Amazon EventBridge 使用者指南中的來源。若要取得有關一般 EventBridge 管的更多資訊，請參閱〈EventBridge 管〉。

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」頁籤。

   「EventBridge 管」頁籤包括目前規劃為使用所選佇列作為來源的所有管道的清單，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。

     這將啟動控制台的「管道詳細信息」頁 EventBridge 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 簡單儲存服務管理大型 Amazon SQS 訊息

您可以使用適用於 Java 的 Amazon SQS 擴充用戶端程式庫和適用於 Python 的 Amazon SQS 擴充用戶端程式庫來傳送大型訊息。這對於消耗 256 KB 和最大 2 GB 的大型訊息承載特別有用。兩個程式庫都會將訊息承載儲存至 Amazon 簡單儲存體服務儲存貯體，並將已存放 Amazon S3 物件的參考傳送至 Amazon SQS 佇列。

> **Note**
>
> Amazon SQS 擴充用戶端程式庫與標準佇列和 FIFO 佇列相容。

主題

- 使用 Java 和 Amazon S3 管理大型 Amazon SQS 消息
- 使用 Python 和 Amazon S3 管理大型 Amazon SQS 消息

# 使用 Java 和 Amazon S3 管理大型 Amazon SQS 消息

您可以使用適用於 Java 的 Amazon SQS 擴展客戶端庫和 Amazon Simple Storage Service (Amazon S3) 來管理大型 Amazon Simple Queue Service (Amazon SQS) 消息。這對於消耗 256 KB 和最大 2 GB 的大型訊息承載特別有用。程式庫會將訊息承載儲存至 Amazon S3 儲存貯體，並將包含已存 Amazon S3 物件參考的訊息傳送至 Amazon Amazon SQS 佇列。

您可以使用適用於 Java 的 Amazon SQS 擴充用戶端程式庫執行下列動作：

- 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。
- 擷取 S3 儲存貯體中的訊息物件
- 從 S3 儲存貯體刪除訊息物件

## 必要條件

下列範例使用 AWS Java SDK。若要安裝和設定 SDK，請參閱 AWS SDK for Java 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，請先設定您的 AWS 認證。如需其他資訊，請參閱《AWS SDK for Java 開發人員指南》中的設定開發用的 AWS 憑證和區域。

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 AWS SDK for Java 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 開發套件來執行此操作。

## AWS適用於 Java 1.x 的開發套件範例：使用 Amazon S3 管理大型 Amazon SQS 訊息

以下適用於 Java 2.x 的AWS開發套件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 MyQueue 的佇列，並傳送儲存在 S3 儲存貯體且超過 256 KB 的隨機訊息至佇列。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。

```
/*
 * Copyright 2010-2022 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 com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;

import java.util.Arrays;
```

```
import java.util.List;
import java.util.UUID;

public class SQSExtendedClientExample {

// Create an Amazon S3 bucket with a random name.
private final static String S3_BUCKET_NAME = 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 AmazonS3 s3 = AmazonS3ClientBuilder.defaultClient();

    /*
     * 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 BucketLifecycleConfiguration.Rule expirationRule =
            new BucketLifecycleConfiguration.Rule();
    expirationRule.withExpirationInDays(14).withStatus("Enabled");
    final BucketLifecycleConfiguration lifecycleConfig =
            new BucketLifecycleConfiguration().withRules(expirationRule);

    // Create the bucket and allow message objects to be stored in the bucket.
    s3.createBucket(S3_BUCKET_NAME);
    s3.setBucketLifecycleConfiguration(S3_BUCKET_NAME, lifecycleConfig);
    System.out.println("Bucket created and configured.");

    /*
     * Set the Amazon SQS extended client configuration with large payload
     * support enabled.
     */
    final ExtendedClientConfiguration extendedClientConfig =
            new ExtendedClientConfiguration()
                    .withLargePayloadSupportEnabled(s3, S3_BUCKET_NAME);

    final AmazonSQS sqsExtended =
            new AmazonSQSExtendedClient(AmazonSQSClientBuilder
```

```
                .defaultClient(), extendedClientConfig);

    /*
     * Create a long string of characters for the message object which will
     * be stored in the bucket.
     */
    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().toString();
    final CreateQueueRequest createQueueRequest =
            new CreateQueueRequest(QueueName);
    final String myQueueUrl = sqsExtended
            .createQueue(createQueueRequest).getQueueUrl();
    System.out.println("Queue created.");

    // Send the message.
    final SendMessageRequest myMessageRequest =
            new SendMessageRequest(myQueueUrl, myLongString);
    sqsExtended.sendMessage(myMessageRequest);
    System.out.println("Sent the message.");

    // Receive the message.
    final ReceiveMessageRequest receiveMessageRequest =
            new ReceiveMessageRequest(myQueueUrl);
    List<Message> messages = sqsExtended
            .receiveMessage(receiveMessageRequest).getMessages();

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

    // Delete the message, the queue, and the bucket.
    final String messageReceiptHandle = messages.get(0).getReceiptHandle();
    sqsExtended.deleteMessage(new DeleteMessageRequest(myQueueUrl,
            messageReceiptHandle));
```

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

    sqsExtended.deleteQueue(new DeleteQueueRequest(myQueueUrl));
    System.out.println("Deleted the queue.");

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

private static void deleteBucketAndAllContents(AmazonS3 client) {

    ObjectListing objectListing = client.listObjects(S3_BUCKET_NAME);

    while (true) {
        for (S3ObjectSummary objectSummary : objectListing
                .getObjectSummaries()) {
            client.deleteObject(S3_BUCKET_NAME, objectSummary.getKey());
        }

        if (objectListing.isTruncated()) {
            objectListing = client.listNextBatchOfObjects(objectListing);
        } else {
            break;
        }
    }

    final VersionListing list = client.listVersions(
            new ListVersionsRequest().withBucketName(S3_BUCKET_NAME));

    for (S3VersionSummary s : list.getVersionSummaries()) {
        client.deleteVersion(S3_BUCKET_NAME, s.getKey(), s.getVersionId());
    }

    client.deleteBucket(S3_BUCKET_NAME);
}
}
```

## AWS適用於 Java 2.x 的開發套件範例：使用 Amazon S3 管理大型 Amazon SQS 訊息

以下適用於 Java 2.x 的AWS開發套件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 MyQueue 的佇列，並傳送儲存在 S3 儲存貯體且超過 256 KB 的隨機訊息至佇列。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。

```
/*
 * Copyright 2010-2022 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 S3_BUCKET_NAME = 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(S3_BUCKET_NAME).build());

 s3.putBucketLifecycleConfiguration(PutBucketLifecycleConfigurationRequest.builder()
                .bucket(S3_BUCKET_NAME)
```

```
                .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, S3_BUCKET_NAME);

        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(S3_BUCKET_NAME).build());

        listObjectsResponse.contents().forEach(object -> {

 client.deleteObject(DeleteObjectRequest.builder().bucket(S3_BUCKET_NAME).key(object.key()).bui
        });

        ListObjectVersionsResponse listVersionsResponse =
 client.listObjectVersions(ListObjectVersionsRequest.builder().bucket(S3_BUCKET_NAME).build());

        listVersionsResponse.versions().forEach(version -> {

 client.deleteObject(DeleteObjectRequest.builder().bucket(S3_BUCKET_NAME).key(version.key()).ve
        });


 client.deleteBucket(DeleteBucketRequest.builder().bucket(S3_BUCKET_NAME).build());
    }
}
```

您可以使用阿帕奇 Maven 為您的 Java 項目配置和構建 Amazon SQS 擴展客戶端，或者自己構建 SDK。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 消息

您可以使用適用於 Python 的 Amazon 簡單佇列服務延伸用戶端程式庫和亞馬遜簡單儲存服務來管理大型 Amazon SQS 訊息。這對於消耗 256 KB 和最大 2 GB 的大型訊息承載特別有用。程式庫會將訊息承載儲存至 Amazon S3 儲存貯體，並將包含已存 Amazon S3 物件參考的訊息傳送至 Amazon Amazon SQS 佇列。

您可以使用 Python 的擴充用戶端程式庫執行下列動作：

- 指定承載是一律存放在 Amazon S3 中，還是只在承載大小超過 256 KB 時才存放在 S3
- 傳送參考存放在 Amazon S3 儲存貯體中的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

## 必要條件

以下是使用適用於 Python 的 Amazon SQS 擴充用戶端程式庫的先決條件：

- 具有必要憑據的AWS帳戶。若要建立AWS帳戶，請瀏覽至AWS首頁，然後選擇 [建立AWS帳戶]。遵循指示。如需認證的相關資訊，請參閱認證。
- 一個 AWS SDK：這個頁面上的示例使用 AWS Python SDK 博托 3。若要安裝和設定開發套件，請參閱適用於 Python 的 AWS SDK 開發人員指南中的AWS開發套件說明文件。
- Python 3.x（或更高版本）和`pip`.
- 適用於 Python 的 Amazon SQS 擴展用戶端程式庫，可從 PyPI 取得

> **Note**
>
> 您可以使用 Amazon SQS 擴充用戶端程式庫，只使用適用於 Python 的AWS開發套件管理 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開發套件來執行此操作。

## 設定訊息儲存

Amazon SQS 擴充用戶端使用下列訊息屬性來設定 Amazon S3 訊息儲存選項：

- `large_payload_support`：用於存放大型訊息的 Amazon S3 儲存貯體名稱。
- `always_through_s3`：如果是`True`，則所有訊息都存放在 Amazon S3 中。如果`False`，小於 256 KB 的訊息將不會序列化到 s3 儲存貯體。預設值為 `False`。
- `use_legacy_attribute`: 如果`True`，所有已發佈的郵件都使用舊版保留訊息屬性 (`SQSLargePayloadSize`) 而非目前保留的訊息屬性 (`ExtendedPayloadSize`)。

## 使用 Python 擴充用戶端程式庫管理大型 Amazon SQS 訊息

下列範例會使用隨機名稱建立 Amazon S3 儲存貯體。然後，它會建立名為的 Amazon SQS 佇列，MyQueue並將存放在 S3 儲存貯體且超過 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 = "S3_BUCKET_NAME"
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
```

# 設定 Amazon SQS 佇列 (主控台)

使用 Amazon SQS 主控台來設定和管理 Amazon Simple Queue Service (Amazon SQS) 佇列和功能。您也可以使用主控台來設定伺服器端加密等功能、將無效字母佇列與佇列產生關聯，或設定觸發程序來調用 AWS Lambda 函數。

主題



# Amazon SQS 的屬性型存取控制 (ABAC)

## ABAC 是什麼？

屬性型存取控制 (ABAC) 是一種授權程序，根據可連接至使用者和 AWS 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 AWS 中的標記作業，請參閱AWS標記策略和 Amazon SQS 成本分配標籤。

> **Note**
>
> Amazon SQS 的 ABAC 目前在提供 Amazon SQS 的所有 AWS 商業區域均可使用，但下列情況除外：
>
> - 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

- Amazon SQS 的 ABAC 條件索引鍵
- 為存取控制加上標籤
- 建立 IAM 使用者和 Amazon SQS 佇列
- 測試屬性型存取控制

## Amazon SQS 的 ABAC 條件索引鍵

您可以使用以下條件索引鍵來控制函數動作：

| ABAC 條件索引鍵 | 描述 | Policy type (政策類型) | Amazon SQS 操作 |
| --- | --- | --- | --- |
| AWS：ResourceTag | Amazon SQS 佇列上的標籤 (鍵和值) 與政策中的標籤 (鍵和值) 或標籤模式相符 | 僅限 IAM 政策 | Amazon SQS 佇列資源操作 |
| AWS：RequestTag | Amazon SQS 佇列資源操作上的標籤 (鍵和值) 與政策中的標籤 (鍵和值) 或標籤模式相符 | 佇列政策和 IAM 政策 | TagQueue, UntagQueue, CreateQueue |
| AWS：TagKeys | 請求中的標籤索引鍵與政策中的標籤索引鍵相符 | 佇列政策和 IAM 政策 | TagQueue, UntagQueue, CreateQueue |

## 為存取控制加上標籤

以下是如何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，請參閱具有標籤和 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 AWS CloudFormation 控制對 Amazon SQS 的存取。

## 使用 AWS Management Console

### 建立 IAM 使用者

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 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，以附加之內嵌政策和 Amazon SQS 佇列建立 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"
```

## 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，以測試使用設定為 environment 的標籤鍵和設定為 prod 的標籤值建立的佇列。如果您沒有 AWS CLI，則可以為您的機器下載並設定。

```
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 MD5OfMessageBody 和 MessageId。

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

## 設定佇列參數 (主控台)

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 開發套件。

- 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 短期和長輪詢。
- 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 佇列入門。
- 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 FIFO 佇列的高輸送量 及 訊息相關配額。
- 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 Amazon SQS 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 FIFO 佇列的高輸送量 及 訊息相關配額。
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。

# 設定存取政策 (主控台)

編輯佇列時，可以設定其存取政策。

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. 在輸入方塊中編輯存取政策陳述式。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 SQL 受管加密金鑰，為佇列設定伺服器端加密 (SSE) (主控台)

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)。

# 為佇列設定伺服器端加密 (SSE) (主控台)

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 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。

# 設定 Amazon SQS 佇列的 Amazon SQS 佇列的成本分配標記 (主控台)

為了幫助組織和識別您的 Amazon SQS 佇列，您可以為它們添加成本分配標籤。如需詳細資訊，請參閱 Amazon SQS 成本分配標籤。

在佇列的詳細資訊頁面上，標記索引標籤會顯示佇列的標籤。

建立或編輯佇列時，可以為其設定標籤。

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。

# 為 Amazon SQS 佇列訂閱 Amazon SNS 主題 (主控台)

您可以為 Amazon SQS 佇列訂閱一或多個 Amazon Simple Notification Service (Amazon SNS) 主題。當您發佈訊息到主題，Amazon SNS 會傳送訊息給每個訂閱佇列。Amazon SQS 會管理訂閱和任何必要的許可。如需 Amazon SNS 的詳細資訊，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 佇列到 SNS 主題，Amazon SNS 會使用 HTTPS 轉發訊息到 Amazon SQS。如需使用 Amazon SNS 搭配加密 Amazon SQS 佇列的詳細資訊，請參閱 設定 AWS 服務的 KMS 權限。

> **Important**
>
> Amazon SQS 支援每個存取政策最多 20 個陳述式。訂閱 Amazon SNS 主題會新增一則這樣的陳述式。超過此數量將導致主題訂閱傳送失敗。

訂閱佇列至 SNS 主題 (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 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. 從為此佇列選單指定可用的 Amazon SNS 主題選單中，為您的佇列選擇 SNS 主題。

   如果選單中未列出 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. 若要確認訂閱結果，可以發佈至主題並檢視主題傳送至佇列的訊息。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。

如果您的 Amazon SQS 佇列和 SNS 主題位於不同的 AWS 帳戶，主題擁有者必須先確認訂閱。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的確認訂閱。

如需訂閱跨區域 SNS 主題的相關資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的將 Amazon SNS 訊息傳送至位於不同區域的 Amazon SQS 佇列或 AWS Lambda 函數

# 設定佇列以觸發 AWS Lambda 函數 (主控台)

使用 AWS Lambda 函數來處理來自 Amazon SQS 佇列的訊息。Lambda 輪詢佇列並使用包含佇列訊息的事件同步調用 Lambda 函數。為讓您的函數有時間處理每個記錄批次，請將來源佇列的可見性逾時設定為您在函數上設定之逾時的至少六倍。萬一您的函數在處理前一個批次時遭到調節，額外的時間可允許 Lambda 重試。

您可以指定另一個佇列做為 Lambda 函數無法處理之訊息的無效字母佇列。

Lambda 函數可以處理來自多個佇列的項目 (每個佇列使用一個 Lambda 事件來源)。您可以使用具有多個 Lambda 函數的相同佇列。

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。

請注意以下限制：

- 您的佇列和 Lambda 函數必須位於相同的 AWS 區域。
- 使用預設金鑰 (Amazon SQS 的 AWS 受管 KMS 金鑰) 的加密佇列無法調用位於不同 AWS 帳戶 的 Lambda 函數。

如需實作 Lambda 函數的相關資訊，請參閱《AWS Lambda 開發人員指南》中的搭配使用 AWS Lambda 與 Amazon SQS。

## 必要條件

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。

# 傳送具有屬性的訊息 (主控台)

對於標準和 FIFO 佇列，您可以在訊息中加入結構化中繼資料 (例如時間戳記、地理空間資料、簽章及識別符)。如需更多詳細資訊，請參閱 Amazon SQS 訊息屬性。

若要傳送具有屬性的訊息到佇列 (主控台)

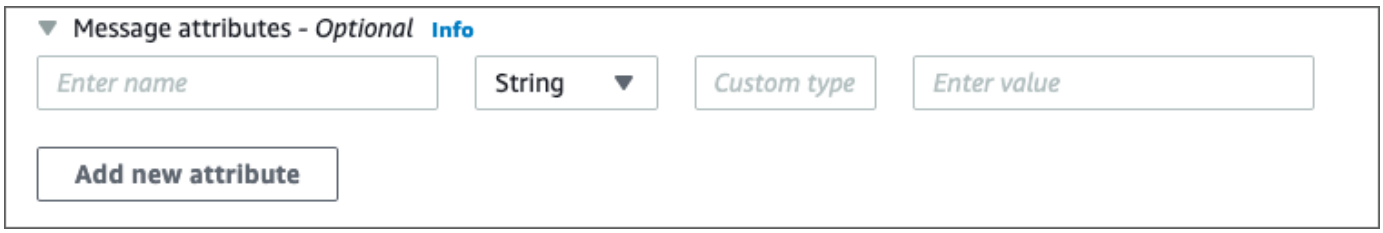
1. 在 https://console.aws.amazon.com/sqs/ 開啟 Amazon SQS 主控台。
2. 在導覽窗格中，選擇 Queues (佇列)。
3. 在佇列頁面上，選擇佇列。
4. 選擇傳送及接收訊息。
5. 定義訊息屬性參數。

   a. 在名稱文字方塊中輸入最多 256 個字元的唯一名稱。

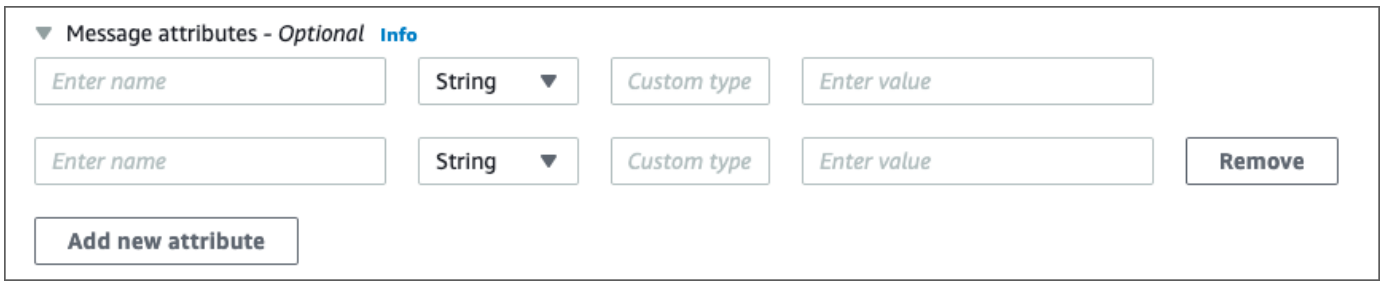
   b. 對於屬性類型，請選擇字串、數字或二進位。

   c. (選用) 輸入自訂資料類型。例如，您可以新增 **`byte`**、**`int`** 或 **`float`** 做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。




6. 若要新增另一個屬性，請選擇新增屬性。




7. 您可以在傳送訊息之前修改屬性值。
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，請選擇檢視詳細資料。選擇完成，關閉訊息詳細資料對話方塊。

# Amazon SQS 的最佳實務

這些最佳實務可協助您充分利用 Amazon SQS。

主題



## 對於 Amazon SQS 標準和 FIFO 佇列的建議

以下最佳實務可協助您使用 Amazon SQS 有效率地降低成本和處理訊息。

主題



### 處理 Amazon SQS 訊息

以下準則可協助您使用 Amazon SQS 有效率地處理訊息。

主題



#### 即時處理訊息

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。

## 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，則您已有可供使用的自動重試和退避邏輯。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 開發套件的重試和退避功能，則在未收到訊息、逾時或收到來自 Amazon SQS 的錯誤訊息後要重試 `ReceiveMessage` 動作之前，可允許先暫停一會 (例如 200 毫秒)。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)。

## 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。如需更多詳細資訊，請參閱 Amazon SQS 短期和長輪詢。

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)。若您並非使用 AWS 開發套件來存取 Amazon SQS，或您將 AWS 開發套件的等待時間設得較短，您可能需要修改 Amazon SQS 用戶端，以允許較長的請求，或是對長輪詢使用較短的等待時間。
- 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。

## 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。

## 設定無效字母佇列保留期間

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。將訊息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，最佳作法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。

### 避免訊息處理不一致

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。

### 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：

- 不要每則訊息都建立回覆佇列。而是在一開始為每個生產者建立回覆佇列，並使用相互關聯 ID 訊息屬性將回覆對應到請求。
- 不要讓您的生產者共用回覆佇列。這可能會導致生產者收到發給另一個生產者的回應訊息。

如需使用暫時佇列用戶端實作請求-回應模式的資訊，請參閱 請求-回應訊息模式 (虛擬佇列)。

## 降低 Amazon SQS 成本

以下最佳實務可協助您降低成本，並利用額外的潛在降低成本和近乎即時的回應。

### 批次訊息動作

若要降低成本，請以批次方式處理您的訊息動作：

- 若要使用單一動作來傳送、接收和刪除訊息，以及變更多個訊息的訊息可見性逾時，請使用 Amazon SQS 批次 API 動作。
- 若要結合用戶端緩衝和請求批次處理，請使用長輪詢搭配中隨附的緩衝非同步用戶端AWS SDK for Java。

> **Note**
>
> Amazon SQS 緩衝非同步用戶端目前不支援 FIFO 佇列。

### 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。

## 從 Amazon SQS 標準佇列移到 FIFO 佇列

如果您未在每則訊息設定 `DelaySeconds` 參數，則可以透過提供每則已傳送訊息的訊息群組 ID 來移到 FIFO 佇列。

如需更多詳細資訊，請參閱 從標準佇列移到 FIFO 佇列。

# 對於 Amazon SQS FIFO 佇列的其他建議

以下最佳實務可協助您以最佳方式使用訊息重複資料刪除 ID 和訊息群組 ID。如需詳細資訊，請參閱 Amazon Simple Queue Service API 參考中 `SendMessage` 和 `SendMessageBatch` 動作。

主題

- 使用 Amazon SQS 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

# 使用 Amazon SQS 訊息重複資料刪除 ID

重複資料刪除 ID 是用於已傳送訊息重複資料刪除的權杖。如果成功傳送含有特定訊息重複資料刪除識別碼的訊息，則會成功接受使用相同訊息重複資料刪除 ID 傳送的任何訊息，但在 5 分鐘的重複資料刪除間隔期間不會傳遞。

> **Note**
>
> Amazon SQS 會繼續追蹤重複資料刪除 ID，即使訊息已經被刪除也一樣。

## 提供訊息重複資料刪除 ID

生產者應該為在以下情況的每則訊息提供訊息重複資料刪除 ID 值：

- 使用 Amazon SQS 必須視為唯一的相同訊息內文傳送的訊息。
- 使用相同內容但 Amazon SQS 必須視為唯一的不同訊息屬性傳送的訊息。
- 使用 Amazon SQS 必須視為重複的不同內容 (例如，訊息內文中包含的重試計數) 傳送的訊息。

## 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，而且因為訊息內文中包含應用程式特定訊息 ID 因此訊息是唯一的，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。
- 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。

## 中斷復原案例的設計

FIFO 佇列中的重複資料刪除程序分秒必爭。在設計應用程式時，請確保生產者和消費者都能在發生用戶端或網路中斷時復原。

- 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。

### 使用可見性逾時

如需最佳效能，請將可見性逾時設為大於 AWS 軟體開發套件讀取逾時。這適用於搭配短輪詢或長輪詢使用 `ReceiveMessage` API 動作。

## 使用 Amazon SQS 訊息群組 ID

`MessageGroupId` 是指定訊息屬於特定訊息組的標籤。屬於相同訊息群組的訊息一律依相對於訊息群組的嚴格順序逐一處理 (不過屬於不同訊息群組的訊息可能會依相對於訊息群組的嚴格順序逐一處理)。

### 交錯多個排序訊息群組

若要在單一 FIFO 佇列中交錯多個排序訊息群組，請使用訊息群組 ID 值 (例如，多個使用者的工作階段資料)。在這種情況下，多位消費者可以處理佇列，但每個使用者的工作階段資料是以 FIFO 方式處理。

> **Note**
>
> 當屬於特定訊息群組 ID 的訊息未顯示，其他消費者都不能處理相同訊息群組 ID 的訊息。

### 避免在多個生產者/消費者系統中處理重複訊息

為了避免在擁有多個生產者和消費者的系統 (其傳輸量和延遲比順序重要) 中處理重複的訊息，生產者應該為每則訊息產生唯一的訊息群組 ID。

> **Note**
>
> 在這種情況下，可以消除重複訊息。不過，無法保證訊息的順序。
> 如果工作者未在可見性逾時期間內處理訊息且訊息變成可供另一個工作者使用，多個生產者和消費者的情況就會提高不小心傳遞重複訊息的風險。

### 避免發生具有相同訊息群組 ID 的大量待處理訊息

對於 FIFO 佇列，最多可有 20,000 則傳送中訊息 (消費者從佇列接收，但尚未從佇列中刪除)。如果達到此配額，Amazon SQS 不會傳回任何錯誤訊息。FIFO 佇列會查看前 20k 個訊息，以判斷可用的訊息群組。這表示，如果您在單一訊息群組中有待處理的訊息，您不能使用稍後傳送至佇列的其他訊息群組的訊息，直到您成功使用積壓的訊息為止。

> **Note**
>
> 具有相同訊息群組 ID 的待處理訊息，可能會因為消費者無法成功處理訊息而堆積。訊息處理問題可能會因為訊息內容的問題，或因為消費者的技術問題而發生。
> 若要移開無法重複處理的訊息，並取消阻止處理具有相同訊息群組 ID 的其他訊息，請考慮設定無效字母佇列政策。

### 避免在虛擬佇列中重複使用相同的訊息群組 ID

為了防止在相同主機佇列中，傳送到不同虛擬佇列而使用相同訊息群組 ID 的訊息發生彼此封鎖的情況，請避免在虛擬佇列中重複使用相同的訊息群組識別碼。

## 使用 Amazon SQS 接收請求嘗試 ID

接收請求嘗試 ID 是用於刪除 `ReceiveMessage` 呼叫重複資料的權杖。

在造成您的軟體開發套件和 Amazon SQS 之間連線問題的長期網路中斷問題中，最佳實務是提供接收請求嘗試 ID，並在軟體開發套件操作失敗時使用相同的接收請求嘗試 ID 來重試。

# Amazon SQS Java 開發套件範例

您可以使用 AWS SDK for Java 建置與 Amazon Simple Queue Service (Amazon SQS) 和其他 AWS 服務互動的 Java 應用程式。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 Amazon SQS 佇列操作的範例，例如如何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。

本主題中的範例示範其他 Amazon SQS 功能，例如伺服器端加密 (SSE)、成本分配標籤和訊息屬性。

主題



## 使用伺服器端加密 (SSE)

您可以使用 AWS SDK for Java 將伺服器端加密 (SSE) 新增到 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 金鑰之角色的詳細資訊，請參閱 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，請確定您有設定您的 AWS 憑證。如需其他資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的設定開發用的 AWS 憑證和區域。

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

// Get the URL of your queue.
String myQueueName = "my queue";
```

```
GetQueueUrlResponse getQueueUrlResponse =

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

// 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");

// Create the SetQueueAttributesRequest.
SetQueueAttributesRequest set_attrs_request = SetQueueAttributesRequest.builder()
          .queueUrl(queueUrl)
          .attributes(attributes)
          .build();

sqsClient.setQueueAttributes(set_attrs_request);
```

## 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，請確定您有設定您的 AWS 憑證。如需其他資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的設定開發用的 AWS 憑證和區域。

```
// 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。

## 為佇列設定標籤

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 AWS SDK for Java 設定標籤。如需更多詳細資訊，請參閱 Amazon SQS 成本分配標籤。

執行範例程式碼之前，請確定您有設定您的 AWS 憑證。如需其他資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的設定開發用的 AWS 憑證和區域。

### 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);
```

## 傳送訊息屬性

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、簽章和識別符)。如需更多詳細資訊，請參閱 Amazon SQS 訊息屬性。

執行範例程式碼之前，請確定您有設定您的 AWS 憑證。如需其他資訊，請參閱《AWS SDK for Java 2.x 開發人員指南》中的設定開發用的 AWS 憑證和區域。

### 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及訊息屬性資料類型。

AWS SDK for Java 會自動計算訊息內文和訊息屬性檢查總和，並與 Amazon SQS 傳回的資料做比較。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 230.00000000000000001 的值。

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

### 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。

# 使用 JMS 和 Amazon SQS

Amazon SQS Java 訊息程式庫是適用於 Amazon SQS 的 Java 訊息服務 (JMS) 界面，可讓您在已經使用 JMS 的應用程式中充分運用 Amazon SQS。此界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 AWS SDK for Java，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 支援的 JMS 1.1 實作 以及 Amazon SQS 常見問答集。

主題



## 必要條件

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 使用 JMS 搭配 Amazon SQS 標準佇列實際可行的 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

  使用適用於 Amazon SQS 的 AWS Management Console、`CreateQueue` API，或 Amazon SQS Java 訊息程式庫內的包裝 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 Amazon SQS Java 訊息程式庫入門。

# Amazon SQS Java 訊息程式庫入門

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

1. 請建立連現工廠並對工廠呼叫 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 JMSException，讓如此可更輕鬆地由預期 JMSException 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. 爪哇 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` 屬性的詳細資訊，請參閱恰好一次的處理。

## 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。如需更多詳細資訊，請參閱 重要用語。

     ```
     // 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。如需更多詳細資訊，請參閱 重要用語。

   ```
   // 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、使用 JMS 搭配 Amazon SQS 標準佇列實際可行的 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. 爪哇 和 SyncMessageReceiver. 爪哇。

## 異步接收訊息

在 Amazon SQS Java 訊息程式庫入門 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 (JMSException 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 Amazon SQS Java 訊息程式庫入門 範例相同。如需異步消費者的完整資訊，請參閱 AsyncMessageReceiver.java 的 使用 JMS 搭配 Amazon SQS 標準佇列實際可行的 Java 範例。

此範例的輸出結果類似如下：

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

## 使用用戶端認可模式

Amazon SQS Java 訊息程式庫入門 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 Amazon SQS Java 訊息程式庫入門 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 使用 JMS 搭配 Amazon SQS 標準佇列實際可行的 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。如需更多詳細資訊，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
```

# 使用 Amazon SQS Java 訊息服務 (JMS) 用戶端搭配其他 Amazon SQS 用戶端

使用 Amazon SQS Java 訊息服務 (JMS) 用戶端搭配 AWS 開發套件，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。如需更多詳細資訊，請參閱 使用 Java 和 Amazon S3 管理大型 Amazon SQS 消息。

以下 Java 範本程式碼會建立擴充用戶端程式庫的 JMS 提供者：

```
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();
```

# 使用 JMS 搭配 Amazon SQS 標準佇列實際可行的 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)。如需有關使用 FIFO 佇列的詳細資訊，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。)

## ExampleConfiguration. 爪哇

下列 Java SDK v 1.x 程式碼範例會設定預設佇列名稱、區域以及與其他 Java 範例搭配使用的認證。

```
/*
 * Copyright 2010-2022 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. 爪哇

下列 Java 程式碼範例會先建立文字訊息生產者。

```
/*
 * Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class TextMessageSender {
    public static void main(String args[]) throws JMSException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("TextMessageSender", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
```

```
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer =
 session.createProducer( session.createQueue( config.getQueueName() ) );

        sendMessages(session, producer);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static void sendMessages( Session session, MessageProducer producer ) {
        BufferedReader inputReader = new BufferedReader(
            new InputStreamReader( System.in, Charset.defaultCharset() ) );

        try {
            String input;
            while( true ) {
                System.out.print( "Enter message to send (leave empty to exit): " );
                input = inputReader.readLine();
                if( input == null || input.equals("" ) ) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
                System.out.println( "Send message " + message.getJMSMessageID() );
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
            System.err.println( "Failed reading input: " + e.getMessage() );
        } catch (JMSException e) {
            System.err.println( "Failed sending message: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

## SyncMessageReceiver. 爪哇

下列 Java 程式碼範例會建立同步訊息消費者。

```
/*
 * Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SyncMessageReceiver {
public static void main(String args[]) throws JMSException {
    ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

    ExampleCommon.setupLogging();

    // Create the connection factory based on the config
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                    .withRegion(config.getRegion().getName())
                    .withCredentials(config.getCredentialsProvider())
            );

    // Create the connection
    SQSConnection connection = connectionFactory.createConnection();

    // Create the queue if needed
    ExampleCommon.ensureQueueExists(connection, config.getQueueName());

    // Create the session
    Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
    MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

    connection.start();
```

```
    receiveMessages(session, consumer);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println( "Connection closed" );
}

private static void receiveMessages( Session session, MessageConsumer consumer ) {
    try {
        while( true ) {
            System.out.println( "Waiting for messages");
            // Wait 1 minute for a message
            Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        }
    } catch (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
```

## AsyncMessageReceiver. 爪哇

下列 Java 程式碼範例會建立異步訊息消費者。

```
/*
 * Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
```

```
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class AsyncMessageReceiver {
    public static void main(String args[]) throws JMSException, InterruptedException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

        // No messages are processed until this is called
        connection.start();

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
```

```
    }


    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();

        public void waitForOneMinuteOfSilence() throws InterruptedException {
            for(;;) {
                long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
                long remainingTillOneMinuteOfSilence =
                    TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
                if( remainingTillOneMinuteOfSilence < 0 ) {
                    break;
                }
                TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
            }
        }


        @Override
        public void onMessage(Message message) {
            try {
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message " +
 message.getJMSMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSException e) {
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}
```

## SyncMessageReceiverClientAcknowledge. 爪哇

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 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
```

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

/**
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also
 acknowledged.
 *
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

        // Setup logging for the example
```

```
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with client acknowledge mode
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));
```

```
        // Attempt to receive another message and acknowledge it. This results in
receiving no messages since
        // we have acknowledged the second message. Although we didn't explicitly
acknowledge the first message,
        // in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the
explicitly acknowledged message
        // are also acknowledged. Therefore, we have implicitly acknowledged the first
message.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
(TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
"Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSException {
        // Receive a message
```

```
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

## SyncMessageReceiverUnorderedAcknowledge. 爪哇

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 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for
 received messages. This example
 * complements the example given in {@link SyncMessageReceiverClientAcknowledge} for
 CLIENT_ACKNOWLEDGE mode.
 *
```

```
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that the first message received in the
 prior attempt is returned again
 * for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
 explicitly acknowledged no matter what
 * the order they're received.
 *
 * This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverClientAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with unordered acknowledge mode
```

```
        Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving the first message since
        // we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE
mode, all the messages must
        // be explicitly acknowledged.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
```

```
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
 messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
 "Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

## SpringExampleConfiguration.xml

以下 XML 程式碼範例是 SpringExample. 爪哇 的 Bean 組態檔案。

```
<!--
    Copyright 2010-2022 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. 爪哇

以下 Java 程式碼範例會使用 Bean 組態檔案來初始化物件。

```
/*
 * Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SpringExample {
    public static void main(String args[]) throws JMSException {
        if( args.length != 1 || !args[0].endsWith(".xml")) {
            System.err.println( "Usage: " + SpringExample.class.getName() + " <spring
 config.xml>" );
            System.exit(1);
```

```
        }

        File springFile = new File( args[0] );
        if( !springFile.exists() || !springFile.canRead() ) {
            System.err.println( "File " + args[0] + " doesn't exist or isn't
readable.");
            System.exit(2);
        }

        ExampleCommon.setupLogging();

        FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext( "file://" +
springFile.getAbsolutePath() );

        Connection connection;
        try {
            connection = context.getBean(Connection.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the JMS connection to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        String queueName;
        try {
            queueName = context.getBean("QueueName", String.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the name of the queue to use: " +
e.getMessage() );
            System.exit(3);
            return;
        }

        if( connection instanceof SQSConnection ) {
            ExampleCommon.ensureQueueExists( (SQSConnection) connection, queueName );
        }

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
session.createConsumer( session.createQueue( queueName) );
```

```
        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                    break;
                }
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message" );
            }
        } catch (JMSException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

## ExampleCommon. 爪哇

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 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
```

```
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is
 run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName)
 throws JMSException {
        AmazonSQSMessagingClientWrapper client =
 connection.getWrappedAmazonSQSClient();

        /**
         * In most cases, you can do this with just a createQueue call, but
 GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
 queue
         * already exists. Also many users and roles have permission to call
 GetQueueUrl
         * but don't have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: ");
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ){
```

```
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up to 256k so
 that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
            System.out.println( "\t" +  Base64.encodeAsString( bytes ) );
        } else if( message instanceof ObjectMessage ) {
            ObjectMessage objMessage = (ObjectMessage) message;
            System.out.println( "\t" + objMessage.getObject() );
        }
    }
}
```

# 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 訊息程式庫的支援功能的詳細資訊，請參閱 Amazon SQS 常見問答集。

## 支援的常用界面

- 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

本節提供教學，供您用來探索 Amazon SQS 特性和功能。

主題

- 建立 Amazon SQS 佇列 (AWS CloudFormation)
- 教學課程：從 Amazon Virtual Private Cloud 將訊息傳送到 Amazon SQS 佇列

## 建立 Amazon SQS 佇列 (AWS CloudFormation)

您可以使用 AWS CloudFormation 主控台和 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》中的使用 AWS CloudFormation 範本和 AWS::SQS::Queue 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詳細資訊，請參閱恰好一次的處理。

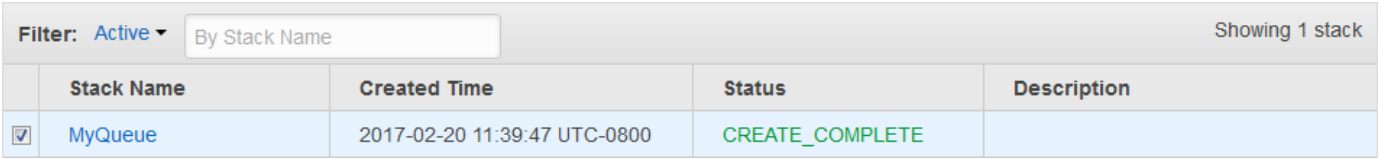
   > **Note**
   >
   > FIFO 佇列名稱結尾必須是 .fifo 尾碼。

```
{
   "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 ，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 會開始建立 `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。



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 將訊息傳送到 Amazon SQS 佇列

在此教學課程中，您將了解如何透過安全的私有網路，將訊息傳送到 Amazon SQS 佇列。這個網路包含一個 VPC，其中包含 Amazon EC2 執行個體。執行個體透過界面 VPC 端點連接到 Amazon SQS，讓您可以連接到 Amazon EC2 執行個體並將訊息傳送到 Amazon SQS 佇列，即使網路與公用網際網路中斷連線也是如此。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。
> - 若您設定 Amazon SQS 從 Amazon VPC 傳送訊息，則必須啟用私有 DNS 並使用 `sqs.us-east-2.amazonaws.com` 格式指定端點。
> - 私有 DNS 不支援 `queue.amazonaws.com` 或 `us-east-2.queue.amazonaws.com` 之類的延遲端點

主題



## 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AWS CloudFormation範本，範本是建立包含AWS資源 (例如 Amazon EC2 執行個體和 Amazon SQS 佇列) 的堆疊的藍圖。

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 隊列

1. 從下載名為的AWS CloudFormation範本`SQS-VPCE-Tutorial-CloudFormation.yaml` GitHub。
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 Stack name (堆疊名稱) 輸入 `SQS-VPCE-Tutorial-Stack`。

b. 對於 KeyName，請選擇 SQS- VPN-教學課程金鑰配對。

c. 選擇下一步。

6. 在 Options (選項) 頁面上，選擇 Next (下一步)。

7. 在 [檢閱] 頁面的 [功能] 區段中，選擇 [我確認AWSCloudFormation 可能會使用自訂名稱建立 IAM 資源]。 ，然後選擇 [建立]。

AWS CloudFormation 會開始建立堆疊並顯示 CREATE_IN_PROGRESS (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

AWS CloudFormation 範本會在 VPC 中啟動名為 `SQS-VPCE-Tutorial-EC2-Instance` 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. 複製在 Public DNS (IPv4) (公有 DNS (IPv4)) 下的主機名稱，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，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至 AWS 網路內的佇列，而不超過公有網際網路。

> **Note**
>
> 該 EC2 執行個體仍無法存取網際網路上其他 AWS 服務和端點。

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 Create Endpoint (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**
>
> 該服務名稱取決於目前 AWS 區域。例如，如果您位於美國東部 (俄亥俄州)，則服務名稱為 com.amazonaws。 ***##-# -2***. 平方米。

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：將消息發送到您的 Amazon SQS 隊列

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"
}
```

如需從 AWS CloudFormation 範本 (例如，VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK) 建立的佇列中接收和刪除訊息的資訊，請參閱 接收和刪除訊息 (主控台)。

如需有關刪除資源的資訊，請參閱下列各項：

- 在 Amazon VPC 使用者指南中刪除虛擬私人雲端端點

- 刪除佇列
- 在 Amazon EC2 執行個體使用者指南中終止您的執行個體
- 在 Amazon VPC 用戶指南中刪除您的 VPC
- 在AWS CloudFormation使用者指南中刪除AWS CloudFormation主控台上的堆疊
- 《適用於 Linux 執行個體的 Amazon EC2 使用者指南》中的刪除金鑰對

# 自動化和疑難排解 Amazon SQS 佇列

本節提供有關自動化和疑難排解 Amazon SQS 佇列的資訊。

主題



## 使用 Amazon EventBridge 自動化從 AWS 服務到 Amazon SQS 的通知

Amazon EventBridge 可讓您自動化 AWS 服務以及回應系統事件 (例如應用程式可用性的問題或資源的變動)。AWS 服務的事件會以接近即時的方式傳送到 EventBridge。您可編寫簡單的規則，來指示您在意的事件，以及當事件符合規則時所要自動執行的動作。

EventBridge 接可讓您設定各種目標—例如 Amazon SQS 標準和 FIFO 佇列—這些目標會以 JSON 格式接收事件。如需詳細資訊，請參閱 Amazon EventBridge 使用者指南中的 Amazon EventBridge 目標。

## 使用 AWS X-Ray 對 Amazon Simple Queue Service 佇列進行疑難排解

AWS X-Ray 收集有關您的應用程式服務所請求的資料，並讓您檢視和篩選資料以識別潛在問題和進行優化的機會。您可以查看任何應用程式追蹤請求的詳細資訊；包含請求和回應，以及對下游 AWS 資源、微服務、資料庫和 HTTP Web API 的呼叫等相關資訊。

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作業：

- 使用 `X-Amzn-Trace-Id` 追蹤標頭。
- 使用 `AWSTraceHeader` 訊息系統屬性。

若要收集有關錯誤和延遲的資料，您必須使用 AWS X-Ray SDK 來檢測 `AmazonSQS` 用戶端。

您可以使用 AWS X-Ray 主控台以檢視 Amazon SQS 和應用程式所使用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。如需更多詳細資訊，請參閱《AWS X-Ray 開發人員指南》中的Amazon SNS 與 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



## 資料保護

AWS 共同責任模型適用於 Amazon 簡單佇列服務中的資料保護。如此模型中所述， AWS 負責保護執行所有 AWS 雲端. 您負責維護在此基礎設施上託管內容的控制權。您也必須負責您所使用 AWS 服務的安全組態和管理任務。如需資料隱私權的相關資訊，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，我們建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 登入資料並設定個別使用者。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 資源進行通訊。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。
- 使用 AWS 加密解決方案以及其中的所有默認安全控制 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。
- 如果您在透過命令列介面或 API 存取時需要經 AWS 過 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需 FIPS 和 FIPS 端點的相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

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。這包括當您使用主控台、API 或 AWS 開發套件 AWS 服務 使用 Amazon SQS 或

其他方式時。 AWS CLI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。

主題



# 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。您可以要求 Amazon SQS 先加密訊息，然後再將訊息儲存到其資料中心的加密檔案系統，藉此保護靜態資料。Amazon SQS 建議使用 SSE 進行最佳化的資料加密。

主題



## 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 使用 SQL 管理的加密金鑰 (SSE-SQS) 或在 (SSE-KMS) 中管理的金鑰來保護佇列中的訊息內容。 AWS Key Management Service 如需有關使用管理 SSE 的詳細資訊 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 AWS SDK for Java (和`CreateQueue`、`SetQueueAttributes`和`GetQueueAttributes`動作) 管理 SSE 的相關資訊，請參閱下列範例：

- 使用伺服器端加密 (SSE)
- 設定的 KMS 權限 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) 的加密佇列無法叫用不同 AWS 帳戶的 Lambda 函數。
> 可以使用 AWS Security Token Service `AssumeRole`動作將通知傳送至 Amazon SQS 的某些 AWS 服務功能與 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 權限 與您的服務文件。

AWS KMS 結合安全、高可用性的硬體和軟體，提供專為雲端擴充的金鑰管理系統。搭配使用 Amazon SQS 時 AWS KMS，加密訊息資料的資料金鑰也會加密，並與其保護的資料一起存放。

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，每個帳戶和區域都是唯一的。
- AWS KMS 安全標準可協助您符合與加密相關的合規性要求。

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？。

主題

- 加密範圍
- 重要用語

### 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 Encryption SDK 開發人員指南中的AWS Key Management Service 開發人員指南中的資料索引鍵。

資料金鑰重複使用期間

Amazon SQS 可以重複使用資料金鑰加密或解密訊息的時間長度 (以秒為單位)， AWS KMS 然後再次呼叫。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**
>
> 在不太可能無法連線的情況下 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、金鑰識別碼或金鑰 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 會使用適用於 Amazon SQS 的 AWS 受管 KMS 金鑰。
> - 第一次使用為佇列指 AWS Management Console 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， AWS KMS 會為 Amazon SQS 建立 AWS 受管 KMS 金鑰。

您可以建立 KMS 金鑰、定義控制 KMS 金鑰使用方式的原則，以及使用主控 AWS KMS 台的 [客戶受管金鑰] 區段或`CreateKey` AWS KMS 動作稽核 KMS 金鑰使用情況。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KeyId中的。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 估算成 AWS KMS 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 使用 KMS 金鑰為資料金鑰加密，然後使用加密的訊息存放加密的資料金鑰。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。

## 金鑰管理

Amazon SQS 與 AWS Key Management Service (KMS) 整合以管理用於伺服器端加密 (SSE) 的 KMS 金鑰。如需 SSE 資訊和金鑰管理定義，請參閱 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。

主題

- 設定 AWS KMS 許可
- 了解資料金鑰重複使用期間
- 估算成 AWS KMS 本
- AWS KMS 錯誤

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 受 AWS 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。如需有關 AWS KMS 權限的詳細資訊，請參閱AWS Key Management Service 開發人員指南中的AWS KMS 資源和作業或 AWS KMS API 權限參考資料。

或者，您也可以在指派給主體 (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**
>
> 雖然您可以設定傳送到 Amazon SQS 和從 Amazon SQS 接收的全域許可，但 AWS KMS 需要在 IAM 政策的一`Resource`節中明確命名特定區域的 KMS 金鑰完整 ARN。

### 設定 AWS 服務的 KMS 權限

數個 AWS 服務充當事件來源，可將事件傳送至 Amazon SQS 佇列。若要允許這些事件來源與加密佇列搭配使用，您必須建立客戶管理的 KMS 金鑰，並在金鑰原則中新增權限，讓服務使用必要的 AWS KMS API 方法。執行下列步驟來設定許可。

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:GenerateDataKey`和 `kms:Decrypt` API 方法，請將下列陳述式新增至 KMS 金鑰原則。

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": "service.amazonaws.com"
         },
         "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
         ],
         "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 活動 | `events.amazonaws.com` |
| Amazon S3 事件通知 | `s3.amazonaws.com` |

| 事件來源 | 服務名稱 |
| --- | --- |
| 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 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:GenerateDataKey 和 kms:Decrypt。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:GenerateDataKey 和 kms:Decrypt 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:GenerateDataKey",
            "kms:Decrypt"
         ],
         "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"
      }]
}
```

### 設定消費者的 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來設定 KMS 許可權

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 活動
- 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。

- 較短的重複使用期可提供更好的安全性，但會導致更多通話 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，下次呼叫SendMessage或SendMessageBatch通常會觸發呼叫方 AWS KMS GenerateDataKey法，以取得新的資料金鑰。此外，下一次呼叫SendMessage和都ReceiveMessage會觸發呼叫，以 AWS KMS Decrypt便在使用資料金鑰之前驗證資料金鑰的完整性。
- 主體 (AWS 帳戶 或使用者) 不會共用資料索引鍵 (由唯一主體傳送的訊息永遠會取得唯一的資料金鑰)。因此，資料金鑰重複使用期間內所使用之唯一主體數目的倍數： AWS KMS

### 估算成 AWS KMS 本

為了預測成本並更好地瞭解 AWS 帳單，您可能想知道 Amazon SQS 使用 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 個主體和 1 個佇列的 AWS KMS API 呼叫次數

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：計算多個生產者和取用者以及 2 個佇列的 AWS KMS API 呼叫數

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 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

## 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。

主題

- 適用於 Amazon SQS 的 Amazon Virtual Private Cloud 端點
- 為 Amazon SQS 建立 VPC 端點政策

## 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 和 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 將訊息傳送到 Amazon SQS 佇列 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。
> - 若您設定 Amazon SQS 從 Amazon VPC 傳送訊息，則必須啟用私有 DNS 並使用 `sqs.us-east-2.amazonaws.com` 格式指定端點。
> - 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，讓管理員能夠安全地控制對 AWS 資源的存取權。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 是一種您可以免費使用的 AWS 服務。

## 物件

AWS Identity and Access Management (IAM) 的使用方式會不同，取決於您在 Amazon SQS 中所執行的工作。

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。瞭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

身分驗證是使用身分憑證登入 AWS 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者身分，或擔任 IAM 角色進行 驗證 (登入至 AWS)。

您可以使用透過身分來源 AWS IAM Identity Center 提供的憑證，以聯合身分登入 AWS。(IAM Identity Center) 使用者、貴公司的單一登入身分驗證和您的 Google 或 Facebook 憑證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。您 AWS 藉由使用聯合進行存取時，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。如需有關登入至 AWS 的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

如果您是以程式設計的方式存取 AWS，AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以便使用您的憑證透過密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，您必須自行簽署請求。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》中的簽署 AWS API 請求。

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) 以提高帳戶的安全。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》中的在 AWS 中使用多重要素驗證 (MFA)。

### AWS 帳戶 根使用者

如果是建立 AWS 帳戶，您會先有一個登入身分，可以完整存取帳戶中所有 AWS 服務 與資源。此身分稱為 AWS 帳戶 根使用者，使用建立帳戶時所使用的電子郵件地址和密碼即可登入並存取。強烈建議您不要以根使用者處理日常作業。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)搭配身分提供者使用聯合功能，使用暫時憑證來存取 AWS 服務。

聯合身分是來自您企業使用者目錄的使用者、Web 身分供應商、AWS Directory Service、Identity Center 目錄或透過身分來源提供的憑證來存取 AWS 服務 的任何使用者。聯合身分存取 AWS 帳戶 時，會擔任角色，並由角色提供暫時憑證。

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，也可以連線並同步到自己身分來源中的一組使用者和群組，以便在您的所有 AWS 帳戶和應用程式中使用。如需 IAM Identity Center 的相關資訊，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 使用者是您 AWS 帳戶 中的一種身分，具備單一人員或應用程式的特定許可。建議您盡可能依賴暫時憑證，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。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#rotate-credentials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。您無法以群組身分登入。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過程變得更為容易。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，但角色僅提供暫時憑證。如需進一步了解，請參閱《IAM 使用者指南》中的建立 IAM 使用者 (而非角色) 的時機。

## IAM 角色

IAM 角色是您 AWS 帳戶 中的一種身分，具備特定許可。它類似 IAM 使用者，但不與特定的人員相關聯。您可以在 AWS Management Console 中透過切換角色來暫時取得 IAM 角色。您可以透過呼叫 AWS CLI 或 AWS API 操作，或是使用自訂 URL 來取得角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色。

使用暫時憑證的 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，並取得由角色定義的許可。如需有關聯合角色的詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-idp.html中的為第三方身分供應商建立角色。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。
- 跨帳户存取權 – 您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶中的資源。角色是授予跨帳户存取權的主要方式。但是，針對某些 AWS 服務，您可以將政策直接連接到資源 (而非使用角色作為代理)。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 角色與資源類型政策的差異。
- 跨服務存取 – 有些 AWS 服務 會使用其他 AWS 服務 中的功能。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。
  - 轉發存取工作階段 (FAS)：當您使用 IAM 使用者或角色在 AWS 中執行動作時，系統會將您視為主體。當您使用某些服務時，您可能會執行一個動作，而該動作之後會在不同的服務中啟動另一個動作。FAS 使用主體的許可呼叫 AWS 服務，搭配請求 AWS 服務 以向下游服務發出請求。只有在服務收到需要與其他 AWS 服務 或資源互動才能完成的請求之後，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給 AWS 服務 服務。
  - 服務連結角色 – 服務連結角色是一種連結到 AWS 服務 的服務角色類型。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 AWS 帳戶 中，並由該服務所擁有。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 – 針對在 EC2 執行個體上執行並提出 AWS CLI 和 AWS API 請求的應用程式，您可以使用 IAM 角色來管理暫時憑證。這是在 EC2 執行個體內儲存存取金鑰的較好方式。如需指派 AWS 角色給 EC2 執行個體並提供其所有應用程式使用，您可以建立連接到執行個體的執行個體設定檔。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暫時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱《IAM 使用者指南》中的建立 IAM 角色 (而非使用者) 的時機。

# 使用政策管理存取權

您可以透過建立政策並將其附加到 AWS 身分或資源，在 AWS 中控制存取。政策是 AWS 中的一個物件，當其和身分或資源建立關聯時，便可定義其許可。AWS 會在主體 (使用者、根使用者或角色工作階段) 發出請求時評估這些政策。政策中的許可，決定是否允許或拒絕請求。大部分政策以 JSON 文件形式儲存在 AWS 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哪些內容。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。若要授與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。具備該政策的使用者便可以從 AWS Management Console、AWS CLI 或 AWS API 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。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

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。受管政策則是獨立的政策，您可以將這些政策附加到 AWS 帳戶 中的多個使用者、群組和角色。受管政策包含 AWS 管理政策和客戶管理政策。如需瞭解如何在受管政策及內嵌政策間選擇，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。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。主體可以包括帳戶、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) 可控制哪些委託人 (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 Simple Storage Service (Amazon S3)、AWS WAF 和 Amazon VPC 是支援 ACL 的服務範例。若要進一步了解 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 支援其他較少見的政策類型。這些政策類型可設定較常見政策類型授與您的最大許可。

- 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。如需許可範圍的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可範圍。
- 服務控制政策 (SCP) – SCP 是 JSON 政策，可指定 AWS Organizations 中組織或組織單位 (OU) 的最大許可。AWS Organizations 服務可用來分組和集中管理您企業所擁有的多個 AWS 帳戶。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 AWS 帳戶根使用者。如需組織和 SCP 的更多相關資訊，請參閱《AWS Organizations 使用者指南》中的 SCP 運作方式。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合使用者的暫時工作階段時，作為參數傳遞。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。如需更多資訊，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。如需瞭解 AWS 在涉及多種政策類型時如何判斷是否允許一項請求，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

每項 AWS 資源均由某個 AWS 帳戶 帳戶所持有，而建立或存取資源的許可則由許可政策管理。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。

主題

- Amazon Simple Queue Service 資源和操作
- 了解資源所有權
- 管理資源存取
- 指定政策元素：動作、效果、資源和主體

## 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 | `arn:aws:sqs:` *`region`*`:`*`account_id`*`:`*`queue_name`* |

以下是佇列的 ARN 格式範例：

- 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

AWS 帳戶 擁有在帳戶內所建立的資源，無論資源的建立者是誰。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 AWS 帳戶 的根帳戶憑證來建立 Amazon SQS 佇列，則您的 AWS 帳戶 即為資源擁有者 (在 Amazon SQS 中，資源為 Amazon SQS 佇列)。
- 如果您在 AWS 帳戶 中建立使用者，並授與使用者建立佇列的許可，使用者即可建立佇列。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。
- 如果您在自己的 AWS 帳戶 中建立 IAM 角色，且該角色具有建立 Amazon SQS 佇列的許可，則任何可以承擔該角色的人都能建立佇列。您的 AWS 帳戶 (也是該角色所屬的帳戶) 擁有該佇列資源。

## 管理資源存取

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型政策 (IAM 政策)，而連接到資源的政策稱為資源型政策。

身分型政策 (IAM 政策和 Amazon SQS 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。
- 將許可政策連接至另一個 AWS 帳戶 中的使用者 - 若要授與使用者建立 Amazon SQS 佇列的許可，您可以將 Amazon SQS 許可政策連接至另一個 AWS 帳戶 中的使用者。

  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
- 將許可政策連接至角色 (授與跨帳戶許可) - 若要授與跨帳戶許可，您可以將身分型許可政策連接至 IAM 角色。例如，AWS 帳戶 A 管理員可以建立角色，將跨帳戶許可授予 AWS 帳戶 B (或 AWS 服務)，如下所示：
  - 帳戶 A 管理員建立 IAM 角色，並將許可政策 (可授與帳戶 A 中資源的許可) 連接到角色。
  - 帳戶 A 管理員將信任政策連接至角色，該角色識別帳戶 B 為可以擔任該角色的主體。
  - 帳戶 B 管理員將擔任該角色的許可委派給帳戶 B 中的任何使用者。這麼做可讓帳戶 B 中的使用者建立或存取帳戶 A 的佇列。

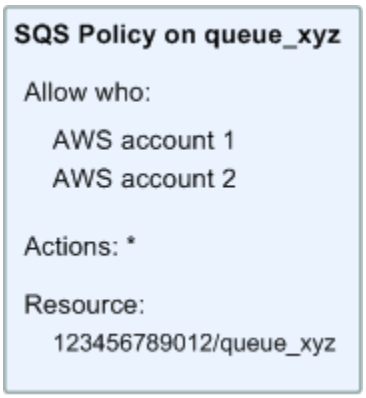
> **Note**
>
> 如果您想要將擔任該角色的許可授予 AWS 服務，則信任政策的主體也可以是 AWS 服務主體。

如需使用 IAM 來委派許可的詳細資訊，請參閱《IAM 使用者指南》中的存取管理。

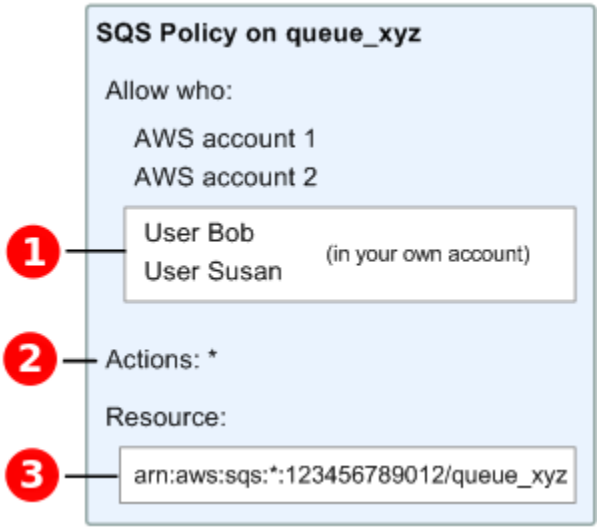
雖然 Amazon SQS 可與 IAM 政策搭配使用，但它有自己的政策基礎設施。您可以使用 Amazon SQS 政策搭配佇列，以指定哪些 AWS 帳戶可以存取佇列。您可以指定存取權類型和條件 (例如，如果請求在 2010 年 12 月 31 日之前提出則授與使用 `SendMessage`、`ReceiveMessage` 之許可的條件)。您可以授予許可的特定動作是整體 Amazon SQS 動作清單的一部分。當您撰寫 Amazon SQS 政策並將 * 指定為「允許所有 Amazon SQS 動作」，這表示使用者可以執行這個子集中的所有動作。

下圖說明其中一個涵蓋動作子集的基本 Amazon SQS 政策概念。政策適用於 `queue_xyz`，並提供 AWS 帳戶 1 和 AWS 帳戶 2 對指定的佇列使用任何允許動作的許可。

> **Note**
>
> 政策中的資源指定為 `123456789012/queue_xyz`，其中 `123456789012` 是擁有佇列之帳戶的 AWS 帳戶 ID。




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 如需有關將許可權授予不同帳戶中的使用者的詳細資訊，請參閱 IAM 使用者指南中的教學課程：使用 IAM 角色跨 AWS 帳戶委派存取權。

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。

例如，根據上圖中的 Amazon SQS 政策，擁有 AWS 帳戶 1 或 AWS 帳戶 2 安全憑證的任何人都能存取 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 |
| 服務角色 | 是 |
| 服務連結角色 | 否 |

如要全面了解 Amazon SQS 和其他 AWS 服務如何使用大多數的 IAM 功能，請參閱 IAM 使用者指南中的搭配 IAM 運作的 AWS 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 Simple Storage Service (Amazon S3)、AWS WAF 和 Amazon VPC 是支援 ACL 的服務範例。若要進一步了解 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 帳戶可將其許可委派給其帳戶下的使用者。跨帳戶存取權可讓您共用 AWS 資源的存取權，而無需管理其他使用者。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 自訂政策的限制。

## Amazon SQS 身分型政策

| | |
|---|---|
| 支援身分型政策 | 是 |

身分型政策是可以連接到身分 (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。若要瞭解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

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，因為這會套用至附加的使用者或角色。如要瞭解您在 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

| | |
|---|---|
| 支援以資源基礎的政策 | 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。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。主體可以包括帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

若要啟用跨帳戶存取權，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，作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。當主體和資源在不同的 AWS 帳戶 中時，受信任帳戶中的 IAM 管理員也必須授與主體實體 (使用者或角色) 存取資源的許可。其透過將身分型政策附加到實體來授予許可。不過，如果資源型政策會為相同帳戶中的主體授與存取，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 角色與資源型政策有何差異。

## 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哪些內容。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。政策動作的名稱通常會和相關聯的 AWS API 操作相同。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些操作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與執行相關聯操作的許可。

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
```

如需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哪些內容。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 (例如列出作業)，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索引鍵 | 是 |

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。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。若您為單一條件索引鍵指定多個值，AWS 會使用邏輯 OR 操作評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在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與該 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 支援全域條件索引鍵和服務特定的條件索引鍵。若要查看 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。您可以將標籤附加到 IAM 實體 (使用者或角色)，以及許多 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。

若要根據標籤控制存取，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索引鍵，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索引鍵，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的什麼是 ABAC?。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 |

您使用臨時憑證進行登入時，某些 AWS 服務 無法運作。如需詳細資訊，包括那些 AWS 服務 搭配臨時憑證運作，請參閱《IAM 使用者指南》中的可搭配 IAM 運作的 AWS 服務。

如果您使用使用者名稱和密碼之外的任何方法登入 AWS Management Console，則您正在使用臨時憑證。例如，當您使用公司的單一登入(SSO)連結存取 AWS 時，該程序會自動建立臨時憑證。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 使用者指南中的切換至角色 (主控台)。

您可使用 AWS CLI 或 AWS API，手動建立臨時憑證。接著，您可以使用這些臨時憑證來存取 AWS。AWS 建議您動態產生臨時憑證，而非使用長期存取金鑰。如需詳細資訊，請參閱 IAM 中的暫時性安全憑證。

## 適用於 Amazon SQS 的轉寄存取工作階段

| | |
|---|---|
| 支援轉寄存取工作階段 (FAS) | 是 |

當您使用 IAM 使用者或角色在 AWS 中執行動作時，您會被視為主體。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 使用主體的許可呼叫 AWS 服務，搭配請求 AWS 服務 以向下游服務發出請求。只有在服務收到需要與其他 AWS 服務 或資源互動才能完成的請求之後，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給 AWS 服務 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 |

服務連結角色是一種連結到 AWS 服務 的服務角色類型。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的 AWS 帳戶 中，並由該服務所擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服務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。

## Amazon SQS 的 AWS 受管政策更新

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 受管政策的更多相關資訊，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。您無法更改 AWS 受管政策中的許可。服務偶爾會在 AWS 受管政策中新增其他許可以支援新功能。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，服務很可能會更新 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 支援跨越多項服務之任務職能的受管政策。例如，ReadOnlyAccess 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，AWS 會為新的操作和資源新增唯讀許可。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管理策略：亞馬遜 FullAccess

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。

若要檢視此政策的權限，請參閱AWS受管政策參考FullAccess中的 AmazonSQL。

### AWS管理策略：亞馬遜 ReadOnlyAccess

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。

若要檢視此政策的權限，請參閱AWS受管政策參考ReadOnlyAccess中的 AmazonSQL。

### Amazon SQS 的 AWS 受管政策更新

檢視自此服務開始追蹤 Amazon SQS AWS 受管政策更新以來的變更詳細資訊。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 |
| --- | --- | --- |
| 亞馬遜人 ReadOnlyAccess | 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 9 日 |

## 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。

主題

- 我未獲授權，不得在 Amazon SQS 中執行動作
- 我沒有授權執行 iam：PassRole
- 我想要允許 AWS 帳戶 外的人員存取我的 Amazon SQS 資源

## 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。

如需任何協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

## 我沒有授權執行 iam：PassRole

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 允許您傳遞現有的角色至該服務，而無須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，該動作要求服務具備服務角色授與的許可。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。

如需任何協助，請聯絡您的 AWS 管理員。您的管理員提供您的登入憑證。

## 我想要允許 AWS 帳戶 外的人員存取我的 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的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。
- 如需了解如何存取您擁有的所有 AWS 帳戶 所提供的資源，請參閱《IAM 使用者指南》中的將存取權提供給您所擁有的另一個 AWS 帳戶 中的 IAM 使用者。
- 如需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擁有的 AWS 帳戶。
- 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 角色與資源型政策的差異。

# 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。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，Amazon SQS 動作皆支援資源層級許可。如需詳細資訊，請參閱 Amazon SQS API 許可：動作和資源參考。

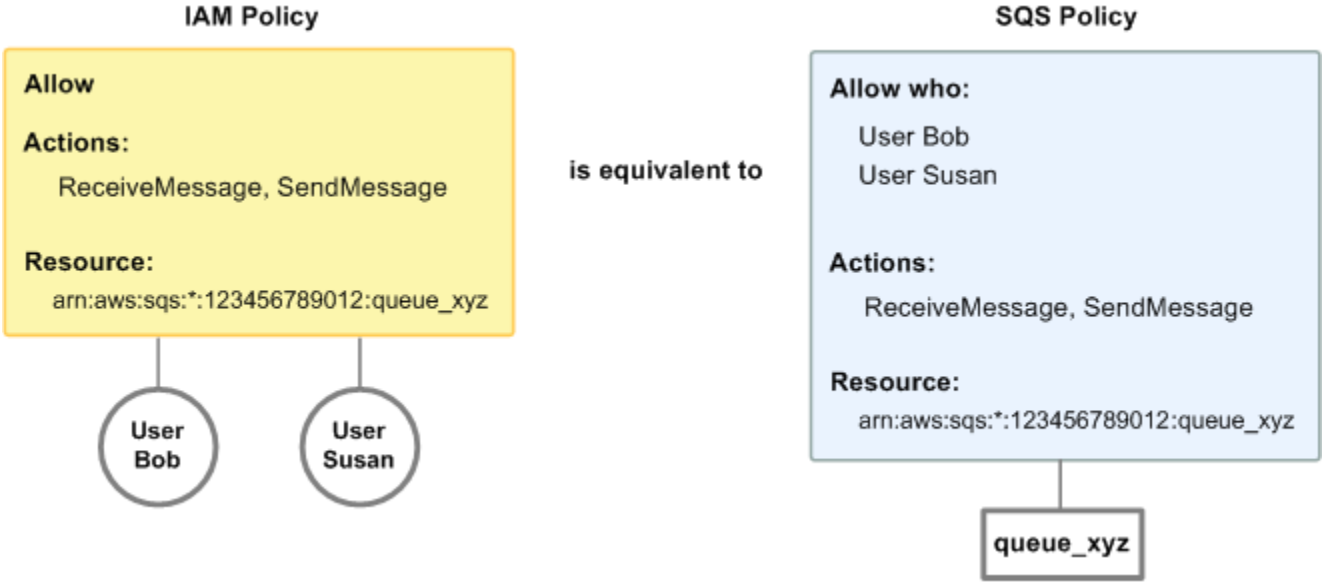
主題



## 使用 Amazon SQS 和 IAM 政策

有兩種方式可讓您將 Amazon SQS 資源的許可授予給使用者：使用 Amazon SQS 政策系統，以及使用 IAM 政策系統。您可以使用其中一種或兩者。大部分情況下，您可以使用任一個達成相同結果。

例如，下圖顯示 IAM 政策和相當的 Amazon SQS 政策。IAM 政策授予您 AWS 帳戶中稱為 queue_xyz 之佇列的 Amazon SQS ReceiveMessage 和 SendMessage 動作權利，而該政策連接到名為 Bob 和 Susan 的使用者 (Bob 和 Susan 擁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。

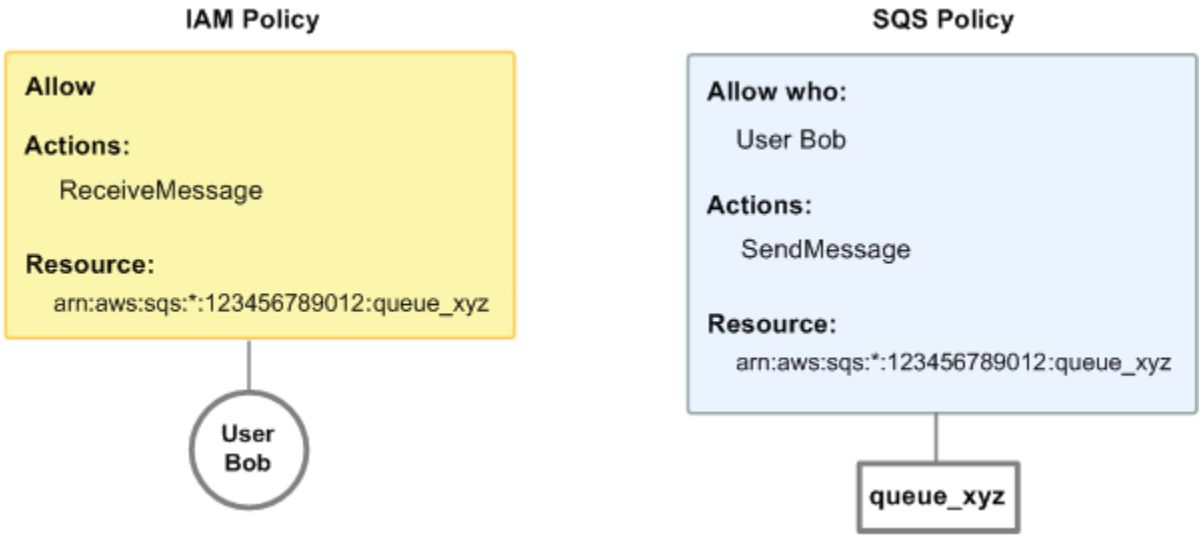



> **Note**
>
> 此範例顯示沒有條件的簡易政策。您可以在任一政策中指定特定條件，並取得相同結果。

IAM 和 Amazon SQS 政策間有一點主要不同：Amazon SQS 政策系統可讓您授予許可給其他 AWS 帳戶，而 IAM 政策無法做到這一點。

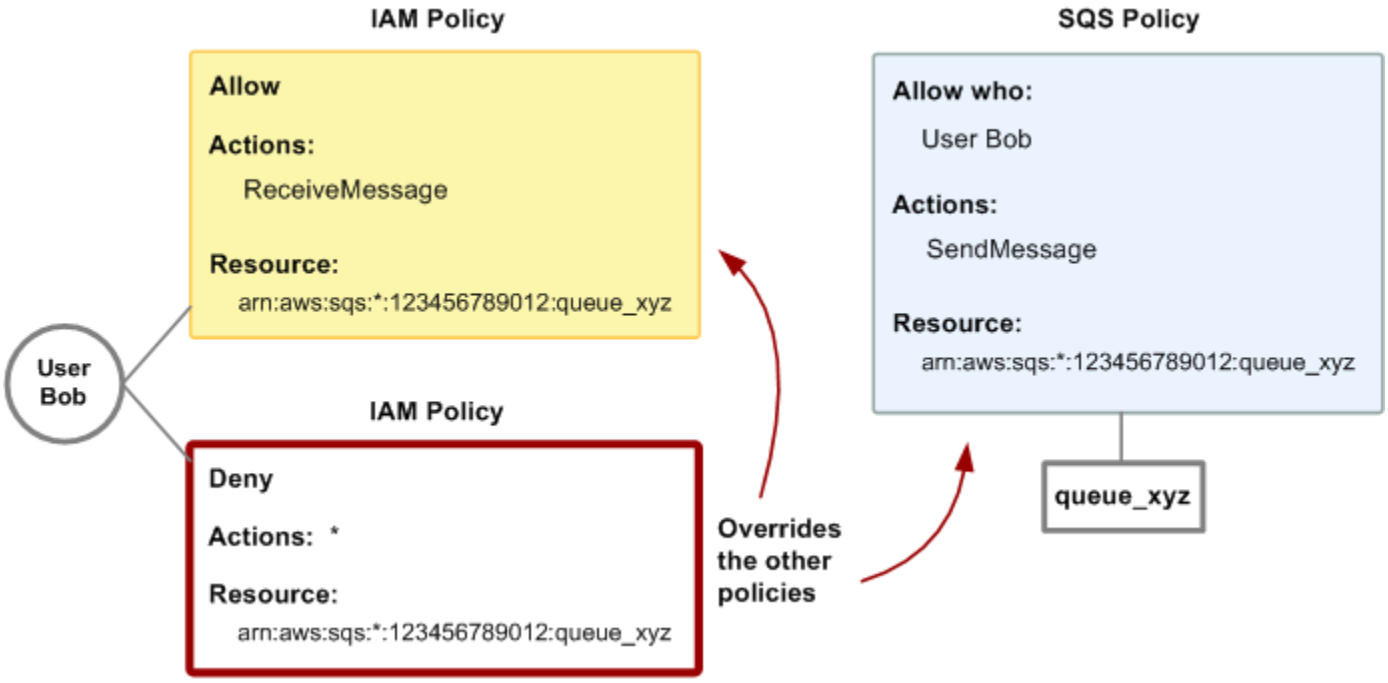
由您決定如何一起使用兩個系統來管理您的許可。以下範例顯示兩個政策系統如何搭配運作。

- 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詳細資訊，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，您不需要允許其最基本的主控台許可。

## 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 執行任務。若要授與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。

若要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。

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，請參閱適用於 Linux 執行個體的 Amazon EC2 使用者指南. 中的 Amazon EC2 Auto Scaling Lifecycle Hook。

主題

- 政策最佳實務
- 使用 Amazon SQS 主控台
- 允許使用者檢視他們自己的許可

- 允許使用者建立佇列
- 允許開發人員將消息寫入共享隊列
- 允許管理者取得佇列的一般大小
- 允許合作夥伴將訊息傳送至特定佇列

### 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 受管政策並朝向最低權限許可的目標邁進：如需開始授予許可給使用者和工作負載，請使用 AWS 受管政策，這些政策會授予許可給許多常用案例。它們可在您的 AWS 帳戶 中使用。我們建議您定義特定於使用案例的 AWS 客戶管理政策，以便進一步減少許可。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。
- 套用最低許可許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的權限。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。您也可以使用條件來授予對服務動作的存取權，前提是透過特定AWS 服務 (例如 AWS CloudFormation)使用條件。如需更多資訊，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，可協助您編寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 IAM Access Analyzer 政策驗證。
- 需要多重要素驗證 (MFA)：如果存在需要 AWS 帳戶中 IAM 使用者或根使用者的情況，請開啟 MFA 提供額外的安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 API 存取。

有關 IAM 中最佳實務的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 最佳安全實務。

### 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您 AWS 帳戶 中 Amazon SQS 資源的詳細資訊。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，您不需要允許其最基本主控台許可。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。

為確保使用者和角色仍然可以使用 Amazon SQS 主控台，請同時將 Amazon SQS AmazonSQSReadOnlyAccess AWS 受管政策附加到實體。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 使用者檢視附加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上，或是使用 AWS CLI 或 AWS API 透過編寫程式的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_*"
   }]
}
```

允許開發人員將消息寫入共享隊列

在下列範例中，我們為開發人員建立群組，並連接政策以讓該群組使用 Amazon SQS `SendMessage` 動作，但僅能用於屬於指定之 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`、`Recei` 和 `SendMessage`。

> **Note**
>
> 儘管 * 包含由其他許可類型提供的存取權限，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 提供的存取權限。
> 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。

允許管理者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，並連接政策以讓該群組使用 Amazon SQS GetQueueAttributes 動作，搭配屬於指定之 AWS 帳戶的所有佇列。

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

允許合作夥伴將訊息傳送至特定佇列

您可以使用 Amazon SQS 政策或 IAM 政策來完成這項工作。如果您的合作夥伴擁有 AWS 帳戶，可能使用 Amazon SQS 政策會比較容易。不過，合作夥伴公司中持有 AWS 安全憑證的任何使用者，都可傳送訊息到佇列。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. 建立一個名為代表 WidgetCo 合作夥伴公司的群組。
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**
>
> 建議您使用兩個瀏覽器視窗：一個用於授予許可，另一個則透過使用者的憑證登入 AWS Management Console，以便驗證對使用者授予的許可。

### 範例 1：將一個許可授予給一個 AWS 帳戶

以下範例政策將對於位在美國東部 (俄亥俄) 區域中名為 444455556666/queue1 之佇列的 SendMessage 許可，授予 AWS 帳戶 編號 111122223333。

```
{
   "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：將兩個許可授予給一個 AWS 帳戶

以下範例政策將對於名為 444455556666/queue1 之佇列的 SendMessage 和 ReceiveMessage 許可，授予 AWS 帳戶編號 111122223333。

```
{
   "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：將全部許可授予給兩個 AWS 帳戶

以下範例政策將對於位在美國東部 (俄亥俄) 區域中名為 123456789012/queue1 的佇列使用 Amazon SQS 允許共用存取之所有動作的許可，授予兩個不同的 AWS 帳戶編號 (111122223333 和 444455556666)。

```
{
   "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

以下範例政策授予AWS 帳戶 編號 111122223333 下的 role1 和 username1 跨帳戶權限，以使用 Amazon SQS 允許對美國東部 (俄亥俄) 區域中名為 123456789012/queue1 的佇列進行共用存取權的所有操作。

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

如果您只想要根據 AWS 帳戶 ID 和基本許可 (例如 SendMessage 或 ReceiveMessage) 來允許 Amazon SQS 存取權，便不需要撰寫自己的政策。您只需使用 Amazon SQS AddPermission 動作。

如果您想要根據更具體的條件 (例如，請求傳入的時間或申請者的 IP 地址) 來明確拒絕或允許存取權，則需要使用 Amazon SQS SetQueueAttributes 動作來撰寫您自己的 Amazon SQS 政策並上傳至 AWS 系統。

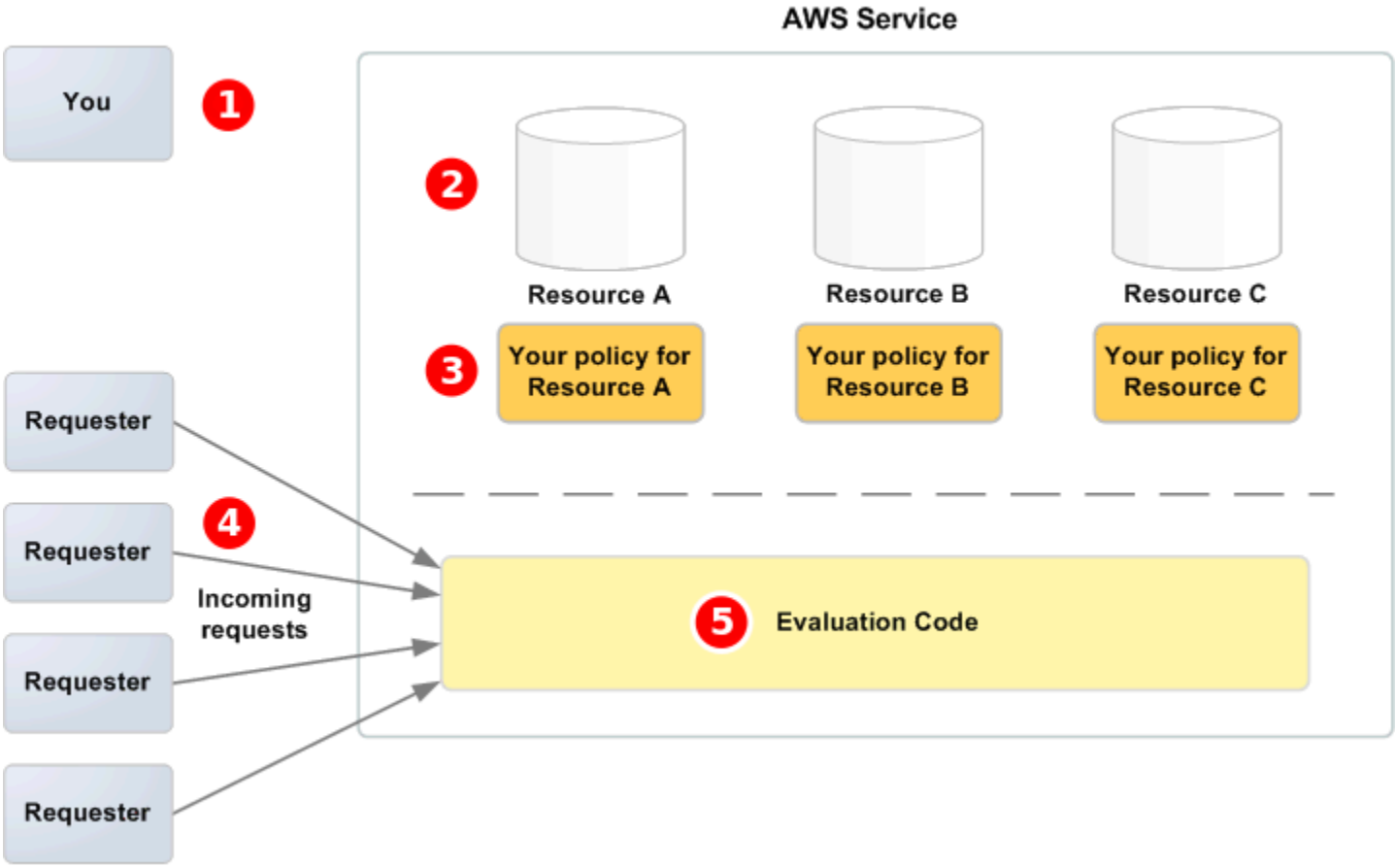
主題

- 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
- 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

包

含在 AWS 服務內的您的資源 (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 服務提供您用來上傳和管理您的政策的 API。

4

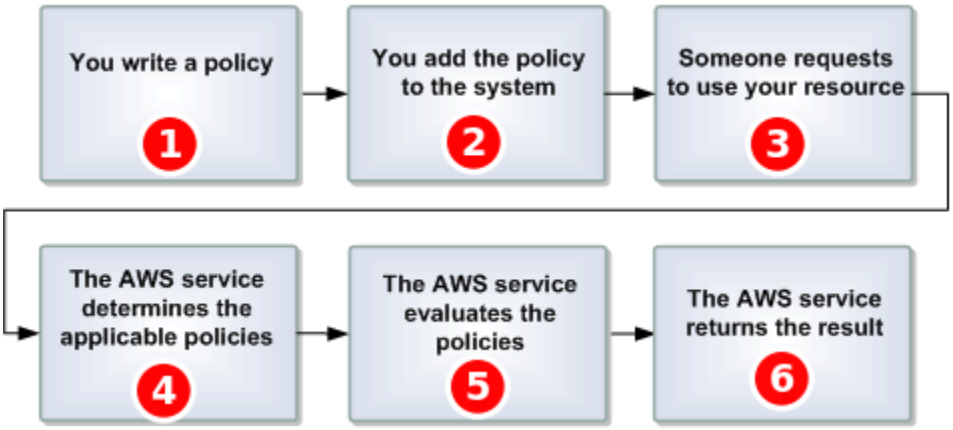
申請者及其傳入的 AWS 服務請求。

5 存

取政策語言評估代碼。這是 AWS 服務內的代碼集，根據適用的政策評估傳入的請求並判定是否允許申請者存取資源。

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 您

為您的佇列撰寫 Amazon SQS 政策。

2 您

上傳您的政策到 AWS。AWS 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 Amazo

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，發行者一律為資源擁有者。AWS 不允許 Amazon SQS 使用者為不是自己所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

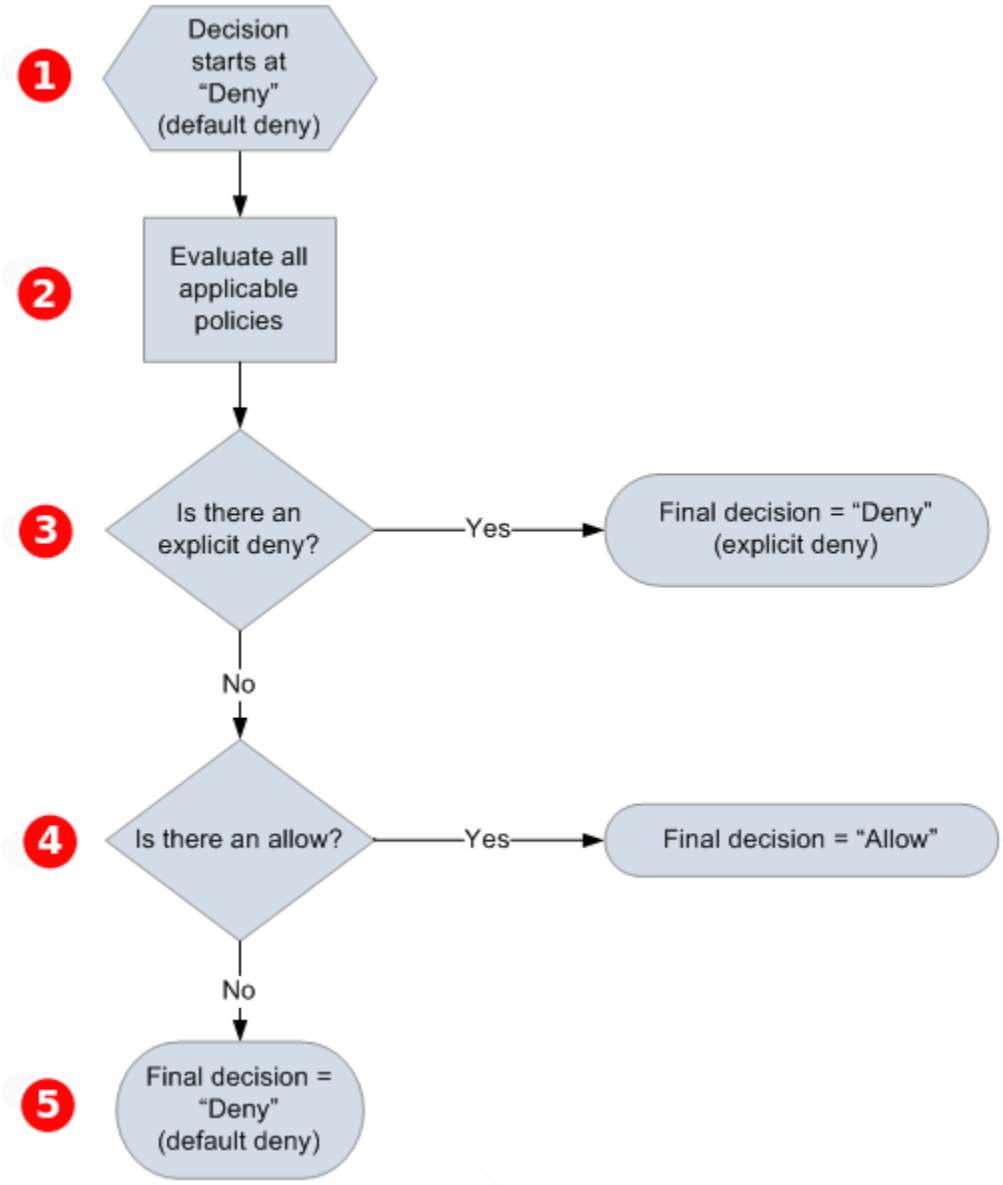
傳送存取Resource請求的使用者。

## 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)。

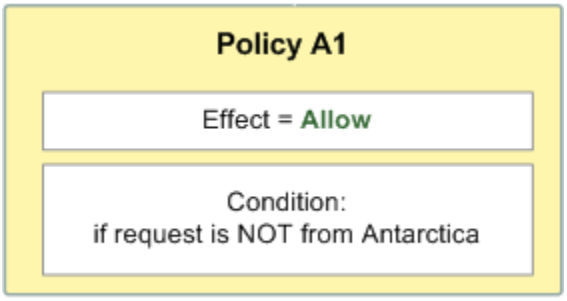
5

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)。

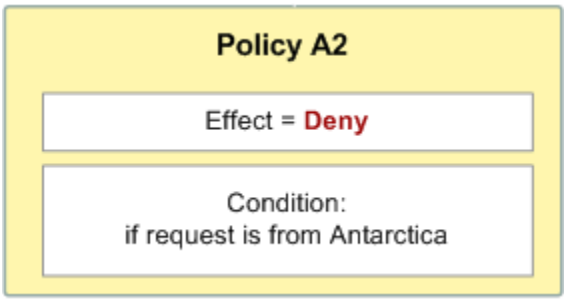
Amazon SQS 存取政策語言中明確拒絕和預設拒絕之間的關係

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。

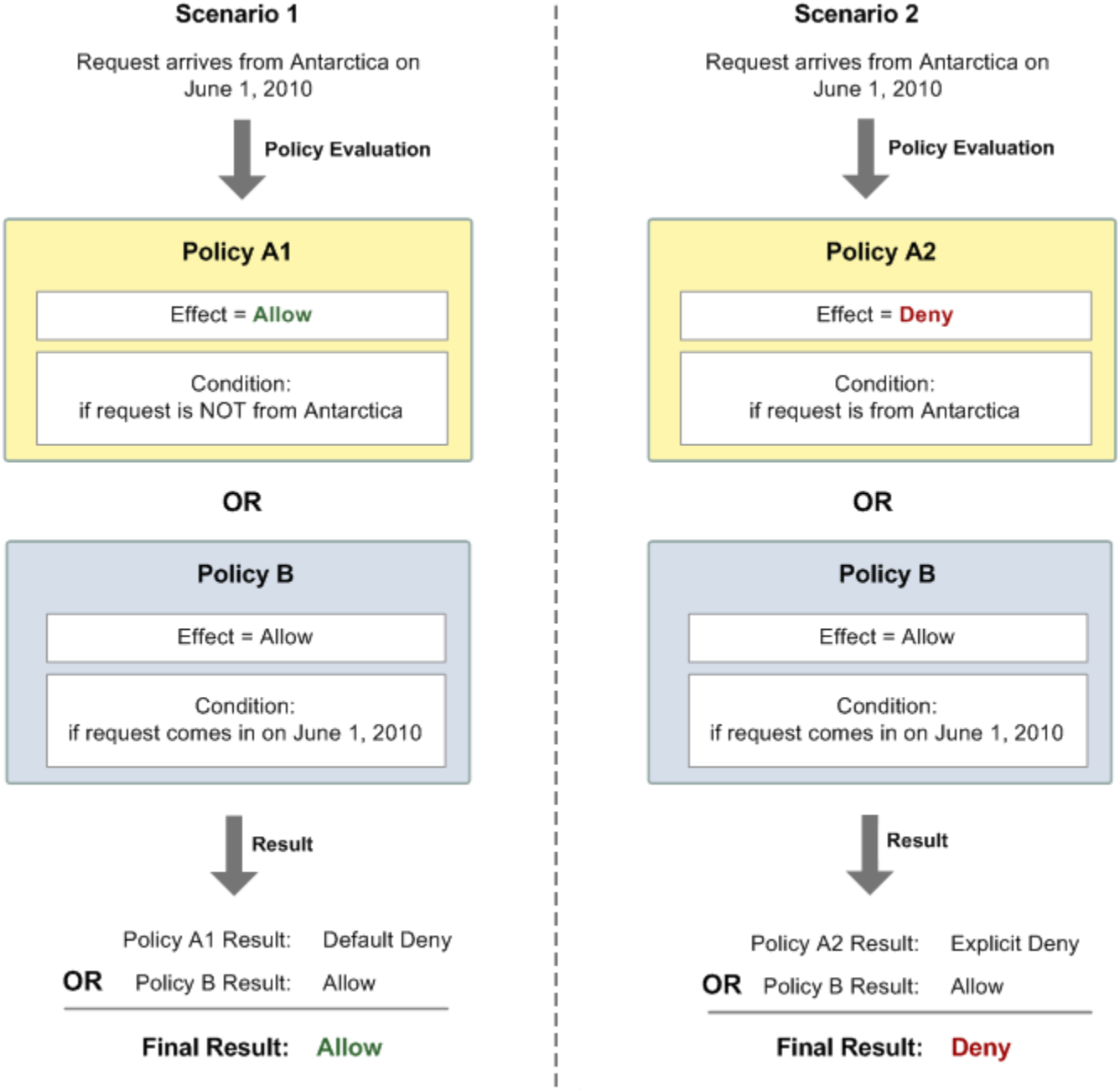



如果使用者從美國傳送請求，則符合條件 (請求非來自南極)，請求會產生允許。不過，如果使用者從南極傳送請求，便不符合條件，請求預設為預設拒絕。您可以撰寫政策 A2 以明確拒絕來自南極的請求，將結果變更為明確拒絕。下圖說明該政策。




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。

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

以下範例 Amazon SQS 政策提供一或多個 AWS 帳戶 在特定時間期限內，存取您帳戶擁有之佇列的許可。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，它限制對 IP 範圍 203.0.113.0/24 的存取。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

以下範例 Amazon SQS 政策拒絕特定 AWS 帳戶 對佇列的存取權。此範例是根據「範例 1：提供許可給一個帳戶」範例：拒絕所指定 AWS 帳戶 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

除了以使用者自己的安全憑證建立使用者之外，IAM 也允許您將臨時安全憑證授予任何使用者，以允許此使用者存取您的 AWS 服務與資源。您可以管理擁有 AWS 帳戶 的使用者。您也可以管理您系統中沒有 AWS 帳戶 的使用者 (聯合身分使用者)。此外，您建立用來存取 AWS 資源的應用程式，也可以視為「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。

必要條件

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。如需臨時安全登入資料之 IAM 支援的詳細資訊，請參閱 IAM 使用者指南中的授與臨時存取權給您的 AWS 資源。

若要使用臨時安全憑證呼叫 Amazon SQS 查詢 API 動作

1. 使用 AWS Identity and Access Management 請求臨時安全權杖。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=wJalrXUtnFEMI/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
```

## 使用最低權限 Amazon SQS 政策和 AWS KMS 金鑰政策管理對加密之 Amazon SQS 佇列的存取

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 Amazon SQS 與 AWS KMS 整合，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。

有多重 AWS 服務可做為將事件傳送到 Amazon SQS 佇列的事件來源。若要讓事件來源存取加密的 Amazon SQS 佇列，您需要使用客戶受管 AWS KMS 金鑰設定佇列。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。

以下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。

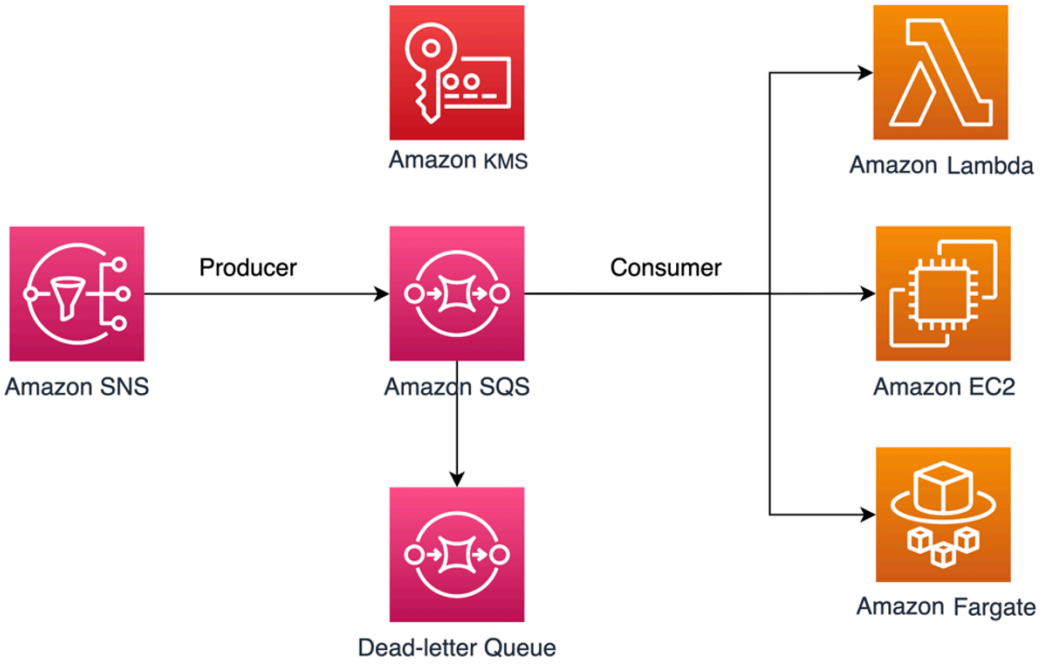
主題

- 概要
- 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

## 概要

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 `DenyReceivingIfNotThroughVPCE` 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。本指南不提供有關如何建立 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

本節說明在 AWS KMS 中用於加密 Amazon SQS 佇列之客戶受管金鑰的所需最低權限的許可。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：

- 將管理員權限授予 AWS KMS 金鑰
- 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

#### 將管理員權限授予 AWS KMS 金鑰

若要建立 AWS KMS 金鑰，您需要為用來部署 AWS KMS 金鑰的 IAM 角色提供 AWS KMS 管理員許可。這些管理員權定義於下列 `AllowKeyAdminPermissions` 政策陳述式中。將此聲明新增至 AWS KMS 金鑰政策時，請務必將 *<admin-role ARN>* 取代為用於部署 AWS KMS 金鑰、管理 AWS KMS 金鑰或二者的 IAM 角色的 Amazon Resource Name (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 *，意即 「這個 AWS KMS 金鑰」。星號 (*) 會識別金鑰政策所連接的 AWS KMS 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，下列聲明可讓您列出帳戶中的所有 AWS KMS 金鑰，以供稽核之用。此陳述式將 AWS 根使用者唯讀存取權授予金鑰中繼資料。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": "AllowReadAcesssToKeyMetaData",
  "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 許可權限，允許其使用 AWS KMS 金鑰發佈到 Amazon SQS 佇列。確實將 *<account-ID>* 取代為您自己的資訊。

> **Note**
>
> 陳述式中的 `Condition` 限制只能存取相同 AWS 帳戶中的 Amazon SNS 服務。

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
  ],
  "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) 來存取您帳戶中的資源，則 AWS 提供的工具可協助您保護帳戶。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 中的特定資源、特定帳戶或特定組織所擁有的許可權。

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 金鑰授予外部實體的存取權時提醒您。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。這個信任區域可以是 AWS 帳戶或您啟用 IAM Access Analyzer 時指定的 AWS Organizations 內的組織。

IAM Access Analyzer 會使用邏輯推理來識別與外部主體共用的資源，以分析 AWS 環境中的資源型政策。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，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。如需有關 AWS IAM Access Analyzer 如何識別非預定存取 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相應動作，以及您可以授予許可的 AWS 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

本節提供記錄和監控 Amazon SQS 佇列的相關資訊。

主題



## 使用 AWS CloudTrail 記錄 Amazon SQS API 呼叫

Amazon SQS 與整合，可記錄AWS CloudTrail來自使用者、角色或AWS服務的 Amazon SQS 呼叫。CloudTrail 將與 Amazon SQS 標準和 FIFO 佇列相關的 API 呼叫擷取為事件，包括透過 Amazon SQS 主控台啟動的互動，以及透過程式設計方式呼叫 Amazon SQS API。

### Amazon SQS 信息 CloudTrail

CloudTrail 當您建立AWS帳戶時，預設為開啟。當受支援的 Amazon SQS 事件活動發生時，它會與其他AWS服務 CloudTrail 事件一起記錄在事件歷史記錄中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱AWS CloudTrail使用指南中的檢視具有 CloudTrail 事件歷程記錄的事件。

呼叫佇列管理操作 (例如) 的 Amazon SQS API `AddPermission` 會歸類為管理事件，預設會登 CloudTrail 入。Amazon SQS API 是在 Amazon SQS 佇列上執行的大量操作，例如會歸類`SendMessage`為資料事件，並在您選擇加入後記錄。 CloudTrail

使用 CloudTrail 收集的資訊，您可以識別 Amazon SQS API 的特定請求、請求者的 IP 地址或身分，以及請求的日期和時間。如果您設定 CloudTrail 追蹤，您可以透過選擇性的交付到 Amazon CloudWatch 日誌和，持續將 CloudTrail事件傳遞到 Amazon S3 儲存貯體AWSEventBridge。如果您未設定追蹤，則只能在主控台中檢視事件中管理事件的事件歷史記 CloudTrail 錄。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的建立追蹤的概觀。

### 管理事件 CloudTrail

Amazon SQS 會將下列 API 動作記錄為管理事件：

- `AddPermission`
- `CreateQueue`
- `CancelMessageMoveTask`

- DeleteQueue
- ListMessageMoveTasks
- PurgeQueue
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

不支援下列 Amazon SQS API 進行記 CloudTrail 錄：

- GetQueueAttributes
- GetQueueUrl
- ListDeadLetterSourceQueues
- ListQueueTags
- ListQueues

## 資料事件 CloudTrail

資料事件提供有關在資源上或資源中執行的資源操作的資訊，例如傳送至或接收自 Amazon SQS 佇列的 Amazon SQS 訊息。資料事件是預設 CloudTrail 不會記錄的大量活動。您可以使用 CloudTrail API 為 SQS 佇列啟用資料事件 API 動作記錄。如需詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的記錄資料事件。

您可以使用進階事件選取器來決定要記錄和記錄哪些 Amazon SQS API 活動。 CloudTrail若要記錄 Amazon SQS 資料事件，您必須包含資源類型 `AWS::SQS::Queue`。設定此選項後，您可以藉由選取要記錄的特定資料事件 (例如使用 `eventName` 篩選條件追蹤 `SendMessage` 事件)，進一步調整記錄偏好設定。如需詳細資訊，請參閱 AWS CloudTrail API 參考中的 AdvancedEventSelector。

Amazon SQS 資料事件：

- SendMessage
- SendMessageBatch
- ReceiveMessage

- DeleteMessage
- DeleteMessageBatch
- ChangeMessageVisibility
- ChangeMessageVisibilityBatch

資料事件需支付額外的費用。如需詳細資訊，請參閱AWS CloudTrail 定價。

## 範例：適用於 Amazon SQS 的 CloudTrail 管理事件

下列範例顯示支援 API 的 CloudTrail 記錄項目：

AddPermission

下列範例顯示 AddPermission API 呼叫的 CloudTrail 記錄項目。


```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Alice",
         "accountId": "123456789012",
         "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
         "userName": "Alice"
       },
       "eventTime": "2018-06-28T22:23:46Z",
       "eventSource": "sqs.amazonaws.com",
       "eventName": "AddPermission",
       "awsRegion": "us-east-2",
       "sourceIPAddress": "203.0.113.0",
       "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
       "requestParameters": {
         "actions": [
           "SendMessage"
         ],
         "AWSAccountIds": [
           "123456789012"
         ],
         "label": "MyLabel",
```

```
        "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
      },
      "responseElements": null,
      "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
      "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
    }
  ]
}
```

### CreateQueue

下列範例顯示 `CreateQueue` API 呼叫的 CloudTrail 記錄項目。

```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Alejandro",
         "accountId": "123456789012",
         "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
         "userName": "Alejandro"
       },
       "eventTime": "2018-06-28T22:23:46Z",
       "eventSource": "sqs.amazonaws.com",
       "eventName": "CreateQueue",
       "awsRegion": "us-east-2",
       "sourceIPAddress": "203.0.113.1",
       "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
       "requestParameters": {
         "queueName": "MyQueue"
       },
       "responseElements": {
         "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
       },
       "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
       "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
     }
   ]
 }
```

### DeleteQueue

下列範例顯示 DeleteQueue API 呼叫的 CloudTrail 記錄項目。

```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Carlos",
         "accountId": "123456789012",
         "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
         "userName": "Carlos"
       },
       "eventTime": "2018-06-28T22:23:46Z",
       "eventSource": "sqs.amazonaws.com",
       "eventName": "DeleteQueue",
       "awsRegion": "us-east-2",
       "sourceIPAddress": "203.0.113.2",
       "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
       "requestParameters": {
         "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
       },
       "responseElements": null,
       "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
       "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
     }
   ]
 }
```

### RemovePermission

下列範例顯示 RemovePermission API 呼叫的 CloudTrail 記錄項目。

```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
```

```
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Jane",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Jane"
      },
      "eventTime": "2018-06-28T22:23:46Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "RemovePermission",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "203.0.113.3",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
      "requestParameters": {
        "label": "label",
        "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
      },
      "responseElements": null,
      "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
      "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
    }
  ]
}
```

### SetQueueAttributes

下列範例顯示下列項目的 CloudTrail 記錄項目`SetQueueAttributes`：

```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Maria",
         "accountId": "123456789012",
         "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
         "userName": "Maria"
       },
       "eventTime": "2018-06-28T22:23:46Z",
       "eventSource": "sqs.amazonaws.com",
       "eventName": "SetQueueAttributes",
       "awsRegion": "us-east-2",
```

```
      "sourceIPAddress": "203.0.113.4",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
      "requestParameters": {
        "attributes": {
          "VisibilityTimeout": "100"
        },
        "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
      },
      "responseElements": null,
      "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
      "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
    }
  ]
}
```

## 範例：適用於 Amazon SQS 的 CloudTrail 資料事件

以下是 Amazon SQS 資料事件 API 的特定事件範例： CloudTrail

### SendMessage

下列範例顯示的 CloudTrail 資料事件SendMessage。

```
   {
 "eventVersion": "1.09",
 "userIdentity": {
   "type": "AssumedRole",
   "principalId": "EXAMPLE_PRINCIPAL_ID",
   "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/SessionName",
   "accountId": "123456789012",
   "accessKeyId": "ACCESS_KEY_ID",
   "sessionContext": {
     "sessionIssuer": {
       "type": "Role",
       "principalId": "AKIAI44QH8DHBEXAMPLE",
       "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed",
       "accountId": "123456789012",
       "userName": "RoleToBeAssumed"
     },
     "attributes": {
       "creationDate": "2023-11-07T22:13:06Z",
       "mfaAuthenticated": "false"
```

```
      }
    }
  },
  "eventTime": "2023-11-07T23:59:11Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessage",
  "awsRegion": "ap-southeast-4",
  "sourceIPAddress": "10.0.118.80",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/
linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/
pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue",
    "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageDeduplicationId": "MsgDedupIdSdk1ae1958f2-bbe8-4442-83e7-4916e3b035aa",
    "messageGroupId": "MsgGroupIdSdk16"
  },
  "responseElements": {
    "mD5OfMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
    "mD5OfMessageSystemAttributes": "f88f0587f951b7f5551f18ae699c3a9d",
    "messageId": "93bb6e2d-1090-416c-81b0-31eb1faa8cd8",
    "sequenceNumber": "18881790870905840128"
  },
  "requestID": "c4584600-fe8a-5aa3-a5ba-1bc42f055fae",
  "eventID": "98c735d8-70e0-4644-9432-b6ced4d791b1",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::SQS::Queue",
      "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sqs.ap-southeast-4.amazonaws.com"
  }
```

## SendMessageBatch

下列範例顯示的 CloudTrail 資料事件SendMessageBatch。


```
    {
  "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": "2023-11-07T23:59:05Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessageBatch",
  "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": {
    "delaySeconds": 0,
    "entries": [
      {
        "id": "0",
        "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
        "messageAttributes": [
          {
            "name": "HIDDEN_DUE_TO_SECURITY_REASONS"
```

```
          }
        ],
        "messageDeduplicationId": "MsgDedupIdSdk1027092b6-b6f6-41af-a084-e72d572a6d4b",
        "messageGroupId": "MsgGroupIdSdk12"
      }
    ],
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue"
  },
  "responseElements": {
    "successful": [
      {
        "id": "0",
        "messageId": "9048ab28-e38d-46da-b9fe-f70b3873f888",
        "mD5OfMessageBody": "0f1a575a56eb5cf5072a8dedc585d2dd",
        "mD5OfMessageAttributes": "6e1d6d5d774a05efe9df5eb000639db7",
        "sequenceNumber": "18881790869375471872",
        "mD5OfMessageSystemAttributes": "6f540b6e375dcda1aad2d4aaff28ebf8"
      }
    ]
  },
  "requestID": "b5a386a4-2d4a-5de3-9910-db60fcc368ee",
  "eventID": "20f5ecbe-2b0b-4d0b-a6f7-365bc94c4ca5",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue",
      "type": "AWS::SQS::Queue"
    }
  ],
  "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"
  }
}
```

### ReceiveMessage

下列範例顯示的 CloudTrail 資料事件`ReceiveMessage`。


```
    {
  "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": "2023-11-07T23:59:24Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "ReceiveMessage",
  "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",
    "maxNumberOfMessages": 10
  },
  "responseElements": null,
  "requestID": "8b4d4643-8f49-52cd-a6e8-1b875ed54b99",
  "eventID": "f3f23ab7-b0a4-4b71-afc0-141209c49206",
  "readOnly": true,
  "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"
  }
}
```

### DeleteMessage

下列範例顯示的 CloudTrail 資料事件`DeleteMessage`。

```
    {
  "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": "2023-11-07T23:58:42Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "DeleteMessage",
  "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": {
    "receiptHandle": "AQEBfgYUKTy3dyOAewC4wI3lQZEB3oUDuv8M8FWhObnr3lqRsFBiZ57mmxO1/
dWfdlvGgDW7sRSry6HHxWrNfHItQMUHtWX3a/vEjJ6sWC/5Mf36I/B2HBLCT2zGO/
IZTywxFmUT4HUudWKcGpuZb/Kcl3Fom6hYU8PxxzPxLOKPtFwrVU+G2Spvf/
Tbuyj27h5+AkNxfaAhu/dnvXnAJcDJErGsJTjSS1i6iRzFq+jg6K5Fw6T578QJZcx/
ZLaCyohmj2HaOOktwhbqQc4j+2gKSfxrACgXCu6De5bCtwgtGdhMEh4DtVIQh88qGUCaofQ3t/
eRBIvIFJIa61JCVNWSBqOtELEIfxaHpSvo0c1IEecKDt1IJ08Cij3euLFMIzmUot24IViZt8ntKVAZ6KBLlLedrVlxOhNVc
WG0jfbqz3iBS1T1AD1zJKT7ICIA+edgaYJpOZw4=",
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue"
  },
  "responseElements": null,
  "requestID": "fbd23ff4-a107-536d-8fcb-623070754bc0",
  "eventID": "9951fed7-365f-4046-bc71-e5bf065a9b47",
  "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"
  }
}
```

DeleteMessageBatch

下列範例顯示的 CloudTrail 資料事件DeleteMessageBatch。


```
    {
  "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": "2023-11-07T23:59:24Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "DeleteMessageBatch",
  "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",
    "entries": [
      {
        "id": "0",
        "receiptHandle": "AQEBefxM1O4zyZGF87DehbRbmri91w2W7mMdD0GrBjQa8e/
hpb4RbXHPZ9tLBVleECbChQIE5NtaDuoZhZPOkTy0eN46EyRR4jXDzE3AlkbPlX1mA9f2fUuTrXx8aeCoCA3I3woNg3fXXA
hlLS94tjAZqV2krc4BaC2pYgjyHWcW019HwIV8T/bjNMIeZoQwOM5V
+o9vHPfewz5QGr5SKpDo7uE7Umyk5n5CJZvcn1efp/
```

```
mrwtaCIb9M7cCQUYcZm2ZmZDnIO9XpGTAi3m2dQ0M83pnNh0nvDfpkHpoa+hX1TrUmxCupCWHJwA8HFJ1O/
CCJsodMNFthLBA9S57dkBZCsw41G8jAmgQ0MkvZ0UL5mg0OFQQd1Yrw0zvthjCgiwdzn0yXoMzxIZMBxkY14E4nVVZ7N5XE
h8oRk2C7gByzg2kYJ0LnUvLJFT8DQE28JZppEC9klvrdR/BWiPT7asc="
      }
    ]
  },
  "responseElements": {
    "successful": [
      {
        "id": "0"
      }
    ],
    "failed": []
  },
  "requestID": "fe423091-5642-5ba5-9256-6d5587de52f1",
  "eventID": "88c8020d-d769-4985-8ecb-ee0b59acc418",
  "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"
  }
}
```

### ChangeMessageVisibility

下列範例顯示的 CloudTrail 資料事件`ChangeMessageVisibility`。

```
    {
  "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": "2023-11-07T23:59:24Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "ChangeMessageVisibility",
  "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",
    "receiptHandle": "AQEBy+2qnmQQVxcXrEwN7t6dXkjGAllr5DuSpGlvHx9s/vwbwp+RIr3dD6vRvlsU/
lteIulKHBs6DEIR7KL+J3mACfB+RRpRlWPlguiCdLKNKSVpdhkSBDDvkRHfycTHjuszGIebGdl+tYYjPrlz
+DSePmpty0EdhqtorW1xAc0Xf0GZbt0FtkbRFK3ql5lETIHgthBCABoxuOCNvMElz9rYQ9m5OY30Z5YOZvQ/
coPHYl+9HhNV/A6Fs+/d6mVx9v6TomTh5L03wXqtjA8b0gkGftclQh/tJBAxqY/S8YG9OKtY4NDP0SQBtYF/
vCCsCq9+5fiUfiYyvtdHSlwP9AyRotenCGrUKaRFiRhxDm1D6up0UaBs2d8wgHdKFf/5mENTdeqrXQdZfwkFazW1a8ifWJm
+HzhfA9EJcdgWSS72WCMaerydsCxaX+E08B2ubL6oiafMYW4gK0GIRxYZ0+eeXKWy4TxkReW3j7k=",
    "visibilityTimeout": 1272
  },
  "responseElements": null,
  "requestID": "6fbefbde-55d9-5640-98d1-a61a84457f14",
  "eventID": "72275c61-bfc0-4606-934b-a6b7397aef20",
  "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"
  }
}
```

### ChangeMessageVisibilityBatch

下列範例顯示的 CloudTrail 資料事件ChangeMessageVisibilityBatch。

```
    {
  "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": "2023-11-07T23:59:01Z",
```

```
  "eventSource": "sqs.amazonaws.com",
  "eventName": "ChangeMessageVisibilityBatch",
  "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": {
    "visibilityTimeout": 0,
    "entries": [
      {
        "id": "0",
        "receiptHandle":
 "AQEB2M5cVYg5gslhWME6537hdjcaPnOYPA5M0W460TTb0DzPle631yPWm8qxd4O1hDj/
B4ntTMnsgBTa95t14tNx7Vn96jKJ5rIoZ7iI8TRmkT1caKodKIPs8w9yndZq50c2FPQxtyH+2L3UHf/
abV3szqVWXOLZR4PwX8zZkWVQGNCNnY2q2lGCG586F8QwvrOFYoXNwB8ymd1t77e1PDPknq1Io3JFuzkEsndkkETy4fV1Qq
l5PHX17nXxaC+DURVlMPXOuSFACGmWqAoyk50HKwGOjLQgpySL/
TcnQXClvFq8kNXGwyVzJsbwHpOHxI7oce69vaD6DaWFP75d3hx+PJeG9pauQCKzVP3skt3Hw/
zDC7YfKcALD3aCwMmeNDwT3w0BUG6XZdG5lYhtFtTQYV7YuS3i/
Jh3HShGbtm07JKOEFiPkxv2+XNaAX3gFEpbng6zamTanfyMXCJIiglAEqiyWHQ=",
        "visibilityTimeout": 2271
      }
    ],
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue"
  },
  "responseElements": {
    "successful": [
      {
        "id": "0"
      }
    ]
  },
  "requestID": "d49ab65f-9dc7-54b8-875c-eb9b4c42988b",
  "eventID": "ca16c8c2-c4ba-4eb5-a54c-e650a10266d4",
  "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"
  }
}
```

# 使用監控 Amazon SQS 佇列 CloudWatch

整合了 Amazon SQS 和亞馬遜 CloudWatch ，因此您可以用 CloudWatch 來檢視和分析 Amazon SQS 佇列的指標。您可以使用或使用 API，從 Amazon SQS 主控台、主控台檢視和分析佇列的指標。CloudWatch AWS CLICloudWatch 您也可以為 Amazon SQS 指標設定 CloudWatch 警示。

CloudWatch Amazon SQS 佇列的指標會以一分鐘的 CloudWatch 間隔自動收集和推送到。這些量度會收集在符合作用中 CloudWatch 準則的所有佇列上。 CloudWatch如果佇列包含任何訊息或有任何動作存取，則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時，Amazon SQS 服務會被視為休眠狀態，並停止向服務傳送指標。 CloudWatch 在 Amazon SQS 佇列處於非作用中的期間內，無法在 Amazon SQS 的 CloudWatch 指標中視覺化遺失的資料或代表零的資料。

> **Note**
>
> - 從非作用中狀態啟動佇列時， CloudWatch 量度中會發生最多 15 分鐘的延遲。
> - 中 CloudWatch報告的 Amazon SQS 指標不收取任何費用。這是 Amazon SQS 服務的一部分。
> - CloudWatch 標準佇列和 FIFO 佇列都支援指標。

主題

- 存取 CloudWatch Amazon SQS 的指標
- 為 Amazon SQS 指標建立 CloudWatch 警示
- Amazon SQS 的可用 CloudWatch 指標

# 存取 CloudWatch Amazon SQS 的指標

整合了 Amazon SQS 和亞馬遜 CloudWatch ，因此您可以用 CloudWatch 來檢視和分析 Amazon SQS 佇列的指標。您可以使用或使用 API，從 Amazon SQS 主控台、主控台檢視和分析佇列的指標。CloudWatch AWS CLICloudWatch 您也可以為 Amazon SQS 指標設定 CloudWatch 警示。

Amazon SQS 主控台

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。

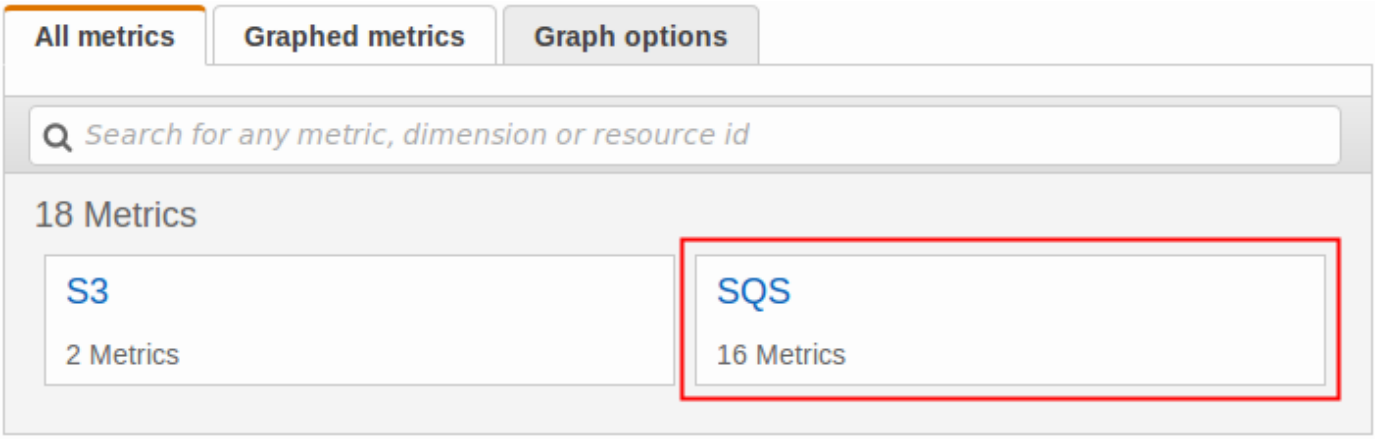
   SQS metrics (SQS 指標) 部分會顯示各種圖形。
4. 若要了解特定圖形代表的意義，請將滑鼠的游標移至所需圖形旁的

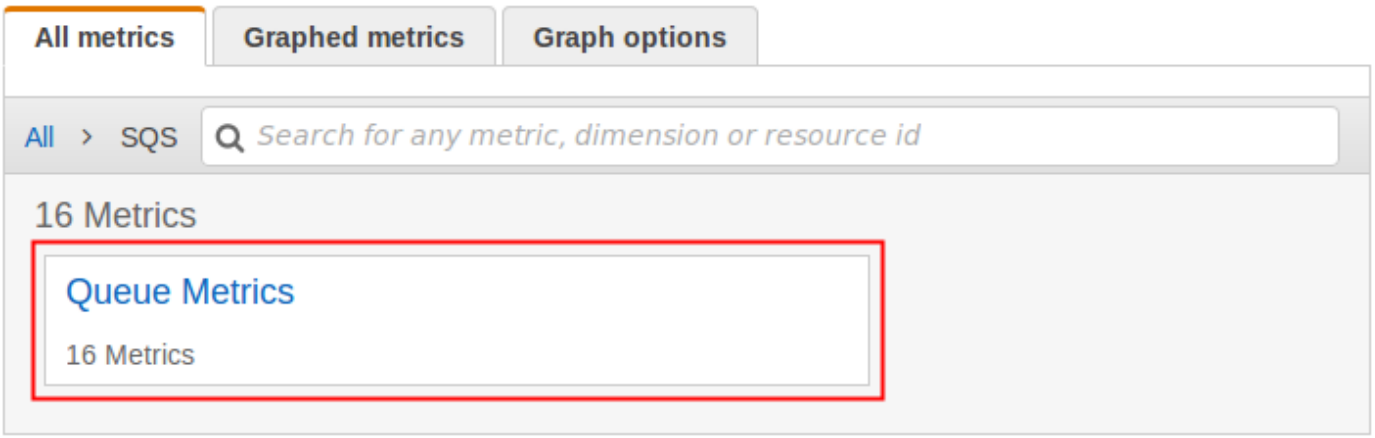
   上，或請參閱 Amazon SQS 的可用 CloudWatch 指標。
5. 若要同時變更所有圖表的時間範圍，請在 Time Range (時間範圍) 選擇所需的時間範圍 (例如 Last Hour (前一小時))。
6. 若要查看個別圖表的其他統計資料，請選擇圖表。
7. 在 CloudWatch Monitoring Details (監控詳細資訊) 對話方塊中，選取 Statistic (統計資料) (例如 Sum (總和))。如需支援的統計資料清單，請參閱Amazon SQS 的可用 CloudWatch 指標。
8. 若要變更個別圖形顯示的時間範圍和時間間隔 (例如，若要顯示過去 24 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)。
9. (選擇性) 若要使用其他 CloudWatch 功能，請在監視索引標籤上選擇檢視所有測 CloudWatch量結果，然後依照Amazon CloudWatch 遊戲程序中的指示執行。

Amazon CloudWatch 遊戲

1. 登入 CloudWatch 主控台。
2. 在導覽面板上，選擇 Metrics (指標)。
3. 選取 SQS 指標命名空間。

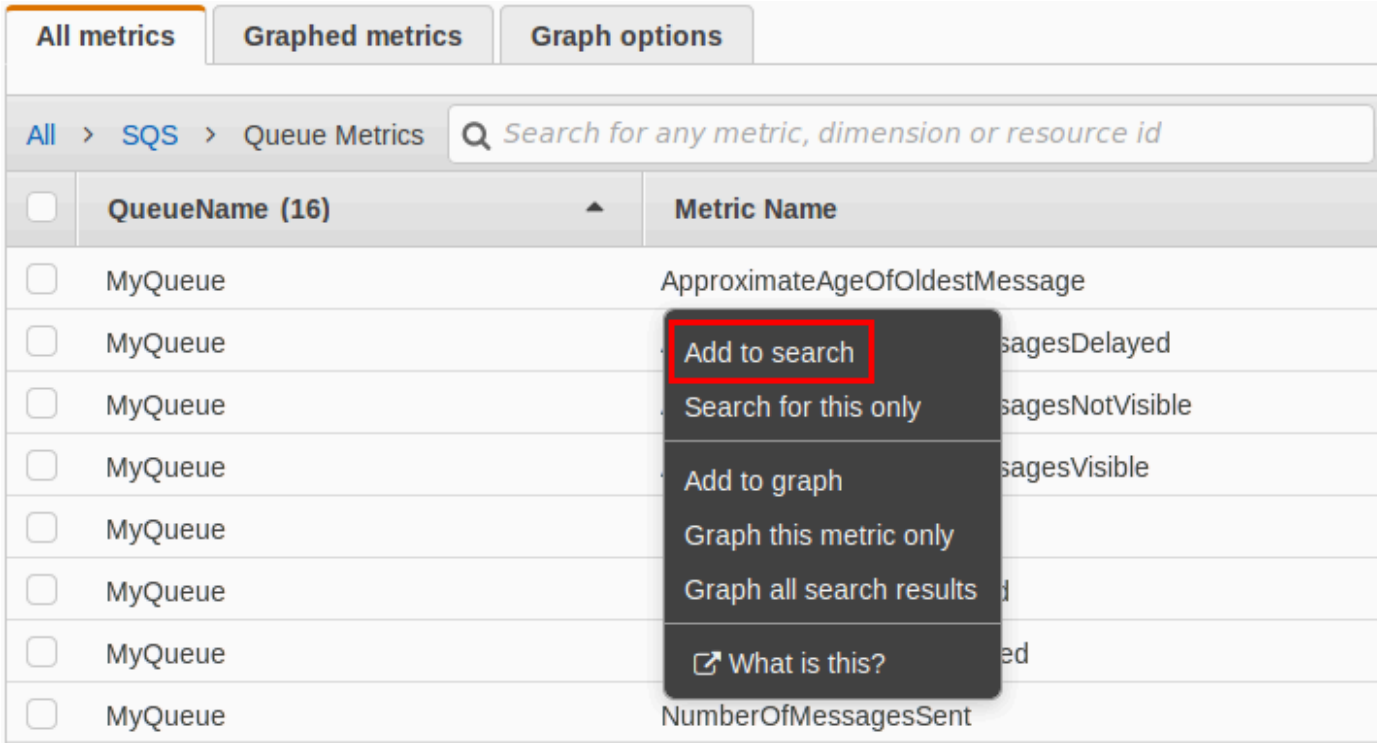



4. 選取 Queue Metrics (佇列指標) 指標維度。




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 使用者指南中的圖形指標和使 CloudWatch 用 Amazon CloudWatch 儀表板。

AWS Command Line Interface

若要使用 AWS CLI 存取 Amazon SQS 指標，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。

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。

## 為 Amazon SQS 指標建立 CloudWatch 警示

CloudWatch 可讓您根據量度臨界值觸發警示。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. 請登入AWS Management Console並開啟 CloudWatch 主控台，網址為 https://console.aws.amazon.com/cloudwatch/。
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 > 佇列測量結果，請選擇要設定警示的QueueName和測量結果名稱，然後選擇下一步。如需可用指標的清單，請參閱 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 CloudWatch 要在觸發警示時傳送通知，請選取現有的 Amazon SNS 主題或選擇 [新增清單]，然後輸入以逗號分隔的電子郵件地址。

   > **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 Create Alarm (建立警示)。

   警示已建立。

## Amazon SQS 的可用 CloudWatch 指標

Amazon SQS 會將下列指標傳送到 CloudWatch。

> **Note**
>
> 對於標準佇列，由於 Amazon SQS 的分散式架構，結果是近似值。在大多數情況下，計數應接近佇列中的實際訊息數目。
> 對於 FIFO 佇列，結果是準確的。

### Amazon SQS 指標

AWS/SQS 命名空間包含下列指標。

| 指標 | 描述 |
| --- | --- |
| ApproximateAgeOfOldestMessage | 佇列中最舊的未刪除訊息的大約存在時間。<br><br>**Note**<br><br>• 如果收到訊息三次 (或更多) 且未處理，此訊息則會移至位於尚未接收超過三次之第二久訊息的佇列和 ApproximateAgeOfOldestMessage 指標節點的後方。即使佇列有再驅動政策，仍會發生此動作。<br>• 由於單一毒丸訊息 (已接收多次但從未刪除) 可能導致此指標曲解，因此在成功消耗毒丸訊息之前，不會將毒丸訊息的存留期計入指標。<br>• 當佇列有再驅動政策時，訊息便會在達到設定的接收數目上限後移至無效字母佇列。當訊息移至無效字母佇列時，無效字母佇列的 ApproximateAgeOfOl |

| 指標 | 描述 |
| --- | --- |
| | destMessage 指標表示訊息移至無效字母佇列的時間 (而非傳送訊息的原始時間)。<br>• 對於 FIFO 佇列，訊息不會移至佇列的後面，因為這會破壞 FIFO 順序保證。若有設定，則該訊息會移至 DLQ。否則，會阻止該訊息群組，直到成功刪除或到期為止。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：秒<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量) |
| ApproximateNumberOfMessagesDelayed | 佇列中延遲且無法立即讀取的訊息數量。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。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：計數<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量) |

| 指標 | 描述 |
| --- | --- |
| ApproximateNumberOfMessagesNotVisible | 傳送中的訊息數。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。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：計數<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量) |
| ApproximateNumberOfMessagesVisible | 要處理的訊息數。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：計數<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量)<br>處理序的訊息數沒有限制，不過您可以將此待處理項目限制在保留期間內。 |
| NumberOfEmptyReceives [1] | 未傳回訊息的 ReceiveMessage API 呼叫的數量。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：計數<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量) |

| 指標 | 描述 |
| --- | --- |
| NumberOfMessagesDeleted [1] | 已從佇列刪除的訊息數量。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：計數<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量)<br>Amazon SQS 會針對使用有效接收控點 (包括重複刪除) 的每個成功刪除作業發出 NumberOfMessagesDeleted 指標。下列情況可能導致 NumberOfMessagesDeleted 指標值高於預期：<br><br>• 呼叫屬於相同訊息的不同接收控點上的 DeleteMessage 動作：如果訊息未在可見性逾時到期之前進行處理，訊息將可再次提供能夠處理與刪除該訊息的其他消費者使用，增加 NumberOfMessagesDeleted 指標值。<br>• 呼叫相同接收控點上的 DeleteMessage 動作：如果訊息已處理並刪除，但您使用相同接收控點呼叫 DeleteMessage 動作，將會傳回成功狀態，增加 NumberOfMessagesDeleted 指標值。 |

| 指標 | 描述 |
| --- | --- |
| NumberOfMessagesReceived [1] | 呼叫 ReceiveMessage 動作所傳回的訊息數量。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：計數<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量) |
| NumberOfMessagesSent [1] | 已新增至佇列的訊息數量。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：計數<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量) |

| 指標 | 描述 |
| --- | --- |
| SentMessageSize [1] | 已新增至佇列的訊息大小。<br><br>報告準則：如果佇列處於作用中狀態，則會報告非負值。<br><br>單位：位元組<br><br>有效統計資訊：平均、最小值、最大值、總和、資料樣本 (在 Amazon SQS 主控台顯示為範例數量)<br><br>**Note**<br>SentMessageSize 在至少將一個訊息傳送至對應佇列之前，不會在 CloudWatch 主控台中顯示為可用的測量結果。 |

[1] 這些指標是從服務的觀點計算出來的，可包括重試次數。請勿依賴這些指標的絕對值，或用它們來估算目前的佇列狀態。

Amazon SQS 指標的維度

Amazon SQS 發送到的唯一維度 CloudWatch 是QueueName。這表示所有可用的統計皆以 QueueName 進行過濾。

# Amazon SQS 的合規驗證

要瞭解 AWS 服務 是否在特定法規遵循方案範圍內，請參閱法規遵循方案範圍內的 AWS 服務，並選擇您感興趣的法規遵循方案。如需一般資訊，請參閱 AWS 法規遵循方案。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱 AWS Artifact 中的下載報告。

您使用 AWS 服務 時的法規遵循責任取決於資料的敏感度、您的公司的合規目標，以及適用的法律和法規。AWS 提供以下資源協助您處理法規遵循事宜：

- 安全與合規快速入門指南 – 這些部署指南討論在 AWS 上部署以安全及合規為重心的基準環境的架構考量和步驟。
- Amazon Web Services 的 HIPAA 安全與法規遵循架構：本白皮書說明公司可如何運用 AWS 來建立符合 HIPAA 規定的應用程式。

> **Note**
>
> 並非全部的 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源：這組手冊和指南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的責任模式。這份指南橫跨多個架構 (包含國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準組織 (ISO))，總結保護 AWS 服務的最佳實務並將指導方針對應至安全控制。
- AWS Config 開發人員指南中的使用規則評估資源：AWS Config 服務可評估您的資源組態對於內部實務、業界準則和法規的合規狀態。
- AWS Security Hub – 此 AWS 服務 可供您全面檢視 AWS 中的安全狀態。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 controls reference。
- AWS Audit Manager – 此 AWS 服務 可協助您持續稽核 AWS 使用情況，以簡化管理風險與法規與業界標準的法規遵循方式。

# Amazon SQS 的恢復能力

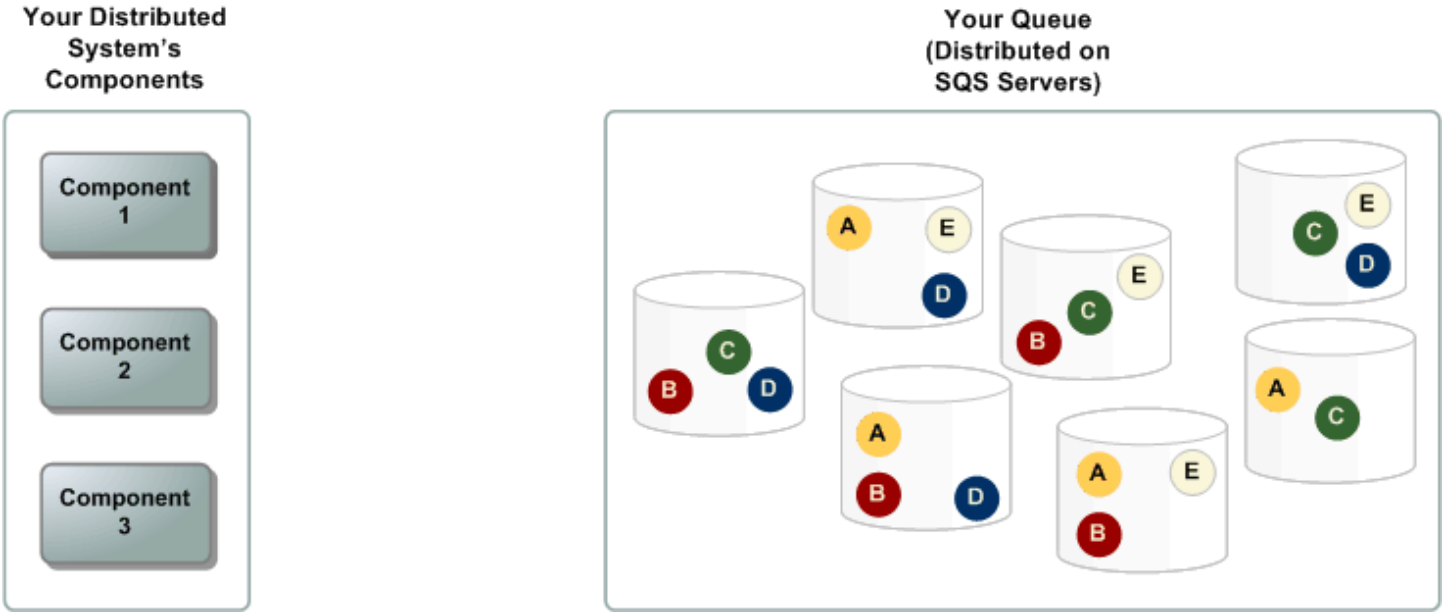
AWS 全球基礎設施是以 AWS 區域與可用區域為中心建置的。AWS區域提供多個分開且隔離的實際可用區域，並以低延遲、高輸送量和高度備援聯網功能相互連結。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。如需有關 AWS 區域與可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

除了 AWS 全球基礎設施之外，Amazon SQS 也提供分散式佇列。

## 分散式佇列

分散式傳訊系統有三個主要部分：分散式系統的元件、您的佇列 (分散在 Amazon SQS 伺服器上)，以及佇列中的訊息。

在以下情況中，您的系統有數個生產者 (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

Amazon SQS 是受到 AWS 全球網路安全程序所保護的受管服務，如 Amazon Web Services：安全程序概觀白皮書所述。

您可使用 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。這會有效隔離 AWS 網路內特定 VPC 對給定 Amazon SQS 佇列的網路存取。如需更多詳細資訊，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 提供了許多安全性功能，您應該在自己的安全性政策內容中檢閱這些功能。

> **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。

# 預防性最佳實務

以下是 Amazon SQS 的預防性安全最佳實務。

主題



## 確保佇列無法公開存取

除非您明確要求網際網路上的任何人都能夠讀取或寫入您的 Amazon SQS 佇列，否則您應確保您的佇列無法公開存取 (全世界所有人或任何已驗證的 AWS 使用者都能存取)。

- 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 存取權的應用程式和 AWS 服務使用 IAM 角色

若為要存取 Amazon SQS 佇列的應用程式或 AWS 服務 (例如 Amazon EC2)，必須在其 AWS API 請求中使用有效的 AWS 憑證。由於這些登入資料不會自動輪換，因此不得將 AWS 登入資料直接儲存在應用程式或 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，您不必將長期登入資料 (如使用者名稱、密碼和存取金鑰) 分發至 EC2 執行個體或 AWS Lambda 等 AWS 服務。相反，該角色提供應用程式在呼叫其他 AWS 資源時可以使用的臨時許可。

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 會使用 AWS Key Management Service 中管理的金鑰。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，請參閱靜態加密及金鑰管理。

## 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。

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。

# 使用 Amazon SQS API

本節提供建構 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。

具有 AWS JSON 通訊協定的 Amazon SQS API

Amazon SQS 使用 AWS JSON 通訊協定做為指定 AWS 開發套件版本上所有 Amazon SQS API 的傳輸機制。AWSJSON 協議提供更高的吞吐量，更低的延遲和更快的 application-to-application 通信。AWS與 AWS 查詢通訊協定相比，JSON 通訊協定在請求和回應的序列化/還原序列化方面更有效。如果您仍想要搭配 SQS API 使用 AWS 查詢通訊協定，請參閱支援 Amazon SQS AWS 查詢通訊協定之 AWS 開發套件版本的 Amazon SQS API 中使用的 AWS JSON 通訊協定支援哪些語言？。

Amazon SQS 使用 AWS JSON 協議在 AWS SDK 客戶端（例如，Java，Python，金 JavaScript）和 Amazon SQS 服務器之間進行通信。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 更簡單，更快，更有效地在用戶端和伺服器之間傳輸數據。

- AWS JSON 通訊協定充當 Amazon SQS 用戶端和伺服器之間的調解器。
- 伺服器無法瞭解建立 Amazon SQS 作業時所使用的程式設計語言，但它瞭解 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) 和還原序列化 (將 JSON 格式轉換為物件)。

如需有關 Amazon SQS 的 AWS JSON 通訊協定的詳細資訊，請參閱Amazon SQS AWS JSON 通訊協定常見問答集。

AWS JSON 通訊協定在指定的 AWS 開發套件版本上可用。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

主題

- 使用 AWS JSON 通訊協定發出查詢 API 請求
- 使用查詢協議發出 AWS 查詢 API 請求
- 對請求進行身分驗證
- Amazon SQS 批次動作

# 使用 AWS JSON 通訊協定發出查詢 API 請求

在本節中，您將了解如何建構 Amazon SQS 端點、提出 POST 請求以及解讀回應。

> **Note**
>
> 大多數語言變體都支援 AWS 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？。

主題

- 建構端點
- 提出 POST 請求
- 解譯 Amazon SQS JSON API 回應
- Amazon SQS AWS JSON 通訊協定常見問答集

## 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

為了回應動作請求，Amazon SQS 會傳回 JSON 資料結構，內有回應動作請求的結果。如需詳細資訊，請參閱中 Amazon Simple Queue Service API 參考 和 Amazon SQS AWS JSON 通訊協定常見問答集 的個別動作。

主題

- 成功的 JSON 回應結構
- JSON 錯誤回應結構

### 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 通訊協定常見問答集

有關與 Amazon SQS 搭配使用 AWS JSON 通訊協定的常見問答集。

### 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 做為 AWS SDK 用戶端 (例如 Java、Python、黃金 JavaScript) 和 Amazon SQS 伺服器之間進行通訊的媒介。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調解器。
- 伺服器無法瞭解建立 Amazon SQS 作業時所使用的程式設計語言，但它瞭解 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 版本以加快 Amazon SQS 的傳訊速度，請將您的 AWS SDK 升級至指定的版本，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。

以下是適用於與 Amazon SQS API 搭配使用的 AWS JSON 通訊協定之各種語言變體的 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++ | aw/ aws-sdk-cpp | 1.11.98 | 適用於 C++ 的 AWS 開發套件 |
| Golang 1.x | aw/ aws-sdk-go | v1.47.7 | 適用於 Go 的 AWS 軟體開發套件 |
| Golang 2.x | aw/ 2 aws-sdk-go-v | v1.28.0 | AWS SDK for Go V2 |
| Java 1.x | aw/ aws-sdk-java | 1.12.585 | 適用於 Java 的 AWS 開發套件 |
| Java 2.x | aw/ 2 aws-sdk-java-v | 2.21.19 | 適用於 Java 的 AWS 開發套件 |
| JavaScript v2.x | aw/ aws-sdk-js | v2.1492.0 | JavaScript 上 AWS |
| JavaScript v3.x | aw/ 3 aws-sdk-js-v | v3.447.0 | JavaScript 上 AWS |
| .NET | aw/ aws-sdk-net | 3.7.681.0 | 適用於 .NET 的 AWS 軟體開發套件 |

| 語言 | SDK 用戶端儲存庫 | 必要的 SDK 用戶端版本 | 指南 |
| --- | --- | --- | --- |
| PHP | aw/ aws-sdk-php | 3.285.2 | 適用於 PHP 的 AWS 開發套件 |
| Python-boto3 | boto/boto3 | 1.28.82 | 適用於 Python (Boto3) 的 AWS 開發套件 |
| Python-botocore | boto/botocore | 1.31.82 | 適用於 Python (Boto3) 的 AWS 開發套件 |
| awscli | AWS CLI | 1.29.82 | AWS 命令列介面 |
| Ruby | aw/ aws-sdk-ruby | 1.67.0 | 適用於 Ruby 的 AWS 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，與產生 AWS 查詢型 (也稱為 XML 型) 回應的 Amazon SQS 互動，則可能與 AWS JSON 通訊協定不相容。如果遇到任何問題，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。如需詳細資訊，請參閱 如何開始使用 Amazon SQS 的 AWS JSON 通訊協定？。AWS如何開始使用 Amazon SQS 的 AWS JSON 通訊協定？ 中列出的 SDK 版本針對 Amazon SQS API 使用 JSON 有線通訊協定。如果您將 AWS SDK 變更為舊版，您的 Amazon SQS API 將會使用 AWS 查詢。

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

Amazon SQS 支援所有一般可用的 (GA) 的 AWS SDK 的語言。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 在提供 Amazon SQS 的所有 AWS 區域都支援 AWS JSON 通訊協定。

### 使用 AWS JSON 通訊協定升級到指定的 Amazon SQS 的 AWS SDK 版本時，可以預期哪些延遲改善？

與 AWS 查詢通訊協定相比，AWS JSON 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。根據針對 5 KB 訊息承載的AWS效能測試，Amazon SQS 的 JSON 通訊協定可將 end-to-end 訊息處理延遲降低多達 23%，並減少應用程式用戶端 CPU 和記憶體使用量。

### AWS 查詢通訊協定會被棄用嗎？

AWS 查詢通訊協定會繼續受到支援。只要您的 AWS SDK 版本設定為 如何開始使用 Amazon SQS 的 AWS JSON 通訊協定？ 中列出的版本之外的任何先前版本，您就可以繼續使用 AWS 查詢通訊協定。

### 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 使用 AWS JSON 通訊協定發出查詢 API 請求。

# 使用查詢協議發出 AWS 查詢 API 請求

在本節中，您將了解如何建構 Amazon SQS 端點、提出 GET 和 POST 請求以及解讀回應。

主題



## 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，請參閱中的以下頁面：Amazon Web Services 一般參考

- 區域端點
- Amazon Simple Queue Service 端點和配額

每個 Amazon SQS 端點都是完全獨立的。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 請求的簽章。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。每個參數之間用 & 符號 (&) 隔開，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

做為對於動作請求的回應，Amazon SQS 會傳回內有回應動作請求的結果的 XML 資料結構。如需詳細資訊，請參閱 Amazon Simple Queue Service API 參考。

主題

- 成功的 XML 回應結構

- XML 錯誤回應結構

## 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 *ActionName*Response)。

此元素內含以下子元素：

- ***ActionName*Result** – 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** - 內含 RequestId，後者又內含請求的全域唯一識別符 (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-4c88daeea257</RequestId>
</ErrorResponse>
```

# 對請求進行身分驗證

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，AWS 會確認生產者的身分以及生產者是否有使用 AWS 註冊 (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，AWS 將按照以下程序進行：

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。

主題



## 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：

- 用於辨識您 AWS 帳戶 身分的 AWS 存取金鑰 ID，AWS 將用以查詢您的私密存取金鑰。
- 使用您的私密存取金鑰計算得出的 HMAC-SHA 請求簽章 (私密存取金鑰只有您和 AWS 知道；如需詳細資訊，請參閱 RFC2104)。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更佳的 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);
    }
}
```

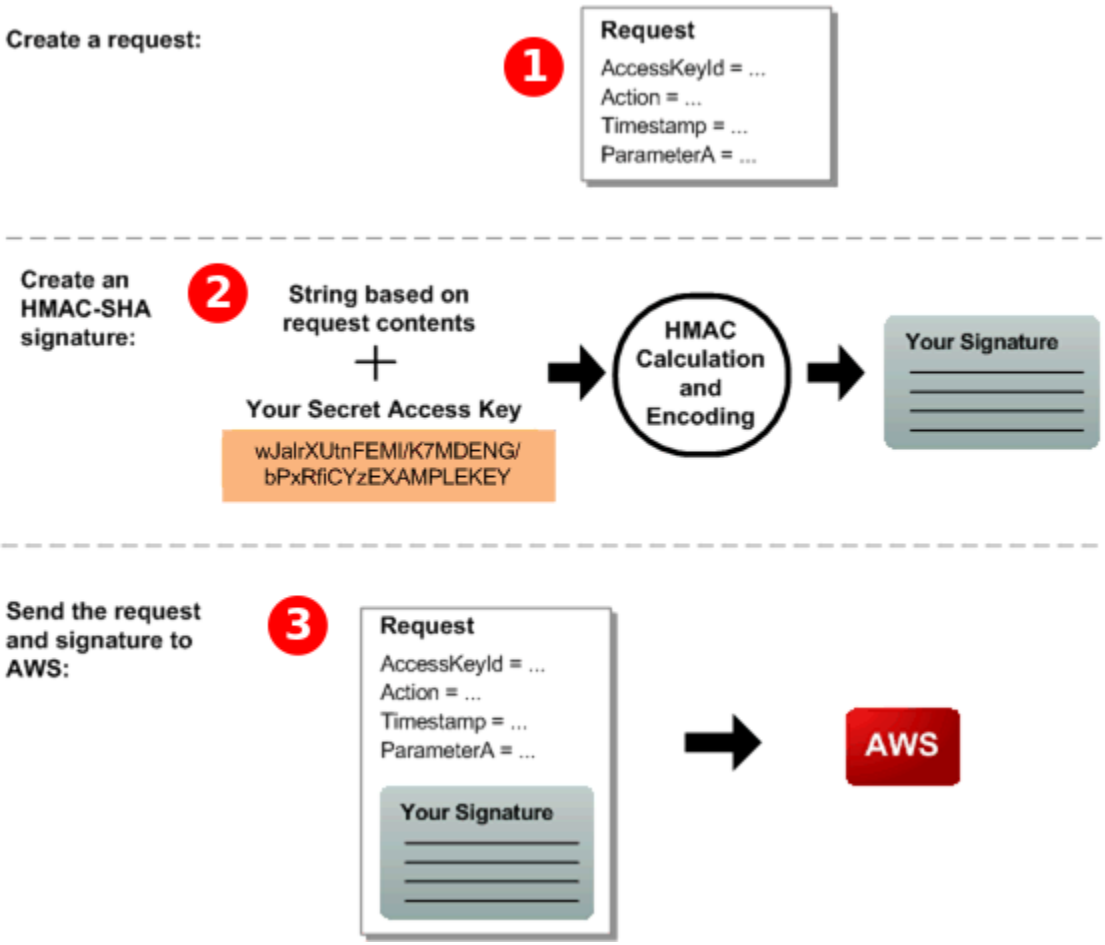
- 請求的時間戳記 (或過期時間)。在請求中使用的時間戳記必須是 dateTime 物件，有完整的日期，包含小時、分鐘、秒鐘。例如：2007-01-31T23:59:59Z。儘管並非必要，但建議您使用國際標準時間 (格林威治標準時間) 時區提供此物件。

> **Note**
>
> 請確定您的伺服器時間設定正確。如果您指定了時間戳記 (而不是過期時間)，請求便會在指定的時間過後 15 分鐘自動過期 (若請求的時間戳記比 AWS 伺服器目前的時間早 15 分鐘以上，AWS 便不會處理請求)。

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

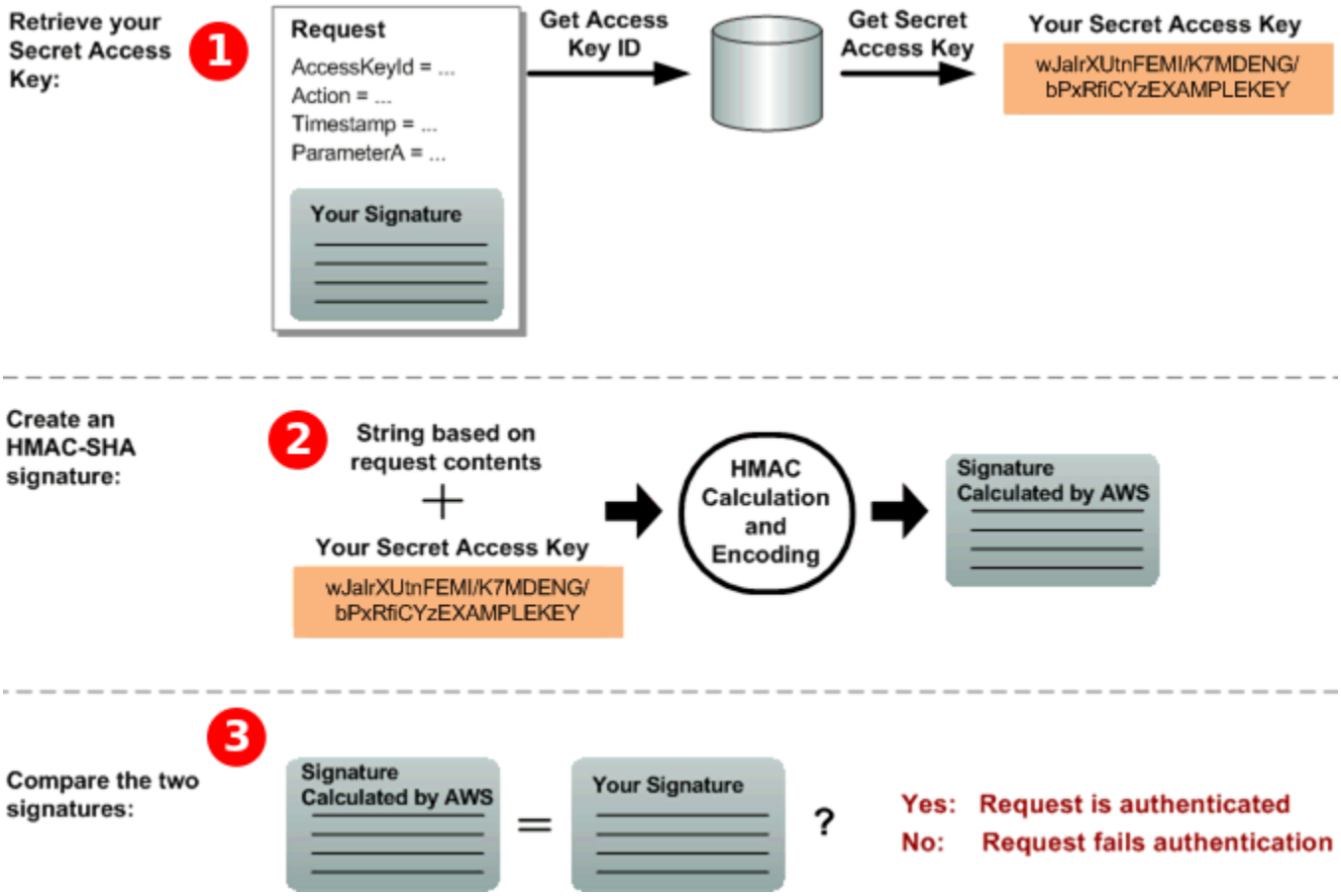
以下是使用 HMAC-SHA 請求簽章對 AWS 請求進行身分驗證時必須經歷的過程。




1. 建構要向 AWS 提出的請求。
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. 在請求中加入簽章和您的存取金鑰 ID，再將請求傳送至 AWS。

## 第 2 部分：來自 AWS 的回應

AWS 開始以下程序做出回應。




1. AWS 使用存取金鑰 ID 查詢您的私密存取金鑰。
2. AWS 使用與您用於計算請求中傳送之簽章相同的演算法，根據請求資料及私密存取金鑰產生簽章。
3. 以下其中一種情況將發生：
   - 若 AWS 產生的簽章與請求中傳送的簽章相符，AWS 即會將請求視為通過驗證。
   - 若比對失敗，則會捨棄該請求，且 AWS 將傳回錯誤。

# Amazon SQS 批次動作

為了降低成本或要透過單次動作處理多達 10 則訊息，您可以使用下列動作：

- SendMessageBatch
- DeleteMessageBatch
- ChangeMessageVisibilityBatch

您可以使用查詢 API 或是支援 Amazon SQS 批次動作的 AWS 開發套件，善加利用批次功能。

> **Note**
>
> 單次 SendMessageBatch 呼叫所傳送的訊息總量不得超過 262,144 位元組 (256 KB)。
> 您無法明示設定對 SendMessageBatch、DeleteMessageBatch 或 ChangeMessageVisibilityBatch 的許可。設定 SendMessage、DeleteMessage 或 ChangeMessageVisibility 的許可就會對相應批次版本的動作設定許可。
> Amazon SQS 主控台不支援批次動作。

主題



## 啟用用戶端緩衝與請求批次處理

AWS SDK for Java 包含可存取 Amazon SQS 的 AmazonSQSBufferedAsyncClient。此用戶端可使用用戶端緩衝功能 (即用戶端發出的呼叫會先經過緩衝處理)，再以批次請求的形式送往 Amazon SQS，如此一來請求的批次處理便更加簡易。

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。如需更多詳細資訊，請參閱 利用水平擴展和動作批次處理提高傳輸量。

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。

主題

## 使用 AmazonSQSBufferedAsyncClient

開始之前，請完成 設定 Amazon SQS 中的步驟。

> **Important**
>
> AWS SDK for Java 2.x 目前與 AmazonSQSBufferedAsyncClient 不相容。

您可以根據 AmazonSQSAsyncClient 建立一個新的 AmazonSQSBufferedAsyncClient，例如：

```
// 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.setMessageBody( body );
request.setQueueUrl(res.getQueueUrl());

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

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

## 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，AmazonSQSBufferedAsyncClient 在消費訊息時會嘗試使用長輪詢。 |
| longPollWaitTimeoutSeconds | 20 秒 | 在傳回空的接收結果前，ReceiveMessage 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)。<br>**Note**<br>停用長輪詢時，此設定沒有作用。 |
| maxBatchOpenMs | 200 毫秒 | |

| 參數 | 預設值 | 描述 |
| --- | --- | --- |
| | | 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)。<br><br>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)。<br><br>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。 |
| `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。<br><br>**Note**<br>Amazon SQS 允許的上限值為每批次 10 個請求。 |

| 參數 | 預設值 | 描述 |
| --- | --- | --- |
| maxBatchSizeBytes | 256 KB | 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。<br><br>**Note**<br>Amazon SQS 允許的上限值為 256 KB。 |
| maxDoneReceiveBatches | 10 個批次 | AmazonSQSBufferedAsyncClient 在用戶端預取並存放的接收批次數上限。<br><br>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)。<br><br>**Note**<br>0 表示所有的訊息預取動作都將停用，而僅按需求消費訊息。 |

| 參數 | 預設值 | 描述 |
| --- | --- | --- |
| maxInflightOutboundBatches | 5 個批次 | 可同時處理的活動中傳出批次的上限。<br><br>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。 |
| maxInflightReceiveBatches | 10 個批次 | 可同時處理的活動中接收批次的上限。<br><br>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 AmazonSQSBufferedAsyncClient 所耗用的執行緒越多。<br><br>**Note**<br>0 表示所有的訊息預取動作都將停用，而僅按需求消費訊息。 |

| 參數 | 預設值 | 描述 |
| --- | --- | --- |
| visibilityTimeoutSeconds | -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。<br><br>**Note**<br>-1 表示將選擇佇列預設的設定。<br>可見性逾時不可設為 0。 |

## 利用水平擴展和動作批次處理提高傳輸量

Amazon SQS 佇列可提供極高的輸送量。如需輸送量配額的詳細資訊，請參閱 訊息相關配額。

若要達到高傳輸量，必須水平擴展訊息生產者和消費者 (增加更多生產者和消費者)。

主題

- 水平擴展
- 動作批次處理
- 單次操作和批次請求實際可行的 Java 範例

### 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。

> **Note**
>
> 進行水平擴展時，您必須確定您的 Amazon SQS 用戶端有足夠的連線和執行緒，能支援傳送請求和接收回應的並存訊息生產者與消費者的數目。舉例而言，預設中 AWS SDK for Java `AmazonSQSClient` 類別的執行個體最多可維護大約 50 條連至 Amazon SQS 的連線。若要建立更多的並存生產者和消費者，必須調整 `AmazonSQSClientBuilder` 物件所允許的生產者和消費者執行緒數目上限，例如：
>
> ```
> final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
>         .withClientConfiguration(new ClientConfiguration()
>                 .withMaxConnections(producerCount + consumerCount))
>         .build();
> ```
>
> 對於 `AmazonSQSAsyncClient`，您還必須確保有足夠的執行緒可用。
> 此示例僅適用於 Java 版本 1.x。

## 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 2010-2022 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。

# 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 | 可搭配佇列使用的 AWS CLI 命令的說明。 |
| 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 | AWS Premium Support 相關資訊的主要網頁，本支援服務是一對一的快速回應支援管道，可協助您在 AWS 基礎設施服務建立並執行應用程式。 |

# 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。

服務功能會逐步發佈至提供服務的 AWS 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資訊，並訂閱更新通知，請參閱 AWS 的最新消息?

| 變更 | 描述 | 日期 |
| --- | --- | --- |
| 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 |
| 已新增以下章節，說明 Amazon SQS 的 AWS 受管政策和以下政策的更新 | 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 | 2021 年 12 月 29 日 |

| | | |
|---|---|---|
| | 時，會根據 KMS 政策中的條件進行檢查。 | |
| 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 支持 listQueues 和分頁 listDeadLetterSourceQueues | 您可以指定要從 listQueues 或listDeadLetterSourceQueues要求傳回的結果數目上限。 | 2020 年 6 月 22 日 |

| | | |
|---|---|---|
| Amazon SQS 在所有AWS 區域支援 1 分鐘的 Amazon CloudWatch 指標，但 AWS GovCloud (美國) 區域除外 | Amazon SQS 的一分鐘 CloudWatch 指標適用於所有區域 (區域除外)。AWS GovCloud (US) | 2020 年 1 月 9 日 |
| Amazon SQS 支援 1 分鐘 CloudWatch 指標 | Amazon SQS 的一分鐘 CloudWatch 指標目前僅在下列區域提供：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 |
| 現已提供適用於 Amazon SQS FIFO 佇列的 AWS Lambda 觸發 | 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 |
| 適用於 Amazon SQS 的 Amazon 虛擬私有雲端 (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) 區域的 Amazon VPC 將訊息傳送到 Amazon SQS 佇列。 | 2019 年 9 月 5 日 |

| | | |
|---|---|---|
| Amazon SQS 允許使用訊息系統屬性對使用 AWS X-Ray 的佇列進行疑難排解 | 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 |
| 適用於 Amazon SQS 的 SSE 可在 AWS GovCloud (美國東部) 區域使用 | 適用於 Amazon SQS 的伺服器端加密 (SSE) 可在 AWS GovCloud (美國東部) 區域使用。 | 2019 年 6 月 20 日 |

| | | |
|---|---|---|
| 先進先出佇列適用於亞太區域 (香港)、中國 (北京)、(美國東部) 和 AWS GovCloud AWS GovCloud (美國西部) 區域 | FIFO 佇列適用於亞太區域 (香港)、中國 (北京)、(美國東部) 和 AWS GovCloud AWS GovCloud (美國西部) 區域。 | 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 |

# AWS 詞彙表

如需最新的 AWS 術語，請參閱《AWS 詞彙表 參考》中的 AWS 詞彙表。

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。