aws

開發人員指南

Amazon Kinesis Data Streams

Amazon Kinesis Data Streams: 開發人員指南

Table of Contents

什麼是 Amazon Kinesis Data Streams？

您可以使用 Amazon Kinesis Data Streams 即時收集並處理大型的資料記錄串流。您可以建立資料處理應用程式，又稱為 Kinesis Data Streams 應用程式。典型的 Kinesis Data Streams 應用程式會從資料串流讀取資料記錄形式的資料。這類應用程式可使用 Kinesis Client Library，而且能在 Amazon EC2 執行個體上執行。您可以將處理過的記錄傳送至儀表板、使用這些記錄產生提醒、動態變更定價和廣告策略，或將資料傳送至其他各種 AWS 服務。如需 Kinesis Data Streams 功能和定價的相關資訊，請參閱 Amazon Kinesis Data Streams。

Kinesis Data Streams 是 Kinesis 串流資料平台的一部分，以及 Firehose、Kinesis Video Streams 和 Managed Service for Apache Flink。

如需 AWS 大數據解決方案的詳細資訊，請參閱 大數據 AWS。如需 AWS 串流資料解決方案的詳細資訊，請參閱什麼是串流資料？。

主題

我可以使用 Kinesis Data Streams 做什麼？

Kinesis Data Streams 可讓您快速且持續擷取和彙總資料。使用的資料類型可包括 IT 基礎架構日誌資料、應用程式日誌、社交媒體、市場資料摘要和 web 點擊流資料。由於資料擷取和處理的回應時間為即時，處理通常是輕量型。

以下是使用 Kinesis Data Streams 的典型案例：

加速日誌和資料饋送的擷取與處理

生產者可直接推送資料至串流。例如，推送系統及應用程式日誌，僅需數秒後即可供進行處理。這可防止日誌資料因前端或應用程式伺服器故障造成遺失。Kinesis Data Streams 提供加速資料饋送的擷取，因為您在提交資料以備擷取之前，未由伺服器上批次處理資料。

即時指標與報告

您可以使用收集到 Kinesis Data Streams 的資料進行即時的簡易資料分析與報告。例如，您的資料處理應用程式可處理系統及應用程式日誌的指標和報告，因為資料是以串流方式傳入，而非等待接收各個批次的資料。

即時資料分析

此案例結合了並行處理的強大功能與即時資料的價值。例如，即時處理網站點擊流，然後使用多個不同的 Kinesis Data Streams 應用程式並行執行，以便分析網站可用性參與度。

複雜的串流處理

您可以建立 Kinesis Data Streams 應用程式和資料串流的有向無環圖 (DAG)。這通常涉及到從多個 Kinesis Data Streams 應用程式將資料放入另一串流，以供其他 Kinesis Data Streams 應用程式進行下游處理。

使用 Kinesis Data Streams 的優點

儘管使用 Kinesis Data Streams 可解決各種串流資料問題，但其常見用途是即時彙整資料，然後將彙整資料載入資料倉儲或對應縮減叢集。

資料將放入 Kinesis 資料串流，以確保耐用性與彈性。從記錄放入串流到記錄可供擷取的這段期間，延遲 (put-to-get 延遲) 通常不到 1 秒。換言之，資料一旦加入後，Kinesis Data Streams 應用程式便幾乎能立即開始從串流取用資料。Kinesis Data Streams 的受管服務層面可減輕您建立和執行資料擷取管道的操作負擔。您可以建立串流對應縮減類型的應用程式。Kinesis Data Streams 的彈性讓您能夠擴展或縮減串流規模，以確保資料記錄過期前絕不會遺失任何記錄。

多個 Kinesis Data Streams 應用程式可以從單一串流取用資料，使得多項動作 (如封存和處理) 能夠同時各自進行。例如，兩個應用程式可從同一串流讀取資料。第一個應用程式計算累計彙整值並更新 Amazon DynamoDB 資料表，第二個應用程式則壓縮資料後封存至 Amazon Simple Storage Service (Amazon S3) 之類的資料存放區。接著，儀表板將讀取具有累計彙整值的 DynamoDB 資料表以取得最新的報告。

Kinesis Client Library 支援以容錯方式從串流取用資料，並且為 Kinesis Data Streams 應用程式提供擴展支援。

相關服務

如需如何使用 Amazon EMR 叢集直接讀取及處理 Kinesis 資料串流的資訊，請參閱 Kinesis 連接器。

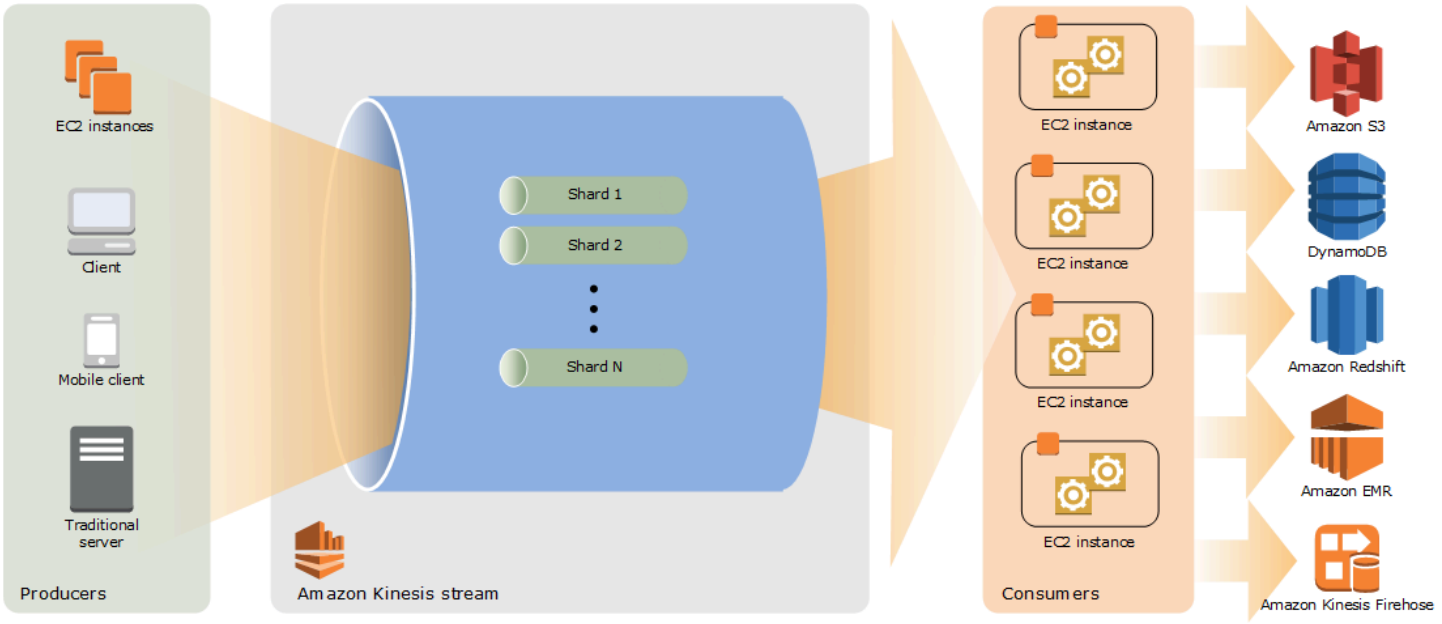
Amazon Kinesis Data Streams 術語和概念

開始使用 Amazon Kinesis Data Streams 之前，請先了解其架構和術語。

主題

檢閱 Kinesis Data Streams 的高階架構

下圖說明 Kinesis Data Streams 的概要架構。生產者會持續推送資料至 Kinesis Data Streams，而取用者將即時處理資料。消費者（例如在 Amazon EC2 或 Amazon Data Firehose 交付串流上執行的自訂應用程式）可以使用 Amazon DynamoDB、Amazon Redshift 或 Amazon S3 等 AWS 服務來儲存其結果。



熟悉 Kinesis Data Streams 的術語

Kinesis Data Stream

Kinesis 資料串流是一組碎片。每個碎片都有一連串的資料記錄。每筆資料記錄具有由 Kinesis Data Streams 指派的序號。

資料記錄

Kinesis 資料串流存放資料的單位是資料記錄。資料記錄由序號、分割區索引鍵和資料 Blob (不可變的位元組序列) 所組成。Kinesis Data Streams 絲毫不會檢查、解譯或變更 Blob 中的資料。資料 Blob 最多可達 1 MB。

容量模式

資料串流容量模式會決定資料串流容量的管理方式，以及如何針對資料串流使用量收費。目前，在 Kinesis Data Streams 中，您可以選擇隨需模式和資料串流的佈建模式。如需詳細資訊，請參閱選擇要在 中串流的正確模式。

使用隨需模式時，Kinesis Data Streams 會自動管理碎片，以提供必要的輸送量。您只需為使用的實際輸送量付費，Kinesis Data Streams 會在工作負載增加或減少時自動滿足您的工作負載輸送量需求。如需詳細資訊，請參閱隨需標準模式功能和使用案例。

採用佈建模式，必須指定資料串流的碎片數目。資料串流的總容量是其碎片容量的總和。您可以依需要增加或減少資料串流中的碎片數目，並按小時費率向您收取碎片數目費用。如需詳細資訊，請參閱佈建模式功能和使用案例。

保留期間

保留期間是資料記錄加入至串流之後可供存取的時間長度。串流建立後，其保留期間將設為預設值 24 小時。您可以使用 IncreaseStreamRetentionPeriod 操作將保留期間增加到最長 8760 小時 (365 天)，使用 DecreaseStreamRetentionPeriod 操作將保留期間減少到最短 24 小時。串流的保留期間若設為 24 小時以上，即需支付額外的費用。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 定價。

生產者

生產者將記錄放入 Amazon Kinesis Data Streams。例如，傳送日誌資料至串流的 web 伺服器即是生產者。

消費者

取用者會從 Amazon Kinesis Data Streams 取得記錄，並加以處理。這些消費者稱為 Amazon Kinesis Data Streams 應用程式。

Amazon Kinesis Data Streams 應用程式

Amazon Kinesis Data Streams 應用程式是通常在 EC2 執行個體機群上執行的串流取用者。

您可以開發的消費者有兩種：共用廣發功能消費者和強化廣發功能消費者。若要了解此兩者間的差異以及如何建立每一種消費者，請參閱從 Amazon Kinesis Data Streams 讀取資料。

Kinesis Data Streams 應用程式的輸出可以是另一串流的輸入，這使您能夠建立複雜的拓撲以便即時處理資料。應用程式也可以將資料傳送至各種 AWS 其他服務。單一串流可以有多個應用程式，每個應用程式均能同時各自從串流取用資料。

碎片

碎片是串流中的資料記錄的唯一識別序列。串流由一個或多個碎片所組成，每個碎片提供固定單位的容量。每個碎片最多可支援每秒 5 筆交易進行讀取，最大總資料讀取速率為每秒 2 MB，寫入時最多可支援每秒 1,000 筆記錄，最大總資料寫入速率為每秒 1 MB (包括分割區索引鍵)。串流的資料容量是您為該串流指定的碎片數目的函數。串流的總容量是其碎片容量的總和。

如果您的資料速率增加，您可以增加或減少配置給串流的碎片數目。如需詳細資訊，請參閱重新分片串流。

分割區索引鍵

分割區索引鍵用於依據串流中的碎片將資料分組。Kinesis Data Streams 會將屬於某一串流的資料記錄分隔到多個碎片中。本服務使用與每筆資料記錄相關聯的分割區索引鍵，判斷特定的資料記錄屬於哪個碎片。分割區索引鍵是 Unicode 字串，每個索引鍵的長度上限為 256 個字元。MD5 雜湊函數用於將分割區索引鍵對應到 128 位元整數值，並使用碎片的雜湊索引鍵範圍將相關聯的資料記錄對應到碎片。應用程式將資料放入串流時必須指定分割區索引鍵。

序號

每筆資料記錄在其碎片中的每個分割區索引鍵都有獨一無二的序號。Kinesis Data Streams 會在您使用 `client.putRecords` 或 `client.putRecord` 寫入串流之後指派序號。同一分割區索引鍵的序號通常會隨著時間而增加。逐次寫入請求的間隔期間愈長，序號將變得愈大。

> **Note**
>
> 序號不能用做為同一串流中各資料集的索引。若要按照邏輯分隔資料集，請使用分割區索引鍵或為每個資料集建立個別串流。

Kinesis Client Library

Kinesis Client Library 會編譯至您的應用程式，以允許容錯使用來自串流的資料。Kinesis Client Library 會確保每個碎片都有記錄處理器正在執行並處理該碎片。本程式庫還可簡化從串流讀取資料的過程。Kinesis Client Library 使用 Amazon DynamoDB 資料表來存放與資料消耗相關的中繼資料。它為每個正在處理資料的應用程式建立三個資料表。如需詳細資訊，請參閱使用 Kinesis 用戶端程式庫。

Application Name (應用程式名稱)

Amazon Kinesis Data Streams 應用程式的名稱可識別該應用程式。您的每個應用程式都必須有一個唯一的名稱，範圍限定於應用程式所使用的 AWS 帳戶和區域。此名稱將做為 Amazon DynamoDB 中控制資料表的名稱以及 Amazon CloudWatch 指標的命名空間。

伺服器端加密

Amazon Kinesis Data Streams 可在生產者將機密資料輸入串流時自動為資料加密。Kinesis Data Streams 使用 AWS KMS 主金鑰進行加密。如需詳細資訊，請參閱Amazon Kinesis Data Streams 中的資料保護。

> **Note**
>
> 若要讀取或寫入已加密的串流，生產者和消費者應用程式必須具備存取主金鑰的許可。如需如何對生產者和消費者應用程式授予許可的資訊，請參閱the section called “使用使用者產生之 KMS 金鑰的許可”。

> **Note**
>
> 使用伺服器端加密會產生 AWS Key Management Service (AWS KMS) 成本。如需更多資訊，請參閱 AWS Key Management Service 定價。

配額和限制

下表說明 Amazon Kinesis Data Streams 的串流和碎片配額和限制。

配額	隨需模式	佈建模式
資料串流數目	您 AWS 帳戶中的串流數量沒有上限配額。根據預設，透過使用隨需容量模式，您可以建立最多 50 個資料串流。如果您需要提高此配額，請提出支援票證。	帳戶中佈建模式的串流數量沒有上限配額。
碎片數量	沒有上限。碎片數目取決於擷取的資料量和您所需要的輸送量層級。Kinesis Data Streams 會自動擴展碎片數目，以回應資料量和流量的變更。	沒有上限。下列 AWS 帳戶項目的預設碎片配額為每個 20，000 個碎片 AWS 區域： • 美國東部 (維吉尼亞北部) • 美國西部 (奧勒岡) • 歐洲 (愛爾蘭) 對於所有其他區域，預設碎片配額為每個 1，000 或 6，000 個碎片 AWS 帳戶。您可以透過 Service Quotas 主控台檢視您帳戶的碎片配額和使用率，網址為 https：//https://console.aws.amazon.com/servicequotas/。 若要請求提高碎片配額，請使用 Service Quotas 主控台 或 AWS CLI。如需詳細資訊，請參閱請求增加配額。

配額	隨需模式	佈建模式
資料串流輸送量	依預設，使用隨需容量模式建立的新資料串流具有 4 MB/s 的寫入和 8 MB/s 的讀取輸送量。在美國東部（維吉尼亞北部）、美國西部（奧勒岡）和歐洲（愛爾蘭）AWS 區域，具有隨需容量模式的資料串流可擴展至每秒 10 GB 的寫入和每秒 20 GB 的讀取輸送量。對於其他 區域，具有隨需容量模式的資料串流可擴展至 200 MB/s 的寫入和 400 MB/s 的讀取輸送量。如果您需要為這些區域增加高達 10 GB/s 的寫入和 20 GB/s 的讀取容量，請提交支援票證。	沒有上限。輸送量上限取決於為串流所佈建的碎片數目。每個碎片最多可支援每秒 1 MB 或每秒 1,000 筆記錄的寫入輸送量，或最高每秒 2 MB 或每秒 2,000 筆記錄的讀取輸送量。如果您需要更多擷取容量，您可以使用 AWS 管理主控台或 UpdateShardCount API 輕鬆擴展串流中的碎片數量。
資料承載大小	之前記錄的資料承載大小上限為 base64-encoding 10 MiB。Kinesis 旨在使用高載容量處理間歇性大型記錄（大小為 1-10MiB)。	
GetRecords 交易大小	GetRecords 每次呼叫可從單一碎片擷取最多 10 MB 的資料，每次呼叫最多 10,000 筆記錄。每呼叫一次 GetRecords 即計為一筆讀取交易。每個碎片每秒可支援最多 5 筆讀取交易。每筆讀取交易可提供多達 10,000 筆記錄，每筆交易的配額上限為 10 MiB。	
每個碎片的資料讀取率	每個碎片透過 GetRecords 每秒可支援最多 2 MB 的總資料讀取速率。如果呼叫 GetRecords 傳回 10 MB，在接下來的 5 秒內發出的後續呼叫將擲回例外狀況。	
每個資料串流已註冊的取用者數目	使用 Kinesis 隨需優勢模式，您最多可以建立 50 個已註冊的消費者（增強廣發）。使用 Kinesis 隨需標準和 Kinesis 佈建模式，您可以為每個資料串流建立最多 20 個已註冊消費者（增強廣發限制）。	

配額	隨需模式	佈建模式
在佈建和隨需模式之間切換	對於您 AWS 帳戶中的每個資料串流，您可以在 24 小時內在隨需和佈建容量模式之間切換兩次。	

API 限制

如同大多數 AWS APIs，Kinesis Data Streams API 操作會受到速率限制。下列限制適用於每個區域的每個 AWS 帳戶。如需有關 Kinesis Data Streams API 的詳細資訊，請參閱 Amazon Kinesis API 參考。

KDS 控制平面 API 限制

下列各節將描述 KDS 控制平面 API 的限制。KDS 控制平面 APIs 可讓您建立和管理資料串流。這些限制適用於每個 AWS 區域每個帳戶。

控制平面 API 限制

API	API 呼叫限制	每個帳戶/串流	Description
AddTagsToStream	每秒 5 次交易 (TPS)	每個帳戶	每個資料串流 50 個標籤
CreateStream	5 TPS	每個帳戶	帳戶中可擁有的串流數目沒有配額上限。當您嘗試執行下列其中一項操作，您會在發出 `CreateStream` 請求時收到 `LimitExceededException`： • 在任何時間點有超過五個處於 `CREATING` 狀態的串流。

API	API 呼叫限制	每個帳戶/串流	Description
			• 建立超過帳戶授權數量的碎片。
DecreaseStreamRetentionPeriod	5 TPS	每個串流	資料串流保留期間的最小值為 24 小時。
DeleteResourcePolicy	5 TPS	每個帳戶	如果您需要增加此限制，請提出支援票證。
DeleteStream	5 TPS	每個帳戶	
DeregisterStreamConsumer	5 TPS	每個串流	
DescribeAccountSettings	5 TPS	每個帳戶	
DescribeLimits	1 TPS	每個帳戶	
DescribeStream	10 TPS	每個帳戶	
DescribeStreamConsumer	20 TPS	每個串流	
DescribeStreamSummary	20 TPS	每個帳戶	
DisableEnhancedMonitoring	5 TPS	每個串流	
EnableEnhancedMonitoring	5 TPS	每個串流	
GetResourcePolicy	5 TPS	每個帳戶	如果您需要增加此限制，請提出支援票證。

API	API 呼叫限制	每個帳戶/串流	Description
IncreaseStreamRetentionPeriod	5 TPS	每個串流	串流保留期間的最大值為 8760 小時 (365 天)。
ListShards	1000 TPS	每個串流	
ListStreamConsumers	5 TPS	每個串流	
ListStreams	5 TPS	每個帳戶	
ListTagsForStream	5 TPS	每個串流	
MergeShards	5 TPS	每個串流	僅適用於已佈建的項目。
PutResourcePolicy	5 TPS	每個帳戶	如果您需要增加此限制，請提出支援票證。
RegisterStreamConsumer	5 TPS	每個串流	您可以對每個資料串流註冊最多 20 個消費者。一個指定消費者一次只能註冊一個資料串流。只能同時建立 5 個消費者。換句話說，您不能同時有超過 5 個處於 CREATING 狀態的取用者。
RemoveTagsFromStream	5 TPS	每個串流	
SplitShard	5 TPS	每個串流	僅適用於已佈建的項目

API	API 呼叫限制	每個帳戶/串流	Description
StartStreamEncryption		每個串流	您可以連續 24 小時成功套用新的 AWS KMS 金鑰進行伺服器端加密 25 次。
StopStreamEncryption		每個串流	在輪換的 24 小時期間，您可以成功停用伺服器端加密 25 次。
UpdateShardCount		每個串流	僅適用於已佈建的項目。碎片數量的預設限制為 10，000。還有此 API 的其他限制。如需詳細資訊，請參閱 UpdateShardCount。
UpdateStreamMode		每個串流	對於您 AWS 帳戶中的每個資料串流，您可以在 24 小時內在隨需和佈建容量模式之間切換兩次。
UpdateStreamWarmThroughput	5 TPS	每個帳戶	可設定的暖輸送量上限是帳戶和區域的隨需模式資料串流輸送量限制。
UpdateAccountSettings	5 TPS	每個帳戶	啟用或停用帳戶設定，例如隨需優勢模式。

KDS 資料平面 API 限制

下節描述 KDS 資料平面 API 的限制。KDS 資料平面 API 可讓您使用資料串流來即時收集和處理資料記錄。這些限制適用於資料串流內的每個碎片。

資料平面 API 限制

API	API 呼叫限制	承載限制	其他詳細資訊
GetRecords	5 TPS	每個呼叫可傳回的記錄數目上限為 10,000。GetRecords 可傳回的資料大小上限為 10 MB。	如果呼叫傳回了此資料數量，在接下來 5 秒內發出的後續呼叫會擲回 ProvisionedThroughputExceededException 。如果串流的佈建輸送量不足，在接下來的 1 秒內進行的後續呼叫會擲出 ProvisionedThroughputExceededException 。
GetShardIterator	5 TPS		分片反覆運算器會在傳回給要求者之後 5 分鐘到期。如果太常發出 GetShardIterator 請求，則會收到 ProvisionedThroughputExceededException。
PutRecord	1000 TPS	每個碎片最多可支援每秒 1，000 筆記錄的寫入，最多可支援每秒 10MiB 的資料寫入總數。	Kinesis 旨在使用高載容量處理間歇性大型記錄（大小為 1-10MiB)。

API	API 呼叫限制	承載限制	其他詳細資訊
PutRecords		每個 PutRecords 請求最高可支援 500 筆記錄。請求中的每個記錄最多可達 10 MiB，整個請求的上限為 10 MiB，包括分割區索引鍵。每個碎片可支援最高每秒 1,000 筆記錄的寫入數目，最高每秒 1 MB 的總計資料寫入上限。	Kinesis 旨在使用高載容量處理間歇性大型記錄（大小為 1-10MiB)。
SubscribeToShard	您可以在每個碎片每個註冊消費者每秒對 SubscribeToShard 發出一個呼叫。		如果您使用相同 ConsumerARN 和 ShardId 在成功呼叫的 5 秒內再次呼叫 SubscribeToShard，將會收到 ResourceInUseException。

提高配額

如果配額可調整，您可以使用 Service Quotas 要求增加配額。有些請求會自動解決，有些則提交給 AWS Support。您可以追蹤提交給 AWS Support 的配額增加請求狀態。提高 Service Quotas 的請求不會獲得優先順序支援。如果您有緊急請求，請聯絡 AWS Support。如需詳細資訊，請參閱什麼是 Service Quotas？

若要請求增加服務配額，請遵循請求增加配額中概述的程序。

完成設定 Amazon Kinesis Data Streams 的先決條件

第一次使用 Amazon Kinesis Data Streams 之前，請先完成下列任務以設定您的環境。

任務

- 註冊 AWS
- 下載程式庫和工具
- 設定您的開發環境

註冊 AWS

當您註冊 Amazon Web Services (AWS) 時， AWS 您的帳戶會自動註冊所有 服務 AWS，包括 Kinesis Data Streams。您只需支付實際使用服務的費用。

如果您已經有 AWS 帳戶，請跳到下一個任務。若您尚未擁有 AWS 帳戶，請使用下列程序建立帳戶。

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或簡訊，並在電話鍵盤輸入驗證碼。

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

下載程式庫和工具

以下程式庫和工具可協助您使用 Kinesis Data Streams：

- Amazon Kinesis API 參考是 Kinesis Data Streams 所支援的一組基本操作。如需使用 Java 程式碼執行基本操作的詳細資訊，請參閱以下主題：
  - 使用 Amazon Kinesis Data Streams API 搭配 開發生產者 適用於 Java 的 AWS SDK
  - 使用 開發消費者 適用於 Java 的 AWS SDK
  - 建立和管理 Kinesis 資料串流

- Gohttps://docs.aws.amazon.com/sdk-for-go/api/service/kinesis/、Java、JavaScript、.NET、PHP、Python 和 Ruby AWS SDKs 包含 Kinesis Data Streams 支援和範例。如果您的 版本 適用於 Java 的 AWS SDK 不包含 Kinesis Data Streams 的範例，您也可以從 GitHub 下載。
- Kinesis Client Library (KCL) 提供一套易用的程式設計模型以處理資料。KCL 可協助您以 Java、Node.js、.NET、Python 和 Ruby 快速上手使用 Kinesis Data Streams。如需詳細資訊，請參閱從串流讀取資料。
- AWS Command Line Interface 支援 Kinesis Data Streams。 AWS CLI 可讓您從命令列控制多個 AWS 服務，並透過指令碼將其自動化。

設定您的開發環境

若要使用 KCL，請確認您的 Java 開發環境符合以下要求：

- Java 1.7 (Java SE 7 JDK) 或更新版本。您可前往 Oracle 網站從 Java SE 下載頁面下載最新版 Java 軟體。
- Apache Commons 套件 (程式碼、HTTP 用戶端和日誌記錄)
- Jackson JSON 處理器

請注意，適用於 Java 的 AWS SDK 已將 Apache Commons 和 Jackson 納入第三方資料夾。不過，適用於 Java 的開發套件適用於 Java 1.6，而 Kinesis Client Library 則需要 Java 1.7。

使用 AWS CLI 執行 Amazon Kinesis Data Streams 操作

本節說明如何使用 執行基本 Amazon Kinesis Data Streams 操作 AWS Command Line Interface。您將了解 Kinesis Data Streams 資料流程的基礎原理，以及將資料放入 Kinesis 資料串流和從中取得資料的必要步驟。

如果您是 Kinesis Data Streams 的新手，請先熟悉一下 Amazon Kinesis Data Streams 術語和概念 中所介紹的概念和術語。

主題

- 教學課程：安裝和設定 Kinesis Data Streams AWS CLI 的
- 教學課程：使用 執行基本 Kinesis Data Streams 操作 AWS CLI

對於 CLI 存取，您需要存取金鑰 ID 和私密存取金鑰。盡可能使用臨時憑證，而不是長期存取金鑰。臨時憑證包含存取金鑰 ID、私密存取金鑰，以及指出憑證何時到期的安全符記。如需詳細資訊，請參閱《IAM 使用者指南》中的將臨時登入資料與 AWS 資源搭配使用。

您可在建立 IAM 使用者中找到 IAM 和安全金鑰設定的詳細逐步說明。

本節中討論到的特定命令將一字不差地提供，但其每次執行時具體的數值必然會有所不同。此外，雖然範例使用 美國西部 (奧勒岡) 區域，但本節中的步驟在支援 Kinesis Data Streams 的任一區域中都適用。

教學課程：安裝和設定 Kinesis Data Streams AWS CLI 的

安裝 AWS CLI

如需如何為 Windows AWS CLI 和 Linux、OS X 和 Unix 作業系統安裝 的詳細步驟，請參閱安裝 AWS CLI。

使用以下命令列出可用的選項和服務：

```
aws help
```

您將使用 Kinesis Data Streams 服務，因此您可以使用下列命令檢閱與 Kinesis Data Streams 相關的 AWS CLI 子命令：

```
aws kinesis help
```

該命令產生的輸出將包括可用的 Kinesis Data Streams 命令：

```
AVAILABLE COMMANDS

       o add-tags-to-stream

       o create-stream

       o delete-stream

       o describe-stream

       o get-records

       o get-shard-iterator

       o help

       o list-streams

       o list-tags-for-stream

       o merge-shards

       o put-record

       o put-records

       o remove-tags-from-stream

       o split-shard

       o wait
```

上述命令清單與《Amazon Kinesis 服務 API 參考》所記載的 Kinesis Data Streams API 相對應。例如，`create-stream` 命令對應至 `CreateStream` API 動作。

現在 AWS CLI 已成功安裝 ，但未設定。請繼續閱讀下一節的說明。

設定 AWS CLI

對於一般用途， `aws configure`命令是設定 AWS CLI 安裝的最快方式。如需詳細資訊，請參閱設定 AWS CLI。

教學課程：使用 執行基本 Kinesis Data Streams 操作 AWS CLI

本節說明如何使用 AWS CLI透過命令列對 Kinesis 資料串流執行基本操作。請您務必先熟悉 Amazon Kinesis Data Streams 術語和概念 所討論的概念。

> **Note**
>
> 建立串流之後，您的帳戶會產生 Kinesis Data Streams 用量的名目費用，因為 Kinesis Data Streams 不符合 AWS 免費方案的資格。完成本教學課程後，請刪除您的 AWS 資源以停止產生費用。如需詳細資訊，請參閱步驟 4：清理。

主題

步驟 1：建立串流

您的第一個步驟是建立串流並確認其是否已成功建立。使用以下命令建立名為 "Foo" 的串流：

```
aws kinesis create-stream --stream-name Foo
```

接著，發出以下命令檢查串流的建立進度：

```
aws kinesis describe-stream-summary --stream-name Foo
```

您應會看到類似如下範例的輸出：

```
{
    "StreamDescriptionSummary": {
        "StreamName": "Foo",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
        "StreamStatus": "CREATING",
        "RetentionPeriodHours": 48,
```

```
        "StreamCreationTimestamp": 1572297168.0,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "OpenShardCount": 3,
        "ConsumerCount": 0
    }
}
```

在此範例中，串流具有狀態 CREATING，這表示尚未準備好使用。請於幾分鐘後再次檢查，屆時您應會看到類似如下範例的輸出：

```
{
    "StreamDescriptionSummary": {
        "StreamName": "Foo",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 48,
        "StreamCreationTimestamp": 1572297168.0,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "OpenShardCount": 3,
        "ConsumerCount": 0
    }
}
```

此輸出中有您不需要在本教學課程中的資訊。現在的重要資訊是 "StreamStatus": "ACTIVE"，它會告訴您串流已準備好可供使用，以及您請求的單一碎片上的資訊。您也可以使用 list-streams 命令確認新串流是否存在，如下所示：

```
aws kinesis list-streams
```

輸出：

```
{
    "StreamNames": [
        "Foo"
    ]
}
```

步驟 2：放置記錄

現在您已有了作用中的串流，即可準備開始放入一些資料。本教學課程使用最簡單可行的 put-record 命令，將包含文字 "testdata" 的單一資料記錄放入串流：

```
aws kinesis put-record --stream-name Foo --partition-key 123 --data testdata
```

此命令若成功，將產生類似如下範例的輸出：

```
{
    "ShardId": "shardId-000000000000",
    "SequenceNumber": "49546986683135544286507457936321625675700192471156785154"
}
```

恭喜，您剛已順利加入資料至串流！接下來您將了解如何從串流取出資料。

步驟 3：取得記錄

GetShardIterator

您必須先取得您感興趣的碎片的碎片反覆運算器，才能從串流取得資料。碎片疊代運算代表了消費者 (本例中為 get-record 命令) 將從中讀取資料的串流及碎片的位置。您將使用 get-shard-iterator命令，如下所示：

```
aws kinesis get-shard-iterator --shard-id shardId-000000000000 --shard-iterator-type
 TRIM_HORIZON --stream-name Foo
```

如前所述，aws kinesis 命令與 Kinesis Data Streams API 相對應，所以如果您對任何顯示的參數感興趣，均可閱讀 GetShardIterator API 參考主題以詳加了解。成功執行將產生類似下列範例的輸出：

```
{
```

```
    "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd
+efGN2aHFdkH1rJl4BL9Wyrk+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg="
}
```

一長串看似隨機字元的字串就是碎片疊代運算 (您的字串會有所出入)。您必須將碎片疊代運算複製/貼上至 get 命令，如下所示。碎片疊代運算的有效期為 300 秒，這段時間應足夠讓您將碎片疊代運算複製/貼入下一個命令中。在貼上至下一個命令之前，您必須從碎片疊代運算中移除任何新行。如果您收到錯誤訊息，指出碎片疊代運算不再有效，請再次執行 `get-shard-iterator`命令。

GetRecords

`get-records` 命令會從串流取得資料，並將解析成 Kinesis Data Streams API 的 `GetRecords` 呼叫。碎片迭代運算指定了碎片中您希望開始循序讀取資料記錄的位置。如果疊代運算所指向的碎片部分沒有可用的記錄，`GetRecords` 將傳回空白清單。可能需要多次呼叫才能到達包含記錄的部分碎片。

在下列 `get-records`命令範例中：

```
aws kinesis get-records --shard-iterator
 AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9ZG4lNR1EMi
+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg=
```

如果您是從像是 bash 的 Unix 類型命令處理器執行本教學課程，您可以使用巢狀命令自動取得碎片迭代器，如下所示：

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000 --
shard-iterator-type TRIM_HORIZON --stream-name Foo --query 'ShardIterator')

aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

如果您是從支援 PowerShell 的系統執行本教學課程，您可以使用下列命令自動取得碎片迭代器：

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-id
 shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo).split('"')
[4])
```

`get-records` 命令的成功結果會從串流請求您在取得碎片迭代器時所指定碎片的記錄，如下列範例所示：

```
{
```

```
  "Records":[ {
    "Data":"dGVzdGRhdGE=",
    "PartitionKey":"123",
    "ApproximateArrivalTimestamp": 1.441215410867E9,
    "SequenceNumber":"49544985256907370027570885864065577703022652638596431874"
  } ],
  "MillisBehindLatest":24000,

 "NextShardIterator":"AAAAAAAAAAEDOW3ugseWPE4503kqN1yN1UaodY8unE0sYslMUmC6lX9hlig5+t4RtZM0/
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvPOZvrUIudb8UkH3V
}
```

請注意， get-records如上所述為 請求，這表示即使串流中有記錄，您也可能收到零筆或多筆記錄。傳回的任何記錄可能不會代表目前串流中的所有記錄。這是正常的，生產程式碼會以適當的間隔輪詢串流以取得記錄。此輪詢速度會根據您的特定應用程式設計需求而有所不同。

在教學課程的這個部分中，您會注意到資料似乎是垃圾，這不是testdata我們傳送的純文字。這是因為 put-record 採用 Base64 編碼方式，讓您能夠傳送二進位資料。不過， 中的 Kinesis Data Streams 支援 AWS CLI 不提供 Base64 解碼，因為 Base64 解碼為列印到 stdout 的原始二進位內容可能會導致某些平台和終端機上不必要的行為和潛在的安全問題。若您使用 Base64 解碼器 (如 https://www.base64decode.org/) 手動解碼 dGVzdGRhdGE=，就會看到實際原文是 testdata。就本教學課程而言，這就已足夠，因為實際上， AWS CLI 很少用於使用資料。通常，它會用來監控串流的狀態並取得資訊，如先前所示 (describe-stream 和 list-streams)。如需有關 KCL 的詳細資訊，請參閱使用 KCL 開發具有共用輸送量的自訂取用者。

get-records 不一定會傳回指定串流/碎片中的所有記錄。若發生這種情況，請由最近的結果使用 NextShardIterator 以取得下一組記錄。如果有更多資料放入串流，這是生產應用程式中的正常情況，您可以get-records在每次使用 持續輪詢資料。不過，如果您未在 300 秒的碎片疊代運算生命週期內get-records使用下一個碎片疊代運算呼叫 ，您會收到錯誤訊息，而且必須使用 get-shard-iterator命令來取得新的碎片疊代運算。

上述輸出中還提供了 MillisBehindLatest，這是從串流的頂端回應 GetRecords 操作的毫秒數，表示取用者落後目前時間有多久。值為零表示記錄處理已跟上進度，此時沒有任何新記錄可供處理。在本教學課程中，若您一邊閱讀內容一邊操作，可能會看到這個數值相當大。根據預設，資料記錄會保留在串流中 24 小時，等待您擷取。此時間範圍稱為保留期間，最長可設定成 365 天。

NextShardIterator 即使串流中目前沒有更多記錄，成功get-records結果一律會有 。這是假定生產者可能在任何特定時間內將更多記錄放入串流的一種輪詢模式。您儘管可自行撰寫輪詢常式，但若您使用前述的 KCL 開發消費者應用程式，該程式庫將會為您處理妥輪詢事宜。

如果您在串流和碎片中沒有更多記錄get-records之前呼叫 ，您會看到具有類似下列範例之空白記錄的輸出：

```
{
    "Records": [],
    "NextShardIterator": "AAAAAAAAAAGCJ5jzQNjmdhO6B/YDIDE56jmZmrmMA/r1WjoHXC/
kPJXc1rckt3TFL55dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCAjQ6s7ZupjXeJGoUFs5oCuFwhP+Wul/
EhyNeSs5DYXLSSC5XCapmCAYGFjYER69QSdQjxMmBPE/hiybFDi5qtkT6/PsZNz6kFoqtDk="
}
```

步驟 4：清理

刪除您的串流以釋出資源，並避免您的帳戶產生意外費用。每當您建立串流且不會使用它時，請執行此操作，因為無論您是否使用它來放置和取得資料，每個串流都會產生費用。清除命令如下所示：

```
aws kinesis delete-stream --stream-name Foo
```

成功不會產生輸出。使用 describe-stream檢查刪除進度：

```
aws kinesis describe-stream-summary --stream-name Foo
```

如果您在刪除命令之後立即執行此命令，您會看到類似下列範例的輸出：

```
{
    "StreamDescriptionSummary": {
        "StreamName": "samplestream",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/samplestream",
        "StreamStatus": "ACTIVE",
```

串流完全刪除後，describe-stream 將導致「找不到」的錯誤。

```
A client error (ResourceNotFoundException) occurred when calling the
 DescribeStreamSummary operation:
Stream Foo under account 123456789012 not found.
```

Amazon Kinesis Data Streams 入門教學課程

Amazon Kinesis Data Streams 提供多種不同的解決方案，用於從 Kinesis 資料串流擷取和取用資料。本節中的教學課程旨在進一步協助您了解 Amazon Kinesis Data Streams 概念和功能，並識別符合您需求的解決方案。

主題

教學課程：使用 KPL 和 KCL 2.x 處理即時庫存資料

本教學課程的案例涉及將股票交易擷取到資料串流，以及撰寫在串流上執行計算的基本 Amazon Kinesis Data Streams 應用程式。您將了解如何將記錄串流傳送至 Kinesis Data Streams，並實作應用程式，以近乎即時的方式取用和處理記錄。

> **Important**
>
> 建立串流後，您的帳戶會產生 Kinesis Data Streams 用量的名目費用，因為 Kinesis Data Streams 不符合 AWS 免費方案的資格。取用者應用程式啟動之後，也會象徵性地收取 Amazon DynamoDB 使用費。取用者應用程式使用 DynamoDB 追蹤處理狀態。當您使用此應用程式完畢後，請刪除您的 AWS 資源以避免其產生費用。如需詳細資訊，請參閱清除資源。

程式碼不會存取實際股票市場資料，而是模擬股票交易串流。其做法是使用隨機股票交易產生器，以截至 2015 年 2 月為止市值排名前 25 位的股票實際市場資料做為起始點。若您能夠存取即時股票交易串流，可能會希望從該串流衍生出實用且及時的統計資料。例如，您可能想要執行滑動時段分析，以得知前 5 分鐘內購買的最熱門股票。或者，您可能希望在銷售訂單過大 (即股份過多) 時接獲通知。您可透過擴展此系列程式碼以提供這類功能。

您可以在桌上型電腦或筆記型電腦上演練本教學課程的步驟，然後在同一部電腦或任何支援既定要求的平台上同時執行生產者和消費者程式碼。

以下所示範例使用美國西部 (奧勒岡) 區域，但在任何支援 Kinesis Data Streams 的AWS 區域內均能運作。

任務

- 完成事前準備
- 建立資料串流
- 建立 IAM 政策和使用者
- 下載並建置程式碼
- 實作生產者
- 實作消費者
- （選用）擴展消費者
- 清除資源

完成事前準備

您必須符合下列要求才能完成本教學課程：

建立和使用 Amazon Web Services 帳戶

開始之前，請確定您已熟悉 中討論的概念Amazon Kinesis Data Streams 術語和概念，特別是串流、碎片、生產者和消費者。完成下列指南中的步驟也會有幫助：教學課程：安裝和設定 Kinesis Data Streams AWS CLI 的。

您必須擁有 AWS 帳戶和 Web 瀏覽器才能存取 AWS 管理主控台。

對於主控台存取權，請使用您的 IAM 使用者名稱和密碼從 IAM 登入頁面登入 AWS 管理主控台。如需有關 AWS 安全登入資料的資訊，包括程式設計存取和長期登入資料的替代方案，請參閱《IAM 使用者指南》中的AWS 安全登入資料。如需登入 的詳細資訊 AWS 帳戶，請參閱AWS 登入 《 使用者指南》中的如何登入 AWS 。

如需 IAM 和安全金鑰設定指示的詳細資訊，請參閱建立 IAM 使用者。

滿足系統軟體需求

用於執行應用程式的系統必須安裝 Java 7 或更高版本。若要下載並安裝最新版 Java 開發套件 (JDK)，請前往 Oracle 的 Java SE 安裝網站。

您需要有最新版本適用於 Java 的 AWS SDK。

取用者應用程式需要 Kinesis Client Library (KCL) 2.2.9 版或更高版本，您可前往 GitHub 從 https://github.com/awslabs/amazon-kinesis-client/tree/master 取得此程式庫。

後續步驟

建立資料串流

建立資料串流

首先，您必須建立將在本教學課程的後續步驟中使用的資料串流。

建立串流

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/kinesis 的 Kinesis 主控台。
2. 在導覽窗格中選擇資料串流。
3. 在導覽列中，展開區域選擇工具，然後選擇一個區域。
4. 選擇 Create Kinesis stream (建立 Kinesis 串流)。
5. 輸入資料串流的名稱 (例如 **`StockTradeStream`**)。
6. 輸入 **`1`**以取得碎片數量，但請維持預估您需要收合的碎片數量。
7. 選擇 Create Kinesis stream (建立 Kinesis 串流)。

建立串流時，在 Kinesis 串流清單頁面上，該串流的狀態會是 `CREATING`。當串流就緒可供使用後，其狀態會變成 `ACTIVE`。

如果您選擇串流的名稱，則在隨後出現的頁面中，Details (詳細資訊) 索引標籤會顯示資料串流組態的摘要。Monitoring (監控) 區段則顯示串流的監控資訊。

後續步驟

建立 IAM 政策和使用者

建立 IAM 政策和使用者

指定 AWS 使用精細許可來控制對不同資源之存取的安全最佳實務。 AWS Identity and Access Management (IAM) 可讓您管理 中的使用者和使用者許可 AWS。IAM 政策將明確列出允許的動作以及各項動作所適用的資源。

以下是 Kinesis Data Streams 生產者和取用者一般需要的最低許可。

生產者

動作	資源	用途
DescribeStream ，DescribeStreamSummary ，DescribeStreamConsumer	Kinesis 資料串流	嘗試讀取記錄之前，消費者應先檢查資料串流是否存在、是否及資料串流中是否含有碎片。
SubscribeToShard ，RegisterStreamConsumer	Kinesis 資料串流	為消費者訂閱和註冊碎片。
PutRecord ，PutRecords	Kinesis 資料串流	將記錄寫入 Kinesis Data Streams。

消費者

動作	Resource	用途
DescribeStream	Kinesis 資料串流	嘗試讀取記錄之前，消費者應先檢查資料串流是否存在、是否及資料串流中是否含有碎片。
GetRecords ，GetShardIterator	Kinesis 資料串流	從碎片讀取記錄。
CreateTable ，DescribeTable ，GetItem, PutItem, Scan, UpdateItem	Amazon DynamoDB 資料表	如果取用者是使用 Kinesis Client Library (KCL) (1.x 或 2.x 版對 DynamoDB 資料表的許可，才能追蹤應用程式的處理狀態
DeleteItem	Amazon DynamoDB 資料表	在取用者對 Kinesis Data Streams 碎片執行分割/合併操作的
PutMetricData	Amazon CloudWatch 日誌	KCL 還會上傳指標至 CloudWatch，這將有助於監控應用程式

在本教學課程中，您將建立單一 IAM 政策，授予所有上述許可。在生產環境中，您可能要建立兩項政策，一項政策用於生產者，另一項政策用於消費者。

建立 IAM 政策

1. 尋找您在上一個步驟中建立的新資料串流的 Amazon Resource Name (ARN)。您可以在 Details (詳細資訊) 標籤頂端所列的 Stream ARN (串流 ARN) 中找到此 ARN。ARN 格式如下：

   ```
   arn:aws:kinesis:region:account:stream/name
   ```

   region

   AWS 區域碼，例如 us-west-2。如需詳細資訊，請參閱區域與可用區域的概念。

   帳戶

   AWS 帳戶 ID，如帳戶設定所示。

   name

   您在上一個步驟中建立的資料串流名稱，即 StockTradeStream。
2. 決定要供取用者使用的 DynamoDB 資料表 (將由第一個取用者執行個體所建立) 的 ARN。其值必須為以下格式：

   ```
   arn:aws:dynamodb:region:account:table/name
   ```

   區域和帳戶 ID 與您用於本教學課程的資料串流 ARN 中的值相同，但名稱是消費者應用程式建立和使用的 DynamoDB 資料表名稱。KCL 使用應用程式名稱作為資料表名稱。在此步驟中，使用 StockTradesProcessor 作為 DynamoDB 資料表名稱，因為這是本教學課程稍後步驟中使用的應用程式名稱。
3. 在 IAM 主控台的政策 (https://console.aws.amazon.com/iam/home#policies) 中，選擇建立政策。如果這是您第一次使用 IAM 政策，請選擇開始使用、建立政策。
4. 從 Policy Generator (政策產生器) 旁選擇 Select (選取)。
5. 選擇 Amazon Kinesis 做為 AWS 服務。
6. 選取 DescribeStream、GetShardIterator、GetRecords、PutRecord 和 PutRecords 做為允許的動作。
7. 輸入您在本教學課程中使用的資料串流 ARN。
8. 使用 Add Statement (新增陳述式) 指定以下各項：

AWS 服務	動作	ARN
Amazon DynamoDB	`CreateTable` ，`DeleteItem` ，`DescribeTable` ，`GetItem`, `PutItem`, `Scan`, `UpdateItem`	您在此程序的步驟 2 中建立的 DynamoDB 資料表 ARN。
Amazon CloudWatch	`PutMetricData`	*

在無須指定 ARN 的情況下要使用星號 (*)。此處使用星號是因為 CloudWatch 中沒有任何可對其叫用 `PutMetricData` 動作的特定資源。

9. 選擇 Next Step (後續步驟)。
10. 將 Policy Name (政策名稱) 更改為 `StockTradeStreamPolicy` 並檢閱程式碼，然後選擇 Create Policy (建立政策)。

產生的政策文件應該如下所示：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt123",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards",
                "kinesis:DescribeStreamSummary",
                "kinesis:RegisterStreamConsumer"
            ],
            "Resource": [
```

```
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream"
            ]
        },
        {
            "Sid": "Stmt234",
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream/
*"
            ]
        },
        {
            "Sid": "Stmt456",
            "Effect": "Allow",
            "Action": [
                "dynamodb:*"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:table/
StockTradesProcessor"
            ]
        },
        {
            "Sid": "Stmt789",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

建立 IAM 使用者

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。

2. 在 Users (使用者) 頁面上，選擇 Add user (新增使用者)。
3. 對於使用者名稱，輸入 StockTradeStreamUser。
4. 對於 Access type (存取類型)，選擇 Programmatic access (程式設計存取)，然後選擇 Next: Permissions (下一步：許可)。
5. 選擇直接連接現有政策。
6. 依名稱搜尋您在上述程序中建立的政策 (StockTradeStreamPolicy。選取政策名稱左側的方塊，然後選擇 Next: Review (下一步：檢閱)。
7. 檢閱詳細資訊及摘要，然後選擇 Create user (建立使用者)。
8. 複製 Access key ID (存取金鑰 ID) 並將其私下儲存。從 Secret access key (私密存取金鑰) 下方選擇 Show (顯示)，然後一併將該金鑰私下儲存。
9. 將存取金鑰和私密金鑰貼入本機檔案中，存放於只有您能夠存取的安全處。針對此應用程式，請建立名為 ~/.aws/credentials (具有嚴格許可) 的檔案。該檔案應採用以下格式：

```
[default]
aws_access_key_id=access key
aws_secret_access_key=secret access key
```

將 IAM 政策附接至使用者

1. 在 IAM 主控台，開啟政策並選擇 政策動作。
2. 依序選擇 StockTradeStreamPolicy 和 Attach (附加)。
3. 依序選擇 StockTradeStreamUser 和 Attach Policy (附加政策)。

後續步驟

下載並建置程式碼

下載並建置程式碼

本主題提供擷取至資料串流的範例股票交易 (生產者) 和此資料處理 (消費者) 的範例實作程式碼。

下載並建置程式碼

1. 將原始碼從 https://github.com/aws-samples/amazon-kinesis-learning GitHub 存放庫下載到您的電腦。
2. 依照提供的目錄結構，在您的 IDE 中使用原始碼建立一個專案。

3. 將以下程式庫加入至該專案：

   - Amazon Kinesis Client Library (KCL)
   - AWS 開發套件
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (適用於 Java 的 Google 核心程式庫)
   - Jackson Annotations
   - Jackson Core
   - Jackson Databind
   - Jackson Dataformat：CBOR
   - Joda Time
4. 視您的 IDE 而定，系統可能會自動建置專案。如果沒有，請根據您的 IDE 使用適當步驟建置專案。

若您已順利完成上述步驟，即可移往下一節進行the section called “實作生產者”。

後續步驟

實作生產者

本教學課程使用真實情境的股票市場交易監控。以下原則簡要說明此情境如何對應到生產者及其支援的程式碼結構。

請查看原始碼並對照檢閱以下資訊。

StockTrade 類別

單次股票交易是由 StockTrade 類別的執行個體表示。此執行個體包含若干屬性，如股票代號、價格、股份數、交易類型 (買進或賣出) 以及唯一識別該交易的 ID。程式碼已為您實作此類別。

串流記錄

串流是一連串的記錄。記錄則是 JSON 格式的序列化 `StockTrade` 執行個體。例如：

```
{
  "tickerSymbol": "AMZN",
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator 類別

StockTradeGenerator 具有 getRandomTrade() 方法，每次叫用後將傳回隨機產生的新股票交易。程式碼已為您實作此類別。

StockTradesWriter 類別

生產者的 main 方法 StockTradesWriter 會持續擷取隨機交易，然後透過執行以下任務將該交易傳送至 Kinesis Data Streams：

1. 讀取資料串流名稱和區域名稱做為輸入。
2. 使用 KinesisAsyncClientBuilder設定區域、登入資料和用戶端組態。
3. 檢查串流是否存在且處於作用中狀態 (否則將結束此方法並顯示錯誤)。
4. 在連續迴圈中依序呼叫 StockTradeGenerator.getRandomTrade() 方法和 sendStockTrade 方法，每隔 100 毫秒將交易傳送至串流。

sendStockTrade 類別的 StockTradesWriter 方法含有以下程式碼：

```
private static void sendStockTrade(StockTrade trade, KinesisAsyncClient
 kinesisClient,
            String streamName) {
        byte[] bytes = trade.toJsonAsBytes();
        // The bytes could be null if there is an issue with the JSON serialization
 by the Jackson JSON library.
        if (bytes == null) {
            LOG.warn("Could not get JSON bytes for stock trade");
            return;
        }

        LOG.info("Putting trade: " + trade.toString());
        PutRecordRequest request = PutRecordRequest.builder()
```

```
                .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol
as the partition key, explained in the Supplemental Information section below.
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(bytes))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            LOG.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            LOG.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }
```

請參閱以下的程式碼詳解：

- PutRecord API 需要位元組陣列，您必須將交易轉換為 JSON 格式。這一行程式碼將執行該項操作：

```
byte[] bytes = trade.toJsonAsBytes();
```

- 傳送交易之前，您必須先建立新的 PutRecordRequest 執行個體 (本例中稱為要求)。每個 request 都需要串流名稱、分割區索引鍵和資料 Blob。

```
PutPutRecordRequest request = PutRecordRequest.builder()
    .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol as the
 partition key, explained in the Supplemental Information section below.
    .streamName(streamName)
    .data(SdkBytes.fromByteArray(bytes))
    .build();
```

此範例使用股票代號做為分割區索引鍵，將記錄映射至特定的碎片。實際上，每個碎片應該會有成千上百的分割區索引鍵，使記錄均勻地分佈於串流中。如需如何加入資料至串流的詳細資訊，請參閱[將資料寫入 Amazon Kinesis Data Streams]。

現在 request 已準備好傳送至用戶端 (put 操作)：

```
   kinesisClient.putRecord(request).get();
```

- 錯誤檢查和日誌記錄肯定是頗為實用的附加功能。此程式碼將記錄錯誤情況：

```
if (bytes == null) {
    LOG.warn("Could not get JSON bytes for stock trade");
    return;
}
```

在 put 操作的周圍加入 try/catch 區塊：

```
try {
  kinesisClient.putRecord(request).get();
} catch (InterruptedException e) {
            LOG.info("Interrupted, assuming shutdown.");
} catch (ExecutionException e) {
            LOG.error("Exception while sending data to Kinesis. Will try again
 next cycle.", e);
}
```

這麼做是因為 Kinesis Data Streams put 操作可能由於網路錯誤或是資料串流達到其傳輸量限制並受到調節而導致失敗。建議您仔細考慮put操作的重試政策，以避免資料遺失，例如使用重試。

- 狀態記錄也很實用，但可有可無：

```
LOG.info("Putting trade: " + trade.toString());
```

此處所示的生產者是使用 Kinesis Data Streams API 的單一記錄功能 PutRecord。實際上，如果個別的生產者將產生許多記錄，則使用 PutRecords 的多筆記錄功能並一次性傳送各個批次的記錄通常會更有效率。如需詳細資訊，請參閱[將資料寫入 Amazon Kinesis Data Streams]。

執行生產者

1. 確認在建立 IAM 政策和使用者中擷取的存取金鑰和秘密金鑰對已儲存在檔案 ~/.aws/credentials 中。
2. 使用以下引數執行 StockTradeWriter 類別：

```
StockTradeStream us-west-2
```

如果您是在 us-west-2 以外的區域建立串流，則此處必須改為指定該區域。

您應該會看到類似下列的輸出：

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

您的股票交易現在正由 Kinesis Data Streams 擷取中。

後續步驟

實作消費者

實作消費者

本教學課程中的消費者應用程式會持續處理您的資料串流中的股票交易。隨後，其將輸出每分鐘買進和賣出最多的熱門股票。此應用程式是使用 Kinesis Client Library (KCL) 所建置，由該程式庫執行取用者應用程式常見的諸多繁重工作。如需詳細資訊，請參閱KCL 1.x 和 2.x 資訊。

請查看原始碼並對照檢閱以下資訊。

StockTradesProcessor 類別

提供給您的消費者主要類別，其會執行下列任務：

- 讀取做為引數傳入的應用程式、資料串流和區域名稱。
- 使用區域名稱建立KinesisAsyncClient執行個體。
- 建立 StockTradeRecordProcessorFactory 執行個體以提供由 ShardRecordProcessor 執行個體實作的 StockTradeRecordProcessor 執行個體。
- 使用 KinesisAsyncClient、ApplicationName、 StreamName和 ConfigsBuilder執行個體建立StockTradeRecordProcessorFactory執行個體。這對於使用預設值建立所有組態非常有用。
- 使用 ConfigsBuilder 執行個體建立 KCL 排程器 (之前，在 KCL 版本 1.x 中，它被稱為 KCL 工作者)。
- 排程器會為每個碎片 (已指派給此消費者執行個體) 建立新的執行緒，以持續循環從資料串流讀取記錄。接著，其將叫用 StockTradeRecordProcessor 執行個體以處理收到的各個批次記錄。

StockTradeRecordProcessor 類別

StockTradeRecordProcessor 執行個體的實作，而此執行個體將實作五個必要的方法：initialize、processRecords、leaseLost、shardEnded 和 shutdownRequested。

KCL 使用 initialize 和 shutdownRequested 方法，讓記錄處理器得知何時應準備好開始接收記錄以及何時應停止接收記錄，好讓程式庫能夠執行任何應用程式特定的設定和終止任務。leaseLost 和 shardEnded 則用於實作當遺失租約或處理已達碎片結尾時應執行什麼操作的任何邏輯。在此範例中，我們只記錄指出這些事件的訊息。

我們會提供這些方法的程式碼。主要處理任務在 processRecords 方法中進行，而此方法將使用 processRecord 處理每筆記錄。後一種方法以幾乎全空的架構程式碼提供，讓您於下一個步驟進行實作，屆時將會有更詳細的說明。

另請注意 processRecord 支援方法的實作：reportStats 和 resetStats，其最初的原始碼為全空。

程式碼已為您實作 processRecords 方法，並將執行以下步驟：

- 對每一筆傳入的記錄，它會呼叫其上的 processRecord。
- 若自從上次報告後已歷時至少 1 分鐘，請先呼叫 reportStats() 列印出最新統計資料，接著呼叫 resetStats() 清除統計資料以使下一個間隔僅包含新記錄。

- 設定下一次報告時間。
- 若自從最後一個檢查點過後已歷時至少 1 分鐘，請呼叫 checkpoint()。
- 設定下一次檢查點作業時間。

此方法使用 60 秒的間隔做為報告及檢查點作業率。如需有關檢查點的詳細資訊，請參閱使用 Kinesis Client Library。

StockStats 類別

此類別針對一段時間內最熱門的股票提供資料保留與統計資料追蹤。其程式碼已為您提供且包含下列方法：

- addStockTrade(StockTrade)：將給定的 StockTrade 注入目前統計資料。
- toString()：以格式化字串的形式傳回統計資料。

此類別透過保留每個股票交易總數的執行中計數和最大計數，來追蹤最熱門的股票。每當股票交易達成時，其將更新這些計數。

為 StockTradeRecordProcessor 類別的各個方法加入程式碼，如以下步驟所示。

實作消費者

1. 實作 processRecord 方法，藉此執行個體化正確大小的 StockTrade 物件並將記錄資料加入該物件，且於發生問題時記錄警告。

```
byte[] arr = new byte[record.data().remaining()];
record.data().get(arr);
StockTrade trade = StockTrade.fromJsonAsBytes(arr);
    if (trade == null) {
        log.warn("Skipping record. Unable to parse record into StockTrade.
 Partition Key: " + record.partitionKey());
        return;
        }
stockStats.addStockTrade(trade);
```

2. 實作 reportStats方法。修改輸出格式以符合您的偏好設定。

```
System.out.println("****** Shard " + kinesisShardId + " stats for last 1 minute
 ******\n" +
```

```
stockStats + "\n" +
"**********************************************************\n");
```

3. 實作 resetStats 方法以便建立新的 stockStats 執行個體。

```
stockStats = new StockStats();
```

4. 實作ShardRecordProcessor界面所需的下列方法：

```
@Override
public void leaseLost(LeaseLostInput leaseLostInput) {
    log.info("Lost lease, so terminating.");
}

@Override
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        log.info("Reached shard end checkpointing.");
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at shard end. Giving up.", e);
    }
}

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    log.info("Scheduler is shutting down, checkpointing.");
    checkpoint(shutdownRequestedInput.checkpointer());
}

private void checkpoint(RecordProcessorCheckpointer checkpointer) {
    log.info("Checkpointing shard " + kinesisShardId);
    try {
        checkpointer.checkpoint();
    } catch (ShutdownException se) {
        // Ignore checkpoint if the processor instance has been shutdown (fail
 over).
        log.info("Caught shutdown exception, skipping checkpoint.", se);
    } catch (ThrottlingException e) {
```

```
        // Skip checkpoint when throttled. In practice, consider a backoff and
 retry policy.
        log.error("Caught throttling exception, skipping checkpoint.", e);
    } catch (InvalidStateException e) {
        // This indicates an issue with the DynamoDB table (check for table,
 provisioned IOPS).
        log.error("Cannot save checkpoint to the DynamoDB table used by the Amazon
 Kinesis Client Library.", e);
    }
}
```

執行消費者

1. 執行您在時撰寫的生產者，將模擬的股票交易記錄注入您的串流。
2. 確認稍早 (建立 IAM 使用者 使用者時) 擷取的存取金鑰和私密金鑰對是否已儲存至 `~/.aws/credentials` 檔案。
3. 使用以下引數執行 `StockTradesProcessor` 類別：

   ```
   StockTradesProcessor StockTradeStream us-west-2
   ```

   請注意，如果您是在 `us-west-2` 以外的區域建立串流，則此處必須改為指定該區域。

一分鐘後，您應會看到類似以下內容的輸出，而且此後每分鐘將重新整理一次輸出：

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
****************************************************************
```

後續步驟

[（選用）擴展消費者]

（選用）擴展消費者

此一選用章節示範如何就更為複雜的情境擴充消費者程式碼。

如果您想知道每分鐘銷售量最高的訂單，則可修改三處位置的 StockStats 類別以納入這項新的優先等級。

擴充消費者

1. 加入新的執行個體變數：

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. 為 addStockTrade 添加以下程式碼：

```
if (type == TradeType.SELL) {
     if (largestSellOrderStock == null || trade.getQuantity() >
 largestSellOrderQuantity) {
         largestSellOrderStock = trade.getTickerSymbol();
         largestSellOrderQuantity = trade.getQuantity();
     }
 }
```

3. 修改 toString 方法以列印其他資訊：

```
public String toString() {
    return String.format(
        "Most popular stock being bought: %s, %d buys.%n" +
        "Most popular stock being sold: %s, %d sells.%n" +
        "Largest sell order: %d shares of %s.",
        getMostPopularStock(TradeType.BUY),
 getMostPopularStockCount(TradeType.BUY),
        getMostPopularStock(TradeType.SELL),
 getMostPopularStockCount(TradeType.SELL),
        largestSellOrderQuantity, largestSellOrderStock);
}
```

若您現在執行消費者 (記得也要執行生產者)，應會看到類似以下內容的輸出：

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
Largest sell order: 996 shares of BUD.
****************************************************************
```

後續步驟

清除資源

清除資源

由於使用 Kinesis 資料串流需要支付費用，當您使用串流完畢後，請務必刪除該串流和對應的 Amazon DynamoDB 資料表。作用中的串流即便並未傳送及取得記錄，也會象徵性地收取費用。這是因為作用中的串流會持續「監聽」傳入的記錄和取得記錄的請求，以致將耗用資源。

刪除串流和資料表

1. 關閉您可能仍在執行的任何生產者和消費者。
2. 在以下網址開啟 Kinesis 主控台：https://console.aws.amazon.com/kinesis。
3. 選擇您為此應用程式所建立的串流 (`StockTradeStream`)。
4. 選擇 Delete Stream (刪除串流)。
5. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
6. 刪除 `StockTradesProcessor` 資料表。

摘要

近乎即時地處理大量資料，不需要撰寫複雜的程式碼或開發巨大的基礎設施。它與編寫邏輯來處理少量資料（例如寫入 `processRecord(Record)`）一樣基本，但使用 Kinesis Data Streams 進行擴展，使其適用於大量串流資料。您不必擔心如何擴展處理方式，因為這一切 Kinesis Data Streams 都將為您代勞。您只需要傳送串流記錄至 Kinesis Data Streams 並撰寫邏輯以處理每一筆收到的新記錄。

以下是對此應用程式可行的一些強化功能。

跨所有碎片進行彙整

目前您是透過彙整單一工作者從單一碎片接收的資料記錄來取得統計資料 (任一碎片不能同時由單一應用程式中的多個工作者處理)。當然，您若擴展為具有多個碎片，可能會想要跨所有碎片進行彙整。為此，您可以採用某種流程架構，將每個工作者的輸出都饋送至具有單一碎片的另一串流，而碎片則由彙整第一個階段輸出的工作者處理。由於取自第一個階段的資料有限 (每個碎片每分鐘一次取樣)，單一碎片就能輕鬆處理這些資料。

擴展處理

當串流擴展為具有多個碎片後 (因為有許多生產者傳送資料)，擴展處理的方式即是增加更多工作者。您可以在 Amazon EC2 執行個體上執行工作者並使用 Auto Scaling 群組。

使用連接器到 Amazon S3/DynamoDB/Amazon Redshift/Storm

隨著串流持續處理，其輸出可以傳送到其他目的地。 AWS 提供連接器，將 Kinesis Data Streams 與其他 AWS 服務和第三方工具整合。

教學課程：使用 KPL 和 KCL 1.x 處理即時庫存資料

本教學課程的情境涉及將股票交易擷取至資料串流和撰寫簡單的 Amazon Kinesis Data Streams 應用程式對該串流執行計算。您將了解如何將記錄串流傳送至 Kinesis Data Streams，並實作應用程式，以近乎即時的方式取用和處理記錄。

> **Important**
>
> 建立串流後，您的帳戶會產生 Kinesis Data Streams 用量的名目費用，因為 Kinesis Data Streams 不符合 AWS 免費方案的資格。取用者應用程式啟動之後，也會象徵性地收取 Amazon DynamoDB 使用費。取用者應用程式使用 DynamoDB 追蹤處理狀態。當您使用此應用程式完畢後，請刪除您的 AWS 資源以避免其產生費用。如需詳細資訊，請參閱清除資源。

程式碼不會存取實際股票市場資料，而是模擬股票交易串流。其做法是使用隨機股票交易產生器，以截至 2015 年 2 月為止市值排名前 25 位的股票實際市場資料做為起始點。若您能夠存取即時股票交易串流，可能會希望從該串流衍生出實用且及時的統計資料。例如，您可能想要執行滑動時段分析，以得知前 5 分鐘內購買的最熱門股票。或者，您可能希望在銷售訂單過大 (即股份過多) 時接獲通知。您可透過擴展此系列程式碼以提供這類功能。

您可以在桌上型電腦或筆記型電腦上演練本教學課程的步驟，然後在同一部電腦或任何支援既定要求的平台如 Amazon Elastic Compute Cloud (Amazon EC2) 上同時執行生產者和取用者程式碼。

以下所示範例使用美國西部 (奧勒岡) 區域，但在任何支援 Kinesis Data Streams 的AWS 區域內均能運作。

任務

- 完成事前準備
- 建立資料串流
- 建立 IAM 政策和使用者
- 下載並建置實作程式碼
- 實作生產者
- 實作消費者
- （選用）擴展消費者
- 清除資源

完成事前準備

以下是完成教學課程：使用 KPL 和 KCL 1.x 處理即時庫存資料的要求。

建立和使用 Amazon Web Services 帳戶

開始之前，請務必先熟悉 Amazon Kinesis Data Streams 術語和概念所討論的概念，特別是串流、碎片、生產者和消費者。事先完成教學課程：安裝和設定 Kinesis Data Streams AWS CLI 的亦將有所助益。

您需要 AWS 帳戶和 Web 瀏覽器才能存取 AWS 管理主控台。

對於主控台存取權，請使用您的 IAM 使用者名稱和密碼從 IAM 登入頁面登入 AWS 管理主控台。如需有關 AWS 安全登入資料的資訊，包括程式設計存取和長期登入資料的替代方案，請參閱《IAM 使用者指南》中的AWS 安全登入資料。如需登入 的詳細資訊 AWS 帳戶，請參閱AWS 登入 《 使用者指南》中的如何登入 AWS 。

如需 IAM 和安全金鑰設定指示的詳細資訊，請參閱建立 IAM 使用者。

滿足系統軟體需求

用於執行應用程式的系統必須安裝 Java 7 或更高版本。若要下載並安裝最新版 Java 開發套件 (JDK)，請前往 Oracle 的 Java SE 安裝網站。

若您已有 Java IDE 如 Eclipse，便可開啟原始碼進行編輯、建置並執行。

您需要有最新版本適用於 Java 的 AWS SDK。若您使用 Eclipse 做為 IDE，則可改為安裝 AWS Toolkit for Eclipse。

取用者應用程式需要 Kinesis Client Library (KCL) 1.2.1 版或更高版本，您可前往 GitHub 從 Kinesis Client Library (Java) 取得此程式庫。

後續步驟

建立資料串流

建立資料串流

在教學課程：使用 KPL 和 KCL 1.x 處理即時庫存資料的第一個步驟中，您將建立串流以供後續步驟使用。

建立串流

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/kinesis 的 Kinesis 主控台。
2. 在導覽窗格中選擇資料串流。
3. 在導覽列中，展開區域選擇工具，然後選擇一個區域。
4. 選擇 Create Kinesis stream (建立 Kinesis 串流)。
5. 輸入串流的名稱 (例如 **StockTradeStream**)。
6. 輸入 **1**表示碎片數量，但請保持估計您需要收合的碎片數量。
7. 選擇 Create Kinesis stream (建立 Kinesis 串流)。

建立串流時，在 Kinesis 串流清單頁面上，該串流的狀態會是 CREATING。當串流就緒可供使用後，其狀態會變成 ACTIVE。選擇串流名稱。在隨後出現的頁面上，Details (詳細資訊) 標籤將顯示串流組態的摘要。Monitoring (監控) 區段則顯示串流的監控資訊。

碎片的其他資訊

在本教學課程外開始使用 Kinesis Data Streams 時，您可能需要更仔細地規劃串流建立程序。您應於佈建碎片時規劃預計的最大需求。以此處的情境為例，美國股票市場某一天 (東部時間) 的交易流量峰值以及需求估計值應該從當天的時間內取樣。接著，您即可選擇佈建最大預計需求，或是擴展或縮減串流規模以因應需求波動。

碎片是傳輸容量的單位。在建立 Kinesis 串流頁面上，展開 Estimate the number of shards you'll need (估計所需的碎片數目)。依照以下準則輸入平均記錄大小、每秒寫入記錄數上限與取用端應用程式數目：

平均記錄大小

您的記錄計算出的平均大小估計值。如果您不知道此值，請使用估計的最大記錄大小做為此值。

寫入記錄上限

考慮提供資料的實體數量，以及每個實體每秒產生的大約記錄數量。例如，假設您從 20 部交易伺服器取得股票交易資料且每部伺服器每秒產生 250 次交易，則每秒的交易 (記錄) 總數為 5000。

取用端應用程式數目

單獨從串流進行讀取以透過不同方式處理串流並產生不同輸出的應用程式數目。每個應用程式可有多個執行個體在不同的電腦上執行 (亦即在叢集內執行)，以便能及時處理高容量串流。

如果顯示的碎片估計數目超出您目前的碎片限額，您可能需要提交請求以提高該限制，然後才能建立具有此碎片數目的串流。若要請求提升您的碎片限額，請使用 Kinesis Data Streams 限制表單。如需串流和碎片的詳細資訊，請參閱 建立和管理 Kinesis 資料串流。

後續步驟

建立 IAM 政策和使用者

建立 IAM 政策和使用者

指定 AWS 使用精細許可來控制對不同資源之存取的安全最佳實務。 AWS Identity and Access Management (IAM) 可讓您管理 中的使用者和使用者許可 AWS。IAM 政策將明確列出允許的動作以及各項動作所適用的資源。

以下是 Kinesis Data Streams 生產者和取用者一般需要的最低許可。

生產者

動作	資源	用途
`DescribeStream`，`DescribeStreamSummary`，`DescribeStreamConsumer`	Kinesis 資料串流	嘗試撰寫記錄之前，生產者應先檢查串流是否存在且處於作 含有碎片，以及串流是否有消費者。

動作	資源	用途
SubscribeToShard ，RegisterStreamConsumer	Kinesis 資料串流	訂閱並將消費者註冊至 Kinesis 資料串流碎片。
PutRecord ，PutRecords	Kinesis 資料串流	將資料寫入至 Kinesis Data Streams。

消費者

動作	Resource	用途
DescribeStream	Kinesis 資料串流	嘗試讀取記錄之前，消費者應先檢查串流是否存在且處於作 是否含有碎片。
GetRecords ，GetShardIterator	Kinesis 資料串流	從 Kinesis Data Streams 碎片讀取記錄。
CreateTable ，DescribeTable ，GetItem, PutItem, Scan, UpdateItem	Amazon DynamoDB 資料表	如果取用者是使用 Kinesis Client Library (KCL) 進行開發， 資料表的許可，才能追蹤應用程式的處理狀態。第一個消費 。
DeleteItem	Amazon DynamoDB 資料表	在取用者對 Kinesis Data Streams 碎片執行分割/合併操作的
PutMetricData	Amazon CloudWatch 日誌	KCL 還會上傳指標至 CloudWatch，這將有助於監控應用程

針對此應用程式，您要建立單一 IAM 政策以授予上述所有許可。實際上，您可能要考慮建立兩項政策，一項政策用於生產者，另一項政策用於消費者。

建立 IAM 政策

1. 找出新串流的 Amazon Resource Name (ARN)。您可以在 Details (詳細資訊) 標籤頂端所列的 Stream ARN (串流 ARN) 中找到此 ARN。ARN 格式如下：

   ```
   arn:aws:kinesis:region:account:stream/name
   ```

   region

   區域代碼，例如 us-west-2。如需詳細資訊，請參閱區域與可用區域的概念。

   帳戶

   AWS 帳戶 ID，如帳戶設定所示。

   name

   取自建立資料串流的串流名稱，即 StockTradeStream。

2. 決定要供取用者使用的 DynamoDB 資料表 (由第一個取用者執行個體所建立) 的 ARN。其值必須為以下格式：

   ```
   arn:aws:dynamodb:region:account:table/name
   ```

   區域及帳戶與前一步驟如出一轍，但此處的 name 為消費者應用程式所建立與使用的資料表的名稱。消費者所用的 KCL 將使用應用程式名稱做為資料表名稱。請使用 StockTradesProcessor，即稍後將使用的應用程式名稱。

3. 在 IAM 主控台的政策 (https://console.aws.amazon.com/iam/home#policies) 中，選擇建立政策。如果這是您第一次使用 IAM 政策，請選擇開始使用、建立政策。

4. 從 Policy Generator (政策產生器) 旁選擇 Select (選取)。

5. 選擇 Amazon Kinesis 做為 AWS 服務。

6. 選取 DescribeStream、GetShardIterator、GetRecords、PutRecord 和 PutRecords 做為允許的動作。

7. 輸入您在步驟 1 建立的 ARN。

8. 使用 Add Statement (新增陳述式) 指定以下各項：

AWS 服務	動作	ARN
Amazon DynamoDB	`CreateTable` ，`DeleteItem` ，`DescribeTable` ，`GetItem`, `PutItem`, `Scan`, `UpdateItem`	您在步驟 2 建立的 ARN
Amazon CloudWatch	`PutMetricData`	*

在無須指定 ARN 的情況下要使用星號 (*)。此處使用星號是因為 CloudWatch 中沒有任何可對其叫用 `PutMetricData` 動作的特定資源。

9. 選擇 Next Step (後續步驟)。
10. 將 Policy Name (政策名稱) 更改為 `StockTradeStreamPolicy` 並檢閱程式碼，然後選擇 Create Policy (建立政策)。

產生的政策文件應類似於以下內容：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt123",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards",
                "kinesis:DescribeStreamSummary",
                "kinesis:RegisterStreamConsumer"
            ],
            "Resource": [
```

```
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream"
            ]
        },
        {
            "Sid": "Stmt234",
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream/
*"
            ]
        },
        {
            "Sid": "Stmt456",
            "Effect": "Allow",
            "Action": [
                "dynamodb:*"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:table/
StockTradesProcessor"
            ]
        },
        {
            "Sid": "Stmt789",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

建立 IAM 使用者

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。

2. 在 Users (使用者) 頁面上，選擇 Add user (新增使用者)。
3. 對於使用者名稱，輸入 `StockTradeStreamUser`。
4. 對於 Access type (存取類型)，選擇 Programmatic access (程式設計存取)，然後選擇 Next: Permissions (下一步：許可)。
5. 選擇直接連接現有政策。
6. 依名稱搜尋您所建立的政策。選取政策名稱左側的方塊，然後選擇 Next: Review (下一步：檢閱)。
7. 檢閱詳細資訊及摘要，然後選擇 Create user (建立使用者)。
8. 複製 Access key ID (存取金鑰 ID) 並將其私下儲存。從 Secret access key (私密存取金鑰) 下方選擇 Show (顯示)，然後一併將該金鑰私下儲存。
9. 將存取金鑰和私密金鑰貼入本機檔案中，存放於只有您能夠存取的安全處。針對此應用程式，請建立名為 `~/.aws/credentials` (具有嚴格許可) 的檔案。該檔案應採用以下格式：

```
[default]
aws_access_key_id=access key
aws_secret_access_key=secret access key
```

將 IAM 政策附接至使用者

1. 在 IAM 主控台，開啟政策並選擇 政策動作。
2. 依序選擇 `StockTradeStreamPolicy` 和 Attach (附加)。
3. 依序選擇 `StockTradeStreamUser` 和 Attach Policy (附加政策)。

後續步驟

下載並建置實作程式碼

下載並建置實作程式碼

我們已提供用於進行the section called “教學課程：使用 KPL 和 KCL 1.x 處理即時庫存資料”的架構程式碼。其內容包含用以擷取股票交易串流 (生產者) 和處理資料 (消費者) 的 stub 實作。下列程序說明如何完成實作。

下載並建置實作程式碼

1. 將原始碼下載到您的電腦。

2. 依照提供的目錄結構，在您慣用的 IDE 中使用原始碼建立一個專案。
3. 將以下程式庫加入至該專案：

   - Amazon Kinesis Client Library (KCL)
   - AWS 開發套件
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (適用於 Java 的 Google 核心程式庫)
   - Jackson Annotations
   - Jackson Core
   - Jackson Databind
   - Jackson Dataformat：CBOR
   - Joda Time
4. 視您的 IDE 而定，系統可能會自動建置專案。如果沒有，請根據您的 IDE 使用適當步驟建置專案。

若您已順利完成上述步驟，即可移往下一節進行the section called “實作生產者”。如果您在建置的任何階段出現錯誤，請先查明原因並予修正後再繼續。

後續步驟

實作生產者

教學課程：使用 KPL 和 KCL 1.x 處理即時庫存資料所述的應用程式使用真實情境的股票市場交易監控。以下原則簡要說明此情境如何對應到生產者和支援的程式碼結構。

請查看原始碼並對照檢閱以下資訊。

StockTrade 類別

單次股票交易是由 `StockTrade` 類別的執行個體表示。此執行個體包含若干屬性，如股票代號、價格、股份數、交易類型 (買進或賣出) 以及唯一識別該交易的 ID。程式碼已為您實作此類別。

串流記錄

串流是一連串的記錄。記錄則是 JSON 格式的序列化 StockTrade 執行個體。例如：

```
{
  "tickerSymbol": "AMZN",
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator 類別

StockTradeGenerator 具有 getRandomTrade() 方法，每次叫用後將傳回隨機產生的新股票交易。程式碼已為您實作此類別。

StockTradesWriter 類別

生產者的 main 方法 StockTradesWriter 會持續擷取隨機交易，然後透過執行以下任務將該交易傳送至 Kinesis Data Streams：

1. 讀取串流名稱和區域名稱做為輸入。
2. 建立 AmazonKinesisClientBuilder。
3. 使用用戶端建置器設定區域、登入資料和用戶端組態。
4. 使用用戶端建置器建置 AmazonKinesis 用戶端。
5. 檢查串流是否存在且處於作用中狀態 (否則將結束此方法並顯示錯誤)。
6. 在連續迴圈中依序呼叫 StockTradeGenerator.getRandomTrade() 方法和 sendStockTrade 方法，每隔 100 毫秒將交易傳送至串流。

sendStockTrade 類別的 StockTradesWriter 方法含有以下程式碼：

```
private static void sendStockTrade(StockTrade trade, AmazonKinesis kinesisClient,
 String streamName) {
    byte[] bytes = trade.toJsonAsBytes();
    // The bytes could be null if there is an issue with the JSON serialization by
 the Jackson JSON library.
    if (bytes == null) {
        LOG.warn("Could not get JSON bytes for stock trade");
        return;
```

```
    }

    LOG.info("Putting trade: " + trade.toString());
    PutRecordRequest putRecord = new PutRecordRequest();
    putRecord.setStreamName(streamName);
    // We use the ticker symbol as the partition key, explained in the Supplemental
 Information section below.
    putRecord.setPartitionKey(trade.getTickerSymbol());
    putRecord.setData(ByteBuffer.wrap(bytes));

    try {
        kinesisClient.putRecord(putRecord);
    } catch (AmazonClientException ex) {
        LOG.warn("Error sending record to Amazon Kinesis.", ex);
    }
}
```

請參閱以下的程式碼詳解：

- PutRecord API 需要位元組陣列，您必須trade轉換為 JSON 格式。這一行程式碼將執行該項操作：

  ```
  byte[] bytes = trade.toJsonAsBytes();
  ```

- 傳送交易之前，您必須先建立新的 PutRecordRequest 執行個體 (本例中其名稱為 putRecord)：

  ```
  PutRecordRequest putRecord = new PutRecordRequest();
  ```

  每次呼叫 PutRecord 都需要串流名稱、分割區索引鍵和資料 Blob。以下程式碼使用 putRecord 物件的 setXxxx() 方法填入這些欄位：

  ```
  putRecord.setStreamName(streamName);
  putRecord.setPartitionKey(trade.getTickerSymbol());
  putRecord.setData(ByteBuffer.wrap(bytes));
  ```

  本範例使用股票代號做為分割區索引鍵，將記錄對應到特定碎片。實際上，每個碎片應該會有成千上百的分割區索引鍵，使記錄均勻地分佈於串流中。如需如何加入資料至串流的詳細資訊，請參閱將資料新增至串流。

  現在 putRecord 已準備好傳送用戶端 (put 操作)：

```
kinesisClient.putRecord(putRecord);
```

- 錯誤檢查和日誌記錄肯定是頗為實用的附加功能。此程式碼將記錄錯誤情況：

```
if (bytes == null) {
    LOG.warn("Could not get JSON bytes for stock trade");
    return;
}
```

在 put 操作的周圍加入 try/catch 區塊：

```
try {
       kinesisClient.putRecord(putRecord);
} catch (AmazonClientException ex) {
       LOG.warn("Error sending record to Amazon Kinesis.", ex);
}
```

這麼做是因為 Kinesis Data Streams put 操作可能由於網路錯誤或是串流達到其輸送量限制並受到調節而導致失敗。我們建議您仔細考慮put操作的重試政策，以避免資料遺失，例如使用重試。

- 狀態記錄也很實用，但可有可無：

```
LOG.info("Putting trade: " + trade.toString());
```

此處所示的生產者是使用 Kinesis Data Streams API 的單一記錄功能 PutRecord。實際上，如果個別的生產者將產生許多記錄，則使用 PutRecords 的多筆記錄功能並一次性傳送各個批次的記錄通常會更有效率。如需詳細資訊，請參閱將資料新增至串流。

執行生產者

1. 確認稍早 (建立 IAM 使用者 使用者時) 擷取的存取金鑰和私密金鑰對是否已儲存至 ~/.aws/credentials 檔案。
2. 使用以下引數執行 StockTradeWriter 類別：

```
StockTradeStream us-west-2
```

如果您是在 us-west-2 以外的區域建立串流，則此處必須改為指定該區域。

您應該會看到類似下列的輸出：

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

您的股票交易串流現在正由 Kinesis Data Streams 擷取中。

後續步驟

實作消費者

實作消費者

教學課程：使用 KPL 和 KCL 1.x 處理即時庫存資料所述的消費者應用程式會持續處理您在時建立的股票交易串流。隨後，其將輸出每分鐘買進和賣出最多的熱門股票。此應用程式是使用 Kinesis Client Library (KCL) 所建置，由該程式庫執行取用者應用程式常見的諸多繁重工作。如需詳細資訊，請參閱開發 KCL 1.x 消費者。

請查看原始碼並對照檢閱以下資訊。

StockTradesProcessor 類別

供您使用的消費者主要類別，將執行以下任務：

- 讀取以引數形式傳入的應用程式名稱、串流名稱和區域名稱。
- 從 `~/.aws/credentials` 讀取登入資料。
- 建立 `RecordProcessorFactory` 執行個體以提供由 `RecordProcessor` 執行個體實作的 `StockTradeRecordProcessor` 執行個體。
- 使用 `RecordProcessorFactory` 執行個體和標準組態 (包括串流名稱、憑證及應用程式名稱) 建立 KCL 工作者。

- 工作者會為每個碎片 (已指派給此取用者執行個體) 建立新的執行緒，以持續循環從 Kinesis Data Streams 讀取記錄。接著，其將叫用 RecordProcessor 執行個體以處理收到的各個批次記錄。

StockTradeRecordProcessor 類別

RecordProcessor 執行個體的實作，而此執行個體將實作三個必要的方法：initialize、processRecords 和 shutdown。

顧名思義，initialize 和 shutdown 分別供 Kinesis Client Library 用於使記錄處理器得知何時應準備好開始接收記錄以及何時應停止接收記錄，好讓程式庫能夠執行任何應用程式特定的設定和終止任務。這些方法的程式碼已為您提供。主要處理任務在 processRecords 方法中進行，而此方法將使用 processRecord 處理每筆記錄。後一種方法以幾乎全空的架構程式碼提供，讓您於下一個步驟進行實作，屆時將會有進一步說明。

另請注意 processRecord 的支援方法 reportStats 和 resetStats 的實作，其最初的原始碼為全空。

程式碼已為您實作 processRecords 方法，並將執行以下步驟：

- 對每一筆傳入的記錄呼叫 processRecord。
- 若自從上次報告後已歷時至少 1 分鐘，請先呼叫 reportStats() 列印出最新統計資料，接著呼叫 resetStats() 清除統計資料以使下一個間隔僅包含新記錄。
- 設定下一次報告時間。
- 若自從最後一個檢查點過後已歷時至少 1 分鐘，請呼叫 checkpoint()。
- 設定下一次檢查點作業時間。

此方法使用 60 秒的間隔做為報告及檢查點作業率。如需檢查點作業的詳細資訊，請參閱有關消費者的其他資訊。

StockStats 類別

此類別針對一段時間內最熱門的股票提供資料保留與統計資料追蹤。其程式碼已為您提供且包含下列方法：

- addStockTrade(StockTrade)：將給定的 StockTrade 注入目前統計資料。
- toString()：以格式化字串的形式傳回統計資料。

此類別透過保留每個股票交易總數的執行中計數和最大計數，來追蹤最熱門的股票。每當股票交易達成時，其將更新這些計數。

為 StockTradeRecordProcessor 類別的各個方法加入程式碼，如以下步驟所示。

實作消費者

1. 實作 processRecord 方法，藉此執行個體化正確大小的 StockTrade 物件並將記錄資料加入該物件，且於發生問題時記錄警告。

```
StockTrade trade = StockTrade.fromJsonAsBytes(record.getData().array());
if (trade == null) {
    LOG.warn("Skipping record. Unable to parse record into StockTrade. Partition
 Key: " + record.getPartitionKey());
    return;
}
stockStats.addStockTrade(trade);
```

2. 實作簡易的 reportStats 方法。輸出格式可依照您的偏好逕自修改。

```
System.out.println("****** Shard " + kinesisShardId + " stats for last 1 minute
 ******\n" +
                   stockStats + "\n" +

 "*****************************************************************\n");
```

3. 最後，實作 resetStats 方法以便建立新的 stockStats 執行個體。

```
stockStats = new StockStats();
```

執行消費者

1. 執行您在時撰寫的生產者，將模擬的股票交易記錄注入您的串流。
2. 確認稍早 (建立 IAM 使用者 使用者時) 擷取的存取金鑰和私密金鑰對是否已儲存至 ~/.aws/credentials 檔案。
3. 使用以下引數執行 StockTradesProcessor 類別：

```
StockTradesProcessor StockTradeStream us-west-2
```

請注意，如果您是在 us-west-2 以外的區域建立串流，則此處必須改為指定該區域。

一分鐘後，您應會看到類似以下內容的輸出，而且此後每分鐘將重新整理一次輸出：

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
****************************************************************
```

有關消費者的其他資訊

如果您已熟悉 Kinesis Client Library 的優點 (誠如 開發 KCL 1.x 消費者 及其他各處的介紹)，可能會質疑為何應該在此使用該程式庫。縱然您只使用單一碎片串流和單一取用者執行個體進行處理，但使用 KCL 實作取用者還是會更加輕鬆。將生產者一節的程式碼實作步驟對比消費者，您會發現實作消費者相較之下容易些。這主要是因為 KCL 提供的服務。

在此應用程式中，您專注於實作可處理個別記錄的記錄處理器類別。您不必擔心如何從 Kinesis Data Streams 擷取記錄；每當有新記錄可用時，KCL 就會擷取該記錄並調用記錄處理器。此外，您也不需要為碎片和消費者執行個體的數目傷腦筋。如果串流已擴展，您無須重新撰寫應用程式就能處理多個碎片或多個消費者執行個體。

檢查點一詞表示將串流中的點記錄到到目前為止已耗用和處理的資料記錄。如果應用程式當機，則會從該點讀取串流，而不是從串流的開頭讀取。檢查點作業的主題及各種設計模式與最佳實務已超出本章討論範圍。不過，生產環境可能要面臨這方面的問題。

正如 一節所述，Kinesis Data Streams API 的 put 操作接受分割區索引鍵做為輸入。Kinesis Data Streams 使用分割區索引鍵做為跨多個碎片分割記錄的機制 (若串流中有多個碎片)。相同的分割區索引鍵一律會路由至同一碎片。這使您能夠憑藉以下假定狀況，設計用於處理特定碎片的消費者：具有相同分割區索引鍵的記錄只會傳送至該消費者，凡是具有相同分割區索引鍵的記錄終究不會抵達任何其他消費者。因此，消費者的工作者可彙整具有相同分割區索引鍵的所有記錄，而不必擔心會遺失所需的資料。

在此應用程式中，取用者對記錄的處理並不密集，所以能夠使用單一碎片並由 KCL 本身的同一執行緒進行處理。然而若是實際應用，請首先考慮擴展碎片數目。在某些情況下，您可能要切換由另一執行緒處理，或者預料將需密集處理記錄時使用執行緒集區。藉此，KCL 便能更快速擷取新記錄，而其他執行緒則可並行處理記錄。多執行緒設計並不微不足道，應該使用進階技術來處理，因此增加碎片計數通常是向上擴展的最有效方法。

後續步驟

（選用）擴展消費者

（選用）擴展消費者

教學課程：使用 KPL 和 KCL 1.x 處理即時庫存資料所述的應用程式可能已足以達到您的目的。此一選用章節示範如何就更為複雜的情境擴充消費者程式碼。

如果您想知道每分鐘銷售量最高的訂單，則可修改三處位置的 StockStats 類別以納入這項新的優先等級。

擴充消費者

1. 加入新的執行個體變數：

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. 為 addStockTrade 添加以下程式碼：

```
if (type == TradeType.SELL) {
    if (largestSellOrderStock == null || trade.getQuantity() >
largestSellOrderQuantity) {
        largestSellOrderStock = trade.getTickerSymbol();
        largestSellOrderQuantity = trade.getQuantity();
    }
}
```

3. 修改 toString 方法以列印其他資訊：

```
public String toString() {
    return String.format(
            "Most popular stock being bought: %s, %d buys.%n" +
            "Most popular stock being sold: %s, %d sells.%n" +
            "Largest sell order: %d shares of %s.",
            getMostPopularStock(TradeType.BUY),
getMostPopularStockCount(TradeType.BUY),
            getMostPopularStock(TradeType.SELL),
getMostPopularStockCount(TradeType.SELL),
            largestSellOrderQuantity, largestSellOrderStock);
}
```

若您現在執行消費者 (記得也要執行生產者)，應會看到類似以下內容的輸出：

```
****** Shard shardId-000000000001 stats for last 1 minute ******
 Most popular stock being bought: WMT, 27 buys.
 Most popular stock being sold: PTR, 14 sells.
 Largest sell order: 996 shares of BUD.
 ****************************************************************
```

後續步驟

清除資源

清除資源

由於使用 Kinesis 資料串流需要支付費用，當您使用串流完畢後，請務必刪除該串流和對應的 Amazon DynamoDB 資料表。作用中的串流即便並未傳送及取得記錄，也會象徵性地收取費用。這是因為作用中的串流會持續「監聽」傳入的記錄和取得記錄的請求，以致將耗用資源。

刪除串流和資料表

1. 關閉任何可能仍在執行中的生產者和消費者。
2. 在以下網址開啟 Kinesis 主控台：https://console.aws.amazon.com/kinesis。
3. 選擇您為此應用程式所建立的串流 (`StockTradeStream`)。
4. 選擇 Delete Stream (刪除串流)。
5. 請在 https://console.aws.amazon.com/dynamodb/ 開啟 DynamoDB 主控台。
6. 刪除 `StockTradesProcessor` 資料表。

摘要

近乎即時地處理大量資料，不需要撰寫複雜的程式碼或開發巨大的基礎設施。它與編寫邏輯來處理少量資料（例如寫入 `processRecord(Record)`）一樣基本，但使用 Kinesis Data Streams 進行擴展，使其適用於大量串流資料。您不必擔心如何擴展處理方式，因為這一切 Kinesis Data Streams 都將為您代勞。您只需要傳送串流記錄至 Kinesis Data Streams 並撰寫邏輯以處理每一筆收到的新記錄。

以下是對此應用程式可行的一些強化功能。

跨所有碎片進行彙整

目前您是透過彙整單一工作者從單一碎片接收的資料記錄來取得統計資料 (任一碎片不能同時由單一應用程式中的多個工作者處理)。當然，您若擴展為具有多個碎片，可能會想要跨所有碎片進行彙整。為此，您可以採用某種流程架構，將每個工作者的輸出都饋送至具有單一碎片的另一串流，而碎片則由彙整第一個階段輸出的工作者處理。由於取自第一個階段的資料有限 (每個碎片每分鐘一次取樣)，單一碎片就能輕鬆處理這些資料。

擴展處理

當串流擴展為具有多個碎片後 (因為有許多生產者傳送資料)，擴展處理的方式即是增加更多工作者。您可以在 Amazon EC2 執行個體上執行工作者並使用 Auto Scaling 群組。

使用連接器到 Amazon S3/DynamoDB/Amazon Redshift/Storm

隨著串流持續處理，其輸出可以傳送到其他目的地。 AWS 提供連接器，將 Kinesis Data Streams 與其他 AWS 服務和第三方工具整合。

後續步驟

- 如需如何使用 Kinesis Data Streams API 操作的詳細資訊，請參閱 使用 Amazon Kinesis Data Streams API 搭配 開發生產者 適用於 Java 的 AWS SDK、使用 開發共用輸送量消費者 適用於 Java 的 AWS SDK，及 建立和管理 Kinesis 資料串流。
- 如需 Kinesis Client Library 的相關詳細資訊，請參閱 開發 KCL 1.x 消費者。
- 如需如何最佳化應用程式的詳細資訊，請參閱最佳化 Amazon Kinesis Data Streams 取用者。

教學課程：使用 Amazon Managed Service for Apache Flink 分析即時股票資料

本教學課程的情境涉及將股票交易擷取至資料串流和撰寫簡單的 Amazon Managed Service for Apache Flink 應用程式對該串流執行計算。您將了解如何將記錄串流傳送至 Kinesis Data Streams，並實作應用程式，以近乎即時的方式取用和處理記錄。

透過 Amazon Managed Service for Apache Flink，您可以使用 Java 或 Scala 來處理和分析串流資料。此服務可讓您針對串流來源撰寫和執行 Java 或 Scala 程式碼，以執行時間序列分析、饋送即時儀表板，以及建立即時指標。

您可以使用基於 Apache Flink 的開放原始碼程式庫，在 Apache Flink 的受管服務中建置 Flink 應用程式。Apache Flink 是處理資料串流的熱門框架及引擎。

> **Important**
>
> 建立兩個資料串流和應用程式之後，您的帳戶會產生 Kinesis Data Streams 和 Managed Service for Apache Flink 用量的名目費用，因為它們不符合 AWS 免費方案的資格。完成此應用程式後，請刪除您的 AWS 資源以停止產生費用。

程式碼不會存取實際股票市場資料，而是模擬股票交易串流。它會使用隨機股票交易產生器來執行此作業。若您能夠存取即時股票交易串流，可能會希望從該串流衍生出實用且及時的統計資料。例如，您可能想要執行滑動時段分析，以得知前 5 分鐘內購買的最熱門股票。或者，您可能希望在銷售訂單過大 (即股份過多) 時接獲通知。您可透過擴展此系列程式碼以提供這類功能。

顯示的範例使用美國西部 (奧勒岡) 區域，但它們適用於任何支援 Managed Service for Apache Flink 的AWS 區域。

任務

- 完成練習的先決條件
- 設定 AWS 帳戶並建立管理員使用者
- 設定 AWS Command Line Interface (AWS CLI)
- 建立並執行 Managed Service for Apache Flink 應用程式

完成練習的先決條件

若要完成本指南中的步驟，您必須執行下列各項：

- Java 開發套件 (JDK) 版本 8。將 JAVA_HOME 環境變數設為指向您的 JDK 安裝位置。
- 我們建議您使用開發環境 (如 Eclipse Java Neon 或 IntelliJ Idea) 來開發和編譯您的應用程式。
- Git 用戶端。 如果您尚未安裝 Git 用戶端，請先安裝。
- Apache Maven 編譯器外掛程式。Maven 必須在您的工作路徑中。若要測試您的 Apache Maven 安裝，輸入以下資訊：

```
$ mvn -version
```

開始執行，請移至 設定 AWS 帳戶並建立管理員使用者。

設定 AWS 帳戶並建立管理員使用者

首次使用 Amazon Managed Service for Apache Flink 之前，請先完成下列任務：

1. 註冊 AWS
2. 建立 IAM 使用者

註冊 AWS

當您註冊 Amazon Web Services (AWS) 時， AWS 您的帳戶會自動註冊 中的所有服務 AWS，包括 Amazon Managed Service for Apache Flink。您只需支付實際使用服務的費用。

使用 Managed Service for Apache Flink，您僅需按使用的資源量付費。如果您是 AWS 新客戶，可免費開始使用 Managed Service for Apache Flink。如需詳細資訊，請參閱 AWS 免費方案。

如果您已有 AWS 帳戶，請跳到下一個任務。如果您尚未擁有 AWS 帳戶，請依照以下步驟建立一個帳戶。

建立 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或簡訊，並在電話鍵盤輸入驗證碼。

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

請記下 AWS 您的帳戶 ID，因為下一個任務需要它。

建立 IAM 使用者

中的服務 AWS，例如 Amazon Managed Service for Apache Flink，需要您在存取憑證時提供憑證。這樣服務可以確定您是否有權存取該服務所擁有的資源。 AWS 管理主控台 需要您輸入密碼。

您可以為 AWS 您的帳戶建立存取金鑰，以存取 AWS Command Line Interface (AWS CLI) 或 API。不過，我們不建議您 AWS 使用 AWS 帳戶的登入資料來存取 。反之，我們建議您使用 AWS Identity and Access Management (IAM)。建立 IAM 使用者，並將使用者新增至擁有管理許可的 IAM 群組，

然後將管理許可授予您建立的 IAM 使用者。您可以使用特殊 URL 與該 IAM 使用者的登入資料來存取 AWS 。

如果您註冊 AWS，但尚未為自己建立 IAM 使用者，您可以使用 IAM 主控台建立一個使用者。

本指南中的入門練習假設您有一個具備管理員權限的使用者 (adminuser)。請遵循程序在您的帳戶中建立 adminuser。

建立管理員的群組

1. 登入 AWS 管理主控台 ，並在 https：//https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇 Groups (群組)，然後選擇 Create New Group (建立新群組)。
3. 針對 Group Name (群組名稱)，輸入您群組的名稱，例如 **Administrators**，然後選擇 Next Step (下一步)。
4. 在政策清單中，選取 AdministratorAccess 政策旁的核取方塊。您可以使用 Filter (篩選) 功能表和 Search(搜尋) 方塊來篩選政策清單。
5. 選擇 Next Step (下一步)，然後選擇 Create Group (建立群組)。

您的新群組會列在 Group Name (群組名稱) 底下。

為您自己建立 IAM 使用者，將使用者新增至管理員群組，以及建立密碼

1. 在導覽窗格中，選擇 使用者，然後選擇 新增使用者。
2. 在 User name (使用者名稱) 方塊中，輸入使用者名稱。
3. 選擇程式設計存取和 AWS 管理主控台存取。
4. 選擇下一步：許可。
5. 選取 Administrators (管理員) 群組旁的核取方塊。接著選擇 Next: Review (下一步：檢閱)。
6. 選擇 Create user (建立使用者)。

以新的 IAM 使用者登入

1. 登出 AWS 管理主控台。
2. 使用以下 URL 格式來登入主控台：

```
https://aws_account_number.signin.aws.amazon.com/console/
```

aws_account_number 是不含任何連字號 AWS 的帳戶 ID。例如，如果 AWS 您的帳戶 ID 為 1234-5678-9012，請將 *aws_account_number* 取代為 **123456789012**。如需如何尋找帳號的資訊，請參閱《IAM 使用者指南》中的AWS 您的帳戶 ID 及其別名。

3. 輸入您剛才建立的 IAM 使用者名稱和密碼。登入時瀏覽列會顯示 *your_user_name* @ *your_aws_account_id*。

> **Note**
>
> 如果您不希望登入頁面的 URL 包含 AWS 您的帳戶 ID，您可以建立帳戶別名。

建立或移除帳戶別名

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇 Dashboard (儀表板)。
3. 找到 IAM 使用者登入連結。
4. 若要建立別名，請選擇 Customize (自訂)。輸入要用作別名的名稱，然後選擇是，建立。
5. 若要移除別名，請選擇 Customize (自訂)，然後選擇 Yes, Delete (是的，請刪除)。登入 URL 會使用 AWS 您的帳戶 ID 還原為 。

若要在建立帳戶別名後登入，請使用下列 URL：

`https://your_account_alias.signin.aws.amazon.com/console/`

若要驗證帳戶的 IAM 使用者的登入連結，請開啟 IAM 主控台，然後在儀表板的 IAM users sign-in link (IAM 使用者登入連結) 下方檢查。

如需 IAM 的詳細資訊，請參閱下列各項：

- AWS Identity and Access Management (IAM)
- IAM 入門
- IAM 使用者指南

下一步驟

設定 AWS Command Line Interface (AWS CLI)

設定 AWS Command Line Interface (AWS CLI)

在此步驟中，您會下載並設定 AWS CLI 以搭配 Amazon Managed Service for Apache Flink 使用。

> **Note**
>
> 本指南的入門練習均假設您使用帳戶中的管理員登入資料 (adminuser) 來執行操作。

> **Note**
>
> 如果您已 AWS CLI 安裝 ，您可能需要升級 以取得最新功能。如需詳細資訊，請參閱AWS Command Line Interface 《 使用者指南》中的安裝 AWS 命令列界面。若要檢查 的版本 AWS CLI，請執行下列命令：
>
> ```
> aws --version
> ```
>
> 本教學課程中的練習需要下列 AWS CLI 版本 或更新版本：
>
> ```
> aws-cli/1.16.63
> ```

若要設定 AWS CLI

1. 下載和設定 AWS CLI。如需相關指示，請參閱《AWS Command Line Interface 使用者指南》中的下列主題：
   - 安裝 AWS Command Line Interface
   - 設定 AWS CLI
2. 在 AWS CLI 組態檔案中為管理員使用者新增具名設定檔。您在執行 AWS CLI 命令時使用此設定檔。如需具名描述檔的詳細資訊，請參閱《AWS Command Line Interface 使用者指南》中的具名描述檔。

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

如需可用 AWS 區域的清單，請參閱 中的AWS 區域和端點Amazon Web Services 一般參考。

3. 在命令提示字元中輸入下列 help 命令，以驗證設定：

```
aws help
```

設定 AWS 帳戶和 之後 AWS CLI，您可以嘗試下一個練習，在其中設定範例應用程式並測試end-to-end設定。

下一步驟

建立並執行 Managed Service for Apache Flink 應用程式

建立並執行 Managed Service for Apache Flink 應用程式

在本練習中，您會建立 Managed Service for Apache Flink 應用程式，並將資料串流作為來源和目的地。

本節包含下列步驟：

- 建立兩個 Amazon Kinesis 資料串流
- 寫入範例記錄至輸入串流
- 下載並檢查 Apache Flink 串流 Java 程式碼
- 編譯應用程式程式碼
- 上傳 Apache Flink 串流 Java 程式碼
- 建立並執行 Managed Service for Apache Flink 應用程式

建立兩個 Amazon Kinesis 資料串流

為此練習建立 Amazon Managed Service for Apache Flink 之前，請建立兩個 Kinesis 資料串流 (`ExampleInputStream` 和 `ExampleOutputStream`)。您的應用程式會將這些串流用於應用程式來源和目的地串流。

您可以使用 Amazon Kinesis 主控台或以下 AWS CLI 命令來建立這些串流。如需主控台說明，請參閱建立及更新資料串流。

建立資料串流 (AWS CLI)

1. 若要建立第一個串流 (ExampleInputStream)，請使用下列 Amazon Kinesis create-stream AWS CLI 命令。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 若要建立應用程式用來寫入輸出的第二個串流，請執行相同的命令，將串流名稱變更為 ExampleOutputStream。

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

寫入範例記錄至輸入串流

在本節，您會使用 Python 指令碼將範例記錄寫入供應用程式處理的串流。

> **Note**
>
> 本節需要 適用於 Python (Boto) 的 AWS SDK。

1. 使用下列內容建立名為 stock.py 的檔案：

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
```

```
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. 在教學課程後半段，您會執行 stock.py 指令碼來傳送資料至應用程式。

```
$ python stock.py
```

下載並檢查 Apache Flink 串流 Java 程式碼

此範例的 Java 應用程式碼可從 GitHub 下載。若要下載應用程式的程式碼，請執行下列動作：

1. 使用以下指令複製遠端儲存庫：

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-
examples.git
```

2. 導覽至 GettingStarted 目錄。

應用程式碼位於 CustomSinkStreamingJob.java 和 CloudWatchLogSink.java 檔案。請留意下列與應用程式的程式碼相關的資訊：

- 應用程式使用 Kinesis 來源從來源串流讀取。以下程式碼片段會建立 Kinesis 目的地：

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
```

```
                new SimpleStringSchema(), inputProperties));
```

編譯應用程式程式碼

在本節中，您會使用 Apache Maven 編譯器來建立應用程式的 Java 程式碼。如需安裝 Apache Maven 和 Java 開發套件 (JDK) 的相關資訊，請參閱 完成練習的先決條件。

Java 應用程式需要下列元件：

- 專案物件模型 (pom.xml) 檔案。此檔案包含應用程式組態和相依性的相關資訊，包括 Amazon Managed Service for Apache Flink 程式庫。
- 包含應用程式邏輯的 `main` 方法。

> **Note**
>
> 若要將 Kinesis 連接器用於下列應用程式，您必須下載連接器的原始程式碼，並依照 Apache Flink 文件所述進行建置。

建立和編譯應用程式碼

1. 在您的開發環境中建立 Java/Maven 應用程式。如需建立應用程式的詳細資訊，請參閱您開發環境的文件：
   - 建立您的第一個 Java 專案 (Eclipse Java Neon)
   - 建立、執行及封裝您的第一個 Java 應用程式 (IntelliJ Idea)
2. 將以下程式碼用於名為 `StreamingJob.java` 的檔案。

```
package com.amazonaws.services.kinesisanalytics;

import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
import
 org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;
```

```
import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {

    private static final String region = "us-east-1";
    private static final String inputStreamName = "ExampleInputStream";
    private static final String outputStreamName = "ExampleOutputStream";

    private static DataStream<String>
 createSourceFromStaticConfig(StreamExecutionEnvironment env) {
        Properties inputProperties = new Properties();
        inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

 inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
 "LATEST");

        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(), inputProperties));
    }

    private static DataStream<String>
 createSourceFromApplicationProperties(StreamExecutionEnvironment env)
            throws IOException {
        Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(),
                applicationProperties.get("ConsumerConfigProperties")));
    }

    private static FlinkKinesisProducer<String> createSinkFromStaticConfig() {
        Properties outputProperties = new Properties();
        outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
        outputProperties.setProperty("AggregationEnabled", "false");

        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
 SimpleStringSchema(), outputProperties);
        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
        return sink;
    }
```

```
    private static FlinkKinesisProducer<String>
 createSinkFromApplicationProperties() throws IOException {
        Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
 SimpleStringSchema(),
                applicationProperties.get("ProducerConfigProperties"));

        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
        return sink;
    }

    public static void main(String[] args) throws Exception {
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

        /*
         * if you would like to use runtime configuration properties, uncomment the
         * lines below
         * DataStream<String> input = createSourceFromApplicationProperties(env);
         */

        DataStream<String> input = createSourceFromStaticConfig(env);

        /*
         * if you would like to use runtime configuration properties, uncomment the
         * lines below
         * input.addSink(createSinkFromApplicationProperties())
         */

        input.addSink(createSinkFromStaticConfig());

        env.execute("Flink Streaming Java API Skeleton");
    }
}
```

請注意下列關於上述程式碼範例的事項：

- 此檔案包含定義應用程式功能的 `main` 方法。

- 您的應用程式會建立來源與目的地連接器，以使用 StreamExecutionEnvironment 物件來存取外部資源。
- 應用程式會使用靜態屬性來建立來源與目的地連接器。若要使用動態應用程式屬性，請使用 createSourceFromApplicationProperties 和 createSinkFromApplicationProperties 方法來建立連接器。這些方法會讀取應用程式的屬性，來設定連接器。

3. 請將應用程式的程式碼編譯並封裝成 JAR 檔案，以使用應用程式的程式碼。您可以使用下列兩種方式的其中之一，編譯和封裝您的程式碼：

   - 使用命令列 Maven 工具。請在包含 pom.xml 檔案的目錄中執行下列命令，來建立 JAR 檔案：

     ```
     mvn package
     ```

   - 設定開發環境。如需詳細資訊，請參閱您的開發環境文件。

   您可以將您的套件做為 JAR 檔案上傳，或壓縮您的套件並做為 ZIP 檔案上傳。如果您使用 建立應用程式 AWS CLI，您可以指定程式碼內容類型 (JAR 或 ZIP)。

4. 如果編譯時發生錯誤，請確認您的 JAVA_HOME 環境變數是否正確設定。

如果應用程式成功編譯，則會建立下列檔案：

```
target/java-getting-started-1.0.jar
```

上傳 Apache Flink 串流 Java 程式碼

在本節中，您會建立 Amazon Simple Storage Service (Amazon S3) 儲存貯體並上傳您的應用程式的程式碼。

上傳應用程式的程式碼

1. 開啟位於 https://console.aws.amazon.com/s3/ 的 Amazon S3 主控台。
2. 選擇建立儲存貯體。
3. 在儲存貯體名稱欄位中，輸入 **ka-app-code-*<username>***。新增尾碼至儲存貯體名稱，例如您的使用者名稱，使其成為全域唯一的。選擇下一步。
4. 在設定選項步驟中，保留原有設定並選擇 Next (下一步)。
5. 在設定許可步驟中，保留原有設定並選擇 Next (下一步)。
6. 選擇建立儲存貯體。

7. 在 Amazon S3 主控台中，選擇 ka-app-code-***<username>*** 儲存貯體，並選擇上傳。
8. 在選取檔案步驟中，選擇新增檔案。導覽至您在上一步驟中建立的 `java-getting-started-1.0.jar` 檔案。選擇下一步。
9. 在設定許可步驟中，保留原有設定。選擇下一步。
10. 在設定屬性步驟中，保留原有設定。選擇上傳。

您的應用程式的程式碼現在儲存在您的應用程式可以存取的 Amazon S3 儲存貯體中。

建立並執行 Managed Service for Apache Flink 應用程式

您可以使用主控台或 AWS CLI建立和執行 Managed Service for Apache Flink 應用程式。

> **Note**
>
> 當您使用主控台建立應用程式時，系統會為您建立 AWS Identity and Access Management (IAM) 和 Amazon CloudWatch Logs 資源。當您使用 建立應用程式時 AWS CLI，您可以分別建立這些資源。

主題

- 建立並執行應用程式 （主控台）
- 建立並執行應用程式 (AWS CLI)

建立並執行應用程式 （主控台）

依照以下步驟來使用主控台建立、設定、更新及執行應用程式。

建立應用程式

1. 在以下網址開啟 Kinesis 主控台：https://console.aws.amazon.com/kinesis。
2. 在 Amazon Kinesis 儀表板上，選擇建立分析應用程式。
3. 在 Kinesis Analytics - Create application (Kinesis 分析 - 建立應用程式) 頁面，請如下所述提供應用程式詳細資訊：
   - 在應用程式名稱中，輸入 **`MyApplication`**。
   - 對於 Description (說明)，輸入 **`My java test app`**。

- 針對 Runtime (執行時間)，選擇 Apache Flink 1.6。

4. 對於存取許可，請選擇建立/更新 IAM 角色 **kinesis-analytics-MyApplication-us-west-2**。
5. 選擇 建立應用程式。

> **Note**
>
> 當您使用主控台建立 Amazon Managed Service for Apache Flink 應用程式時，您可以選擇為您的應用程式建立 IAM 角色和政策。應用程式使用此角色和政策來存取其相依資源。這些 IAM 資源會如下所述使用您的應用程式名稱和區域命名：
>
> - 政策：kinesis-analytics-service-*MyApplication*-*us-west-2*
> - 角色：kinesis-analytics-*MyApplication*-*us-west-2*

編輯 IAM 政策

編輯 IAM 政策來新增存取 Kinesis 資料串流的許可。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇政策。選擇主控台為您在上一節所建立的 **kinesis-analytics-service-MyApplication-us-west-2** 政策。
3. 在摘要頁面，選擇編輯政策。請選擇 JSON 標籤。
4. 將下列政策範例的反白部分新增至政策。使用您的帳戶 ID 取代範例帳戶 ID (*012345678901*)。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-
started-1.0.jar"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
```

```
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

設定應用程式

1. 在我的應用程式頁面，選擇設定。
2. 在設定應用程式頁面，提供程式碼位置：
   - 對於 Amazon S3 儲存貯體，請輸入 **ka-app-code-*<username>***。
   - 對於 Amazon S3 物件的路徑，請輸入 **java-getting-started-1.0.jar**。
3. 在存取應用程式資源下，對於存取許可，選擇建立/更新 IAM 角色 **kinesis-analytics-MyApplication-us-west-2**。
4. 在屬性下，為群組 ID輸入 **ProducerConfigProperties**。
5. 輸入以下應用程式屬性和數值：

金鑰	值
flink.inputstream.initpos	**LATEST**
aws:region	**us-west-2**
AggregationEnabled	**false**

6. 在監控下，確保監控指標層級設為應用程式。
7. 針對 CloudWatch 記錄，選取啟用核取方塊。
8. 選擇更新。

> **Note**
>
> 當您選擇啟用 CloudWatch 記錄時，Managed Service for Apache Flink 便會為您建立日誌群組和日誌串流。這些資源的名稱如下所示：
>
> - 日誌群組：`/aws/kinesis-analytics/MyApplication`
> - 日誌串流：`kinesis-analytics-log-stream`

執行應用程式

1. 在 MyApplication 頁面，選擇執行。確認動作。
2. 應用程式執行時，重新整理頁面。主控台會顯示 Application graph (應用程式圖形)。

停止應用程式

在 MyApplication 頁面，選擇停止。確認動作。

更新應用程式

您可以使用主控台更新應用程式設定，例如應用程式屬性、監控設定及位置或應用程式 JAR 的檔名。如果需要更新應用程式的程式碼，也可以從 Amazon S3 儲存貯體重新載入應用程式 JAR。

在 MyApplication 頁面，選擇設定。更新應用程式設定，然後選擇更新。

建立並執行應用程式 (AWS CLI)

在本節中，您可以使用 AWS CLI 來建立和執行 Managed Service for Apache Flink 應用程式。Managed Service for Apache Flink 使用 `kinesisanalyticsv2` AWS CLI 命令來建立 Managed Service for Apache Flink 應用程式並與之互動。

建立許可政策

您會先建立具有兩條陳述式的許可政策：一條陳述式授與來源串流上 `read` 動作的許可，而另一條則是授與目的地串流上 `write` 動作的許可。您之後會將政策連接至 IAM 角色 (您會在下一節中建立)。因此，當 Managed Service for Apache Flink 擔任角色時，服務便具有從來源串流讀取並寫入目的地串流的所需許可。

使用以下程式碼來建立 `KAReadSourceStreamWriteSinkStream` 許可政策。以您用於建立 Amazon S3 儲存貯體 (以儲存應用程式的程式碼) 的使用者名稱來取代 *`username`*。使用您的帳戶 ID 取代 Amazon Resource Name (ARN) (*`012345678901`*) 中的帳戶 ID。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

如需建立許可政策的逐步指示，請參閱《IAM 使用者指南》中的教學課程：建立和連接您的第一個客戶管理政策。

> **Note**
>
> 若要存取其他 AWS 服務，您可以使用 適用於 Java 的 AWS SDK。Managed Service for Apache Flink 自動將 SDK 所需的憑證設定為與應用程式相關聯的服務執行 IAM 角色。無須採取額外的步驟。

建立 IAM 角色

在本節中，您將建立 Managed Service for Apache Flink 可擔任的 IAM 角色，以讀取來源串流並寫入目的地串流。

Managed Service for Apache Flink 沒有許可，無法存取串流。您可以透過 IAM 角色來授與這些許可。各 IAM 角色都有連接兩項政策。信任政策會授與擔任角色的 Managed Service for Apache Flink 許可，而許可政策決定了 Managed Service for Apache Flink 在擔任角色後可以執行的作業。

您會將在上一節中建立的許可政策連接至此角色。

若要建立一個 IAM 角色

1. 前往網址 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇角色 、建立角色。
3. 在選取可信身分類型下，選擇 AWS 服務。在選擇將使用此角色的服務下，選擇 Kinesis。在 Select your use case (選取您的使用案例) 下，選擇 Kinesis Analytics (Kinesis 分析)。

   選擇下一步：許可。
4. 在連接許可政策頁面，選擇下一步：檢閱。您會在建立角色後連接許可政策。
5. 在建立角色頁面，輸入 **`KA-stream-rw-role`** 作為角色名稱。選擇建立角色。

   現在您已建立新的 IAM 角色，名為 `KA-stream-rw-role`。您接著會更新角色的信任和許可政策。
6. 將 許可政策連接到角色。

   > **Note**
   >
   > 在此練習中，Managed Service for Apache Flink 擔任從 Kinesis 資料串流 (來源) 讀取資料並將輸出寫入另一個 Kinesis 資料串流的角色。因此您會連接在上一個步驟中建立的政策，the section called “建立許可政策”。

a. 在摘要頁面，選擇許可標籤。

b. 選擇連接政策。

c. 在搜尋方塊中，輸入 **KAReadSourceStreamWriteSinkStream** (您在上一節中建立的政策)。

d. 選擇 KAReadInputStreamWriteOutputStream 政策，然後選擇 Attach policy (連接政策)。

您現在已建立應用程式用於存取資源的服務執行角色。請記下新角色的 ARN。

如需建立角色的逐步說明，請參閱《IAM 使用者指南》中的建立 IAM 角色 (主控台)。

建立 Managed Service for Apache Flink 應用程式

1. 將下列 JSON 程式碼複製到名為 `create_request.json` 的檔案。使用您之前建立之角色的 ARN，取代範例角色 ARN。使用您在上一節中選擇的尾碼取代儲存貯體 ARN 尾碼 (*username*)。使用您的帳戶 ID 取代服務執行角色中的範例帳戶 ID (*012345678901*)。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_6",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
```

```
                }
            },
            {
                "PropertyGroupId": "ConsumerConfigProperties",
                "PropertyMap" : {
                     "aws.region" : "us-west-2"
                }
            }
         ]
      }
    }
}
```

2. 使用前述請求執行 CreateApplication 動作以建立應用程式：

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

應用程式現在已建立。您會在下一個步驟中啟動應用程式。

啟動應用程式

在本節中，您會透過 StartApplication 動作來啟動應用程式。

啟動應用程式

1. 將下列 JSON 程式碼複製到名為 start_request.json 的檔案。

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 以啟動應用程式的上述請求，執行 StartApplication 動作：

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

應用程式現在正在執行。您可以在 Amazon CloudWatch 主控台上查看 Managed Service for Apache Flink 指標，以確認應用程式是否正常運作。

停止應用程式

在本節，您會使用該 StopApplication 動作來停止應用程式。

停止應用程式

1. 將下列 JSON 程式碼複製到名為 stop_request.json 的檔案。

```
{"ApplicationName": "test"
}
```

2. 以停止應用程式的上述請求，執行 StopApplication 動作：

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

現在已停止應用程式。

教學課程： AWS Lambda 搭配 Amazon Kinesis Data Streams 使用

在本教學課程中，您會建立 Lambda 函數，以取用 Kinesis 資料串流中的事件。在此範例案例中，自訂應用程式會將記錄寫入 Kinesis 資料串流。 AWS Lambda 然後輪詢此資料串流，並在偵測到新資料記錄時叫用您的 Lambda 函數。 AWS Lambda 然後， 會透過擔任您在建立 Lambda 函數時指定的執行角色來執行 Lambda 函數。

如需詳細的逐步說明，請參閱教學課程：搭配 Amazon Kinesis 使用 AWS Lambda。

> **Note**
>
> 本教學假設您對基本 Lambda 操作和 AWS Lambda 主控台有一些了解。如果您尚未建立，請遵循 AWS Lambda 入門中的指示來建立您的第一個 Lambda 函數。

使用 AWS Amazon Kinesis 的串流資料解決方案

Amazon Kinesis AWS 的串流資料解決方案會自動設定所需的 AWS 服務，以輕鬆擷取、儲存、處理和交付串流資料。解決方案提供多種選項來解決使用 Kinesis Data Streams AWS Lambda、Amazon API Gateway 和 Amazon Managed Service for Apache Flink 等多項 AWS 服務的串流資料使用案例。

每個解決方案 包含下列元件：

- 部署完整範例的 CloudFormation 套件。
- 用於顯示應用程式指標的 CloudWatch 儀表板。
- CloudWatch 會對最相關的應用程式指標發出警示。
- 所有必要的 IAM 角色和政策。

可以在這裡找到解決方案：適用於 Amazon Kinesis 的串流資料解決方案

建立和管理 Kinesis 資料串流

Amazon Kinesis Data Streams 能即時擷取大量資料、長期存放資料並使資料可供取用。Kinesis Data Streams 存放資料的單位是資料記錄。資料串流代表一組資料記錄。資料串流中的資料記錄分佈於各個碎片。

碎片具有一連串位於串流中的資料記錄。它可作為 Kinesis 資料串流的基本輸送量單位。碎片支援 1 MB/s 和 1000 筆記錄進行寫入，而在隨需和佈建容量模式下讀取則支援 2 MB /s。碎片限制確保實現可預測的效能，讓您更輕鬆地設計和操作高度可靠的資料串流工作流程。

在本節中，您將了解如何設定串流的容量模式，以及如何使用 AWS 管理主控台 或 APIs 建立串流。然後，您可以對串流採取其他動作。

主題

選擇要在 中串流的正確模式

下列主題說明如何為您的應用程式選擇最佳模式，並視需要在模式之間切換。

主題

- 隨需優勢模式功能和使用案例
- 佈建模式功能和使用案例
- 在模式之間切換

Kinesis Data Streams 中有哪些不同的模式？

模式會決定如何管理資料串流的容量，以及如何向您收取資料串流用量的費用。在 Amazon Kinesis Data Streams 中，您可以選擇隨需標準、隨需優勢，並佈建為資料串流的 模式。

- 隨需標準 - 具有隨需模式的資料串流不需要容量規劃，且會自動擴展以處理每分鐘 GB 的寫入和讀取輸送量。使用隨需模式時，Kinesis Data Streams 會自動管理碎片，以提供必要的輸送量。
- 隨需優勢 - 帳戶層級模式，可啟用更多功能，並為隨需串流提供更簡單的定價結構。在此模式中，您可以隨時主動暖機串流的寫入輸送量容量。對於定價，不再收取固定的每個串流層級費用，而且所有隨需串流的資料擷取、資料擷取和延長保留用量至少比隨需標準低 60%。
- 佈建 - 對於具有佈建模式的資料串流，您必須指定資料串流的碎片數量。資料串流的總容量是其碎片容量的總和。您可以視需要增加或減少資料串流中的碎片數量。

您可以使用 Kinesis Data Streams `PutRecord`和 `PutRecords` APIs，以任何模式將資料寫入資料串流。若要擷取資料，這三種模式都支援使用 `GetRecords` API 的預設取用者，以及使用 `SubscribeToShard` API 的增強型廣發 (EFO) 取用者。

隨需和佈建兩種模式都支援所有 Kinesis Data Streams 功能，包括保留模式、加密、監控指標和其他功能。Kinesis Data Streams 可在隨需和佈建容量兩種模式下提供高耐久性和可用性。

隨需標準模式功能和使用案例

具有隨需模式的資料串流不需要規劃容量，而且會自動擴展以處理每分鐘 GB 的寫入和讀取輸送量。隨需模式可簡化以低延遲的方式擷取和儲存大量資料，因為無需佈建和管理伺服器、儲存裝置或輸送量。您每天可以擷取數十億筆記錄，而不會產生任何營運開銷。

隨需模式非常適合應對高度變化且無法預測的應用程式流量需求。您不再需要為尖峰容量佈建這些工作負載，這可能會因為使用率低而導致更高的成本。隨需模式適用於流量模式不可預測且高度變化的工作負載。

使用隨需容量模式，您可以按從資料串流寫入和讀取的資料 GB 數量付費。不需要指定您預期應用程式將進行的讀取和寫入輸送量。Kinesis Data Streams 會在您的工作負載上升或下降時，立即因應。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 定價。

隨需模式下的資料串流最多可容納前 30 天觀察到的尖峰寫入輸送量的兩倍。當資料串流的寫入輸送量達到新的尖峰時，Kinesis Data Streams 會自動擴展資料串流的容量。例如，如果您的資料串流的寫入輸送量在 10 MB/s 和 40 MB/s 之間變化，Kinesis Data Streams 可確保您可以輕鬆提升至先前的尖峰輸送量增加一倍，或是 80 MB/s。如果相同的資料串流維持 50 MB/s 的新尖峰輸送量，則 Kinesis Data Streams 可確保有足夠的容量來擷取 100 MB/s 的寫入輸送量。但是，如果流量在 15 分鐘的持續時間內增加到前一個峰值的兩倍以上，則可能會發生寫入限流。您必須重試這些限流的請求。

使用隨需模式的資料串流彙總讀取容量會與寫入輸送量成比例增加。這有助於確保取用者應用程式永遠具有足夠的讀取輸送量來即時處理傳入資料。與使用 `GetRecords` API 讀取資料相比，您會取得至少兩倍的寫入輸送量。建議您將一個取用者應用程式與 `GetRecord` API 搭配使用，以便在應用程式需要從停機時間復原時，該應用程式有足夠的空間來跟進。對於需要新增多個取用者應用程式的案例，建議您使用 Kinesis Data Streams 的增強型散發功能。增強型散發支援使用 `SubscribeToShard` API 將最多 20 個取用者應用程式新增至資料串流，每個取用者應用程式都有專用輸送量。

處理讀取和寫入輸送量例外狀況

使用隨需模式（與佈建容量模式相同），您必須指定每個記錄的分割區索引鍵，以將資料寫入資料串流。Kinesis Data Streams 會使用您的分割區索引鍵在碎片之間分發資料。Kinesis Data Streams 會監控每個碎片的流量。當傳入流量超過每個碎片 500 KB/s 時，其會在 15 分鐘內對碎片進行分割。父碎片的雜湊索引鍵值在子碎片中均勻地重新分配。

如果傳入流量超過先前峰值的兩倍，即使資料平均分佈在碎片上，您也可以體驗大約 15 分鐘的讀取或寫入例外狀況。建議您重試所有此類請求，以便將所有記錄正確儲存在 Kinesis Data Streams 中。

如果您使用的分割區索引鍵會導致資料分配不均，且指派給特定碎片的記錄超出其限制，則可能會遇到讀取和寫入例外狀況。使用隨需模式時，資料串流會自動適應以處理不均勻的資料分配模式，除非單一分割區索引鍵超過碎片的 1 MB/s 輸送量和每秒 1000 筆記錄限制。

在隨需模式中，Kinesis Data Streams 偵測到流量增加時，會平均分割碎片。但是，它不會偵測和隔離將較高部分傳入流量導向特定碎片的雜湊索引鍵。如果您使用的是高度不均勻的分割區索引鍵，則可能會繼續收到寫入異常。如需此使用案例，建議使用支援精細碎片分割的佈建容量模式。

隨需優勢模式功能和使用案例

隨需優勢是一種帳戶層級設定，可解鎖更多功能，並為區域中所有隨需串流提供不同的定價結構。在此模式中，隨需串流會保留其功能，並繼續根據實際資料用量自動擴展容量。如果您想要主動暖機串流的寫入輸送量容量，您可以設定暖輸送量。例如，如果您的資料串流的寫入輸送量介於 10 MB/s 到 40 MB/s 之間，您可以預期它可以處理高達 80MB/s 的即時輸送量增加，而不會調節。不過，如果您預測即將發生的事件達到大約 200MB/s 的流量峰值，則可以使用 200MB/s 的暖輸送量來設定串流，以確保在資料輸送量到達時有可用的容量。使用暖輸送量不會產生額外費用。

隨需優勢模式的另一個好處是隨需串流會轉換為更簡單的定價結構。啟用 模式後，帳戶將不再看到固定的每個串流費用，而且您只會處理資料擷取、資料擷取和選用的延長保留費用。與隨需標準中的對應維度相比，每個定價維度也會有顯著的折扣。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 定價。

與此模式中的標準資料擷取相比，增強廣發資料擷取也沒有價格溢價。此外，使用隨需優勢模式，每個串流最多可註冊 50 位取用者，以使用增強廣發功能。啟用隨需優勢會將帳戶遞交至所有隨需串流至少每秒 25MiB 的資料擷取和每秒 25MiB 的資料擷取。對於符合最低用量需求的帳戶，Kinesis Data Streams 主控台會檢查您帳戶的用量模式是否適合使用隨需優勢模式。

如果您的帳戶的資料使用量低於要求，將會向您收取差額的費用，但仍會以相同的折扣費率計費。啟用隨需優勢也有最短 24 小時的期間，您才能停用 模式。整體而言，如果您的輸送量使用量接近或高於最低承諾、需要許多廣發消費者，或是使用數百個資料串流操作，則隨需優勢是使用 Kinesis Data Streams 進行串流的最佳方式。

佈建模式功能和使用案例

使用佈建模式時，在建立資料串流之後，您可以使用 或 UpdateShardCount API 動態擴展 AWS 管理主控台 或縮減碎片容量。當有 Kinesis Data Streams 生產者或取用者應用程式正在寫入或從串流讀取資料時，您可以進行更新。

佈建模式適用於容量需求且易於預測的可預測流量。如果您想要對資料在碎片之間分發的方式進行更精細控制，則可以使用佈建模式。

採用佈建模式，必須指定資料串流的碎片數目。若要決定使用佈建模式的資料串流的大小，您需要以下各項輸入值：

- 寫入串流的資料記錄其平均大小 (KB 數)，無條件進位至 1 KB (`average_data_size_in_KB`)。
- 每秒寫入串流以及從串流讀取的資料記錄筆數 (`records_per_second`)。
- 同時各自從串流取用資料的 Kinesis Data Streams 應用程式數目，即取用者數目 (`number_of_consumers`)。
- 單位為 KB 的傳入寫入頻寬 (`incoming_write_bandwidth_in_KB`)，其值等於 `average_data_size_in_KB` 乘以 `records_per_second`。
- 單位為 KB 的傳出讀取頻寬 (`outgoing_read_bandwidth_in_KB`)，其值等於 `incoming_write_bandwidth_in_KB` 乘以 `number_of_consumers`。

您可使用以下公式帶入輸入值，計算串流所需的碎片數目 (`number_of_shards`)。

```
number_of_shards = ceiling(max(incoming_write_bandwidth_in_KiB/1024,
 outgoing_read_bandwidth_in_KiB/2048))
```

如果您未設定處理尖峰輸送量的資料串流，則可能仍會在佈建模式中遇到讀取和寫入輸送量例外狀況。在這種情形下，您必須手動擴展資料串流以適應資料流量。

如果您使用的分割區索引鍵會導致資料分配不均，且指派給碎片的記錄超出其限制，則可能會遇到讀取和寫入例外狀況。若要在佈建模式中解決此問題，請識別此類碎片並手動分割，以更好地容納您的流量。如需詳細資訊，請參閱將串流重新分片。

在模式之間切換

對於 中的每個資料串流 AWS 帳戶，您可以在 24 小時內在隨需和佈建模式之間切換兩次。在模式之間切換不會對使用此資料串流的應用程式造成任何中斷。您可以繼續寫入此資料串流和從中讀取資料。當您在模式之間切換時，從隨需切換到佈建，或從佈建切換到隨需，串流的狀態會設定為更新。您必須等待資料串流狀態變為作用中，才能再次修改其屬性。

當您從佈建容量模式切換到隨需容量模式時，資料串流最初會保留轉換前的碎片計數，從此開始，Kinesis Data Streams 會監控您的資料流量，並根據您的寫入輸送量擴展此隨需資料串流的碎片計數。當您從隨需模式切換到佈建模式時，資料串流一開始也會保留在轉換之前擁有的任何碎片計數，但從這個時間點開始，您必須負責監控和調整此資料串流的碎片計數，以正確容納您的寫入輸送量。

您可以透過啟用帳戶層級設定，從隨需標準切換到隨需優勢模式。啟用時，帳戶會遞交至 區域中所有隨需串流至少 25MiB/s 的資料擷取和 25MiB/s 的資料擷取用量。啟用後，您必須等待至少 24 小時才能停用隨需優勢，但您可以隨時請求變更。如果您想要從隨需優勢切換到隨需標準，您必須先移除隨需串流設定的任何暖輸送量。

使用 建立串流 AWS 管理主控台

您可以使用 Kinesis Data Streams 主控台、Kinesis Data Streams API 或 AWS Command Line Interface (AWS CLI) 建立串流。

使用主控台建立資料串流

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/kinesis 的 Kinesis 主控台。
2. 在導覽列中，展開區域選擇工具，然後選擇一個區域。
3. 選擇 建立資料串流。

4. 在建立 Kinesis 串流頁面上，輸入資料串流的名稱，然後選擇隨需或佈建容量模式。依預設，會選取隨需模式。如需詳細資訊，請參閱選擇要在 中串流的正確模式。

   在隨需模式下，您可以選擇建立 Kinesis 串流來建立資料串流。在佈建模式下，您必須指定所需的碎片數量，然後選擇建立 Kinesis 串流。

   建立串流時，在 Kinesis 串流頁面上，串流的狀態會是正在建立。當串流就緒可供使用後，其狀態將變成作用中。

5. 選擇串流名稱。串流詳細資訊頁面會顯示串流組態的摘要以及監控資訊。

使用 Kinesis Data Streams API 建立串流

- 如需如何使用 Kinesis Data Streams API 建立串流的資訊，請參閱 使用 APIs 建立串流。

使用 建立串流 AWS CLI

- 如需有關使用 建立串流的資訊 AWS CLI，請參閱 create-stream 命令。

使用 APIs 建立串流

使用下列步驟來建立 Kinesis 資料串流。

建置 Kinesis Data Streams 用戶端

在可以使用 Kinesis 資料串流之前，必須建置用戶端物件。以下 Java 程式碼會將用戶端建置器執行個體化，並使用它來設定區域、登入資料和用戶端組態。接著會建置一個用戶端物件。

```
AmazonKinesisClientBuilder clientBuilder = AmazonKinesisClientBuilder.standard();

clientBuilder.setRegion(regionName);
clientBuilder.setCredentials(credentialsProvider);
clientBuilder.setClientConfiguration(config);

AmazonKinesis client = clientBuilder.build();
```

如需詳細資訊，請參閱 AWS 一般參考 中的 Kinesis Data Streams 區域與端點。

建立串流

現在您已建立 Kinesis Data Streams 用戶端，您可以使用 主控台或以程式設計方式建立串流。若要以程式設計方式建立串流，請執行個體化CreateStreamRequest物件並指定串流的名稱。如果您想要使用佈建模式，請指定串流要使用的碎片數量。

- 隨需：

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
```

- 佈建：

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
createStreamRequest.setShardCount( myStreamSize );
```

串流名稱可識別串流。名稱的範圍限定為應用程式所使用的 AWS 帳戶。也受限於區域。也就是說，兩個不同 AWS 帳戶中的兩個串流可以具有相同的名稱，而相同 AWS 帳戶中但兩個不同區域中的兩個串流可以具有相同的名稱，但同一個帳戶和相同區域中的兩個串流不能具有相同的名稱。

串流的輸送量是碎片數量的函數。若要提高佈建輸送量，您需要更多碎片。更多碎片也會增加串流 AWS 的費用。如需有關計算應用程式的適當碎片數量的詳細資訊，請參閱選擇要在 中串流的正確模式。

設定createStreamRequest物件之後，請在用戶端上呼叫 createStream方法來建立串流。呼叫 createStream 之後，等候串流達到 ACTIVE 狀態，之後再對串流執行任何操作。若要查看串流的狀態，請呼叫 describeStream 方法。不過，如果串流不存在，describeStream 會擲出例外狀況。因此，請將 describeStream 呼叫含括在 try/catch 區塊中。

```
client.createStream( createStreamRequest );
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime ) {
  try {
    Thread.sleep(20 * 1000);
  }
```

```
  catch ( Exception e ) {}

  try {
    DescribeStreamResult describeStreamResponse =
 client.describeStream( describeStreamRequest );
    String streamStatus =
 describeStreamResponse.getStreamDescription().getStreamStatus();
    if ( streamStatus.equals( "ACTIVE" ) ) {
      break;
    }
    //
    // sleep for one second
    //
    try {
      Thread.sleep( 1000 );
    }
    catch ( Exception e ) {}
  }
  catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime ) {
  throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

更新串流

您可以使用 Kinesis Data Streams 主控台、Kinesis Data Streams API 或 AWS CLI更新串流詳細資料。

> **Note**
>
> 您可為現有的串流或最近剛建立的串流啟用伺服器端加密。

使用主控台

使用主控台更新資料串流

1. 在 https://console.aws.amazon.com/kinesis 上開啟 Amazon Kinesis 主控台。
2. 在導覽列中，展開區域選擇工具，然後選擇一個區域。

3. 從清單中選擇串流的名稱。Stream Details (串流詳細資訊) 頁面會顯示串流組態的摘要和監控資訊。
4. 若要在資料串流的隨需和佈建容量模式之間進行切換，請在組態索引標籤中選擇編輯容量模式。如需詳細資訊，請參閱選擇要在 中串流的正確模式。

   > **Important**
   >
   > 對於您 AWS 帳戶中的每個資料串流，您可以在 24 小時內在隨需和佈建模式之間切換兩次。

5. 對於具有佈建模式的資料串流，若要編輯碎片數目，請在組態索引標籤中選擇編輯已佈建的碎片，然後輸入新的碎片計數。
6. 若要對資料記錄啟用伺服器端加密，從 Server-side encryption (伺服器端加密) 區段選擇 Edit (編輯)。選擇要使用做為加密主金鑰的 KMS 金鑰，或者使用由 Kinesis 管理的預設主金鑰 aws/kinesis。如果您啟用串流的加密並使用自己的 AWS KMS 主金鑰，請確定您的生產者和取用者應用程式可存取您使用 AWS KMS 的主金鑰。若要為應用程式指派許可使其能夠存取使用者產生的 AWS KMS 金鑰，請參閱 the section called “使用使用者產生之 KMS 金鑰的許可”。
7. 若要編輯資料保留期間，從 Data retention period (資料保留期間) 區段選擇 Edit (編輯)，然後輸入新的資料保留期間。
8. 若您已對自己的帳戶啟用自訂指標，請從 Shard level metrics (碎片級指標) 區段選擇 Edit (編輯)，然後為您的串流指定指標。如需詳細資訊，請參閱the section called “使用 CloudWatch 監控 Kinesis Data Streams 服務”。

使用 API

若要使用 API 更新串流詳細資訊，請參閱下列方法：

- AddTagsToStream
- DecreaseStreamRetentionPeriod
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- IncreaseStreamRetentionPeriod
- RemoveTagsFromStream
- StartStreamEncryption
- StopStreamEncryption

- UpdateShardCount

使用 AWS CLI

如需有關使用 更新串流的資訊 AWS CLI，請參閱 Kinesis CLI 參考。

列出串流

串流的範圍是與用於執行個體化 Kinesis Data Streams 用戶端的 AWS 登入資料相關聯的 AWS 帳戶，以及用戶端指定的區域。 AWS 帳戶一次可以有多個作用中的串流。您可以在 Kinesis Data Streams 主控台中或是以程式設計方式列出串流。本節中的程式碼說明如何列出您 AWS 帳戶的所有串流。

```
ListStreamsRequest listStreamsRequest = new ListStreamsRequest();
listStreamsRequest.setLimit(20);
ListStreamsResult listStreamsResult = client.listStreams(listStreamsRequest);
List<String> streamNames = listStreamsResult.getStreamNames();
```

此程式碼範例會先建立 `ListStreamsRequest` 的新執行個體，並呼叫它的 `setLimit` 方法來指定針對通往 `listStreams` 的每個呼叫，應該傳回最多 20 個串流。如果您不為 `setLimit` 指定值，Kinesis Data Streams 會傳回數量小於或等於帳戶中數量的一些串流。然後該程式碼會將 `listStreamsRequest` 傳遞至用戶端的 `listStreams` 方法。傳回值 `listStreams` 會存放在某個 `ListStreamsResult` 物件中。該程式碼呼叫在此物件上的 `getStreamNames` 方法，並將傳回的串流名稱存放在 `streamNames` 清單中。請注意，即使有較帳戶和區域中更多的串流，Kinesis Data Streams 可能傳回較指定的限制所指定之更少的串流。為了確保會擷取所有串流，請使用下一個程式碼範例中所述的 `getHasMoreStreams` 方法。

```
while (listStreamsResult.getHasMoreStreams())
{
    if (streamNames.size() > 0) {
      listStreamsRequest.setExclusiveStartStreamName(streamNames.get(streamNames.size()
 - 1));
    }
    listStreamsResult = client.listStreams(listStreamsRequest);
    streamNames.addAll(listStreamsResult.getStreamNames());
}
```

此程式碼會呼叫 `getHasMoreStreams` 上的 `listStreamsRequest` 方法，來查看在對 `listStreams` 的初始呼叫中傳回的那些串流以外，是否有額外的可用串流。如果有，程式碼會以之前

對 setExclusiveStartStreamName 呼叫中傳回的最後一個串流的名稱來呼叫 listStreams 方法。setExclusiveStartStreamName 方法會造成接下來對 listStreams 的呼叫在該串流之後開始。然後該呼叫傳回的串流名稱群組會新增到 streamNames 清單。此程序會持續，直到已收集清單中的所有串流名稱為止。

listStreams 傳回的串流可以是以下其中一個狀態：

- CREATING
- ACTIVE
- UPDATING
- DELETING

您可以使用 describeStream 方法來查看串流的狀態，如先前的小節使用 APIs 建立串流中所述。

列出碎片

資料串流可以有一個或多個碎片。從資料串流中列出或擷取碎片的建議方法是使用 ListShards API。下列範例顯示取得資料串流中碎片清單的方法。如需此範例主要操作的完成說明以及所有您可以為該操作設定的參數，請參閱 ListShards。

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ListShardsRequest;
import software.amazon.awssdk.services.kinesis.model.ListShardsResponse;

import java.util.concurrent.TimeUnit;

public class ShardSample {

    public static void main(String[] args) {

        KinesisAsyncClient client = KinesisAsyncClient.builder().build();

        ListShardsRequest request = ListShardsRequest
                .builder().streamName("myFirstStream")
                .build();

        try {
            ListShardsResponse response = client.listShards(request).get(5000,
 TimeUnit.MILLISECONDS);
```

```
            System.out.println(response.toString());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

若要執行前一個程式碼範例，您可以使用如下所示的 POM 檔案。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>kinesis.data.streams.samples</groupId>
    <artifactId>shards</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>kinesis</artifactId>
            <version>2.0.0</version>
        </dependency>
    </dependencies>
</project>
```

透過 `ListShards` API，您可以使用 ShardFilter 參數來篩選掉 API 的回應。一次只可以指定一個篩選條件。

如果您在調用 ListShards API 時使用 ShardFilter 參數，Type 就是必要的屬性，而且必須指定。如果指定 AT_TRIM_HORIZON、FROM_TRIM_HORIZON 或 AT_LATEST 類型，則不需要指定 ShardId 或 Timestamp 選用屬性。

如果指定 AFTER_SHARD_ID 類型，則還必須提供選用 ShardId 屬性的值。該 ShardId 屬性在功能上與 ListShards API 的 ExclusiveStartShardId 參數相同。當指定 ShardId 屬性時，回應包括碎片，從其 ID 緊跟您所提供 ShardId 的碎片開始。

如果指定 AT_TIMESTAMP 或 FROM_TIMESTAMP_ID 類型，則還必須提供選用 Timestamp 屬性的值。如果指定 AT_TIMESTAMP 類型，則會傳回在提供的時間戳記開啟的所有碎片。如果指定 FROM_TIMESTAMP 類型，則會傳回在提供的時間戳記開啟的所有碎片。

Important

DescribeStreamSummary 和 ListShard API 提供了一種更可擴展的方式來檢索有關資料串流的資訊。更具體地說，DescribeStream API 的配額可能會導致限流。如需詳細資訊，請參閱配額和限制。另請注意，DescribeStream配額會跨與您 AWS 帳戶中所有資料串流互動的所有應用程式共用。另一方面，ListShards API 的配額是特定於單個資料串流。因此，您不僅可以使用 ListShards API 獲得更高的 TPS，而且隨著建立更多資料串流，操作可以更好地擴展。

建議您移轉呼叫 DescribeStream API 的所有生產者和取用者，改為調用 DescribeStreamSummary 和 ListShard APIs API。若要識別這些生產者和取用者，建議使用 Athena 剖析 CloudTrail 記錄，因為在 API 呼叫中擷取了 KPL 和 KCL 的使用者客服人員。

```
SELECT useridentity.sessioncontext.sessionissuer.username,
useridentity.arn,eventname,useragent, count(*) FROM
cloudtrail_logs WHERE Eventname IN ('DescribeStream')  AND
eventtime
    BETWEEN ''
        AND ''
GROUP BY
 useridentity.sessioncontext.sessionissuer.username,useridentity.arn,eventname,useragent
ORDER BY  count(*) DESC LIMIT 100
```

我們也建議重新設定 AWS Lambda 和 Amazon Firehose 與叫用 DescribeStream API 的 Kinesis Data Streams 整合，以便改為叫用 DescribeStreamSummary和 。 ListShards具

體而言，對於 AWS Lambda，您必須更新事件來源映射。對於 Amazon Firehose，必須更新對應的 IAM 許可，以便其中包含 `ListShards` IAM 許可。

刪除串流

您可以使用 Kinesis Data Streams 主控台或是以程式設計方式刪除串流。若要以程式設計方式刪除串流，請使用 `DeleteStreamRequest`，如以下程式碼所示。

```
DeleteStreamRequest deleteStreamRequest = new DeleteStreamRequest();
deleteStreamRequest.setStreamName(myStreamName);
client.deleteStream(deleteStreamRequest);
```

請在刪除串流之前，關閉串流上操作中的任何應用程式。如果應用程式嘗試在已刪除的串流上操作時，它會收到 `ResourceNotFound` 例外狀況。此外，如果您後續建立與先前串流具有相同名稱的新串流，並且先前串流上操作的應用程式仍在執行，這些應用程式可能會嘗試與新串流互動，就好像它是舊的串流一般，但結果無法預測。

重新分片串流

> **Important**
>
> 您可以使用 UpdateShardCount API 來將串流重新分片。否則，您可以如此處的說明，繼續執行分割和合併。

Amazon Kinesis Data Streams 支援重新分片，它可讓您調整串流中的碎片數量，以適應通過串流的資料流程速率的變化。重新分片為進階操作。如果您是 Kinesis Data Streams 的新使用者，請在您熟悉 Kinesis Data Streams 的所有其他層面之後，回到此主題。

重新分片操作有兩個類型：碎片分割和碎片合併。在碎片分割中，您會將單一碎片分成兩個碎片。在碎片合併中，您會將兩個碎片結合成單一碎片。重新分片一律為逐對，因為您無法在單一操作中分割成兩個以上的碎片，而且您無法在單一操作合併超過兩個碎片。重新分片操作執行所在的碎片或碎片對，稱為父碎片。重新分片操作所產生的碎片或碎片對，稱為子碎片。

分割會增加串流中碎片的數量，因此增加串流的資料容量。由於您需按碎片付費，分割會增加串流的成本。相同地，合併會減少串流中碎片的數量，因此會減少串流的資料容量和成本。

重新分片通常是由與生產程式 (put) 應用程式和使用程式 (get) 應用程式不同的管理應用程式執行。這類管理應用程式會根據 Amazon CloudWatch 提供的指標或根據從生產程式和使用程式收集的指標，監控串流的整體效能。管理應用程式也需要較使用程式或生產程式更廣泛的一組 IAM 許可，因為使用程式和生產程式通常不應需要存取用於重新分片的 API。如需 Kinesis Data Streams 的 IAM 許可的詳細資訊，請參閱 使用 IAM 控制對 Amazon Kinesis Data Streams 資源的存取。

如需有關重新分片的詳細資訊，請參閱如何變更 Kinesis Data Streams 中開啟的碎片數量？

主題

決定重新分片的策略

Amazon Kinesis Data Streams 中重新分片的目的是讓您的串流適應資料流程速率的變化。您分割碎片，以增加串流的容量 (和成本)。您合併碎片，以減少串流的成本 (和容量)。

重新分片的其中一個方法可能是分割串流中的每個碎片，這樣會讓串流的容量加倍。不過，這可能會提供超過您實際所需額外的容量，因此產生不必要的成本。

您也可以使用指標來判斷哪些是您的熱或冷碎片，也就是說，較預期接收非常多資料或非常少資料的碎片。然後您可以選擇性地分割熱碎片，以增加以這些碎片為目標之雜湊索引鍵的容量。同樣地，您可以合併冷碎片以更明智利用其未使用的容量。

您可以從 Kinesis Data Streams 發佈的 Amazon CloudWatch 指標為串流取得一些效能資料。不過，您也可以為串流收集一些自己的指標。其中一個方法會是記錄分割區索引鍵為資料記錄產生的雜湊索引鍵值。回想您在將記錄新增到串流時指定的分割區索引鍵。

```
putRecordRequest.setPartitionKey( String.format( "myPartitionKey" ) );
```

Kinesis Data Streams 使用 MD5，從分割區索引鍵運算雜湊索引鍵。由於您會指定記錄的分割區索引鍵，您可以使用 MD5 來運算該記錄的雜湊索引鍵鍵值，並記錄它。

您也可以記錄資料記錄獲指派的碎片 ID。您可以使用 `getShardId` 方法傳回的 `putRecordResults` 物件的 `putRecords` 方法，以及 `putRecordResult` 方法傳回的 `putRecord` 物件，來取得碎片 ID。

```
String shardId = putRecordResult.getShardId();
```

有了碎片 ID 和雜湊索引鍵值，您可以判斷哪些碎片和雜湊索引鍵正接收最多或最少的流量。然後，您可以使用重新分片來提供更多或更少的容量，視這些索引鍵的需要而定。

分割碎片

若要在 Amazon Kinesis Data Streams 中分割碎片，您必須指定如何將來自父碎片的雜湊索引鍵值重新分配到子碎片。將資料記錄新增到串流時，它會根據雜湊索引鍵值指派給碎片。雜湊索引鍵值是在您將資料記錄新增至串流時，為資料記錄指定之分割區索引鍵的 MD5 雜湊。具有相同分割區索引鍵的資料記錄也會有相同的雜湊索引鍵值。

指定碎片的可能雜湊索引鍵值會組成一組有次序的連續非負整數。此可能的雜湊索引鍵值範圍假設為：

```
shard.getHashKeyRange().getStartingHashKey();
shard.getHashKeyRange().getEndingHashKey();
```

分割碎片時，您會指定在這個範圍中的值。該雜湊索引鍵值和所有較高的雜湊索引鍵值會分配至其中一個子碎片。所有較低的雜湊索引鍵值會分配至其他子碎片。

以下程式碼示範碎片分割操作，該操作會在每個子碎片之間將雜湊索引鍵平均分配，基本上是將父碎片分割成兩半。這只是分割父碎片的一個可能方式。例如，您可以分割碎片，使得來自父系較少的三分之一的索引鍵前往第一個子碎片，以及來自父系較高的三分之二索引鍵則前往其他子碎片。不過，對於許多應用程式來說，將碎片分割成兩半是有效的方法。

此程式碼假設 myStreamName 擁有串流的名稱，以及物件變數 shard 則擁有要分割的碎片。從將新的 splitShardRequest 物件執行個體化開始，並設定串流名稱和碎片 ID。

```
SplitShardRequest splitShardRequest = new SplitShardRequest();
splitShardRequest.setStreamName(myStreamName);
splitShardRequest.setShardToSplit(shard.getShardId());
```

決定碎片中介於最低和最高值之間一半的雜湊索引鍵值。這是將包含來自父碎片上半雜湊索引鍵之子碎片的開始雜湊索引鍵值。在 setNewStartingHashKey 方法中指定這個值。您只需要指定這個值。Kinesis Data Streams 會將低於這個值的雜湊索引鍵自動分配至分割建立的另一個子碎片。最後一個步驟是呼叫 Kinesis Data Streams 用戶端上的 splitShard 方法。

```
BigInteger startingHashKey = new
 BigInteger(shard.getHashKeyRange().getStartingHashKey());
```

```
BigInteger endingHashKey   = new
 BigInteger(shard.getHashKeyRange().getEndingHashKey());
String newStartingHashKey  = startingHashKey.add(endingHashKey).divide(new
 BigInteger("2")).toString();

splitShardRequest.setNewStartingHashKey(newStartingHashKey);
client.splitShard(splitShardRequest);
```

等待串流再次變成作用中中顯示此程序後的第一個步驟。

合併兩個碎片

碎片合併操作需要兩個指定的碎片，並且會將它們合併為單一碎片。合併之後，該單一子碎片會收到兩個父碎片涵蓋的所有雜湊索引鍵值的資料。

碎片相鄰

若要合併兩個碎片，碎片必須相鄰。如果兩個碎片的雜湊索引鍵範圍的聯結形成連續的一組 (無間隙)，則將其視為相鄰。舉例來說，假設您有兩個碎片，其中一個的雜湊鍵範圍為 276...381，另一個的雜湊鍵範圍為 382...454。您可以將這兩個碎片合併為一個碎片，就會得到範圍為 276...454 的雜湊鍵。

再舉另一個例子，假設您有兩個碎片，其中一個的雜湊鍵範圍為 276...381，另一個的雜湊鍵範圍為 455...560。您可以合併這兩個碎片，因為這樣在這兩個碎片之間就會有一個或多個碎片，範圍涵蓋 382...454。

串流中所有 OPEN 碎片的集合 – 群組形式 – 一律會跨越 MD5 雜湊索引鍵值的整個範圍。如需碎片狀態—例如 CLOSED—的詳細資訊，請參閱 考慮重新碎片之後的資料路由、資料持久性和碎片狀態。

若要識別用於合併之候選項目的碎片，您應該篩選掉處於 CLOSED 狀態的所有碎片。狀態為 OPEN (即非 CLOSED) 的碎片，其結束序號為 null。您可以使用下列方式來測試碎片的結束序號：

```
if( null == shard.getSequenceNumberRange().getEndingSequenceNumber() )
{
  // Shard is OPEN, so it is a possible candidate to be merged.
}
```

篩選掉排序關閉碎片後，將剩餘的碎片依每個碎片支援的最高雜湊索引鍵值排序。您可以使用下列方式來擷取此值：

```
shard.getHashKeyRange().getEndingHashKey();
```

如果在這個經篩選、排序之清單中的兩個碎片為相鄰，則可以將它們合併。

合併操作的程式碼

以下程式碼會合併兩個碎片。此程式碼假設 myStreamName 擁有串流的名稱，以及物件變數 shard1 和 shard2 則擁有要合併的兩個相鄰碎片。

對於合併操作，從將新的 mergeShardsRequest 物件執行個體化開始。使用 setStreamName 方法指定串流名稱。然後，使用 setShardToMerge 和 setAdjacentShardToMerge 方法指定要合併的兩個碎片。最後，呼叫 Kinesis Data Streams 用戶端上的 mergeShards 方法，以執行操作。

```
MergeShardsRequest mergeShardsRequest = new MergeShardsRequest();
mergeShardsRequest.setStreamName(myStreamName);
mergeShardsRequest.setShardToMerge(shard1.getShardId());
mergeShardsRequest.setAdjacentShardToMerge(shard2.getShardId());
client.mergeShards(mergeShardsRequest);
```

等待串流再次變成作用中中顯示此程序後的第一個步驟。

完成重新分片動作

在 Amazon Kinesis Data Streams 中任何種類的重新分片程序之後，以及在繼續正常記錄處理之前，需要其他程序和考量。下列小節描述這些情況。

主題

- 等待串流再次變成作用中
- 考慮重新碎片之後的資料路由、資料持久性和碎片狀態

等待串流再次變成作用中

呼叫重新分片操作 splitShard或 之後mergeShards，您必須等待串流再次變成作用中。要使用的程式碼與建立串流之後，等待串流變得作用中的程式碼相同。該程式碼如下所示：

```
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime )
{
  try {
```

```
    Thread.sleep(20 * 1000);
  }
  catch ( Exception e ) {}

  try {
    DescribeStreamResult describeStreamResponse =
 client.describeStream( describeStreamRequest );
    String streamStatus =
 describeStreamResponse.getStreamDescription().getStreamStatus();
    if ( streamStatus.equals( "ACTIVE" ) ) {
      break;
    }
   //
    // sleep for one second
    //
    try {
      Thread.sleep( 1000 );
    }
    catch ( Exception e ) {}
  }
  catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime )
{
  throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

考慮重新碎片之後的資料路由、資料持久性和碎片狀態

Kinesis Data Streams 是一種即時資料串流服務。您的應用程式應該假設資料持續流經串流中的碎片。執行重新分片時，流動至父碎片的資料記錄會重新路由，以根據資料記錄分割區索引鍵所對應的雜湊索引鍵值，流動至子碎片。不過，重新分片之前在父碎片中的任何資料記錄，會維持在那些碎片中。重新碎片發生時，父碎片不會消失。它們會隨著重新分片之前包含的資料保留。父碎片中的資料記錄可使用 Kinesis Data Streams API 中的 `getShardIterator` 和 `getRecords` 操作或透過 Kinesis Client Library 存取。

> **Note**
>
> 資料記錄可供存取的時間是從新增至串流的時間到目前的保留期間。無論在該時間期間對串流中的碎片進行任何變更，都是如此。如需串流保留期間的詳細資訊，請參閱變更資料保留期間。

在重新分片的程序中，父碎片會從 OPEN 狀態轉換為 CLOSED 狀態然後為 EXPIRED 狀態。

- OPEN：重新分片操作之前，父碎片會處於 OPEN 狀態，這表示資料記錄可同時新增至碎片和從碎片中擷取。
- CLOSED：重新分片操作之後，父碎片會轉換為 CLOSED 狀態。這表示資料記錄不再會新增至碎片。應新增至此碎片的資料記錄，現在會改為新增至子碎片。不過，您仍可以在限定時間內從碎片擷取資料記錄。
- EXPIRED：在串流的保留期過期之後，父碎片中的所有資料記錄會過期，並不再可供存取。此時，碎片本身會轉換為 EXPIRED 狀態。為了列舉串流中碎片對 `getStreamDescription().getShards` 的呼叫，在傳回的清單碎片中不會包含 EXPIRED 碎片。如需串流保留期間的詳細資訊，請參閱變更資料保留期間。

重新分片發生且串流再次處於 ACTIVE 狀態之後，您可以立即開始從子碎片讀取資料。不過，在重新碎片之後保留的父碎片可能仍然包含您尚未讀取，且在重新碎片之前新增至串流的資料。如果您在從父碎片讀取所有資料之前從子碎片讀取資料，則可以不按資料記錄的序號指定的順序讀取特定雜湊索引鍵的資料。因此，假設資料的順序很重要，在重新分片之後，您應該一律持續從父碎片讀取資料，直至耗盡為止。只有這樣您才應該從子碎片開始讀取資料。當 `getRecordsResult.getNextShardIterator` 傳回 `null`，表示您已讀取父碎片中的所有資料。

變更資料保留期間

Amazon Kinesis Data Streams 支援對資料串流變更資料記錄保留期間。Kinesis 資料串流是資料記錄的排序序列，目的是要即時對其寫入和讀取。因此，資料記錄會暫時存放在串流中的碎片。從新增記錄的時間期間，到記錄不再可供存取的時間稱為保留期間。根據預設，Kinesis 資料串流會存放 24 小時的記錄，最長可達 8760 小時 (365 天)。

您可以透過 Kinesis Data Streams 主控台或使用 ncreaseStreamRetentionPeriod 和 DecreaseStreamRetentionPeriod 操作來更新保留期間。使用 Kinesis Data Streams 主控台，您可以同時對多個資料串流的保留期進行大量編輯。您可以使用 IncreaseStreamRetentionPeriod 操作或 Kinesis Data Streams 主控台，將保留期間最多增加 8760 小時 (365 天)。您可以使用 DecreaseStreamRetentionPeriod 操作或 Kinesis Data Streams 主控台，將保留期間減少至最低 24 小時。這兩個操作的請求語法會包含串流名稱和保留期間 (小時)。最後，您可以透過呼叫 DescribeStream 操作來檢查串流目前的保留期間。

以下是使用 AWS CLI來變更保留期間的範例：

```
aws kinesis increase-stream-retention-period --stream-name retentionPeriodDemo --
retention-period-hours 72
```

Kinesis Data Streams 會在增加保留期間的數分鐘內，讓舊保留期間的記錄無法提供存取。例如，將保留期間從 24 小時變更為 48 小時，表示在 23 小時 55 分鐘之前新增到串流的記錄仍會在 24 小時後提供。

Kinesis Data Streams 幾乎會立即讓早於新保留期間的記錄於保留期間減少時無法供使用。因此，呼叫 DecreaseStreamRetentionPeriod 操作時請特別注意。

設定資料保留期間，以確保使用程式能夠在資料過期之前加以讀取 (如果發生問題)。您應該仔細考慮所有可能性，例如，您的記錄處理邏輯或下游相依性關閉很長一段時間的問題。您可以將保留期間想成安全網，讓資料使用程式有更多的時間可進行恢復。保留期間 API 操作可讓您主動設定或回應式地回應操作事件。

串流的保留期間設定如超過 24 小時，將產生額外的費用。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 定價。

標記您的 Amazon Kinesis Data Streams 資源

您可以將自己的中繼資料指派給您在 Amazon Kinesis Data Streams 中以標籤形式建立的串流和增強廣發消費者。標籤是您為串流所定義的索引鍵值組。使用標籤是一種簡單但強大的方法來管理 AWS 資源和組織資料，包括帳單資料。

目錄

檢閱標籤基本概念

您可以標記的 Kinesis Data Streams 資源包括資料串流和增強廣發消費者。您可以使用 Kinesis Data Streams 主控台 AWS CLI或 Kinesis Data Streams API 來完成下列任務：

- 使用標籤建立資源
- 將標籤新增至資源
- 列出資源的標籤
- 從資源移除標籤

> **Note**
>
> 您無法使用 Kinesis Data Streams 主控台將標籤套用至增強型廣發消費者。若要將標籤套用至消費者，請使用 AWS CLI 或 Kinesis Data Streams API。

您可以使用標籤來分類您的 資源。例如，您可以依用途、擁有者或環境來分類資源。由於您定義了每個標籤的鍵和值，您可以建立一組自訂的類別，以符合您的特定需求。例如，您可以定義一組標籤，協助您依擁有者和相關聯的應用程式追蹤資源。以下是數個標籤的範例：

- 專案：專案名稱
- 擁有者：名稱
- 用途：負載測試
- 應用程式：應用程式名稱
- 環境：生產

> **Important**
>
> - 若要在建立串流時新增標籤，您必須包含該串流的 `kinesis:CreateStream`和 `kinesis:AddTagsToStream`許可。建立串流時，您無法使用 `kinesis:TagResource`許可來標記串流。
> - 若要在消費者註冊期間新增標籤，您必須包含 `kinesis:TagResource`和 `kinesis:RegisterStreamConsumer`許可。

使用標記追蹤成本

您可以使用標籤來分類和追蹤您的 AWS 成本。當您將標籤套用至 Kinesis Data Streams 資源時，AWS 成本分配報告會包含依標籤彙總的用量和成本。您可以套用代表商業類別的標籤，例如成本中心、應用程式名稱或擁有者，以跨多個服務組織成本。如需詳細資訊，請參閱《AWS Billing 使用者指南》中的將成本分配標籤用於自訂帳單報告。

了解標籤限制

以下限制適用於標籤：

基本限制

- 每個資源的標籤數上限為 50。
- 標籤鍵與值皆區分大小寫。
- 您無法變更或編輯已刪除資源的標籤。

標籤鍵限制

- 每個標籤鍵都必須是唯一的。如果您新增具有已使用索引鍵的標籤，則新的標籤會覆寫現有鍵值對。
- 您無法以 啟動標籤金鑰，aws:因為此字首保留給 使用 AWS。 會代表您 AWS 建立以此字首開頭的標籤，但您無法編輯或刪除它們。
- 標籤鍵的長度必須介於 1 到 128 個 Unicode 字元之間。
- 標籤鍵必須包含下列字元：Unicode 字母、數字、空格以及下列特殊字元：`_ . / = + - @`。

標籤值限制

- 標籤值的長度必須介於 0 到 255 個 Unicode 字元之間。
- 標籤值可以空白。否則，它們必須包含下列字元：Unicode 字母、數字、空格以及下列任何特殊字元：`_ . / = + - @`。

使用 Kinesis Data Streams 主控台標記串流

您可以使用 Kinesis Data Streams 主控台新增、更新、列出和移除串流上的標籤。

檢視串流的標籤

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/kinesis 的 Kinesis 主控台。
2. 在左側導覽窗格中，選擇資料串流。
3. 在資料串流頁面上，選擇您要標記的串流。
4. 在串流詳細資訊頁面上，選擇組態。
5. 在標籤區段中，檢視套用至串流的標籤。

使用標籤建立資料串流

1. 開啟 Kinesis Data Streams 主控台。
2. 在左側導覽窗格中，選擇資料串流。
3. 選擇 建立資料串流。
4. 在建立資料串流頁面上，輸入資料串流的名稱。
5. 針對資料串流容量，選擇隨需或佈建容量模式。

   如需容量模式的詳細資訊，請參閱 選擇要在 中串流的正確模式。
6. 在標籤區段中，執行下列動作：

   a. 選擇 Add new tag (新增標籤)。
   b. 針對金鑰，輸入標籤，並選擇性地在值欄位中指定值。

      如果您看到錯誤，您指定的標籤索引鍵或值不符合標籤限制。如需詳細資訊，請參閱了解標籤限制。
7. 選擇 建立資料串流。

在串流上新增或更新標籤

1. 開啟 Kinesis Data Streams 主控台。
2. 在左側導覽窗格中，選擇資料串流。
3. 在資料串流頁面上，選擇要新增或更新標籤的串流。
4. 在串流詳細資訊頁面上，選擇組態。
5. 在標籤區段中，選擇管理標籤。
6. 在標籤下，執行下列其中一項操作：

- 若要新增標籤，請選擇新增標籤，然後輸入標籤的金鑰和值資料。視需要重複此步驟。

  您可以為每個串流新增的標籤數量上限為 50。
- 若要更新現有的標籤，請在該標籤金鑰的值欄位中輸入新的標籤值。

如果您看到錯誤，您指定的標籤索引鍵或值不符合標籤限制。如需詳細資訊，請參閱了解標籤限制。

7. 選擇 Save changes (儲存變更)。

從串流移除標籤

1. 開啟 Kinesis Data Streams 主控台。
2. 在左側導覽窗格中，選擇資料串流。
3. 在資料串流頁面上，選擇要從中移除標籤的串流。
4. 在串流詳細資訊頁面上，選擇組態。
5. 在標籤區段中，選擇管理標籤。
6. 尋找您要移除的標籤索引鍵和值對。然後選擇移除。
7. 選擇儲存變更。

使用 標記串流 AWS CLI

您可以使用 新增、列出和移除串流上的標籤 AWS CLI。如需範例，請參閱下列文件。

create-stream

建立具有標籤的串流。

add-tags-to-stream

為指定的串流新增或更新標籤。

list-tags-for-stream

列出所指定串流的標籤。

remove-tags-from-stream

從指定的串流移除標籤。

使用 Kinesis Data Streams APIs 標記串流

您可以使用 Kinesis Data Streams APIs 在串流上新增、列出和移除標籤。如需範例，請參閱下列文件：

CreateStream

建立具有標籤的串流。

AddTagsToStream

為指定的串流新增或更新標籤。

ListTagsForStream

列出所指定串流的標籤。

RemoveTagsFromStream

從指定的串流移除標籤。

使用 標記消費者 AWS CLI

您可以使用 新增、列出和移除消費者上的標籤 AWS CLI。如需範例，請參閱下列文件：

register-stream-consumer

使用標籤註冊 Kinesis 資料串流的取用者。

tag-resource

新增或更新指定 Kinesis 資源的標籤。

list-tags-for-resource

列出指定 Kinesis 資源的標籤。

untag-resource

從指定的 Kinesis 資源移除標籤。

使用 Kinesis Data Streams APIs 標記消費者

您可以使用 Kinesis Data Streams APIs 在消費者上新增、列出和移除標籤。如需範例，請參閱下列文件：

RegisterStreamConsumer

使用標籤註冊 Kinesis 資料串流的取用者。

TagResource

新增或更新指定 Kinesis 資源的標籤。

ListTagsForResource

列出指定 Kinesis 資源的標籤。

UntagResource

從指定的 Kinesis 資源移除標籤。

處理大型記錄

Amazon Kinesis Data Streams 支援高達 10 MB (MiBs的記錄。建議將此功能用於處理超過預設 1 MiB 記錄大小限制的間歇性資料承載。現有和新建立串流的預設記錄大小上限設定為 1 MiB。

此功能有利於物聯網 (IoT) 應用程式、變更資料擷取 (CDC) 管道，以及需要偶爾處理較大資料承載的機器學習工作流程。若要開始在串流中使用大型記錄，請更新串流的記錄大小上限。

> **Important**
>
> 對於寫入，個別碎片輸送量限制為 1 MB/s，對於讀取，2 MB/s 保持不變，但支援更大的記錄大小。Kinesis Data Streams 旨在適應間歇性大型記錄，以及小於或等於 1 MiB 的記錄基準流量。它的設計無法容納持續大量大型記錄擷取。

更新您的串流以使用大型記錄

使用 Kinesis Data Streams 處理較大的記錄

1. 導覽至 Kinesis Data Streams 主控台。
2. 選取您的串流，然後前往組態索引標籤。
3. 按一下編輯，其位於記錄大小上限旁。
4. 設定您的記錄大小上限 （ 最多 10 MiB)。

5. 儲存您的變更。

此設定只會調整此 Kinesis 資料串流的記錄大小上限。在提高此限制之前，請確認所有下游應用程式都可以處理較大的記錄。

您也可以使用 CLI AWS 更新此設定：

```
aws kinesis update-max-record-size \ --stream-arn  \
        --max-record-size-in-ki-b 5000
```

使用大型記錄最佳化串流效能

建議將大型記錄維持在整體流量的 2% 以下。在串流中，每個碎片的輸送量容量為每秒 1 MiB。為了容納大型記錄，Kinesis Data 串流爆量高達 10 MiBs同時平均每秒 1 MiB。支援大型記錄的此容量會持續重新填入串流。重新填入速率取決於大型記錄的大小和基準記錄的大小。為了獲得最佳結果，請使用統一分佈的分割區索引鍵。如需 Kinesis 隨需擴展方式的詳細資訊，請參閱隨需模式功能和使用案例。

使用大型記錄緩解限流

緩解限流

1. 在生產者應用程式中實作具有指數退避的重試邏輯。
2. 使用隨機分割區索引鍵，將大型記錄分散到可用的碎片。
3. 將承載存放在 Amazon S3 中，並僅傳送中繼資料參考至串流，以取得大型記錄的連續串流。如需詳細資訊，請參閱使用 Amazon Kinesis Data Streams 處理大型記錄。

使用 Kinesis Data Streams APIs 處理大型記錄

大型記錄支援引進一個新的 API，並更新兩個現有的控制平面 APIs，以處理高達 10 MiBs 的記錄。

用於修改記錄大小的 API：

- `UpdateMaxRecordSize`：設定現有串流的記錄大小上限，上限為 10 MiBs。

現有 APIs更新：

- `CreateStream`：新增選擇性`MaxRecordSizeInKiB`參數，以在建立串流期間設定記錄大小限制。
- `DescribeStreamSummary`：傳回 `MaxRecordSizeInKiB` 欄位以顯示目前的串流組態。

列出的所有 APIs 都會維持現有串流的回溯相容性。如需完整的 API 文件，請參閱 Amazon Kinesis Data Streams Service API 參考。

AWS 元件與大型記錄相容

下列 AWS 元件與大型記錄相容：

元件	Description
AWS 開發套件	AWS SDK 支援處理大型記錄。您可以使用 AWS SDKs 中的可用方法來更新串流的最大記錄大小，最高可達 10 MiB。如需詳細資訊，請參閱搭配 AWS SDK 使用此服務。
Kinesis 消費者程式庫 (KCL)	從 2.x 版開始，KCL 支援處理大型記錄。若要使用大型記錄支援，請更新串流`maxRecordSize` 的 ，並使用 KCL。如需詳細資訊，請參閱使用 Kinesis Client Library。
Kinesis Producer Library (KPL)	從 1.0.5 版開始，KPL 支援處理大型記錄。若要使用大型記錄支援，請更新串流`maxRecordSize` 的 ，並使用 KPL。如需詳細資訊，請參閱使用 Amazon Kinesis Producer Library (KPL) 開發生產者。
Amazon EMR	Amazon EMR with Apache Spark 支援處理高達 Kinesis Data Streams 限制 (10 MiBs 的大型記錄。若要使用大型記錄支援，請使用 `readStream` 函數。如需詳細資訊，請參閱 Amazon EMR 和 Amazon Kinesis 整合。
Amazon Data Firehose	與 Kinesis Data Streams 搭配使用時，具有大型記錄的 Amazon Data Firehose 行為取決於交付目的地： • Amazon S3：支援交付大型記錄，無需任何其他組態。當您使用資料格式轉換時，Firehose 支援交付大型記錄。當您使用動態分割時，Firehose 不支援交付大型記錄。

元件	Description
	• Lambda：不建議在觸發下游 Lambda 函數時搭配 Firehose 使用大型記錄。這可能會導致間歇性故障。 • HTTP：Firehose 不支援交付大型記錄。 • Snowflake：Firehose 不支援交付大型記錄。 • Amazon Redshift：Firehose 不支援交付大型記錄。 對於需要以大型記錄交付至 Snowflake 或 Redshift 的應用程式，請先將資料交付至 Amazon S3。之後，使用擷取、轉換、載入 (ETL) 程序載入資料。對於所有其他目的地，請在proof-of-concept環境中使用大型記錄測試行為，然後再擴展到生產用量。處理大型記錄會因目的地而異。
AWS Lambda	AWS Lambda 支援最多 6 MiBs承載。此限制包括轉換為 base-64 編碼的 Kinesis 承載，以及與事件來源映射 (ESM) 相關聯的中繼資料。對於小於 6 MiBs的記錄，Lambda 會使用 ESM 來處理它們，而不需要額外的組態。對於大於 6 MiBs的記錄，Lambda 會使用失敗時的目的地來處理它們。您必須使用 ESM 設定失敗時目的地，以處理超過 Lambda 處理限制的記錄。傳送至失敗時目的地的每個事件都是 JSON 文件，其中包含有關失敗呼叫的中繼資料。 無論記錄大小為何，建議在 ESM 中建立失敗時目的地。這可確保不會捨棄任何記錄。如需詳細資訊，請參閱設定失敗呼叫的目的地。

元件	Description
Amazon Redshift	從 Kinesis Data Streams 串流資料時，Amazon Redshift 僅支援小於 1 MiB 的記錄大小。超過此限制的記錄不會處理。未處理的記錄會記錄為 `sys_stream_scan_errors` 。如需詳細資訊，請參閱 SYS_STREAM_SCAN_ERRORS。
Kinesis Data Streams 的 Flink 連接器	使用來自 Kinesis Data Streams 的資料有兩種方法：Kinesis 來源連接器和 Kinesis 接收器連接器。來源連接器支援處理小於 1 MiB 的記錄，最多 10 MiBs 請勿將接收器連接器用於大於 1 MiB 的記錄。如需詳細資訊，請參閱使用連接器透過 DataStream API 在 Amazon Managed Service for Apache Flink 中移動資料。

支援大型記錄的區域

此 Amazon Kinesis Data Streams 功能僅適用於下列 AWS 區域：

AWS 區域	區域名稱
eu-north-1	歐洲 (斯德哥爾摩)
me-south-1	Middle East (Bahrain)
ap-south-1	亞太地區 (孟買)
eu-west-3	Europe (Paris)
ap-southeast-3	亞太地區 (雅加達)
us-east-2	美國東部 (俄亥俄)
af-south-1	非洲 (開普敦)
eu-west-1	歐洲 (愛爾蘭)

AWS 區域	區域名稱
me-central-1	中東 (阿拉伯聯合大公國)
eu-central-1	歐洲 (法蘭克福)
sa-east-1	南美洲 (聖保羅)
ap-east-1	亞太地區 (香港)
ap-south-2	亞太地區 (海德拉巴)
us-east-1	美國東部 (維吉尼亞北部)
ap-northeast-2	亞太地區 (首爾)
ap-northeast-3	亞太地區 (大阪)
eu-west-2	歐洲 (倫敦)
ap-southeast-4	亞太地區 (墨爾本)
ap-northeast-1	亞太地區 (東京)
us-west-2	美國西部 (奧勒岡)
us-west-1	美國西部 (加利佛尼亞北部)
ap-southeast-1	亞太地區 (新加坡)
ap-southeast-2	亞太地區 (悉尼)
il-central-1	以色列 (特拉維夫)
ca-central-1	加拿大 (中部)
ca-west-1	加拿大西部 (卡加利)
eu-south-2	歐洲 (西班牙)
cn-northwest-1	中國 (寧夏)

AWS 區域	區域名稱
eu-central-2	歐洲 (蘇黎世)
us-gov-east-1	AWS GovCloud （美國東部）
us-gov-west-1	AWS GovCloud （美國西部）

使用 執行彈性測試 AWS Fault Injection Service

AWS Fault Injection Service 是一項全受管服務，可協助您在 AWS 工作負載上執行故障注入實驗。與 Amazon Kinesis Data Streams AWS FIS 整合可讓您針對受控環境中常見的 Amazon Kinesis Data Streams API 錯誤測試應用程式彈性。此功能可讓您在遇到失敗之前驗證錯誤處理、重試邏輯和監控系統。如需詳細資訊，請參閱什麼是 AWS Fault Injection Service？。

動作

- API 內部錯誤：這會將內部錯誤注入目標 IAM 角色提出的請求中。特定回應取決於每個服務和 API。動作`aws:fis:inject-api-internal-error`會建立`InternalFailure`錯誤 (HTTP 500)。
- API 調節錯誤：這會將內部錯誤注入目標 IAM 角色提出的請求中。特定回應取決於每個服務和 API。動作`aws:fis:inject-api-throttle-error`會建立`ThrottlingException`錯誤 (HTTP 400)。
- API 無法使用錯誤：這會將內部錯誤注入目標 IAM 角色提出的請求中。特定回應取決於每個服務和 API。動作`aws:fis:inject-api-unavailable-error`會建立`ServiceUnavailable`錯誤 (HTTP 503)。
- API 佈建輸送量例外狀況：這會將內部錯誤注入目標 IAM 角色提出的請求中。特定回應取決於每個服務和 API。動作`aws:kinesis:inject-api-provisioned-throughput-exception`會建立`ProvisionedThroughputExceededException`錯誤 (HTTP 400)。
- API 過期迭代器例外狀況：這會將內部錯誤注入目標 IAM 角色提出的請求中。特定回應取決於每個服務和 API。動作`aws:kinesis:inject-api-expired-iterator-exception`會建立`ExpiredIteratorException`錯誤 (HTTP 400)。

如需詳細資訊，請參閱 Amazon Kinesis Data Streams 動作。

考量事項

- 您可以對 Amazon Kinesis Data Streams 的佈建和隨需方案使用上述動作。
- 一旦實驗完成，您的串流會根據選取的持續時間繼續。您也可以在實驗完成之前停止執行中的實驗。或者，您可以定義停止條件，根據在 Amazon CloudWatch Application Insights 中定義應用程式運作狀態的警示來停止實驗。
- 您最多可以測試 280 個串流。

如需區域支援的詳細資訊，請參閱AWS Fault Injection Service 端點和配額。

佈建輸送量例外狀況錯誤

當 Kinesis 串流的請求速率超過一或多個碎片的輸送量限制時，會發生佈建輸送量超過例外狀況錯誤 (HTTP 400)。每個碎片都有特定的讀取和寫入容量限制，超過這些限制會觸發此例外狀況。導致此例外狀況的情況包括：資料擷取或耗用突然峰值、處理的資料磁碟區碎片容量不足，或分割區索引鍵分佈不均。

處理例外狀況的建議

- 實作指數退避和重試機制。
- 增加碎片數量以適應更高的輸送量。
- 確保分割區索引鍵有適當的分佈。
- 監控串流指標。

此外，使用 Kinesis 隨需容量模式有助於自動調整工作負載，並將此例外狀況的發生降至最低。如需詳細資訊，請參閱什麼是 AWS Fault Injection Service？

> **Note**
>
> 不當的分佈問題不在自動擴展的隨需模式功能範圍內。

執行基本實驗

1. 使用基準指標：在測試之前記錄正常輸送量模式。
2. 建立實驗：使用 `aws:kinesis:inject-api-provisioned-throughput-exception`動作。

3. 設定強度：從 25% 請求限流開始。
4. 監控回應：使用指數退避驗證重試邏輯。
5. 驗證擴展：確認自動擴展會觸發啟用。
6. 檢查警示：確保CloudWatch警示如預期般執行。

應用程式應實作適當的退避策略、監控 WriteProvisionedThroughputExceeded和 ReadProvisionedThroughputExceeded指標，並適時觸發碎片擴展。

動作詳細資訊

- 資源類型：IAM 角色 ARN
- 目標操作：PutRecord、PutRecords、 GetRecords
- 錯誤代碼：ProvisionedThroughputExceededException(HTTP 400)
- 描述：模擬請求率超過碎片容量限制、測試應用程式限流和擴展回應的情況。

Parameters

- IAM 角色 ARN：您的應用程式用於 Kinesis Data Streams 操作的角色。
- 操作：目標操作：PutRecord、PutRecords、GetRecords。
- 資源清單：特定的串流名稱或碎片識別符。
- 持續時間：實驗持續時間，也就是從一分鐘到 12 小時的持續時間。在 AWS FIS API 中，值是 ISO 8601 格式的字串。例如，PT1M 代表一分鐘。在 AWS FIS 主控台中，您可以輸入秒數、分鐘數或小時數。
- 強度：要調節的請求百分比。

所需的許可

- kinesis:InjectApiError

範例實驗範本

下列範例顯示最多 5 個 Kinesis Data 串流具有指定標籤的所有請求的佈建輸送量例外狀況。 會隨機 AWS FIS 選取要影響的串流。5 分鐘後會移除故障。

```
{
```

```
    "description": "Kinesis stream experiment",
    "targets": {
        "KinesisStreams-Target-1": {
            "resourceType": "aws:kinesis:stream",
            "resourceTags": {
                  "tag-key": "tag-value"
            },
            "selectionMode": "COUNT(5)"
        }
    },
    "actions": {
         "kinesis": {
              "actionId": "aws:kinesis:stream-provisioned-throughput-exception",
              "description": "my-stream",
              "parameters": {
                   "duration": "PT5M",
                   "percentage": "100",
                   "service": "kinesis"
              },
              "targets": {
                    "KinesisStreams": "KinesisStreams-Target-1"
              }
         }
   },
   "stopConditions": [
         {
              "source": "none"
         }
   ],
   "roleArn": "arn:aws:iam::111122223333:role/role-name",
   "tags": {},
   "experimentOptions": {
       "accountTargeting": "single-account",
       "emptyTargetResolutionMode": "fail"
   }
}
```

實驗角色許可範例

下列許可可讓您對影響 50% 請求的特定串流執行 aws:kinesis:stream-provisioned-throughput-exception和 aws:kinesis:stream-expired-iterator-exception動作。

迭代器例外錯誤已過期

過期的疊代運算例外錯誤 (HTTP 400) 會在碎片疊代運算過期時發生，而且不再用於在呼叫 時擷取串流記錄GetRecords。當讀取操作之間發生延遲，這是由於長時間執行的資料處理任務、網路問題或應用程式停機時間所造成。

> **Note**
>
> 碎片疊代運算在發出後 5 分鐘內有效。

處理例外狀況的建議

- 在碎片迭代器過期之前重新整理碎片迭代器。
- 整合錯誤處理以取得新的迭代器。
- 使用 Kinesis Kinesis Client Library (KCL) 來自動管理碎片迭代器過期。

如需詳細資訊，請參閱什麼是 AWS Fault Injection Service？

執行基本實驗

1. 建立實驗範本：使用 AWS FIS 主控台。
2. 選取動作：使用 `aws:kinesis:inject-api-expired-iterator-exception`動作。
3. 設定目標：指定 IAM 角色和 Kinesis Data Streamsoperations。
4. 設定持續時間：從初始測試的 5-10 分鐘開始。
5. 新增停止條件：停止條件 AWS FIS。
6. 執行實驗：監控應用程式行為。

動作詳細資訊

- 資源類型：IAM 角色 ARN
- 目標操作： `GetRecords`
- 錯誤代碼：`ExpiredIteratorException`(HTTP 400)
- 描述：提供的迭代器超過允許的最長存留期，模擬記錄處理太慢或檢查點邏輯失敗的案例。

Parameters

- IAM 角色 ARN：您的應用程式用於 Kinesis Data Streams 操作的角色。
- 操作：目標操作： `GetRecords`
- 資源清單：特定的串流名稱或 ARNs。
- 持續時間：實驗持續時間。這是可設定的。
- 強度：要調節的請求百分比。

所需的許可

- `kinesis:InjectApiError`

將資料寫入 Amazon Kinesis Data Streams

生產者是將資料寫入 Amazon Kinesis Data Streams 的應用程式。您可以使用 適用於 Java 的 AWS SDK 和 Kinesis Producer Library (KPL) 為 Kinesis Data Streams 建置生產者。

如果您是 Kinesis Data Streams 的新手，請先熟悉一下 什麼是 Amazon Kinesis Data Streams？ 及 使用 AWS CLI 執行 Amazon Kinesis Data Streams 操作 所介紹的概念和術語。

> **Important**
>
> Kinesis Data Streams 支援對資料串流變更資料記錄保留期間。如需詳細資訊，請參閱變更資料保留期間。

若要將資料放入串流，您必須指定串流的名稱、分割區索引鍵以及要加入至串流的資料 Blob。分割區索引鍵用於決定串流中將要加入資料記錄的碎片。

碎片中的所有資料會傳送給負責處理碎片的同一個工作者。您應使用哪個分割區索引鍵取決於您的應用程式邏輯。通常，分割區索引鍵數目應該要比碎片數目大得多。這是因為分割區索引鍵將用於決定資料記錄如何對應到特定碎片。如果分割區索引鍵數目夠多，資料就能均勻地分佈於串流中的各個碎片。

主題

- 使用 Amazon Kinesis Producer Library (KPL) 開發生產者
- 使用 Amazon Kinesis Data Streams API 搭配 開發生產者 適用於 Java 的 AWS SDK
- 使用 Kinesis Agent 寫入 Amazon Kinesis Data Streams
- 使用其他服務寫入 Kinesis Data Streams AWS
- 使用第三方整合寫入 Kinesis Data Streams
- Amazon Kinesis Data Streams 生產者疑難排解
- 最佳化 Kinesis Data Streams 生產者

使用 Amazon Kinesis Producer Library (KPL) 開發生產者

Amazon Kinesis Data Streams 生產者是將使用者資料記錄放入 Kinesis 資料串流 (又稱為資料擷取) 的應用程式。Amazon Kinesis Producer Library (KPL) 可簡化生產者應用程式開發，讓開發人員達到 Kinesis 資料串流的高寫入輸送量。

您可以使用 Amazon CloudWatch 對 KPL 進行監控。如需詳細資訊，請參閱使用 Amazon CloudWatch 監控 Kinesis Producer Library。

主題

> **Note**
>
> 建議您升級至最新 KPL 版本。KPL 會定期更新為更新的版本，其中包括最新的相依性和安全性修補程式、錯誤修正，以及向後相容的新功能。如需詳細資訊，請參閱 https://github.com/awslabs/amazon-kinesis-producer/releases/。

檢閱 KPL 的角色

KPL 是一套具高度可設定性的易用程式庫，能協助您對 Kinesis 資料串流進行寫入。此程式庫在您的生產者應用程式的程式碼與 Kinesis Data Streams API 動作之間擔任媒介。KPL 將執行以下主要任務：

- 利用可設定的自動重試機制對一個或多個 Kinesis 資料串流進行寫入

- 收集記錄並於每次請求時使用 `PutRecords` 將多筆記錄寫入多個碎片
- 彙整使用者記錄以增加承載大小並提高傳輸量
- 與 Kinesis Client Library (KCL) 無縫整合以取消彙整取用者上的批次記錄
- 代表您提交 Amazon CloudWatch 指標以提供關於生產者效能的資訊

請注意，KPL 與 AWS SDK 所提供的 Kinesis Data Streams API 不同。Kinesis Data Streams API 可協助您管理 Kinesis Data Streams 的許多層面 (包括建立串流、重新分片、放入與取得記錄)，而 KPL 提供專用於擷取資料的抽象層。如需 Kinesis Data Streams API 的相關資訊，請參閱 Amazon Kinesis API 參考。

了解使用 KPL 的優點

以下清單舉出使用 KPL 開發 Kinesis Data Streams 生產者的一些主要優點。

KPL 可用於同步或非同步使用案例。建議您使用具有較高效能的非同步界面，除非有具體的原因需要使用同步操作。如需以上兩種使用案例的詳細資訊及範例程式碼，請參閱使用 KPL 寫入 Kinesis 資料串流。

效能優勢

KPL 可協助建置高效能的生產者。試想以下情況：您的 Amazon EC2 執行個體擔任代理，從數以百計或千計的低功率裝置收集 100 位元組的事件並將記錄寫入 Kinesis 資料串流。這些 EC2 執行個體每秒均須將成千個事件寫入您的資料串流。為達到所需的傳輸量，生產者必須實作複雜的邏輯 (如批次處理或多執行緒) 和重試邏輯並在消費者端取消彙整記錄。KPL 將為您執行所有這些任務。

消費者端易用性

對於使用 KCL 的 Java 取用者端開發人員來說，KPL 整合毫不費力。當 KCL 擷取含有多筆 KPL 使用者記錄的彙整 Kinesis Data Streams 記錄時，會自動叫用 KPL 先擷取個別的使用者記錄，然後再將其傳回給使用者。

取用者端開發人員若未使用 KCL 而是直接使用 API `GetRecords` 操作，則可利用 KPL Java 程式庫擷取個別的使用者記錄後再將其傳回給使用者。

生產者監控

您可使用 Amazon CloudWatch 和 KPL 收集、監控及分析您的 Kinesis Data Streams 生產者。KPL 將代表您向 CloudWatch 發出輸送量指標、錯誤指標及其他指標，並可設定成進行串流層級、碎片層級或生產者層級監控。

非同步架構

由於 KPL 可能會在將記錄傳送到 Kinesis Data Streams 之前緩衝記錄，因此不會強制發起人應用程式封鎖並等待確認記錄已送達伺服器，然後再繼續執行時間。將記錄放入 KPL 的呼叫一律立即傳回，不會等待傳送記錄或接收伺服器的回應。反而，其將建立 `Future` 物件用以稍後接收傳送記錄至 Kinesis Data Streams 的結果。這與 AWS SDK 中的非同步用戶端的行為相同。

了解何時不使用 KPL

KPL 可能引發程式庫中額外的處理延遲，最高達到 `RecordMaxBufferedTime` (使用者可設定)。`RecordMaxBufferedTime` 的值愈大，壓縮效率以及效能愈高。無法容忍此額外延遲的應用程式可能需要直接使用 AWS SDK。如需搭配 Kinesis Data Streams 使用 AWS SDK 的詳細資訊，請參閱 使用 Amazon Kinesis Data Streams API 搭配 開發生產者 適用於 Java 的 AWS SDK。如需 `RecordMaxBufferedTime` 以及 KPL 可供使用者設定的其他各項屬性的詳細資訊，請參閱 設定 Amazon Kinesis Producer Library。

安裝 KPL

Amazon 為 macOS、Windows 和最近的 Linux 發行版本提供預先建置的 C++ Amazon Kinesis Producer Library (KPL) 二進位檔 （如需支援的平台詳細資訊，請參閱下一節）。這些二進位檔封裝於 Java .jar 檔案中，若您使用 Maven 安裝該套件，即會自動叫用並予使用。如要尋找最新版的 KPL 和 KCL，請使用以下 Maven 搜尋連結：

- KPL
- KCL

Linux 二進位檔已使用 GNU 編譯器套件 (GCC) 進行編譯並靜態連結到 Linux 上的 libstdc++。這些二進位檔應能適用於任何附有 glibc 2.5 或更高版本的 64 位元 Linux 發行版本。

舊版 Linux 發行版本的使用者可以使用 GitHub 上與來源一起提供的建置說明來建置 KPL。若要從 GitHub 下載 KPL，請參閱 Amazon Kinesis Producer Library。

> **Important**
>
> Amazon Kinesis Producer Library (KPL) 0.x 將於 2026 年 1 月 30 日end-of-support。我們強烈建議您使用 0.x 版將 KPL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KPL 版本。若要尋

找最新的 KPL 版本，請參閱 Github 上的 KPL 頁面。如需從 KPL 0.x 遷移至 KPL 1.x 的資訊，請參閱 從 KPL 0.x 遷移至 KPL 1.x。

從 KPL 0.x 遷移至 KPL 1.x

本主題提供step-by-step說明，將您的消費者從 KPL 0.x 遷移至 KPL 1.x。KPL 1.x 推出對 適用於 Java 的 AWS SDK 2.x 的支援，同時保持與舊版的介面相容性。您不需要更新核心資料處理邏輯，即可遷移至 KPL 1.x。

1. 請確定您有下列先決條件：

   - Java 開發套件 (JDK) 8 或更新版本
   - 適用於 Java 的 AWS SDK 2.x
   - Maven 或 Gradle 用於相依性管理

2. 新增相依性

   如果您使用的是 Maven，請將下列相依性新增至 pom.xml 檔案。請務必將 groupId 從 更新`com.amazonaws`為 ，`software.amazon.kinesis`並將 版本更新`1.x.x`為最新的 KPL 版本。

   ```
   <dependency>
       <groupId>software.amazon.kinesis</groupId>
       <artifactId>amazon-kinesis-producer</artifactId>
       <version>1.x.x</version> <!-- Use the latest version -->
   </dependency>
   ```

   如果您使用的是 Gradle，請將以下內容新增至您的 `build.gradle` 檔案。請務必將 取代`1.x.x`為最新的 KPL 版本。

   ```
   implementation 'software.amazon.kinesis:amazon-kinesis-producer:1.x.x'
   ```

   您可以在 Maven Central Repository 上檢查最新版本的 KPL。

3. 更新 KPL 的匯入陳述式

   KPL 1.x 使用 適用於 Java 的 AWS SDK 2.x，並使用以 開頭的更新套件名稱`software.amazon.kinesis`，相較於以 開頭的先前 KPL 中的套件名稱`com.amazonaws.services.kinesis`。

將 的匯入取代`com.amazonaws.services.kinesis`為 `software.amazon.kinesis`。下表列出您必須取代的匯入。

匯入替換項目

取代：	取代為：
import com.amazonaws.services.kinesis.producer.Attempt；	import software.amazon.kinesis.producer.Attempt；
import com.amazonaws.services.kinesis.producer.BinaryToHexConverter；	import software.amazon.kinesis.producer.BinaryToHexConverter；
import com.amazonaws.services.kinesis.producer.CertificateExtractor；	import software.amazon.kinesis.producer.CertificateExtractor；
import com.amazonaws.services.kinesis.producer.Daemon；	import software.amazon.kinesis.producer.Daemon；
import com.amazonaws.services.kinesis.producer.DaemonException；	import software.amazon.kinesis.producer.DaemonException；
import com.amazonaws.services.kinesis.producer.FileAgeManager；	import software.amazon.kinesis.producer.FileAgeManager；
import com.amazonaws.services.kinesis.producer.FutureTimedOutException；	import software.amazon.kinesis.producer.FutureTimedOutException；
import com.amazonaws.services.kinesis.producer.GlueSchemaRegistrySerializerInstance；	import software.amazon.kinesis.producer.GlueSchemaRegistrySerializerInstance；
import com.amazonaws.services.kinesis.producer.HashedFileCopier；	import software.amazon.kinesis.producer.HashedFileCopier；
import com.amazonaws.services.kinesis.producer.IKinesisProducer；	import software.amazon.kinesis.producer.IKinesisProducer；

取代：	取代為：
import com.amazonaws.services.kinesis.producer.IrrecoverableError；	import software.amazon.kinesis.producer.IrrecoverableError；
import com.amazonaws.services.kinesis.producer.KinesisProducer；	import software.amazon.kinesis.producer.KinesisProducer；
import com.amazonaws.services.kinesis.producer.KinesisProducerConfiguration；	import software.amazon.kinesis.producer.KinesisProducerConfiguration；
import com.amazonaws.services.kinesis.producer.LogInputStreamReader；	import software.amazon.kinesis.producer.LogInputStreamReader；
import com.amazonaws.services.kinesis.producer.Metric；	import software.amazon.kinesis.producer.Metric；
import com.amazonaws.services.kinesis.producer.ProcessFailureBehavior；	import software.amazon.kinesis.producer.ProcessFailureBehavior；
import com.amazonaws.services.kinesis.producer.UnexpectedMessageException；	import software.amazon.kinesis.producer.UnexpectedMessageException；
import com.amazonaws.services.kinesis.producer.UserRecord；	import software.amazon.kinesis.producer.UserRecord；
import com.amazonaws.services.kinesis.producer.UserRecordFailedException；	import software.amazon.kinesis.producer.UserRecordFailedException；
import com.amazonaws.services.kinesis.producer.UserRecordResult；	import software.amazon.kinesis.producer.UserRecordResult；
import com.amazonaws.services.kinesis.producer.protobuf.Messages；	import software.amazon.kinesis.producer.protobuf.Messages；
import com.amazonaws.services.kinesis.producer.protobuf.Config；	import software.amazon.kinesis.producer.protobuf.Config；

4. 更新 AWS 登入資料提供者類別的匯入陳述式

遷移至 KPL 1.x 時，您必須在以 適用於 Java 的 AWS SDK 1.x 為基礎的 KPL 應用程式程式碼中，將匯入中的套件和類別更新為以 適用於 Java 的 AWS SDK 2.x 為基礎的對應套件和類別。KPL 應用程式中的常見匯入是登入資料提供者類別。如需登入資料提供者變更的完整清單，請參閱 適用於 Java 的 AWS SDK 2.x 遷移指南文件中的登入資料提供者變更。以下是您可能需要在 KPL 應用程式中進行的常見匯入變更。

在 KPL 0.x 中匯入

```
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
```

在 KPL 1.x 中匯入

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
```

如果您根據 適用於 Java 的 AWS SDK 1.x 匯入任何其他登入資料提供者，則必須將其更新為 適用於 Java 的 AWS SDK 2.x 同等登入資料提供者。如果您未從 適用於 Java 的 AWS SDK 1.x 匯入任何類別/套件，您可以忽略此步驟。

5. 在 KPL 組態中更新登入資料提供者組態

KPL 1.x 中的登入資料提供者組態需要 適用於 Java 的 AWS SDK 2.x 登入資料提供者。如果您要透過覆寫預設登入資料提供者`KinesisProducerConfiguration`，在 中傳遞 適用於 Java 的 AWS SDK 1.x 的登入資料提供者，您必須使用 適用於 Java 的 AWS SDK 2.x 登入資料提供者進行更新。如需登入資料提供者變更的完整清單，請參閱 適用於 Java 的 AWS SDK 2.x 遷移指南文件中的登入資料提供者變更。如果您沒有覆寫 KPL 組態中的預設登入資料提供者，您可以忽略此步驟。

例如，如果您使用下列程式碼覆寫 KPL 的預設登入資料提供者：

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration();
// SDK v1 default credentials provider
config.setCredentialsProvider(new DefaultAWSCredentialsProviderChain());
```

您必須使用下列程式碼更新它們，才能使用 適用於 Java 的 AWS SDK 2.x 登入資料提供者：

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration();
// New SDK v2 default credentials provider
config.setCredentialsProvider(DefaultCredentialsProvider.create());
```

轉換至 KPL 的 Amazon Trust Services (ATS) 憑證

於 2018 年 2 月 9 日上午 9:00 (太平洋標準時間)，Amazon Kinesis Data Streams 已安裝 ATS 憑證。若要繼續使用 Amazon Kinesis Producer Library (KPL) 將記錄寫入 Kinesis Data Streams，您必須將 KPL 的安裝升級至 0.12.6 版或更新版本。此變更會影響所有 AWS 區域。

如需有關移至 ATS 的資訊，請參閱如何準備 AWS移至其自己的憑證授權單位。

若您遇到問題而需要技術支援，請透過 AWS Support 中心建立案例。

KPL 支援的平台

Amazon Kinesis Producer Library (KPL) 是以 C++ 撰寫，並做為主要使用者程序的子程序執行。預先編譯的 64 位元原生二進位檔隨附於 Java 版本，由 Java 包裝函式管理。

Java 套件在以下作業系統執行無須安裝任何其他程式庫：

- 核心為 2.6.18 (2006 年 9 月) 及更新版本的 Linux 發行版本
- Apple iOS X 10.9 及更新版本
- Windows Server 2008 及更新版本

> **Important**
>
> 所有 KPL 版本 (最高可達版本 0.14.0) 都支援 Windows Server 2008 和更新版本。從 KPL 版本 0.14.0 或更高版本開始，不支援 Windows 平台。

請注意，KPL 只有 64 位元版本。

來源碼

如果 KPL 安裝所提供的二進位檔未能滿足您的環境，KPL 的核心已編寫成 C++ 模組。C++ 模組的原始碼和 Java 界面是根據 Amazon Public License 發行，可在 GitHub 上的 Amazon Kinesis Producer Library 取得。儘管 KPL 可在已有符合標準的最新版 C++ 編譯器和 JRE 的任何平台上使用，Amazon 仍未正式支援未列於支援的平台清單中的任何平台。

KPL 關鍵概念

下列各節包含了解和受益於 Amazon Kinesis Producer Library (KPL) 的必要概念和術語。

主題

記錄

本指南區分了 KPL 使用者記錄和 Kinesis Data Streams 記錄。當使用不帶限定詞的記錄一詞時，指的就是 KPL 使用者記錄。當我們參照 Kinesis Data Streams 記錄時，我們明確地說 Kinesis Data Streams 記錄。

KPL 使用者記錄是對使用者具有特別含意的資料 Blob。例子包括代表網站 UI 事件或 web 伺服器某個日誌項目的 JSON Blob。

Kinesis Data Streams 記錄是 Kinesis 資料串流服務 API 所定義之 `Record` 資料結構的執行個體。此種記錄包含分割區索引鍵、序號和資料 Blob。

批次處理

批次處理是指對多個項目執行單一動作，而不是對每個單獨項目重複執行該動作。

就此處而言，「項目」是一筆記錄，「動作」則是將記錄傳送至 Kinesis Data Streams。在非批次處理的情況下，您會將每筆記錄放入單獨的 Kinesis Data Streams 記錄中，然後發出一次 HTTP 請求將其傳送至 Kinesis Data Streams。透過批次處理，每次 HTTP 請求可攜帶多筆記錄而不只有一筆。

KPL 支援兩種批次處理方式：

- 彙整 – 將多筆記錄存放於單筆 Kinesis Data Streams 記錄中。
- 收集 – 使用 API `PutRecords` 操作，將多筆 Kinesis Data Streams 記錄傳送至 Kinesis 資料串流中的一個或多個碎片。

KPL 兩種批次處理方式的設計彼此共存並可單獨啟用或停用。預設情況下，兩者將一併啟用。

聚合

彙整是指將多筆記錄儲存於單筆 Kinesis Data Streams 記錄中。彙整使得客戶能夠增加每次 API 呼叫所傳送的記錄筆數，進而有效地提高生產者傳輸量。

Kinesis Data Streams 碎片支援每秒多達 1,000 筆 Kinesis Data Streams 記錄或 1 MB 輸送量。Kinesis Data Streams 每秒記錄筆數限制綁定記錄在 1 KB 以內的客戶。記錄彙整使得客戶能夠將多筆記錄合併為單筆 Kinesis Data Streams 記錄。客戶可藉此提高其每一碎片傳輸量。

假設區域 us-east-1 中有一個碎片目前以每秒 1，000 個記錄的固定速率執行，且每個記錄為 512 個位元組。透過 KPL 彙整，您可以將這 1,000 筆記錄壓縮成只有 10 筆 Kinesis Data Streams 記錄，進而使 RPS 降至 10 (每次 50 KB)。

收集

收集是指批次處理多筆 Kinesis Data Streams 記錄並透過對 API 操作 `PutRecords` 呼叫的單次 HTTP 請求來傳送記錄，而非以各自的 HTTP 請求來傳送每個 Kinesis Data Streams 記錄。

這與不使用收集相比可提高傳輸量，因為其減少了發出多次個別 HTTP 請求的額外負擔。事實上，`PutRecords` 本身即是專為達到此目的而設計。

收集與彙整不同，其將處理 Kinesis Data Streams 記錄群組。正在收集的 Kinesis Data Streams 記錄仍可包含來自使用者的多筆記錄。兩者間的關係可用以下視覺化方式表達：

整合 KPL 與生產者程式碼

Amazon Kinesis Producer Library (KPL) 會以個別程序執行，並使用 IPC 與您的父使用者程序通訊。此架構有時稱為微服務，選用的主要原因有二：

1) 即使 KPL 當機也不會造成您的使用者程序當機

您的程序可執行與 Kinesis Producer Library (KPL) 無關的任務，而且就算 KPL 當機也仍能繼續操作。您的父使用者程序也可重新啟動 KPL 並恢復到完全運作狀態 (此功能由官方包裝函式提供)。

例如，若 web 伺服器傳送指標至 Kinesis Data Streams，就算 Kinesis Data Streams 部分已停止運作，伺服器仍能繼續處理頁面。整部伺服器會由於 KPL 中存在錯誤而當機，以致造成不必要的停機。

2) 任意用戶端均可支援

客戶當中肯定會有人使用非正式支援的語言。這類客戶也應能輕鬆使用 KPL。

建議使用矩陣

下列用量矩陣會列出不同使用者的建議設定，並建議您是否及如何使用 KPL。請切記，如果啟用彙整，則必須連帶使用取消彙整以便在消費者端擷取記錄。

生產者端語言	消費者端語言	KCL 版本	檢查點邏輯	能否使用 KPL？	警告
Java 除外的任何語言	*	*	*	否	N/A
Java	Java	直接使用 Java 開發套件	N/A	是	如果使用彙整，則呼叫 `GetRecords` 後必須使用現成提供的取消彙整程式庫。
Java	Java 除外的任何語言	直接使用軟體開發套件	N/A	是	必須停用彙整。

生產者端語言	消費者端語言	KCL 版本	檢查點邏輯	能否使用 KPL？	警告
Java	Java	1.3.x	N/A	是	必須停用彙整。
Java	Java	1.4.x	不帶任何引數呼叫檢查點	是	無
Java	Java	1.4.x	使用顯式序號呼叫檢查點	是	停用彙整，或將程式碼改為使用擴展序號執行檢查點作業。
Java	Java 除外的任何語言	1.3.x + 多語言協助程式 + 特定語言包裝函式	N/A	是	必須停用彙整。

使用 KPL 寫入 Kinesis 資料串流

下列各節顯示從最基本生產者到完全非同步程式碼的進度中的範例程式碼。

Barebones 生產者程式碼

以下是撰寫最低限度能夠運作的生產者所需的全部程式碼。Amazon Kinesis Producer Library (KPL) 使用者記錄會在背景處理。

```
// KinesisProducer gets credentials automatically like
// DefaultAWSCredentialsProviderChain.
// It also gets region automatically from the EC2 metadata service.
KinesisProducer kinesis = new KinesisProducer();
// Put some records
for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    kinesis.addUserRecord("myStream", "myPartitionKey", data);
}
```

```
// Do other stuff ...
```

同步回應結果

前述範例的程式碼並未檢查 KPL 使用者記錄是否成功。KPL 會就失敗狀況執行任何必要的重試。然而若您想要檢查結果，則可使用 Future 所傳回的 addUserRecord 物件進行檢查，如以下範例所示 (另顯示前述範例以供對照)：

```
KinesisProducer kinesis = new KinesisProducer();

// Put some records and save the Futures
List<Future<UserRecordResult>> putFutures = new
 LinkedList<Future<UserRecordResult>>();
for (int i = 0; i < 100; i++) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    putFutures.add(
        kinesis.addUserRecord("myStream", "myPartitionKey", data));
}

// Wait for puts to finish and check the results
for (Future<UserRecordResult> f : putFutures) {
    UserRecordResult result = f.get(); // this does block
    if (result.isSuccessful()) {
        System.out.println("Put record into shard " +
                           result.getShardId());
    } else {
        for (Attempt attempt : result.getAttempts()) {
            // Analyze and respond to the failure
        }
    }
}
```

非同步回應結果

先前的範例是在Future物件get()上呼叫 ，這會封鎖執行時間。如果您不想封鎖執行時間，可以使用非同步回呼，如下列範例所示：

```
KinesisProducer kinesis = new KinesisProducer();

FutureCallback<UserRecordResult> myCallback = new FutureCallback<UserRecordResult>() {
```

```
    @Override public void onFailure(Throwable t) {
        /* Analyze and respond to the failure  */
    };
    @Override public void onSuccess(UserRecordResult result) {
        /* Respond to the success */
    };
};

for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    ListenableFuture<UserRecordResult> f = kinesis.addUserRecord("myStream",
 "myPartitionKey", data);
    // If the Future is complete by the time we call addCallback, the callback will be
 invoked immediately.
    Futures.addCallback(f, myCallback);
}
```

設定 Amazon Kinesis Producer Library

儘管預設的設定應能適用於大多數使用案例，但您也許想要變更某些預設值，以依照您的需求量身打造 KinesisProducer 的行為。為此，您可以將 KinesisProducerConfiguration 類別的執行個體傳遞給 KinesisProducer 建構函數，例如：

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()
        .setRecordMaxBufferedTime(3000)
        .setMaxConnections(1)
        .setRequestTimeout(60000)
        .setRegion("us-west-1");

final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

您也可以從屬性檔案載入組態：

```
KinesisProducerConfiguration config =
 KinesisProducerConfiguration.fromPropertiesFile("default_config.properties");
```

您可以替換使用者程序能夠存取的任何路徑和檔案名稱。此外，您亦可對以此方式建立的 KinesisProducerConfiguration 執行個體呼叫 set 方法以自訂組態。

屬性檔案指定參數時應使用各參數的帕斯卡命名法名稱。這類名稱與 KinesisProducerConfiguration 類別的 set 方法中所使用的名稱相吻。例如：

```
RecordMaxBufferedTime = 100
MaxConnections = 4
RequestTimeout = 6000
Region = us-west-1
```

如需組態參數用法規則及數值限制的詳細資訊，請參閱 GitHub 上的範例組態屬性。

請注意，一旦 KinesisProducer 初始化，變更使用中的 KinesisProducerConfiguration 執行個體將不會再有任何作用。KinesisProducer 目前不支援動態重新配置。

實作消費者取消彙總

自發行版本 1.4.0 起，KCL 支援自動取消彙整 KPL 使用者記錄。在您更新 KCL 後，使用舊版 KCL 撰寫的取用者應用程式將編譯程式碼而不會進行任何修改。不過，如果生產者端使用了 KPL 彙整，則有一項與檢查點作業相關的細微之處須留意：彙整的記錄中所有的子記錄具有相同的序號，所以若您需要區別子記錄，就必須隨檢查點存放額外的資料。這些額外的資料稱為子序號。

選項

- 從舊版 KCL 遷移
- 使用 KPL 去彙總的 KCL 擴充功能
- 直接使用 GetRecords

從舊版 KCL 遷移

您不需要變更現有的呼叫，即可使用彙總執行檢查點。保證您仍能成功擷取存放於 Kinesis Data Streams 的所有記錄。KCL 現在提供兩個新的檢查點操作，以支援特定使用案例，如下所述。

如果您在 KPL 支援之前為 KCL 編寫現有程式碼，而且您的檢查點操作在沒有引數的情況下呼叫，則相當於對批次中最後一個 KPL 使用者記錄的序號進行檢查點。如果使用序號字串呼叫檢查點操作，則等同於對批次的指定序號及隱含的子序號 0 (零) 執行檢查點作業。

呼叫新的 KCL 檢查點操作而不帶任何引數如 checkpoint() 語意上等同於對批次中其上次 Record 呼叫的序號及隱含的子序號 0 (零) 執行檢查點作業。

呼叫新的 KCL 檢查點操作如 checkpoint(Record record) 語意上等同於對指定 Record 的序號及隱含的子序號 0 (零) 執行檢查點作業。若 Record 呼叫實際為 UserRecord，則會對 UserRecord 序號和子序號執行檢查點作業。

呼叫新的 KCL 檢查點操作如 `checkpoint(String sequenceNumber, long subSequenceNumber)` 會對指定的序號及指定的子序號執行明示檢查點作業。

上述任何情況下，當檢查點已存放於 Amazon DynamoDB 檢查點資料表之後，KCL 便能正確地恢復擷取記錄，就算應用程式當機並重新啟動也沒問題。如果序列中包含多筆記錄，則會從序號最近執行過檢查點作業的記錄中為下一個子序號的記錄開始擷取。若最近的檢查點包括前一序號記錄的最新子序號，將從下一個序號的記錄開始擷取。

下一節討論了消費者的序列和子序列檢查點的詳細資訊，必須避免略過和重複記錄。若停止並重新啟動消費者的記錄處理會略過 (或重複) 記錄無關緊要，您就可以執行現有的程式碼而無須修改。

使用 KPL 去彙總的 KCL 擴充功能

KPL 去彙總可能涉及子序列檢查點。為方便使用子序列檢查點作業，KCL 增加了 `UserRecord` 類別：

```
public class UserRecord extends Record {
    public long getSubSequenceNumber() {
    /* ... */
    }
    @Override
    public int hashCode() {
    /* contract-satisfying implementation */
    }
    @Override
    public boolean equals(Object obj) {
    /* contract-satisfying implementation */
    }
}
```

現已使用此類別代替 `Record`。這不會破壞現有的程式碼，因為其為 `Record` 的子類別。`UserRecord` 類別同時代表實際的子記錄和未彙整的標準記錄。未彙整的記錄可想像成恰有一筆子記錄的已彙整記錄。

此外，`IRecordProcessorCheckpointer` 也增加了兩項新的操作：

```
public void checkpoint(Record record);
public void checkpoint(String sequenceNumber, long subSequenceNumber);
```

若要開始使用子序號檢查點作業，您可進行以下轉換。更改以下形式的程式碼：

```
checkpointer.checkpoint(record.getSequenceNumber());
```

新形式的程式碼：

```
checkpointer.checkpoint(record);
```

建議您使用 checkpoint(Record record) 形式執行子序列檢查點作業。不過，若您已將 sequenceNumbers 存放在字串中用於檢查點作業，則現在亦應存放 subSequenceNumber，如以下範例所示：

```
String sequenceNumber = record.getSequenceNumber();
long subSequenceNumber = ((UserRecord) record).getSubSequenceNumber();  // ... do other
 processing
checkpointer.checkpoint(sequenceNumber, subSequenceNumber);
```

從 轉換為UserRecord Record一律會成功，因為實作一律使用 UserRecord。除非需要對序號進行算術運算，否則這種方式並不建議。

處理 KPL 使用者記錄時，KCL 會將子序號寫入 Amazon DynamoDB 成為每一列的額外欄位。舊版 KCL 是使用 AFTER_SEQUENCE_NUMBER 在恢復檢查點作業時擷取記錄。目前有 KPL 支援的 KCL 則改用 AT_SEQUENCE_NUMBER。在擷取已對序號執行過檢查點作業的記錄時，會檢查已執行檢查點作業的子序號，且將視需要刪除子記錄 (若已對最後一筆子記錄執行檢查點作業，則可能全部刪除)。同樣地，未彙整的記錄可想像成只有一筆子記錄的已彙整記錄，所以同一套演算法對已彙整和未彙整的記錄都適用。

直接使用 GetRecords

您也可以選擇不使用 KCL 而是直接調用 API 操作 GetRecords 來擷取 Kinesis Data Streams 記錄。若要將這些擷取到的記錄解壓縮為原始 KPL 使用者記錄，請呼叫 UserRecord.java 的以下任一項靜態操作：

```
public static List<Record> deaggregate(List<Record> records)

public static List<UserRecord> deaggregate(List<UserRecord> records, BigInteger
 startingHashKey, BigInteger endingHashKey)
```

第一項操作使用 0 的預設值 startingHashKey (零) 以及 2^128 -1 的預設值 endingHashKey。

上述每一項操作都將取消彙總指定的 Kinesis Data Streams 記錄清單，轉成 KPL 使用者記錄清單。傳回的記錄清單將會捨棄顯式雜湊索引鍵或分割區索引鍵落在 startingHashKey (含) 以及 endingHashKey (含) 範圍外的任何 KPL 使用者記錄。

搭配 Amazon Data Firehose 使用 KPL

如果您使用 Kinesis Producer Library (KPL) 來寫入資料到 Kinesis 資料串流，則可使用彙整來合併您寫入至該 Kinesis 資料串流的記錄。如果您接著使用該資料串流做為 Firehose 交付串流的來源，Firehose 會先取消彙總記錄，再將記錄交付至目的地。如果您將交付串流設定為轉換資料，Firehose 會先取消彙總記錄，再將其交付至其中 AWS Lambda。如需更多資訊，請參閱使用 Kinesis Data Streams 寫入至 Amazon Kinesis Data Firehose。

使用 KPL 搭配 AWS Glue 結構描述登錄檔

您可以將 Kinesis 資料串流與 AWS Glue 結構描述登錄檔整合。 AWS Glue 結構描述登錄檔可讓您集中探索、控制和發展結構描述，同時確保已註冊結構描述持續驗證產生的資料。結構描述定義資料記錄的結構和格式。結構描述是可靠的資料發佈、耗用或儲存的版本化規格。 AWS Glue 結構描述登錄檔可讓您改善串流應用程式中end-to-end資料品質和資料控管。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。設定此整合的方法之一是透過 KPL 和 Java 中的 Kinesis Client Library (KCL) 程式庫。

> **Important**
>
> 目前，Kinesis Data Streams AWS Glue 和結構描述登錄整合僅支援使用在 Java 中實作的 KPL 生產者的 Kinesis 資料串流。不提供多語言支援。

如需如何使用 KPL 設定 Kinesis Data Streams 與結構描述登錄檔整合的詳細說明，請參閱「使用 KPL/KCL 程式庫與資料互動」一節：將 Amazon Kinesis Data Streams 與 AWS Glue 結構描述登錄檔整合。

設定 KPL 代理組態

對於無法直接連線至網際網路的應用程式，所有 AWS SDK 用戶端都支援使用 HTTP 或 HTTPS 代理。在一般的企業環境中，所有輸出網路流量都必須經過代理伺服器。如果您的應用程式使用 Kinesis Producer Library (KPL) 在使用代理伺服器 AWS 的環境中收集和傳送資料至 ，您的應用程式將需要 KPL 代理組態。KPL 是建置在 AWS Kinesis SDK 上的高階程式庫。它被分成一個原生處理程序和一個包裝函式。原生處理程序會執行處理和傳送記錄的所有作業，而包裝函式則會管理原生處理序並與其通訊。如需詳細資訊，請參閱使用 Amazon Kinesis Producer Library 實作高效且可靠的生產者。

包裝函式是以 Java 撰寫的，而原生處理程序是使用 Kinesis SDK 以 C++ 撰寫的。KPL 版本 0.14.7 及更高版本現在支援 Java 包裝器中的代理組態，該包裝器可以將所有代理組態傳遞給原生處理程序。如需詳細資訊，請參閱 https://github.com/awslabs/amazon-kinesis-producer/releases/tag/v0.14.7。

您可以使用以下程式碼將代理組態新增到您的 KPL 應用程式。

```
KinesisProducerConfiguration configuration = new KinesisProducerConfiguration();
// Next 4 lines used to configure proxy
configuration.setProxyHost("10.0.0.0"); // required
configuration.setProxyPort(3128); // default port is set to 443
configuration.setProxyUserName("username"); // no default
configuration.setProxyPassword("password"); // no default

KinesisProducer kinesisProducer = new KinesisProducer(configuration);
```

KPL 版本生命週期政策

本主題概述 Amazon Kinesis Producer Library (KPL) 的版本生命週期政策。 會 AWS 定期提供 KPL 版本的新版本，以支援新功能和增強功能、錯誤修正、安全修補程式和相依性更新。我們建議您隨時掌握 KPL up-to-date以掌握最新功能、安全性更新和基礎相依性。我們不建議繼續使用不支援的 KPL 版本。

主要 KPL 版本的生命週期包含下列三個階段：

- 一般可用性 (GA) – 在此階段，完全支援主要版本。 AWS 提供一般次要和修補程式版本，包括對 Kinesis Data Streams 新功能或 API 更新的支援，以及錯誤和安全性修正。
- 維護模式 – AWS 限制修補程式版本，以僅解決重大錯誤修正和安全問題。主要版本不會收到 Kinesis Data Streams 新功能或 APIs 的更新。
- End-of-support – 主要版本將不再收到更新或版本。先前發佈的版本將繼續透過公有套件管理員提供，且程式碼將保留在 GitHub 上。使用者可自行決定是否使用已end-of-support版本。我們建議您升級至最新的主要版本。

主要版本	目前階段	版本日期	維護模式日期	End-of-support日期
KPL 0.x	維護模式	2015-06-02	2025-04-17	2026-01-30
KPL 1.x	一般可用性	2024-12-15	--	--

使用 Amazon Kinesis Data Streams API 搭配 開發生產者 適用於 Java 的 AWS SDK

您可以使用 Amazon Kinesis Data Streams API 搭配適用於 Java 的 AWS SDK 來開發生產者。如果您是 Kinesis Data Streams 的新手，請先熟悉一下 什麼是 Amazon Kinesis Data Streams？ 及 使用 AWS CLI 執行 Amazon Kinesis Data Streams 操作 所介紹的概念和術語。

本文範例會討論 Kinesis Data Streams API 並使用適用於 Java 的AWS SDK 將資料加入 (放入) 串流。不過，對於大部分的使用案例，則應使用 Kinesis Data Streams KPL 程式庫為宜。如需詳細資訊，請參閱使用 Amazon Kinesis Producer Library (KPL) 開發生產者。

本章的 Java 範例程式碼示範如何執行基本的 Kinesis Data Streams API 操作，並依操作類型按照邏輯進行劃分。這些範例不代表可立即生產的程式碼，無法檢查出所有可能的例外狀況，也不可視為任何潛在安全或效能疑慮的原因。此外，您亦可使用其他程式設計語言呼叫 Kinesis Data Streams API。如需所有 AWS SDKs的詳細資訊，請參閱使用 Amazon Web Services 開始開發。

每項任務皆有其先決條件；例如，若要加入資料至串流，您必須先建立串流，而建立串流則需事先建立用戶端。如需詳細資訊，請參閱建立和管理 Kinesis 資料串流。

主題

- 將資料新增至串流
- 使用 AWS Glue 結構描述登錄檔與資料互動

將資料新增至串流

一旦建立了串流之後，您即可將資料以記錄的形式加入至該串流。記錄是一種資料結構，其中包含所要處理的資料 Blob 形式的資料。當您將資料存放於記錄後，Kinesis Data Streams 即絲毫不會檢查、解譯或變更該資料。每筆記錄也各有其相關聯的序號和分割區索引鍵。

Kinesis Data Streams API 提供兩種不同的操作可新增資料至串流：`PutRecords` 和 `PutRecord`。`PutRecords` 操作會透過 HTTP 請求將多筆記錄傳送至您的串流，而單數的 `PutRecord` 操作則是一次傳送一筆記錄至您的串流 (需針對每筆記錄發出單獨的 HTTP 請求)。大多數應用程式均應使用 `PutRecords` 為宜，因為這將使每個資料生產者達到更高的傳輸量。如需上述各項操作的詳細資訊，請分別參閱以下各小節。

主題

- 使用 PutRecords 新增多筆記錄

- 使用 PutRecord 新增單一記錄

務請切記，若您的來源應用程式使用 Kinesis Data Streams API 加入資料至串流，很可能會有一個或多個取用者應用程式同時處理從串流取出的資料。如需有關取用者如何使用 Kinesis Data Streams API 取得資料的詳細資訊，請參閱 從串流取得資料。

> **Important**
>
> 變更資料保留期間

使用 PutRecords 新增多筆記錄

PutRecords 操作會透過單次請求將多筆記錄傳送至 Kinesis Data Streams。藉由使用 PutRecords，生產者傳送資料至其 Kinesis 資料串流時將可達到更高的輸送量。每個 PutRecords 請求最高可支援 500 筆記錄。請求中的每筆記錄最大可為 1 MB，整個請求的最高限制為 5 MB，包括分區索引鍵。如同以下所述的單數 PutRecord 操作，PutRecords 也使用序號和分割區索引鍵。不過，PutRecord 的 SequenceNumberForOrdering 參數並未包含在 PutRecords 呼叫中。PutRecords 操作將嘗試依照請求的自然順序處理所有記錄。

每筆資料記錄都有獨一無二的序號。此序號是由 Kinesis Data Streams 在您呼叫 client.putRecords 將資料記錄加入至串流後所指派。同一分割區索引鍵的序號通常會隨著時間而增加；逐次 PutRecords 請求的間隔期間愈長，序號將變得愈大。

> **Note**
>
> 序號不能用做為同一串流中各資料集的索引。若要按照邏輯分隔資料集，請使用分割區索引鍵或為每個資料集建立個別串流。

PutRecords 請求可以附上具有不同分割區索引鍵的記錄。請求是以整個串流當成範圍；每次請求均能附上任意組合的分割區索引鍵和記錄，總數最多可達到請求的限額。使用許多不同的分割區索引鍵對具有許多不同碎片的串流發出請求，通常會比使用少量的分割區索引鍵對少量的碎片發出請求更快。分割區索引鍵數目應該要比碎片數目大得多，以減少延遲並達到最高的傳輸量。

PutRecords 範例

以下程式碼會建立 100 筆具有循序分割區索引鍵的資料記錄，並將其放入名為 DataStream 的串流。

```
        AmazonKinesisClientBuilder clientBuilder =
AmazonKinesisClientBuilder.standard();

        clientBuilder.setRegion(regionName);
        clientBuilder.setCredentials(credentialsProvider);
        clientBuilder.setClientConfiguration(config);

        AmazonKinesis kinesisClient = clientBuilder.build();

        PutRecordsRequest putRecordsRequest  = new PutRecordsRequest();
        putRecordsRequest.setStreamName(streamName);
        List <PutRecordsRequestEntry> putRecordsRequestEntryList  = new ArrayList<>();
        for (int i = 0; i < 100; i++) {
            PutRecordsRequestEntry putRecordsRequestEntry  = new
PutRecordsRequestEntry();

putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(i).getBytes()));
            putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d",
i));
            putRecordsRequestEntryList.add(putRecordsRequestEntry);
        }

        putRecordsRequest.setRecords(putRecordsRequestEntryList);
        PutRecordsResult putRecordsResult  =
kinesisClient.putRecords(putRecordsRequest);
        System.out.println("Put Result" + putRecordsResult);
```

PutRecords 回應包含回應 Records 的陣列。回應陣列中的每筆記錄依照自然順序 (請求和回應的內容由上而下) 與請求陣列中的某一記錄直接相關。回應 Records 陣列一律包含與請求陣列相同數目的記錄。

使用 PutRecords 時處理失敗

預設情況下，請求中個別記錄的失敗不會造成停止處理 PutRecords 請求中的後續記錄。這表示回應 Records 陣列包含已成功處理和未成功處理的記錄。您必須偵測未成功處理的記錄，並將其納入到後續呼叫中。

成功的記錄包含 SequenceNumber 和 ShardID 值，未成功的記錄則包含 ErrorCode 和 ErrorMessage 值。ErrorCode 參數反映錯誤類型，且可能是下列值的其中之一：ProvisionedThroughputExceededException 或 InternalFailure。ErrorMessage 提供關於ProvisionedThroughputExceededException 例外的更詳細資訊，包括帳戶 ID、串流名

稱、以及遭節制的記錄之碎片 ID。以下範例所示的 PutRecords 請求中有三筆記錄。第二筆記錄發生失敗，會反映在回應中。

Example PutRecords 請求語法

```
{
    "Records": [
        {
     "Data": "XzxkYXRhPl8w",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "AbceddeRFfg12asd",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "KFpcd98*7nd1",
     "PartitionKey": "partitionKey3"
        }
    ],
    "StreamName": "myStream"
}
```

Example PutRecords 回應語法

```
{
    "FailedRecordCount”: 1,
    "Records": [
        {
     "SequenceNumber": "21269319989900637946712965403778482371",
     "ShardId": "shardId-000000000001"

        },
        {
     “ErrorCode":”ProvisionedThroughputExceededException”,
     “ErrorMessage": "Rate exceeded for shard shardId-000000000001 in stream
 exampleStreamName under account 111111111111."

        },
        {
     "SequenceNumber": "21269319989999637946712965403778482985",
     "ShardId": "shardId-000000000002"
        }
```

```
    ]
}
```

未成功處理的記錄可納入到後續的 PutRecords 請求中。首先，查看 putRecordsResult 中的 FailedRecordCount 參數以確認請求中是否有失敗的記錄。若有，即應將 putRecordsEntry 非 ErrorCode 的每個 null 加入至後續的請求。如需此處理常式類型的範例，請參閱以下程式碼。

Example PutRecords 失敗處理常式

```
PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
putRecordsRequest.setStreamName(myStreamName);
List<PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
for (int j = 0; j < 100; j++) {
    PutRecordsRequestEntry putRecordsRequestEntry = new PutRecordsRequestEntry();
    putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(j).getBytes()));
    putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d", j));
    putRecordsRequestEntryList.add(putRecordsRequestEntry);
}

putRecordsRequest.setRecords(putRecordsRequestEntryList);
PutRecordsResult putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);

while (putRecordsResult.getFailedRecordCount() > 0) {
    final List<PutRecordsRequestEntry> failedRecordsList = new ArrayList<>();
    final List<PutRecordsResultEntry> putRecordsResultEntryList =
 putRecordsResult.getRecords();
    for (int i = 0; i < putRecordsResultEntryList.size(); i++) {
        final PutRecordsRequestEntry putRecordRequestEntry =
 putRecordsRequestEntryList.get(i);
        final PutRecordsResultEntry putRecordsResultEntry =
 putRecordsResultEntryList.get(i);
        if (putRecordsResultEntry.getErrorCode() != null) {
            failedRecordsList.add(putRecordRequestEntry);
        }
    }
    putRecordsRequestEntryList = failedRecordsList;
    putRecordsRequest.setRecords(putRecordsRequestEntryList);
    putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);
}
```

使用 PutRecord 新增單一記錄

逐次呼叫 PutRecord 對單一記錄進行操作。PutRecords所述的 使用 PutRecords 新增多筆記錄 操作方為首選，除非您的應用程式具體需要始終透過單次請求傳送單一記錄，或因其他緣故無法使用 PutRecords。

每筆資料記錄都有獨一無二的序號。此序號是由 Kinesis Data Streams 在您呼叫 client.putRecord 將資料記錄加入至串流後所指派。同一分割區索引鍵的序號通常會隨著時間而增加；逐次 PutRecord 請求的間隔期間愈長，序號將變得愈大。

快速連續進行放置操作時，不保證傳回的序號會增加，因為放置操作對 Kinesis Data Streams 基本上是同時發生。為保證同一分割區索引鍵的序號嚴格遞增，請使用 SequenceNumberForOrdering 參數，如 PutRecord 範例的程式碼範例所示。

無論您是否使用 SequenceNumberForOrdering，Kinesis Data Streams 透過 GetRecords 呼叫接收的記錄都將依照序號嚴格排序。

> **Note**
>
> 序號不能用做為同一串流中各資料集的索引。若要按照邏輯分隔資料集，請使用分割區索引鍵或為每個資料集建立個別串流。

分割區索引鍵用於將串流中的資料分組。資料記錄是根據其分割區索引鍵指派給串流中的碎片。具體而言，Kinesis Data Streams 使用分割區索引鍵做為雜湊函數的輸入，由該函數將分割區索引鍵 (和相關聯的資料) 對應到特定碎片。

經過此雜湊處理機制，具有相同分割區索引鍵的所有資料記錄會對應到串流中的同一碎片。然而，若分割區索引鍵數目多過碎片數目，某些碎片即必定包含具有不同分割區索引鍵的記錄。從設計的角度來看，為確保您的所有碎片獲得充分利用，碎片數目 (由 setShardCount 的 CreateStreamRequest 方法指定) 應遠少於獨一分割區索引鍵的數目，且流向單一分割區索引鍵的資料量應遠少於碎片容量。

PutRecord 範例

以下程式碼會建立 10 筆跨兩個分割區索引鍵分佈的資料記錄，並將其放入名為 myStreamName 的串流。

```
for (int j = 0; j < 10; j++)
{
  PutRecordRequest putRecordRequest = new PutRecordRequest();
```

```
  putRecordRequest.setStreamName( myStreamName );
  putRecordRequest.setData(ByteBuffer.wrap( String.format( "testData-%d",
 j ).getBytes() ));
  putRecordRequest.setPartitionKey( String.format( "partitionKey-%d", j/5 ));
  putRecordRequest.setSequenceNumberForOrdering( sequenceNumberOfPreviousRecord );
  PutRecordResult putRecordResult = client.putRecord( putRecordRequest );
  sequenceNumberOfPreviousRecord = putRecordResult.getSequenceNumber();
}
```

上述程式碼範例使用 `setSequenceNumberForOrdering` 保證每個分割區索引鍵內的順序嚴格遞增。為求有效使用此參數，將目前記錄 `SequenceNumberForOrdering` (記錄 n) 設為前一記錄 (記錄 n-1) 的序號。為取得已加入至串流的記錄其序號，則對 `getSequenceNumber` 的結果呼叫 `putRecord`。

`SequenceNumberForOrdering` 參數可確保嚴格遞增分割區索引鍵的序號。`SequenceNumberForOrdering` 不提供跨多個分割區索引鍵的記錄排序。

使用 AWS Glue 結構描述登錄檔與資料互動

您可以將 Kinesis 資料串流與 AWS Glue 結構描述登錄檔整合。 AWS Glue 結構描述登錄檔可讓您集中探索、控制和發展結構描述，同時確保已註冊結構描述持續驗證產生的資料。結構描述定義資料記錄的結構和格式。結構描述是可靠的資料發佈、耗用或儲存的版本化規格。 AWS Glue 結構描述登錄檔可讓您改善串流應用程式中end-to-end資料品質和資料控管。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。設定此整合的其中一個方法是透過 AWS Java SDK 中提供的 `PutRecords` 和 `PutRecord` Kinesis Data Streams API。

如需如何使用 PutRecords 和 PutRecord Kinesis Data Streams APIs 設定 Kinesis Data Streams 與結構描述登錄整合的詳細說明，請參閱使用案例：整合 Amazon Kinesis Data Streams 與 AWS Glue 結構描述登錄檔中的「使用 Kinesis Data Streams APIs 與資料互動」一節。

使用 Kinesis Agent 寫入 Amazon Kinesis Data Streams

Kinesis 代理程式是獨立的 Java 軟體應用程式，可讓您輕鬆收集資料並將資料傳送至 Kinesis Data Streams。此代理程式將持續監控一組檔案，並且傳送新資料到您的串流。代理程式會處理檔案輪換、檢查點，並在故障時重試。它以可靠、及時和簡單的方式提供所有資料。它也會發出 Amazon CloudWatch 指標，協助您更有效地監控串流程序並進行疑難排解。

根據預設，記錄會從各個檔案根據換行符號 (`'\n'`) 字元進行剖析。不過，代理程式也可以設定為剖析多行記錄 (請參閱指定代理程式組態設定)。

您可以在以 Linux 為基礎的伺服器環境安裝代理程式，例如 Web 伺服器、日誌伺服器，及資料庫伺服器。安裝代理程式後，請透過指定要監控的檔案和資料的串流以進行設定。代理程式設定妥後，其將持續從檔案收集資料並以可靠的方式傳送資料至串流。

主題

完成 Kinesis Agent 的先決條件

- 您的作業系統必須是 Amazon Linux AMI 2015.09 版或更新版本，或 Red Hat Enterprise Linux 版本 7 或更新版本。
- 如果您使用 Amazon EC2 執行您的代理程式，則請啟動您的 EC2 執行個體。
- 使用下列其中一種方法管理您的 AWS 登入資料：
  - 當您啟動 EC2 執行個體時，指定 IAM 角色。
  - 設定代理程式時指定 AWS 登入資料（請參閱 awsAccessKeyId 和 awsSecretAccessKey)。
  - 編輯 `/etc/sysconfig/aws-kinesis-agent` 以指定您的區域和 AWS 存取金鑰。
  - 如果您的 EC2 執行個體位於不同的 AWS 帳戶中，請建立 IAM 角色以提供 Kinesis Data Streams 服務的存取權，並在設定代理程式時指定該角色（請參閱 assumeRoleARN 和 assumeRoleExternalId)。使用上述其中一種方法來指定其他帳戶中具有擔任此角色許可之使用者的 AWS 登入資料。
- 您指定的 IAM 角色或 AWS 登入資料必須具有執行 Kinesis Data Streams PutRecords 操作的許可，代理程式才能將資料傳送至您的串流。若您啟用 CloudWatch 監控代理程式，則另需具備執行 CloudWatch PutMetricData 操作的許可。如需詳細資訊，請參閱 使用 IAM 控制對 Amazon Kinesis Data Streams 資源的存取、使用 Amazon CloudWatch 監控 Kinesis Data Streams Agent 運作狀態 和 CloudWatch 存取控制。

下載並安裝代理程式

首先，連接至您的執行個體。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的連線至您的執行個體。如果您無法連線，請參閱《Amazon EC2 使用者指南》中的連線至執行個體的故障診斷。

使用 Amazon Linux AMI 設定代理程式

使用以下命令來下載和安裝代理程式：

```
sudo yum install -y aws-kinesis-agent
```

使用 Red Hat Enterprise Linux 設定代理程式

使用以下命令來下載和安裝代理程式：

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-
latest.amzn2.noarch.rpm
```

使用 GitHub 設定代理程式

1. 從 awlabs/amazon-kinesis-agent 下載代理程式。
2. 瀏覽到下載目錄並執行下列命令以安裝代理程式：

   ```
   sudo ./setup --install
   ```

若要在 Docker 容器中設定代理程式

Kinesis 代理程式也可以透過 amazonlinux 容器基礎在容器中執行。使用以下 Dockerfile，然後執行 `docker build`。

```
FROM amazonlinux

RUN yum install -y aws-kinesis-agent which findutils
COPY agent.json /etc/aws-kinesis/agent.json

CMD ["start-aws-kinesis-agent"]
```

設定和啟動代理程式

設定和啟動代理程式

1. 開啟並編輯組態檔案 (如果使用預設檔案存取許可，即以超級使用者身分執行)：/etc/aws-kinesis/agent.json

   在此組態檔案中，指定代理程式從中收集資料的檔案 ("filePattern")，以及代理程式將向其傳送資料的串流名稱 ("kinesisStream")。請注意，檔案名稱是一種模式，代理程式可辨識檔案輪換。您可以輪換檔案或建立新的檔案，每秒不超過一次。代理程式利用檔案建立時間戳記以判斷要追蹤哪些檔案，然後傳送至您的串流；如果每秒建立新檔案或輪換檔案超過一次，將導致代理程式無法正確區分這些檔案。

   ```
   {
      "flows": [
           {
               "filePattern": "/tmp/app.log*",
               "kinesisStream": "yourkinesisstream"
           }
      ]
   }
   ```

2. 手動啟動代理程式：

   ```
   sudo service aws-kinesis-agent start
   ```

3. (選用) 設定代理程式在系統啟動時開始執行：

   ```
   sudo chkconfig aws-kinesis-agent on
   ```

代理程式現在已做為系統服務在背景執行。其將持續監控指定的檔案，並將資料傳送至指定的串流。代理程式的活動記錄於 /var/log/aws-kinesis-agent/aws-kinesis-agent.log。

指定代理程式組態設定

代理程式支援兩種必要的組態設定 filePattern 和 kinesisStream，以及用於其他功能的選用組態設定。您可以由 /etc/aws-kinesis/agent.json 指定必要及選用的組態。

當您變更組態檔案時，必須使用下列命令停止及啟動代理程式：

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

或者，您可以使用下列命令：

```
sudo service aws-kinesis-agent restart
```

以下是一般組態設定。

組態設定	Description
assumeRoleARN	要由使用者擔任的角色 ARN。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色跨 AWS 帳戶委派存取權。
assumeRoleExternalId	選用的識別符決定誰可以擔任此角色。如需詳細資訊，請參閱《IAM 使用者指南》中的如何使用外部 ID。
awsAccessKeyId	AWS 會覆寫預設登入資料的存取金鑰 ID。此設定優先於所有其他登入資料供應商。
awsSecretAccessKey	AWS 覆寫預設登入資料的私密金鑰。此設定優先於所有其他登入資料供應商。
cloudwatch.emitMetrics	如設定為 (true)，將啟用代理程式發出指標至 CloudWatch。 預設：true
cloudwatch.endpoint	適用於 CloudWatch 的區域端點。 預設：monitoring.us-east-1.amazonaws.com
kinesis.endpoint	適用於 Kinesis Data Streams 的區域端點。 預設：kinesis.us-east-1.amazonaws.com

以下是流程組態設定。

組態設定	Description
dataProcessingOptions	將每個剖析的記錄傳送到串流之前會套用於這些記錄的處理選項的清單。此處理選項會在指定的資料夾執行。如需詳細資訊，請參閱使用代理程式預先處理資料。
kinesisStream	[必要] 串流的名稱。
filePattern	【必要】 必須相符的目錄和檔案模式，才能由代理程式挑選。符合此模式的所有檔案皆需將讀取許可授予 aws-kinesis-agent-user 。對於包含這些檔案的目錄，必須將讀取和執行許可授予 aws-kinesis-agent-user 。
initialPosition	檔案開始進行剖析的初始位置。有效值為 START_OF_FILE 和 END_OF_FILE 。 預設：END_OF_FILE
maxBufferAgeMillis	代理程式將資料傳送到串流之前先緩衝資料的時間上限 (毫秒)。 數值範圍：1,000 到 900,000 (1 秒到 15 分鐘) 預設：60,000 (1 分鐘)
maxBufferSizeBytes	代理程式將資料傳送到串流之前先緩衝資料的容量上限 (位元組)。 數值範圍：1 到 4,194,304 (4 MB) 預設：4,194,304 (4 MB)
maxBufferSizeRecords	代理程式將資料傳送到串流之前先緩衝資料的記錄數上限。 數值範圍：1 到 500 預設：500
minTimeBetweenFilePollsMillis	代理程式輪詢和剖析檔案以找出新資料的時間間隔 (以毫秒為單位)。 數值範圍：1 或以上 預設：100

組態設定	Description
multiLine StartPattern	用於識別記錄開始處的模式。記錄是由符合模式的一列及不符合模式的任何幾列所組成。有效值為常規運算式。根據預設，每個新日誌檔中的新列會剖析為一筆記錄。
partition KeyOption	產生分割區索引鍵的方法。有效值為 RANDOM (隨機產生的整數) 和 DETERMINISTIC (根據資料計算得出的雜湊值)。 預設：RANDOM
skipHeaderLines	代理程式剖析監控檔案開頭部分時略過的列數。 數值範圍：0 或以上 預設：0 (零)
truncated RecordTer minator	記錄大小超過 Kinesis Data Streams 記錄大小限制時，代理程式將用來截斷剖析之記錄的字串。(1,000 KB) 預設：'\n' (換行符號)

監控多個檔案目錄並寫入多個串流

透過指定多個流程組態設定，您可以設定代理程式來監控多個檔案目錄，然後將資料傳送到多個串流。在下列組態範例中，代理程式會監控兩個檔案目錄，並分別將資料傳送至 Kinesis 串流和 Firehose 交付串流。請注意，您可以為 Kinesis Data Streams 和 Firehose 指定不同的端點，因此 Kinesis 串流和 Firehose 交付串流不需要位於相同的區域。

```
{
    "cloudwatch.emitMetrics": true,
    "kinesis.endpoint": "https://your/kinesis/endpoint",
    "firehose.endpoint": "https://your/firehose/endpoint",
    "flows": [
        {
            "filePattern": "/tmp/app1.log*",
            "kinesisStream": "yourkinesisstream"
        },
        {
            "filePattern": "/tmp/app2.log*",
```

```
            "deliveryStream": "yourfirehosedeliverystream"
        }
    ]
}
```

如需搭配 Firehose 使用代理程式的詳細資訊，請參閱搭配 Kinesis 代理程式寫入 Amazon Data Firehose。

使用代理程式預先處理資料

代理程式可預先處理經由受監控檔案所剖析的記錄，然後再將其傳送至您的串流。您可以將 `dataProcessingOptions` 組態設定新增到您的檔案流程以啟用此功能。可新增一或多個處理選項，這些選項將會依照指定的順序執行。

代理程式支援下列處理選項。由於代理程式是開放原始碼，您可以進一步開發和擴展其處理選項。您可以從 Kinesis 代理程式下載代理程式。

處理選項

`SINGLELINE`

藉由移除換行字元、前方空格及結尾空格，將多列記錄轉換為單列記錄。

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

將記錄從分隔符號區隔格式轉換為 JSON 格式。

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[必要] 欄位名稱在每個 JSON 鍵值對中做為鍵。例如，若您指定 `["f1", "f2"]`，記錄「v1, v2」將轉換為 `{"f1":"v1","f2":"v2"}`。

`delimiter`

在記錄做為分隔符號的字串。預設為逗號 (,)。

`LOGTOJSON`

將記錄從日誌格式轉換為 JSON 格式。支援的日誌格式為 Apache Common Log、Apache Combined Log、Apache Error Log、以及 RFC3164 Syslog。

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[必要] 日誌項目格式。以下是可能的值：

- `COMMONAPACHELOG` – Apache Common Log 格式。根據預設，每個日誌項目皆有以下模式：「`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}`」。
- `COMBINEDAPACHELOG` – Apache Combined Log 格式。根據預設，每個日誌項目皆有以下模式：「`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}`」。
- `APACHEERRORLOG` – Apache Error Log 格式。根據預設，每個日誌項目皆有以下模式：「`[%{timestamp}] [%{module}:%{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}`」。
- `SYSLOG` – RFC3164 Syslog 格式。根據預設，每個日誌項目皆有以下模式：「`%{timestamp} %{hostname} %{program}[%{processid}]: %{message}`」。

`matchPattern`

用於從日誌項目擷取值的正規運算式模式。如果您的日誌項目不屬於任一種預先定義的日誌格式，則將使用此設定。使用此設定時，您還必須指定 `customFieldNames`。

`customFieldNames`

自訂欄位名稱在每個 JSON 鍵值對中做為鍵。您可以使用此設定來定義從 `matchPattern` 擷取的值的欄位名稱，或覆寫預先定義的日誌格式的預設欄位名稱。

Example：LOGTOJSON 組態

這裡提供一個 Apache Common Log 項目轉換為 JSON 格式的 LOGTOJSON 組態範例：

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

轉換前：

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
 HTTP/1.1" 200 6291
```

轉換後：

```
{"host":"64.242.88.10","ident":null,"authuser":null,"datetime":"07/
Mar/2004:16:10:02 -0800","request":"GET /mailman/listinfo/hsdivision
 HTTP/1.1","response":"200","bytes":"6291"}
```

Example：使用自訂欄位的 LOGTOJSON 組態

以下是另一個 LOGTOJSON 組態範例：

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

使用此組態設定，前一個範例的相同 Apache Common Log 項目轉換為 JSON 格式如下：

```
{"f1":"64.242.88.10","f2":null,"f3":null,"f4":"07/Mar/2004:16:10:02 -0800","f5":"GET /
mailman/listinfo/hsdivision HTTP/1.1","f6":"200","f7":"6291"}
```

Example：轉換 Apache Common Log 項目

以下流程組態將 Apache Common Log 項目轉換為 JSON 格式的單列記錄：

```
{
    "flows": [
```

```
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
            ]
        }
    ]
}
```

Example：轉換多列記錄

以下流程組態剖析第一行從「[SEQUENCE=」開始的多列記錄。每筆記錄都會先轉換為單列記錄。然後，根據定位鍵分隔符號從記錄中擷取值。擷取的值會對應到指定的 customFieldNames 值以形成 JSON 格式的單列記錄。

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
            ]
        }
    ]
}
```

Example：使用匹配模式的 LOGTOJSON 組態

以下是 Apache Common Log 項目轉換為 JSON 格式的 LOGTOJSON 組態範例，省略最後欄位 (位元組)：

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "matchPattern": "^([\\d.]+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.
+?)\" (\\d{3})",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request",
 "response"]
}
```

轉換前：

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
 200
```

轉換後：

```
{"host":"123.45.67.89","ident":null,"authuser":null,"datetime":"27/Oct/2000:09:27:09
 -0400","request":"GET /java/javaResources.html HTTP/1.0","response":"200"}
```

使用代理程式 CLI 命令

在系統啟動時自動開始執行代理程式：

```
sudo chkconfig aws-kinesis-agent on
```

檢查代理程式的狀態：

```
sudo service aws-kinesis-agent status
```

停止代理程式：

```
sudo service aws-kinesis-agent stop
```

從這個位置讀取代理程式的日誌檔案：

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

解除安裝代理程式：

```
sudo yum remove aws-kinesis-agent
```

常見問答集

是否有適用於 Windows 的 Kinesis 代理程式？

適用於 Windows 的 Kinesis 代理程式是不同於適用於 Linux 平台的 Kinesis 代理程式的軟體。

為什麼 Kinesis 代理程式會減速和/或 **RecordSendErrors** 增加？

這通常是由於來自 Kinesis 的限流。檢查 Kinesis Data Streams 的 `WriteProvisionedThroughputExceeded` 指標或 Firehose Delivery Streams 的 `ThrottledRecords` 指標。這些指標中從 0 開始的任何增量，均表示需要提升串流限制。如需詳細資訊，請參閱 Kinesis Data Stream 限制和 Amazon Firehose 交付串流。

排除限流之後，請查看 Kinesis 代理程式是否設定為追蹤大量小型檔案。Kinesis 代理程式追蹤新檔案時會有延遲，因此 Kinesis 代理程式應追蹤少量較大的檔案。嘗試將日誌檔案合併至較大的檔案中。

為什麼我會遇到 **java.lang.OutOfMemoryError** 例外狀況？

Kinesis 代理程式沒有足夠的記憶體可以處理其目前的工作負載。嘗試增加 `/usr/bin/start-aws-kinesis-agent` 中的 `JAVA_START_HEAP` 和 `JAVA_MAX_HEAP` 並重新啟動代理程式。

為什麼我會遇到 **IllegalStateException : connection pool shut down** 例外狀況？

Kinesis 代理程式沒有足夠的連線可以處理其目前的工作負載。嘗試在位於 `/etc/aws-kinesis/agent.json` 的一般代理程式組態設定中增加 `maxConnections` 和 `maxSendingThreads`。這些欄位的預設值是可用執行期處理器的 12 倍。如需進階代理程式組態設定的詳細資訊，請參閱 AgentConfiguration.java。

如何使用 Kinesis 代理程式對另一個問題進行偵錯？

可以在 `/etc/aws-kinesis/log4j.xml` 中啟用 `DEBUG` 層級日誌。

我應該如何對 Kinesis Agent 進行設定？

`maxBufferSizeBytes` 越小，Kinesis 代理程式傳送資料的頻率就越高。這可能很好，因為這樣會減少記錄的交付時間，但也增加了 Kinesis 的每秒請求。

為什麼 Kinesis 代理程式傳送重複的日誌？

發生這種情況是由於檔案追蹤組態錯誤。請確保每個 `fileFlow's filePattern` 僅與一個檔案相符。如果正在 `copytruncate` 模式中使用 `logrotate` 模式下，也可能發生這種情況。嘗試將模式變更為預設模式，或建立模式以避免重複。如需有關處理重複記錄的詳細資訊，請參閱處理重複記錄。

使用其他服務寫入 Kinesis Data Streams AWS

下列 AWS 服務可以直接與 Amazon Kinesis Data Streams 整合，以將資料寫入 Kinesis 資料串流。檢閱您感興趣的每個服務的資訊，並參考提供的參考。

主題

- 使用 寫入 Kinesis Data Streams AWS Amplify
- 使用 Amazon Aurora 寫入 Kinesis Data Streams
- 使用 Amazon CloudFront 寫入 Kinesis Data Streams
- 使用 Amazon CloudWatch Logs 寫入 Kinesis Data Streams
- 使用 Amazon Connect 寫入 Kinesis Data Streams
- 使用 寫入 Kinesis Data Streams AWS Database Migration Service
- 使用 Amazon DynamoDB 寫入 Kinesis Data Streams
- 使用 Amazon EventBridge 寫入 Kinesis Data Streams
- 使用 寫入 Kinesis Data Streams AWS IoT Core
- 使用 Amazon Relational Database Service 寫入 Kinesis Data Streams
- usingAmazon Pinpoint 寫入 Kinesis Data Streams
- 使用 Amazon Quantum Ledger Database (Amazon QLDB) 寫入 Kinesis Data Streams

使用 寫入 Kinesis Data Streams AWS Amplify

您可以使用 Amazon Kinesis Data Streams 從使用 AWS Amplify 建置的行動應用程式串流資料，以進行即時處理。然後，您可以建立即時儀表板、擷取例外狀況並產生提醒、驅動建議，以及做出其他即時業務或營運決策。您也可以將資料傳送至其他 服務，例如 Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Redshift。

如需詳細資訊，請參閱 AWS Amplify 開發人員中心中的使用 Amazon Kinesis。

使用 Amazon Aurora 寫入 Kinesis Data Streams

您可以使用 Amazon Kinesis Data Streams 來監控 Amazon Aurora 資料庫叢集上的活動。使用資料庫活動串流，您的 Aurora 資料庫叢集可即時將活動推送至 Amazon Kinesis Data Stream。然後，可以建置應用程式以進行合規管理，以取用這些活動、進行稽核並產生提醒。您也可以使用 Amazon Firehose 來存放資料。

如需詳細資訊，請參閱《Amazon Aurora 資料庫開發人員指南》中的資料庫活動串流。

使用 Amazon CloudFront 寫入 Kinesis Data Streams

您可以使用 Amazon Kinesis Data Streams 搭配 CloudFront 即時日誌，並即時取得對分佈提出之請求的相關資訊。然後，您可以建置自己的 Kinesis 資料串流取用者，或使用 Amazon Data Firehose 將日誌資料傳送至 Amazon S3、Amazon Redshift、Amazon OpenSearch Service 或第三方日誌處理服務。

如需詳細資訊，請參閱《Amazon CloudFront 開發人員指南》中的即時日誌。

使用 Amazon CloudWatch Logs 寫入 Kinesis Data Streams

您可以使用 CloudWatch 訂閱從 Amazon CloudWatch Logs 存取日誌事件的即時摘要，並將其交付至 Kinesis 資料串流以進行處理、分析和載入至其他系統。

如需更多資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的使用訂閱即時處理日誌資料。

使用 Amazon Connect 寫入 Kinesis Data Streams

您可以使用 Kinesis Data Streams，從 Amazon Connect 執行個體即時匯出聯絡人記錄和客服人員事件。您也可以從 Amazon Connect Customer Profiles 啟用資料串流，以自動接收 Kinesis 資料串流的更新，以建立新的設定檔或變更現有的設定檔。

然後，您可以建置取用者應用程式，以即時處理和分析資料。例如，使用聯絡記錄和客戶個人檔案資料，您可以讓來源系統資料 (例如 CRM 和行銷自動化工具) 與最新資訊保持同步。使用客服人員事件資料，您可以建立顯示客服人員資訊和事件的儀表板，並觸發特定客服人員活動的自訂通知。

如需詳細資訊，請參閱 《Amazon Connect 管理員指南》中的執行個體資料串流、設定即時匯出和客服人員事件串流。

使用 寫入 Kinesis Data Streams AWS Database Migration Service

您可以使用 AWS Database Migration Service 將資料遷移至 Kinesis 資料串流。然後，可以構建取用者應用程式以即時處理資料記錄。您也可以輕鬆將資料傳送至其他 服務，例如 Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Redshift

如需詳細資訊，請參閱《AWS Database Migration Service 使用者指南》中的使用 Kinesis Data Streams。

使用 Amazon DynamoDB 寫入 Kinesis Data Streams

您可以使用 Amazon Kinesis Data Streams 來擷取 Amazon DynamoDB 的變更。Kinesis Data Streams 會擷取任何 DynamoDB 資料表中的項目層級修改，並將其複製到您選擇的 Kinesis 資料串流。您的取用者應用程式可以存取此串流，以即時檢視項目層級的變更，並在下游傳送這些變更，或根據內容採取動作。

如需詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 Kinesis Data Streams 如何與 DynamoDB 搭配運作。

使用 Amazon EventBridge 寫入 Kinesis Data Streams

使用 Kinesis Data Streams，您可以將 EventBridge 中的 AWS API 呼叫事件傳送至串流、建置取用者應用程式，以及處理大量資料。您也可以在 EventBridge 管道中使用 Kinesis Data Streams 作為目標，並在選擇性篩選和充實之後，從其中一個可用來源交付串流記錄。

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的傳送事件至 Amazon Kinesis 串流和 EventBridge 管道。

使用 寫入 Kinesis Data Streams AWS IoT Core

您可以使用 IoT AWS 規則動作，從 AWS IoT Core 中的 MQTT 訊息即時寫入資料。然後，您可以建置處理資料、分析其內容並產生提醒的應用程式，並將其傳遞至分析應用程式或其他 AWS 服務，

如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 Kinesis Data Streams。

使用 Amazon Relational Database Service 寫入 Kinesis Data Streams

您可以使用 Amazon Kinesis Data Streams 來監控 Amazon RDS 執行個體上的活動。使用資料庫活動串流，Amazon RDS 會即時將活動推送至 Kinesis 資料串流。然後，可以建置應用程式以進行合規管理，以取用這些活動、進行稽核並產生提醒。您也可以使用 Amazon Data Firehose 來存放資料。

如需詳細資訊，請參閱《Amazon RDS 開發人員指南》中的資料庫活動串流。

usingAmazon Pinpoint 寫入 Kinesis Data Streams

您可以將 Amazon Pinpoint 設定為將事件資料傳送至 Amazon Kinesis Data Streams。Amazon Pinpoint 可以傳送用於行銷活動、旅程、交易電子郵件和簡訊的事件資料。然後，您可以將資料擷取至分析應用程式中，或建置自己的取用者應用程式，這些應用程式會根據事件的內容採取動作。

如需詳細資訊，請參閱 《Amazon Pinpoint 開發人員指南》中的串流事件。

使用 Amazon Quantum Ledger Database (Amazon QLDB) 寫入 Kinesis Data Streams

您可以在 Amazon QLDB 中建立串流，擷取遞交至日誌的每個文件修訂，並即時將此資料交付至 Amazon Kinesis Data Streams。QLDB 串流是從總帳日誌傳送到 Kinesis 資料串流資源的連續資料流。然後，可以使用 Kinesis 串流平台或 Kinesis Client Library 來使用串流、處理資料記錄以及分析資料內容。QLDB 串流會以三種記錄類型將您的資料寫入 Kinesis Data Streams：`control`、`block summary` 和 `revision details`。

如需詳細資訊，請參閱《Amazon QLDB 開發人員指南》中的串流。

使用第三方整合寫入 Kinesis Data Streams

您可以使用與 Kinesis Data Streams 整合的下列其中一個第三方選項，將資料寫入 Kinesis Data Streams。選取您想要進一步了解的選項，並尋找相關文件的資源和連結。

主題

- Apache Flink
- Fluentd
- Debezium
- Oracle GoldenGate
- Kafka Connect
- Adobe 體驗
- Striim

Apache Flink

Apache Flink 是一個架構和分散式處理引擎，用於對未限制和有限制資料串流進行狀態運算。如需從 Apache Flink 寫入 Kinesis Data Streams 的詳細資訊，請參閱 Amazon Kinesis Data Streams 連接器。

Fluentd

Fluentd 是用於統一日誌記錄層的開放原始碼資料收集器。如需從 Fluentd 寫入 Kinesis Data Streams 的詳細資訊。如需詳細資訊，請參閱使用 Kinesis 處理串流。

Debezium

Debezium 是一個用於變更資料擷取的開放原始碼分散式平台。如需從 Debezium 寫入 Kinesis Data Streams 的詳細資訊，請參閱將 MySQL 資料變更串流至 Amazon Kinesis。

Oracle GoldenGate

Oracle GoldenGate 是一種軟體產品，可讓您複製、篩選，以及將資料從一個資料庫轉換到另一個資料庫。如需從 Oracle GoldenGate 寫入 Kinesis Data Streams 的詳細資訊，請參閱使用 Oracle GoldenGate 將資料複寫至 Kinesis 資料串流。

Kafka Connect

Kafka Connect 是在 Apache Kafka 和其他系統之間以可擴展和可靠方式串流資料的工具。如需將資料從 Apache Kafka 寫入 Kinesis Data Streams 的詳細資訊，請參閱 Kinesis Kafka 連接器。

Adobe 體驗

Adobe 體驗平台使組織能夠集中和標準化來自任何系統的客戶資料。然後，它會套用資料科學和機器學習，大幅改善豐富、個人化體驗的設計和交付。如需將資料從 Adobe 體驗平台寫入 Kinesis Data Streams 的詳細資訊。請參閱如何建立 Amazon Kinesis 連線。

Striim

Striim 是一個完整的端對端記憶體平台，用於即時收集、篩選、轉換、富集、彙總、分析和交付資料。如需如何從 Striim 將資料寫入 Kinesis Data Streams 的詳細資訊，請參閱 Kinesis 寫入器。

Amazon Kinesis Data Streams 生產者疑難排解

下列主題提供 Amazon Kinesis Data Streams 生產者常見問題的解決方案：

- 我的生產者應用程式寫入的速度比預期慢
- 我收到未經授權的 KMS 主金鑰許可錯誤
- 對生產者的其他常見問題進行故障診斷

我的生產者應用程式寫入的速度比預期慢

寫入輸送量比預期慢的最常見原因是：

- 超過服務限制
- 我想要最佳化我的生產者
- 操作濫用 flushSync()

超過服務限制

若要查明是否超出服務限制，請檢查您的生產者是否由服務擲回了傳輸量例外狀況，並查驗有哪些 API 操作受到調節。請切記，視呼叫而定將有不同的限制，具體如 配額和限制所述。例如，除了讀寫操作眾所周知的碎片層級限制外，另有以下的串流層級限制：

- CreateStream
- DeleteStream
- ListStreams
- GetShardIterator
- MergeShards
- DescribeStream
- DescribeStreamSummary

`CreateStream`、`DeleteStream`、`ListStreams`、`GetShardIterator` 和 `MergeShards` 操作的限制為每秒 5 次呼叫。`DescribeStream` 操作的限制為每秒 10 次呼叫。`DescribeStreamSummary` 操作的限制為每秒 20 次呼叫。

如果上述呼叫不存在問題，請確定您選取了能夠對所有碎片均勻地分佈 put 操作的分割區索引鍵，而且並無任何特定分割區索引鍵不慎達到了服務限制但其餘則未達到限制。對此，您將需要測量尖峰傳輸量並考量到串流中的碎片數目。如需如何管理串流的詳細資訊，請參閱建立和管理 Kinesis 資料串流。

> **Tip**
>
> 請記得，使用單一記錄的 PutRecord 操作時，輸送量限流計算要無條件進位至最接近的 KB 數，而多筆記錄的 PutRecords 操作需對每次呼叫累計的記錄總和進行捨入。例如，放入 600 筆記錄共 1.1 KB 大小的 PutRecords 請求將不會受到調節。

我想要最佳化我的生產者

開始最佳化生產者之前，請先完成下列關鍵任務。首先，根據記錄大小和每秒記錄筆數，確認所需的尖峰傳輸量。接著，排除串流容量為限制因素 (超過服務限制) 的可能性。如果您排除了串流容量，則針對以下兩種常見類型的生產者使用相應的故障診斷技巧及最佳化準則。

大型生產者

大型生產者通常是從內部部署伺服器或 Amazon EC2 執行個體執行。需要由大型生產者提供較高傳輸量的消費者通常會在意每一記錄延遲。處理延遲的策略包括：如果客戶可以微批次/緩衝記錄，在使用單一記錄操作 PutRecord 之前，請使用 Amazon Kinesis Producer Library （具有進階彙總邏輯）、多記錄操作 PutRecords PutRecord或將記錄彙總到較大的檔案中。 PutRecords 如果您無法批次處理/緩衝，請使用多個執行緒同時寫入 Kinesis Data Streams 服務。 適用於 Java 的 AWS SDK 和其他 SDKs 包含可在極少程式碼下執行此操作的非同步用戶端。

小型生產者

小型生產者通常是行動應用程式、IoT 裝置或 web 用戶端。如果是行動應用程式，建議您在 AWS Mobile SDKs 中使用 PutRecords操作或 Kinesis Recorder。如需詳細資訊，請參閱 適用於 Android 的 AWS Mobile SDK 入門指南和 AWS Mobile SDK for iOS 入門指南。行動應用程式必須處理自身固有的斷續連線問題，而且需要某一類的批次 put 操作如 PutRecords。若您由於某些因素無法批次處理，請參閱上述「大型生產者」一節的資訊。如果您的生產者是瀏覽器，產生的資料量通常很少。不過，這樣是將 put 操作放在了應用程式的重要路徑上，而此做法並不建議。

操作濫用 `flushSync()`

`flushSync()` 不正確地使用 可能會大幅影響寫入效能。`flushSync()` 操作專為關閉案例而設計，以確保在 KPL 應用程式終止之前傳送所有緩衝記錄。如果您在每次寫入操作後實作此操作，可能會增

加大量額外延遲，每次寫入大約 500 毫秒。請確定您已`flushSync()`針對應用程式關閉實作 ，以避免寫入效能不必要的額外延遲。

我收到未經授權的 KMS 主金鑰許可錯誤

若生產者應用程式寫入已加密的串流但未具備 KMS 主金鑰的許可，便會發生此錯誤。若要為應用程式指派許可使其能夠存取 KMS 金鑰，請參閱在 AWS KMS 中使用金鑰政策及搭配 AWS KMS 使用 IAM 政策。

對生產者的其他常見問題進行故障診斷

- 為何我的 Kinesis 資料串流會傳回 500 個內部伺服器錯誤？
- 如何對從 Flink 寫入 Kinesis Data Streams 時發生的逾時錯誤進行故障診斷？
- 如何疑難排解 Kinesis Data Streams 中的限流錯誤？
- 為何我的 Kinesis 資料串流會限流？
- 如何使用 KPL 將資料記錄放入 Kinesis 資料串流中？

最佳化 Kinesis Data Streams 生產者

您可以根據您看到的特定行為，進一步最佳化 Amazon Kinesis Data Streams 生產者。檢閱下列主題以識別解決方案。

主題

- 自訂 KPL 重試和速率限制行為
- 將最佳實務套用至 KPL 彙總

自訂 KPL 重試和速率限制行為

當您使用 KPL 操作新增 Amazon Kinesis Producer Library (KPL) 使用者記錄時，系統會為記錄提供時間戳記，並新增至具有 `RecordMaxBufferedTime`組態參數所設定截止日期的緩衝區。`addUserRecord()`此時間戳記/截止日期的組合設定了緩衝區優先順序。記錄將根據下列條件從緩衝區排清：

- 緩衝區優先順序
- 彙整組態
- 收集組態

影響緩衝區行為的彙整和收集組態參數如下：

- `AggregationMaxCount`
- `AggregationMaxSize`
- `CollectionMaxCount`
- `CollectionMaxSize`

然後，清除的記錄會以 Amazon Kinesis Data Streams 記錄的形式，使用對 Kinesis Data Streams API 操作 `PutRecords` 的呼叫來傳送至 Kinesis 資料串流。`PutRecords` 操作向串流傳送的請求偶爾會完全失敗或局部失敗。失敗的記錄會自動加回到 KPL 緩衝區。將根據以下兩個值當中較小者設定新的截止日期：

- 目前 `RecordMaxBufferedTime` 組態減半
- 記錄的存留時間值

此策略使重試的 KPL 使用者記錄得以納入後續 Kinesis Data Streams API 呼叫中，既強制實施了 Kinesis Data Streams 記錄的存留時間值又能提高輸送量並降低複雜性。其間不涉及退避演算法，使得此策略成為相對積極的重試策略。因重試次數過多造成的垃圾郵件會受到速率限制的阻止，相關內容將於下一節談論。

速率限制

KPL 包含速率限制功能，可限制從單一生產者傳送的每個碎片輸送量。速率限制是使用字符儲存貯體演算法搭配用於 Kinesis Data Streams 記錄和位元組的單獨儲存貯體進行實作。每次成功寫入 Kinesis 資料串流都會將一個字符 (或多個字符) 加入各儲存貯體，最多達到特定閾值。此閾值可供設定，但預設情況下設定的值將比實際碎片限額高出 50%，使得來自單一生產者的碎片能夠達到飽和。

您可降低此限制以減少因重試次數過多造成的垃圾郵件。然而，最佳實務是由每個生產者主動重試最大傳輸量，透過擴展串流容量並實作相應的分割區索引鍵策略來處理判定為過多的任何形成調節情況。

將最佳實務套用至 KPL 彙總

雖然產生的 Amazon Kinesis Data Streams 記錄序號方案保持不變， 彙總會導致彙總 Kinesis Data Streams 記錄中包含的 Amazon Kinesis Producer Library (KPL) 使用者記錄索引從 0 （零） 開始；不過， 只要您不依賴序號來唯一識別您的 KPL 使用者記錄， 您的程式碼可以忽略這一點， 作為彙總 (KPL 使用者記錄的 Kinesis Data Streams 記錄） 和後續的取消彙總 (Kinesis Data Streams 記錄的 Kinesis Data Streams 記錄的 KPL 使用者記錄） 會自動為您處理此問題。無論您的消費者使用的

是 KCL 或 AWS SDK，這都適用。若要使用此彙總功能，如果您的取用者是使用 AWS SDK 中提供的 API 進行寫入，則需要將 KPL 的 Java 部分提取到組建中。

若您打算使用序號做為 KPL 使用者記錄的唯一識別符，建議您使用 `Record` 和 `UserRecord` 所提供遵守合約的 `public int hashCode()` 及 `public boolean equals(Object obj)` 操作，對您的 KPL 使用者記錄進行比較。此外，如果想要檢查 KPL 使用者記錄的子序號，您可以將其轉換為 `UserRecord` 執行個體並擷取其子序號。

如需詳細資訊，請參閱實作消費者取消彙總。

從 Amazon Kinesis Data Streams 讀取資料

取用者是處理來自 Kinesis 資料串流的所有資料的應用程式。當消費者使用強化廣發功能時，將取得其自身每秒 2 MB 的讀取傳輸量配額，使得多個消費者能夠並行從同一串流讀取資料，而不必與其他消費者爭用讀取傳輸量。若要使用碎片的強化廣發功能，請參閱開發具有專用輸送量的增強型廣發消費者。

您可以使用 Kinesis Client Library (KCL) 或 為 Kinesis Data Streams 建置取用者 適用於 Java 的 AWS SDK。您也可以使用其他服務開發消費者 AWS Lambda， AWS 例如 Amazon Managed Service for Apache Flink 和 Amazon Data Firehose。Kinesis Data Streams 支援與其他 AWS 服務整合，例如 Amazon EMR、Amazon EventBridge、 AWS Glue和 Amazon Redshift。它還支援第三方整合，包括 Apache Flink、Adobe Experience Platform、Apache Druid、Apache Spark、Databricks、Confluent Platform、Kinesumer 和 Talend。

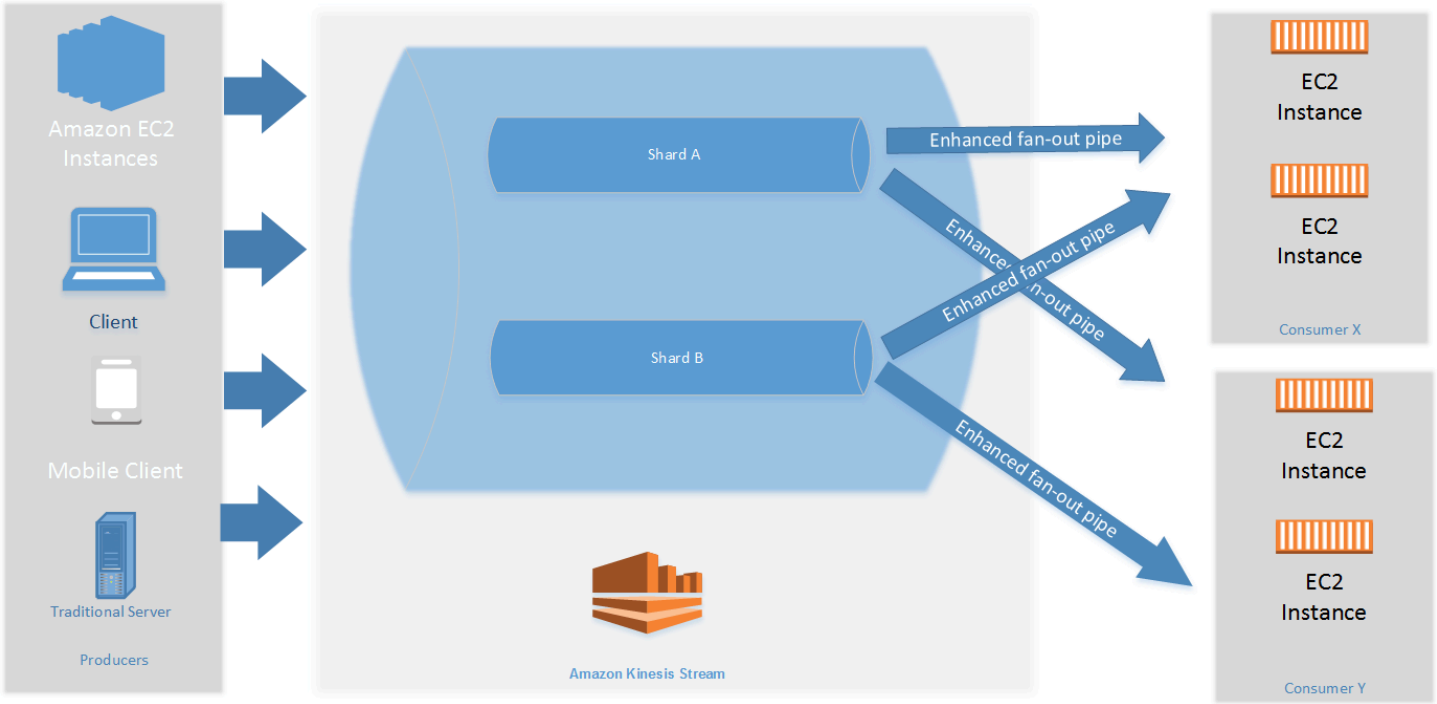
主題

開發具有專用輸送量的增強型廣發消費者

在 Amazon Kinesis Data Streams 中，您可以建置使用所謂增強型散發功能的取用者。此功能可讓消費者接收來自串流的記錄，每個碎片每秒傳輸量最多 2 MB 的資料。此輸送量係屬專用，也就是說使用增強型散發功能的取用者不必與其他從串流接收資料的取用者競爭。Kinesis Data Streams 會將資料記錄從串流推送至使用增強型散發功能的取用者。因此，這類消費者無須輪詢資料。

> **Important**
>
> 使用隨需優勢模式，每個串流最多可註冊 50 位取用者，以使用增強廣發功能。使用隨需標準和佈建串流，每個串流最多可註冊 20 位消費者，以使用增強型廣發功能。

下圖顯示強化廣發功能的架構。若您使用 2.0 版或更新版本的 Amazon Kinesis Client Library (KCL) 建置取用者，KCL 會將取用者設定成使用增強型散發功能從串流中的所有碎片接收資料。如果您是使用 API 建置使用強化廣發功能的消費者，則可以訂閱個別的碎片。



上圖顯示以下項目：

- 具有兩個碎片的串流。
- 使用強化廣發功能從串流接收資料的兩個消費者：消費者 X 和消費者 Y。兩個消費者均已訂閱串流中的所有碎片和所有記錄。若您使用 2.0 版或更新版本的 KCL 建置取用者，KCL 將自動為取用者訂閱串流中的所有碎片。另一方面，如果您是使用 API 建置消費者，則可以訂閱個別的碎片。
- 箭頭代表消費者用於從串流接收資料的強化廣發功能管道。強化廣發功能管道提供每個碎片每秒高達 2 MB 的資料，獨立於任何其他管道且與消費者總數無關。

主題

- 共用輸送量消費者與增強廣發消費者之間的差異

- 支援最多 50 個增強型廣發消費者的區域（僅限隨需優勢）
- 使用 AWS CLI 或 APIs 管理增強型廣發消費者

共用輸送量消費者與增強廣發消費者之間的差異

下表將預設共用輸送量消費者與增強型廣發消費者進行比較。訊息傳播延遲定義為使用承載分派 APIs（例如 PutRecord和 PutRecords) 傳送的承載，透過承載消耗 APIs（例如 GetRecords和）到達取用者應用程式所需的時間，以毫秒為單位SubscribeToShard。

此資料表比較共用輸送量消費者與增強廣發消費者

特性	沒有增強廣發功能的共用輸送量消費者	增強廣發消費者
讀取輸送量	固定為每個碎片每秒總共 2 MB。若有多個消費者從同一碎片讀取，所有這些消費者將共用此傳輸量。消費者每秒從碎片接收的傳輸量總和不超過 2 MB。	隨著消費者註冊使用強化廣發功能而擴展。已註冊使用強化廣發功能的各消費者均接收其自身的每個碎片讀取傳輸量，最多每秒 2 MB，獨立於其他消費者。
訊息傳播延遲	若您有一個來自串流的消費者讀數，平均約為 200 ms。若您有五個消費者，則此平均數會上升至約 1000 ms。	一般而言，無論您有一個或五個消費者，平均為 70 ms。
Cost	不適用	分為資料擷取成本和消費者碎片小時成本。如需詳細資訊，請參閱 Amazon Kinesis Data Streams 定價。
記錄交付模式	使用 GetRecords 透過 HTTP 提取模型。	Kinesis Data Streams 使用 SubscribeToShard 透過 HTTP/2 推送記錄給您。

支援最多 50 個增強型廣發消費者的區域（僅限隨需優勢）

在隨需優勢模式下，最多支援 50 個增強型廣發消費者，僅適用於下列 AWS 區域：

AWS 區域	區域名稱
eu-north-1	歐洲 (斯德哥爾摩)
me-south-1	Middle East (Bahrain)
ap-south-1	亞太地區 (孟買)
eu-west-3	Europe (Paris)
ap-southeast-3	亞太地區 (雅加達)
us-east-2	美國東部 (俄亥俄)
af-south-1	非洲 (開普敦)
eu-west-1	歐洲 (愛爾蘭)
me-central-1	中東 (阿拉伯聯合大公國)
eu-central-1	歐洲 (法蘭克福)
sa-east-1	南美洲 (聖保羅)
ap-east-1	亞太地區 (香港)
ap-south-2	亞太地區 (海德拉巴)
us-east-1	美國東部 (維吉尼亞北部)
ap-northeast-2	亞太地區 (首爾)
ap-northeast-3	亞太地區 (大阪)
eu-west-2	歐洲 (倫敦)
ap-southeast-4	亞太地區 (墨爾本)
ap-northeast-1	亞太地區 (東京)
us-west-2	美國西部 (奧勒岡)

AWS 區域	區域名稱
us-west-1	美國西部 (加利佛尼亞北部)
ap-southeast-1	亞太地區 (新加坡)
ap-southeast-2	亞太地區 (悉尼)
il-central-1	以色列 (特拉維夫)
ca-central-1	加拿大 (中部)
ca-west-1	加拿大西部 (卡加利)
eu-south-2	歐洲 (西班牙)
cn-northwest-1	中國 (寧夏)
eu-central-2	歐洲 (蘇黎世)
us-gov-east-1	AWS GovCloud （美國東部）
us-gov-west-1	AWS GovCloud （美國西部）

使用 AWS CLI 或 APIs 管理增強型廣發消費者

在 Amazon Kinesis Data Streams 中使用增強型散發功能的取用者從資料串流接收記錄時，專用輸送量可高達每個碎片每秒 2 MB 的資料。如需詳細資訊，請參閱開發具有專用輸送量的增強型廣發消費者。

您可以使用 AWS CLI 或 Kinesis Data Streams APIs 來註冊、描述、列出和取消註冊在 Kinesis Data Streams 中使用增強廣發功能的消費者。

使用 管理消費者 AWS CLI

您可以使用 註冊、描述、列出和取消註冊增強型廣發消費者 AWS CLI。如需範例，請參閱下列文件。

register-stream-consumer

註冊 Kinesis 資料串流的取用者。您可以在註冊消費者時套用標籤。

describe-stream-consumer

取得已註冊消費者的描述，其中包含消費者 ARN 或消費者名稱和串流 ARN 組合。

list-stream-consumers

列出使用增強廣發功能從串流接收資料的註冊消費者。

deregister-stream-consumer

使用消費者 ARN 或消費者名稱和串流 ARN 組合取消註冊消費者。

使用 Kinesis Data Streams APIs 管理消費者

您可以使用 Kinesis Data Streams APIs 註冊、描述、列出和取消註冊增強型廣發消費者。如需範例，請參閱下列文件。

RegisterStreamConsumer

使用標籤註冊 Kinesis 資料串流的取用者。您可以在註冊消費者時套用標籤。

DescribeStreamConsumer

取得已註冊消費者的描述，其中包含消費者 ARN 或消費者名稱和串流 ARN 組合。

ListStreamConsumers

列出使用增強廣發功能從串流接收資料的註冊消費者。

DeregisterStreamConsumer

使用消費者 ARN 或消費者名稱和串流 ARN 組合取消註冊消費者。

標記消費者

您可以將自己的中繼資料指派給在 Kinesis Data Streams 中以標籤形式建立的串流和增強廣發消費者。您可以使用標籤來分類和追蹤消費者的成本。您也可以使用具有屬性型存取控制 (ABAC) 的標籤來控制消費者的存取。如需詳細資訊，請參閱標記您的 Amazon Kinesis Data Streams 資源。

在 Kinesis 主控台中使用資料檢視器

Kinesis 管理主控台中的資料檢視器可讓您檢視資料串流指定碎片內的資料記錄，而不必開發取用者應用程式。若要使用資料檢視器，請遵循下列步驟：

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/kinesis 的 Kinesis 主控台。
2. 選擇您要使用「資料檢視器」檢視其記錄的作用中資料串流，然後選擇資料檢視器索引標籤。
3. 在所選作用中資料串流的資料檢視器索引標籤中，選擇要檢視其記錄的碎片，選擇起始位置，然後按一下取得記錄。您可以將起始位置設定為下列其中一個值：

   - 依序號碼：顯示由序號欄位中指定的序號所表示之位置的記錄。
   - 序號後：顯示由序號欄位中指定的序號所表示之位置之後的記錄。
   - 時間戳記：顯示時間戳記欄位中指定的時間戳記所表示的位置記錄。
   - 水平修剪：顯示碎片中最後一個未修剪記錄的記錄，這是碎片中最舊的資料記錄。
   - 最新：在碎片中最近記錄之後顯示記錄，以便您始終讀取碎片中的最新資料。

   然後，與指定的碎片 ID 和起始位置匹配的生成的資料記錄，顯示在控制台的記錄表中。一次最多可顯示 50 筆記錄。若要檢視下一組記錄，請按一下下一步按鈕。
4. 按一下任何個別記錄，即可在個別視窗中以原始資料或 JSON 格式檢視該記錄承載。

請注意，當您按一下資料檢視器中的取得記錄或下一步按鈕時，會呼叫 GetRecords API，這會套用至每秒 5 筆交易的 GetRecords API 限制。

在 Kinesis 主控台中查詢您的資料串流

Kinesis Data Streams 主控台中的 Data Analytics 索引標籤可讓您使用 SQL 查詢資料串流。若要使用此功能，請依照下列步驟進行：

1. 登入 AWS 管理主控台 並開啟位於 https：//https://console.aws.amazon.com/kinesis 的 Kinesis 主控台。
2. 選擇您要使用 SQL 查詢的作用中資料串流，然後選擇資料分析索引標籤。
3. 在資料分析索引標籤中，您可以使用 Managed Apache Flink Studio 筆記本執行串流檢查和視覺化。您可以使用 Apache Zeppelin 執行臨機操作 SQL 查詢，以檢查資料串流並在幾秒鐘內檢視結果。在資料分析索引標籤中，選擇我同意，然後選擇建立筆記本來建立筆記本。
4. 建立筆記本後，選擇在 Apache Zeppelin 中開啟。這將在新的索引標籤中開啟您的筆記本。筆記本是一種互動式界面，您可以在其中提交 SQL 查詢。選擇包含串流名稱的備註。
5. 您將看到一個備註，其中包含範例`SELECT`查詢，以輸出已在執行的串流中的資料。這可讓您檢視資料串流的結構描述。

6. 若要嘗試其他查詢，例如輪轉或滑動視窗，請在資料分析索引標籤中選擇檢視範例查詢。複製查詢，修改它以符合您的資料串流結構描述，然後在 Zeppelin 備註的新段落中執行它。

使用 Kinesis 用戶端程式庫

什麼是 Kinesis Client Library？

Kinesis Client Library (KCL) 是獨立的 Java 軟體程式庫，旨在簡化從 Amazon Kinesis Data Streams 取用和處理資料的程序。KCL 會處理許多與分散式運算相關的複雜任務，讓開發人員專注於實作其商業邏輯以處理資料。它會管理各種活動，例如跨多個工作者的負載平衡、回應工作者故障、檢查點處理過的記錄，以及回應串流中碎片數量的變更。

KCL 經常更新，以納入基礎程式庫的較新版本、安全性改善和錯誤修正。我們建議您使用最新版本的 KCL，以避免已知問題並從所有最新的改進中獲益。若要尋找最新的 KCL 版本，請參閱 KCL Github。

> **Important**
>
> - 我們建議您使用最新的 KCL 版本，以避免已知的錯誤和問題。如果您使用的是 KCL 2.6.0 或更早版本，請升級至 KCL 2.6.1 或更新版本，以避免在串流容量變更時封鎖碎片處理的罕見情況。
> - KCL 是 Java 程式庫。使用名為 MultiLangDaemon 的 Java 型協助程式來支援 Java 以外的語言。MultiLangDaemon 透過 STDIN 和 STDOUT 與 KCL 應用程式互動。如需 GitHub 上 MultiLangDaemon 的詳細資訊，請參閱 使用非 Java 語言開發具有 KCL 的消費者。
> - 請勿將 2.27.19 到 2.27.23 適用於 Java 的 AWS SDK 版與 KCL 3.x 搭配使用。這些版本包含導致與 KCL DynamoDB 用量相關的例外狀況錯誤的問題。我們建議您使用 2 適用於 Java 的 AWS SDK .28.0 版或更新版本，以避免此問題。

KCL 主要功能和優點

以下是 KCL 的主要功能和相關優點：

- 可擴展性：KCL 可將處理負載分散到多個工作者，讓應用程式能夠動態擴展。您可以手動或使用自動擴展來擴展應用程式，而無需擔心負載重新分佈。
- 負載平衡：KCL 會自動平衡可用工作者的處理負載，進而讓工作者的工作分佈均勻。

- 檢查點：KCL 管理已處理記錄的檢查點，讓應用程式能夠從上次成功處理的位置繼續處理。
- 容錯能力：KCL 提供內建容錯能力機制，確保即使個別工作者失敗，資料處理仍會繼續。KCL 也提供at-least-once的交付。
- 處理串流層級變更：KCL 會適應因資料磁碟區變更而可能發生的碎片分割和合併。它透過確保子碎片僅在其父碎片完成和檢查點之後才處理來維持排序。
- 監控：KCL 與 Amazon CloudWatch 整合，以進行消費者層級監控。
- 多語言支援：KCL 原生支援 Java，並透過 MultiLangDaemon 啟用多種非 Java 程式設計語言。

KCL 概念

本節說明 Kinesis Client Library (KCL) 的核心概念和互動。這些概念是開發和管理 KCL 消費者應用程式的基礎。

- KCL 取用者應用程式 – 自訂建置的應用程式，旨在使用 Kinesis Client Library 從 Kinesis 資料串流讀取和處理記錄。
- 工作者 – KCL 取用者應用程式通常會分散，並同時執行一或多個工作者。KCL 會協調工作者以分散式方式使用來自串流的資料，並在多個工作者之間平均平衡負載。
- 排程器 – KCL 工作者用來開始處理資料的高階類別。每個 KCL 工作者都有一個排程器。排程器會初始化和監督各種任務，包括從 Kinesis 資料串流同步碎片資訊、追蹤工作者之間的碎片指派，以及根據指派給工作者的碎片處理串流中的資料。排程器可以採用各種會影響排程器行為的組態，例如要處理 和 AWS 憑證的串流名稱。排程器會啟動將資料記錄從串流交付至記錄處理器。
- 記錄處理器 – 定義 KCL 取用者應用程式如何處理從資料串流接收之資料的邏輯。您必須在記錄處理器中實作自己的自訂資料處理邏輯。KCL 工作者會執行個體化排程器。排程器接著會針對其持有租用的每個碎片，執行個體化一個記錄處理器。工作者可以執行多個記錄處理器。
- 租用 – 定義工作者與碎片之間的指派。KCL 取用者應用程式使用租用將資料記錄處理分散到多個工作者。每個碎片在任何指定時間只能透過租用繫結至一個工作者，而且每個工作者可以同時保留一或多個租用。當工作者因為停止或失敗而停止保留租用時，KCL 會指派另一個工作者來接受租用。若要進一步了解租用，請參閱 Github 文件：租用生命週期。
- 租用資料表 – 是唯一的 Amazon DynamoDB 資料表，用於追蹤 KCL 取用者應用程式的所有租用。每個 KCL 取用者應用程式都會建立自己的租用資料表。租用資料表用於維護所有工作者的狀態，以協調資料處理。如需詳細資訊，請參閱KCL 中的 DynamoDB 中繼資料表和負載平衡。
- 檢查點 – 是將上次成功處理記錄的位置持續儲存在碎片中的程序。KCL 會管理檢查點，以確保在工作者失敗或應用程式重新啟動時，可以從最後一個檢查點位置繼續處理。檢查點會存放在 DynamoDB 租用資料表中，做為租用中繼資料的一部分。這可讓工作者從上一個工作者停止的位置繼續處理。

KCL 中的 DynamoDB 中繼資料表和負載平衡

KCL 會管理工作者的中繼資料，例如租用和 CPU 使用率指標。KCL 會使用 DynamoDB 資料表追蹤這些中繼資料。對於每個 Amazon Kinesis Data Streams 應用程式，KCL 會建立三個 DynamoDB 資料表來管理中繼資料：租用資料表、工作者指標資料表和協調器狀態資料表。

> **Note**
>
> KCL 3.x 推出了兩個新的中繼資料表：工作者指標和協調器狀態表。

> **Important**
>
> 您必須為 KCL 應用程式新增適當的許可，才能在 DynamoDB 中建立和管理中繼資料表。如需詳細資訊，請參閱KCL 取用者應用程式所需的 IAM 許可。
> KCL 取用者應用程式不會自動移除這三個 DynamoDB 中繼資料資料表。當您停用取用者應用程式時，請務必移除 KCL 取用者應用程式建立的這些 DynamoDB 中繼資料表，以避免不必要的成本。

租用資料表

租用資料表是唯一的 Amazon DynamoDB 資料表，用於追蹤 KCL 取用者應用程式的排程器所租用和處理的碎片。每個 KCL 取用者應用程式都會建立自己的租用資料表。根據預設，KCL 會使用取用者應用程式的名稱做為租用資料表的名稱。您可以使用組態設定自訂資料表名稱。KCL 也會使用 leaseOwner 的分割區索引鍵在租用資料表上建立全域次要索引，以實現高效的租用探索。全域次要索引會從基本租用資料表鏡像 leaseKey 屬性。如果應用程式啟動時 KCL 取用者應用程式的租用資料表不存在，則其中一個工作者會為您的應用程式建立租用資料表。

您可以在取用者應用程式執行時使用 Amazon DynamoDB 主控台檢視其租用資料表。

> **Important**
>
> - 每個 KCL 取用者應用程式名稱都必須是唯一的，以防止重複的租用資料表名稱。
> - 您的帳戶除須支付 Kinesis Data Streams 本身的相關費用外，另將收取與 DynamoDB 資料表關聯的費用。

租用資料表中的每一列都代表您取用者應用程式的排程器正在處理的碎片。金鑰欄位包括下列項目：

- leaseKey：對於單一串流處理，這是碎片 ID。對於使用 KCL 的多串流處理，其結構為 `account-id:StreamName:streamCreationTimestamp:ShardId`。 leaseKey 是租用資料表的分割區索引鍵。如需多串流處理的詳細資訊，請參閱 使用 KCL 進行多串流處理。
- checkpoint：碎片的最新檢查點序號。
- checkpointSubSequenceNumber：使用 Kinesis Producer Library 的彙整功能時，此為 checkpoint 的延伸，將追蹤 Kinesis 記錄內的個別使用者記錄。
- leaseCounter：用於檢查工作者目前是否正在主動處理租用。如果將租用所有權轉移給其他工作者，則 leaseCounter 會增加。
- leaseOwner：目前持有此租用的工作者。
- ownerSwitchesSinceCheckpoint：自上次檢查點以來，此租用變更工作者的次數。
- parentShardId：此碎片父項的 ID。在子碎片上開始處理之前，請確定父碎片已完全處理，並維持正確的記錄處理順序。
- childShardId：此碎片分割或合併所產生的子碎片 IDs 清單。用於在重新分片操作期間追蹤碎片歷程和管理處理順序。
- startingHashKey：此碎片的雜湊金鑰範圍下限。
- endingHashKey：此碎片的雜湊金鑰範圍上限。

如果您搭配 KCL 使用多串流處理，您會在租用資料表中看到下列兩個額外欄位。如需詳細資訊，請參閱使用 KCL 進行多串流處理。

- shardID：碎片的 ID。
- streamName：資料串流的識別符，格式如下：`account-id:StreamName:streamCreationTimestamp`。

工作者指標資料表

工作者指標資料表是每個 KCL 應用程式的唯一 Amazon DynamoDB 資料表，用於記錄每個工作者的 CPU 使用率指標。KCL 將使用這些指標來執行有效的租用指派，以產生工作者之間的資源使用率平衡。KCL 預設會將 `KCLApplicationName-WorkerMetricStats`用於工作者指標表的名稱。

協調器狀態資料表

協調器狀態資料表是每個 KCL 應用程式的唯一 Amazon DynamoDB 資料表，用於存放工作者的內部狀態資訊。例如，協調器狀態資料表會儲存有關領導者選擇的資料，或與 KCL 2.x 就地遷移至 KCL 3.x 相關聯的中繼資料。根據預設，KCL 會使用 `KCLApplicationName-CoordinatorState`做為協調器狀態資料表的名稱。

KCL 所建立中繼資料資料表的 DynamoDB 容量模式

根據預設，Kinesis Client Library (KCL) 會使用隨需容量模式建立 DynamoDB 中繼資料表，例如租用表、工作者指標表和協調器狀態表。此模式會自動擴展讀取和寫入容量，以容納流量，而不需要容量規劃。我們強烈建議您將容量模式保留為隨需模式，以便更有效率地操作這些中繼資料表。

如果您決定將租用資料表切換為佈建容量模式，請遵循下列最佳實務：

- 分析用量模式：
  - 使用 Amazon CloudWatch 指標監控應用程式的讀取和寫入模式和用量 (RCU、WCU)。
  - 了解尖峰和平均輸送量需求。
- 計算所需的容量：
  - 根據您的分析估計讀取容量單位 RCUs) 和寫入容量單位 WCUs)。
  - 考慮碎片數量、檢查點頻率和工作者計數等因素。
- 實作自動擴展：
  - 使用 DynamoDB Auto Scaling 自動調整佈建容量，並設定適當的最小和最大容量限制。
  - DynamoDB Auto Scaling 有助於避免 KCL 中繼資料資料表達到容量限制並受到調節。
- 定期監控和最佳化：
  - 持續監控 的 CloudWatch 指標`ThrottledRequests`。
  - 隨著工作負載隨著時間的變化調整容量。

如果您在 KCL 取用者應用程式的中繼資料 DynamoDB 資料表`ProvisionedThroughputExceededException`中遇到 ，則必須增加 DynamoDB 資料表的佈建輸送量。如果您在第一次建立取用者應用程式時設定特定層級的讀取容量單位 (RCU) 和寫入容量單位 (WCU)，則隨著用量的增加，可能還不夠。例如，如果您的 KCL 取用者應用程式經常檢查點，或在具有許多碎片的串流上操作，您可能需要更多容量單位。如需有關 DynamoDB 中佈建輸送量的資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 DynamoDB 輸送量容量和更新資料表。 DynamoDB

KCL 如何指派租用給工作者並平衡負載

KCL 會持續從執行工作者的運算主機收集和監控 CPU 使用率指標，以確保工作負載分佈均勻。這些 CPU 使用率指標會存放在 DynamoDB 的工作者指標表中。如果 KCL 偵測到某些工作者的 CPU 使用率高於其他工作者，則會在工作者之間重新指派租用，以降低高度使用工作者的負載。目標是更平均地平衡整個取用者應用程式機群的工作負載，防止任何單一工作者超載。隨著 KCL 將 CPU 使用率分配到整個取用者應用程式機群，您可以選擇正確的工作者數量或使用自動擴展來有效管理運算容量，以實現更低的成本，從而調整取用者應用程式機群容量的大小。

> **Important**
>
> 只有在符合特定先決條件時，KCL 才能從工作者收集 CPU 使用率指標。如需詳細資訊，請參閱先決條件。如果 KCL 無法從工作者收集 CPU 使用率指標，KCL 將回復為使用每個工作者的輸送量來指派租用，並平衡機群中工作者之間的負載。KCL 會監控每個工作者在指定時間收到的輸送量，並重新指派租用，以確保每個工作者從其指派的租用中取得類似的總輸送量層級。

使用 KCL 開發消費者

您可以使用 Kinesis Client Library (KCL) 來建置取用者應用程式，以處理來自 Kinesis 資料串流的資料。

KCL 提供多種語言。本主題涵蓋如何以 Java 和非 Java 語言開發 KCL 消費者。

- 若要檢視 Kinesis Client Library Javadoc 參考，請參閱 Amazon Kinesis Client Library Javadoc。
- 若要從 GitHub 下載適用於 Java 的 KCL，請參閱適用於 Java 的 Amazon Kinesis 用戶端程式庫。
- 若要在 Apache Maven 上尋找適用於 Java 的 KCL，請參閱 KCL Maven 中央儲存庫。

主題

在 Java 中使用 KCL 開發消費者

先決條件

開始使用 KCL 3.x 之前，請確定您有下列項目：

- Java 開發套件 (JDK) 8 或更新版本
- 適用於 Java 的 AWS SDK 2.x
- Maven 或 Gradle 用於相依性管理

KCL 會從工作者正在執行的運算主機收集 CPU 使用率指標，例如 CPU 使用率，以平衡負載，實現工作者之間的平均資源使用率層級。若要讓 KCL 從工作者收集 CPU 使用率指標，您必須符合下列先決條件：

Amazon Elastic Compute Cloud(Amazon EC2)

- 您的作業系統必須是 Linux 作業系統。
- 您必須在 ECIMDSv22。 EC2

Amazon EC2 上的 Amazon EC2)

- 您的作業系統必須是 Linux 作業系統。
- 您必須啟用 ECS 任務中繼資料端點第 4 版。
- 您的 Amazon ECS 容器代理程式版本必須為 1.39.0 或更新版本。

上的 Amazon ECS AWS Fargate

- 您必須啟用 Fargate 任務中繼資料端點第 4 版。如果您使用 Fargate 平台 1.4.0 版或更新版本，預設會啟用此功能。
- Fargate 平台 1.4.0 版或更新版本。

Amazon EC2 上的 Amazon Elastic Kubernetes Service (Amazon EKS)

- 您的作業系統必須是 Linux 作業系統。

上的 Amazon EKS AWS Fargate

- Fargate 平台 1.3.0 或更新版本。

> **Important**
>
> 如果 KCL 無法從工作者收集 CPU 使用率指標，KCL 會回復為使用每個工作者的輸送量來指派租用，並平衡機群中工作者的負載。如需詳細資訊，請參閱KCL 如何指派租用給工作者並平衡負載。

安裝和新增相依性

如果您使用的是 Maven，請將下列相依性新增至 pom.xml 檔案。請確定您已將 3.x.x 取代為最新的 KCL 版本。

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
</dependency>
```

如果您使用的是 Gradle，請將以下內容新增至您的 build.gradle 檔案。請確定您已將 3.x.x 取代為最新的 KCL 版本。

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

您可以在 Maven Central Repository 上檢查 KCL 的最新版本。

實作消費者

KCL 取用者應用程式包含下列重要元件：

關鍵元件

- RecordProcessor
- RecordProcessorFactory
- 排程器
- 主要消費者應用程式

RecordProcessor

RecordProcessor 是處理 Kinesis 資料串流記錄之業務邏輯所在的核心元件。它定義您的應用程式如何處理從 Kinesis 串流接收到的資料。

主要責任：

- 初始化碎片的處理
- 處理來自 Kinesis 串流的記錄批次
- 碎片的關閉處理 （例如，當碎片分割或合併時，或將租用移交給另一個主機時）
- 處理檢查點以追蹤進度

以下顯示實作範例：

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.*;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class SampleRecordProcessor implements ShardRecordProcessor {
    private static final String SHARD_ID_MDC_KEY = "ShardId";
    private static final Logger log =
 LoggerFactory.getLogger(SampleRecordProcessor.class);
    private String shardId;

    @Override
    public void initialize(InitializationInput initializationInput) {
        shardId = initializationInput.shardId();
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Initializing @ Sequence: {}",
 initializationInput.extendedSequenceNumber());
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Processing {} record(s)", processRecordsInput.records().size());
            processRecordsInput.records().forEach(r ->
```

```
                log.info("Processing record pk: {} -- Seq: {}", r.partitionKey(),
r.sequenceNumber())
            );

            // Checkpoint periodically
            processRecordsInput.checkpointer().checkpoint();
        } catch (Throwable t) {
            log.error("Caught throwable while processing records. Aborting.", t);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Lost lease, so terminating.");
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Reached shard end checkpointing.");
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            log.error("Exception while checkpointing at shard end. Giving up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Scheduler is shutting down, checkpointing.");
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
```

```
            log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }
}
```

以下是範例中使用的每個方法的詳細說明：

initialize(InitializationInput initializationInput)

- 目的：設定處理記錄所需的任何資源或狀態。
- 呼叫時：一次，當 KCL 將碎片指派給此記錄處理器時。
- 重點：
  - `initializationInput.shardId()`：此處理器將處理的碎片 ID。
  - `initializationInput.extendedSequenceNumber()`：開始處理的序號。

processRecords(ProcessRecordsInput processRecordsInput)

- 目的：處理傳入的記錄和選擇性檢查點進度。
- 呼叫時：重複，只要記錄處理器保留碎片的租用即可。
- 重點：
  - `processRecordsInput.records()`：要處理的記錄清單。
  - `processRecordsInput.checkpointer()`：用於檢查點進度。
  - 請務必在處理期間處理任何例外狀況，以防止 KCL 失敗。
  - 此方法應該是等冪的，因為在某些情況下，相同的記錄可能會處理多次，例如在意外工作者當機或重新啟動之前尚未檢查點的資料。
  - 在檢查點之前，請務必清除任何緩衝的資料，以確保資料一致性。

leaseLost(LeaseLostInput leaseLostInput)

- 目的：清除處理此碎片的任何特定資源。
- 呼叫時：當另一個排程器接管此碎片的租用時。
- 重點：

- 此方法不允許檢查點。

shardEnded(ShardEndedInput shardEndedInput)

- 目的：完成此碎片和檢查點的處理。
- 當呼叫時：當碎片分割或合併時，表示已處理此碎片的所有資料。
- 重點：
  - `shardEndedInput.checkpointer()`：用來執行最終檢查點。
  - 若要完成處理，必須使用此方法中的檢查點。
  - 若未在此排清資料和檢查點，可能會在碎片重新開啟時導致資料遺失或重複處理。

shutdownRequested(ShutdownRequestedInput shutdownRequestedInput)

- 目的：KCL 關閉時檢查和清除資源。
- 呼叫時：KCL 關閉時，例如應用程式終止時）。
- 重點：
  - `shutdownRequestedInput.checkpointer()`：用於在關機前執行檢查點。
  - 請務必在 方法中實作檢查點，以便在應用程式停止之前儲存進度。
  - 若未在此排清資料和檢查點，可能會導致應用程式重新啟動時資料遺失或重新處理記錄。

> **Important**
>
> KCL 3.x 透過在前一個工作者關閉之前進行檢查點，確保將租用從一個工作者交付到另一個工作者時，資料重新處理次數更少。如果您未在 `shutdownRequested()`方法中實作檢查點邏輯，則不會看到此好處。請確定您已在 `shutdownRequested()`方法中實作檢查點邏輯。

RecordProcessorFactory

RecordProcessorFactory 負責建立新的 RecordProcessor 執行個體。KCL 使用此工廠為應用程式需要處理的每個碎片建立新的 RecordProcessor。

主要責任：

- 視需要建立新的 RecordProcessor 執行個體

- 確定每個 RecordProcessor 都已正確初始化

以下是實作範例：

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleRecordProcessorFactory implements ShardRecordProcessorFactory {
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new SampleRecordProcessor();
    }
}
```

在此範例中，每次呼叫 shardRecordProcessor() 時，工廠都會建立新的 SampleRecordProcessor。shardRecordProcessor 您可以將此延伸到包含任何必要的初始化邏輯。

排程器

排程器是一種高階元件，可協調 KCL 應用程式的所有活動。它負責資料處理的整體協調。

主要責任：

- 管理 RecordProcessors 的生命週期
- 處理碎片的租用管理
- 座標檢查點
- 在應用程式的多個工作者之間平衡碎片處理負載
- 處理正常關機和應用程式終止訊號

排程器通常會在主要應用程式中建立和啟動。您可以在下一節主要消費者應用程式中查看排程器的實作範例。

主要消費者應用程式

主要消費者應用程式將所有元件連結在一起。它負責設定 KCL 取用者、建立必要的用戶端、設定排程器，以及管理應用程式的生命週期。

主要責任：

- 設定 AWS 服務用戶端 (Kinesis、DynamoDB、CloudWatch)

- 設定 KCL 應用程式
- 建立和啟動排程器
- 處理應用程式關閉

以下是實作範例：

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import java.util.UUID;

public class SampleConsumer {
    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    public SampleConsumer(String streamName, Region region) {
        this.streamName = streamName;
        this.region = region;
        this.kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    public void run() {
        DynamoDbAsyncClient dynamoDbAsyncClient =
 DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
            streamName,
            streamName,
            kinesisClient,
            dynamoDbAsyncClient,
            cloudWatchClient,
            UUID.randomUUID().toString(),
            new SampleRecordProcessorFactory()
        );
```

```
        Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
            configsBuilder.retrievalConfig()
        );

        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();
    }

    public static void main(String[] args) {
        String streamName = "your-stream-name"; // replace with your stream name
        Region region = Region.US_EAST_1; // replace with your region
        new SampleConsumer(streamName, region).run();
    }
}
```

KCL 預設會建立具有專用輸送量的增強型廣發 (EFO) 取用者。如需增強型廣發功能的詳細資訊，請參閱開發具有專用輸送量的增強型廣發消費者。如果您的取用者少於 2 個，或不需要 200 毫秒以下的讀取傳播延遲，您必須在排程器物件中設定下列組態，以使用共用輸送量取用者：

```
configsBuilder.retrievalConfig().retrievalSpecificConfig(new PollingConfig(streamName,
 kinesisClient))
```

下列程式碼是建立使用共用輸送量取用者之排程器物件的範例：

匯入：

```
import software.amazon.kinesis.retrieval.polling.PollingConfig;
```

程式碼：

```
Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
```

```
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
            configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient))
        );/
```

使用非 Java 語言開發具有 KCL 的消費者

本節涵蓋在 Python、Node.js、.NET 和 Ruby 中使用 Kinesis Client Library (KCL) 的消費者實作。

KCL 是 Java 程式庫。使用稱為 的多語言界面提供 Java 以外的語言支援`MultiLangDaemon`。此協助程式以 Java 為基礎，當您使用 KCL 搭配 Java 以外的語言時，會在背景執行。因此，如果您為非 Java 語言安裝 KCL，並完全以非 Java 語言撰寫消費者應用程式，則由於 ，您仍需要在系統上安裝 Java`MultiLangDaemon`。此外， `MultiLangDaemon` 有一些預設設定，您可能需要針對使用案例進行自訂（例如，連線的 AWS 區域）。如需 GitHub `MultiLangDaemon`上 的詳細資訊，請參閱 KCL MultiLangDaemon 專案。

雖然核心概念在語言之間保持不變，但有一些特定語言的考量事項和實作。如需 KCL 消費者開發的核心概念，請參閱 在 Java 中使用 KCL 開發消費者。如需如何在 Python、Node.js、.NET 和 Ruby 中開發 KCL 消費者以及最新更新的詳細資訊，請參閱下列 GitHub 儲存庫：

- Python：amazon-kinesis-client-python
- Node.js：amazon-kinesis-client-nodejs
- .NET：amazon-kinesis-client-net
- Ruby：amazon-kinesis-client-ruby

> **Important**
>
> 如果您使用的是 JDK 8，請勿使用下列非 Java KCL 程式庫版本。這些版本包含與 JDK 8 不相容的相依性（日誌）。
>
> - KCL Python 3.0.2 和 2.2.0
> - KCL Node.js 2.3.0
> - KCL .NET 3.1.0
> - KCL Ruby 2.2.0

使用 JDK 8 時，建議您使用這些受影響版本之前或之後發行的版本。

使用 KCL 進行多串流處理

本節說明 KCL 中的必要變更，可讓您建立可同時處理多個資料串流的 KCL 取用者應用程式。

Important

- 只有 KCL 2.3 或更新版本才支援多串流處理。
- 使用 執行的非 Java 語言撰寫的 KCL 取用者不支援多串流處理`multilangdaemon`。
- 任何 KCL 1.x 版本都不支援多串流處理。

- MultistreamTracker interface
  - 若要建置可以同時處理多個串流的取用者應用程式，您必須實作名為 MultiStreamTracker 的新介面。此介面包含傳回資料串流清單及其組態的 `streamConfigList` 方法，以供 KCL 取用者應用程式處理。請注意，正在處理的資料串流可以在取用者應用程式執行時間期間變更。KCL `streamConfigList`會定期呼叫 ，以了解要處理的資料串流中的變更。
  - 會填入 `streamConfigList` StreamConfig 清單。

```
package software.amazon.kinesis.common;

import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
    private String consumerArn;
}
```

  - `StreamIdentifier` 和 `InitialPositionInStreamExtended`是必要欄位，而 `consumerArn`是選用欄位。`consumerArn` 只有在您使用 KCL 實作增強型廣發消費者應用程式時，才必須提供 。

- 如需 的詳細資訊`StreamIdentifier`，請參閱 https：//https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java#L129。若要建立 `StreamIdentifier`，建議您從`streamArn`和 KCL 2.5.0 或更新版本`streamCreationEpoch`提供的 建立多串流執行個體。在不支援 的 KCL v2.3 和 v2.4 中`streamArm`，使用 格式建立多串流執行個體`account-id:StreamName:streamCreationTimestamp`。從下一個主要版本開始，此格式將被取代，不再受支援。
- MultistreamTracker 也包含刪除租用資料表中舊串流租用的策略 (formerStreamsLeasesDeletionStrategy)。請注意，在取用者應用程式執行期，無法變更策略。如需詳細資訊，請參閱 https：//https://github.com/awslabs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy.java。

•

或者，如果您想實作一個同時處理多個串流的 KCL 取用者應用程式，則可以使用 `MultiStreamTracker` 初始化 ConfigsBuilder。

```
* Constructor to initialize ConfigsBuilder with MultiStreamTracker
     * @param multiStreamTracker
     * @param applicationName
     * @param kinesisClient
     * @param dynamoDBClient
     * @param cloudWatchClient
     * @param workerIdentifier
     * @param shardRecordProcessorFactory
     */
    public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
 String applicationName,
            @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
 dynamoDBClient,
            @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
 workerIdentifier,
            @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
        this.appStreamTracker = Either.left(multiStreamTracker);
        this.applicationName = applicationName;
        this.kinesisClient = kinesisClient;
        this.dynamoDBClient = dynamoDBClient;
```

```
        this.cloudWatchClient = cloudWatchClient;
        this.workerIdentifier = workerIdentifier;
        this.shardRecordProcessorFactory = shardRecordProcessorFactory;
    }
```

- 透過針對 KCL 取用者應用程式實作的多串流支援，應用程式租用資料表的每一列現在都包含碎片 ID 和此應用程式處理之多個資料串流的串流名稱。
- 實作 KCL 取用者應用程式的多串流支援時， leaseKey 會採用下列結構：`account-id:StreamName:streamCreationTimestamp:ShardId`。例如 `111111111:multiStreamTest-1:12345:shardId-000000000336`。

> **Important**
>
> 當您現有的 KCL 取用者應用程式設定為僅處理一個資料串流時， `leaseKey`（這是租用資料表的分割區索引鍵）是碎片 ID。如果您重新設定現有的 KCL 取用者應用程式來處理多個資料串流，它會中斷您的租用資料表，因為`leaseKey`結構必須如下所示： `account-id:StreamName:StreamCreationTimestamp:ShardId`以支援多串流。

使用 AWS Glue 結構描述登錄檔搭配 KCL

您可以將 Kinesis Data Streams 與 AWS Glue 結構描述登錄檔整合。 AWS Glue 結構描述登錄檔可讓您集中探索、控制和發展結構描述，同時確保已註冊結構描述持續驗證產生的資料。結構描述定義資料記錄的結構和格式。結構描述是可靠的資料發佈、耗用或儲存的版本化規格。 AWS Glue 結構描述登錄檔可讓您改善串流應用程式中end-to-end資料品質和資料控管。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。設定此整合的其中一種方法是透過適用於 Java 的 KCL。

> **Important**
>
> - AWS Glue Kinesis Data Streams 的結構描述登錄整合僅支援 KCL 2.3 或更新版本。
> - AWS Glue 使用 執行的非 Java 語言撰寫的 KCL 取用者不支援 Kinesis Data Streams 的結構描述登錄整合`multilangdaemon`。
> - AWS Glue KCL 1.x 的任何版本都不支援 Kinesis Data Streams 的結構描述登錄整合。

如需如何使用 KCL 設定 Kinesis Data Streams 與 AWS Glue 結構描述登錄檔整合的詳細說明，請參閱「使用 KPL/KCL 程式庫與資料互動」一節：將 Amazon Kinesis Data Streams 與 AWS Glue 結構描述登錄檔整合。

KCL 取用者應用程式所需的 IAM 許可

您必須將下列許可新增至與您的 KCL 取用者應用程式相關聯的 IAM 角色或使用者。

指定 AWS 使用精細許可來控制對不同資源之存取的安全最佳實務。 AWS Identity and Access Management (IAM) 可讓您管理 中的使用者和使用者許可 AWS。IAM 政策將明確列出允許的動作以及各項動作所適用的資源。

下表顯示 KCL 取用者應用程式通常所需的最低 IAM 許可：

KCL 取用者應用程式的最低 IAM 許可

服務	動作	資源 ARNs)	用途

Amazon Kinesis Data Streams	`DescribeStream`
	
	`DescribeStreamSummary`
	
	`RegisterStreamConsumer`
		KCL 應用程式將從中處理資料的 Kinesis 資料串流。
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			嘗試讀取記錄之前，消費者應先檢查資料串流是否存在、是否處於作用中狀態，以及資料串流中是否含有碎片。
			
			將消費者註冊到碎片。

Amazon Kinesis Data Streams	`GetRecords`
	
	`GetShardIterator`
	
	`ListShards`
		KCL 應用程式將從中處理資料的 Kinesis 資料串流。
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			從碎片讀取記錄。

Amazon Kinesis Data Streams	`SubscribeToShard`	KCL 應用程式將從中處理資料的 Kinesis 資	為增強型廣發 (EFO) 消費者訂閱碎片。

服務	動作	資源 ARNs)	用途
	DescribeStreamConsumer	料串流。只有在您使用增強型廣發 (EFO) 取用者時，才新增此動作。 arn:aws:kinesis:region:account:stream/StreamName/consumer/*	
Amazon DynamoDB	CreateTable DescribeTable UpdateTable Scan GetItem PutItem UpdateItem DeleteItem	租用資料表 (KCL 在 DynamoDB 中建立的中繼資料資料表。 arn:aws:dynamodb:region:account:table/KCLApplicationName	KCL 需要這些動作，才能管理在 DynamoDB 中建立的租用資料表。

服務	動作	資源 ARNs)	用途

Amazon DynamoDB	`CreateTable`
	
	`DescribeTable`
	
	`Scan`
	
	`GetItem`
	
	`PutItem`
	
	`UpdateItem`
	
	`DeleteItem`
		工作者指標和協調器狀態資料表 (DynamoDB 中的中繼資料資料表），由 KCL 建立。
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`
			KCL 需要這些動作來管理 DynamoDB 中的工作者指標和協調器狀態中繼資料表。

Amazon DynamoDB	`Query`	租用資料表上的全域次要索引。 `arn:aws:dynamodb:region:account:table/KCLApplicationName/index/*`	KCL 需要此動作，才能讀取在 DynamoDB 中建立之租用資料表的全域次要索引。

服務	動作	資源 ARNs)	用途
Amazon CloudWatch	`PutMetricData`	*	將指標上傳至 CloudWatch，這對於監控應用程式非常有用。使用星號 (*) 是因為 CloudWatch 中沒有叫用`PutMetricData` 動作的 spcific 資源。

> **Note**
>
> 將 ARNs 中的 "region"、"account"、"StreamName" 和 "KCLApplicationName" 分別取代為您自己的 AWS 區域、 AWS 帳戶 number、Kinesis 資料串流名稱和 KCL 應用程式名稱。KCL 3.x 在 DynamoDB 中建立另外兩個中繼資料表。如需 KCL 建立之 DynamoDB 中繼資料資料表的詳細資訊，請參閱 KCL 中的 DynamoDB 中繼資料表和負載平衡。如果您使用組態來自訂 KCL 建立的中繼資料表名稱，請使用這些指定的資料表名稱，而不是 KCL 應用程式名稱。

以下是 KCL 取用者應用程式的範例政策文件。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:DescribeStreamSummary",
                "kinesis:RegisterStreamConsumer",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
```

```
            "Resource": "arn:aws:kinesis:us-
east-1:123456789012:stream/STREAM_NAME"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": "arn:aws:kinesis:us-
east-1:123456789012:stream/STREAM_NAME/consumer/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME-
WorkerMetricStats",
    "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME-
CoordinatorState"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:Query"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME/
index/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        }
    ]
}
```

使用此範例政策之前，請檢查下列項目：

- 將 REGION 取代為您的 AWS 區域（例如 us-east-1)。
- 以您的 AWS 帳戶 ID 取代 ACCOUNT_ID。
- 以 Kinesis 資料串流的名稱取代 STREAM_NAME。
- 以消費者的名稱取代 CONSUMER_NAME，通常是使用 KCL 時的應用程式名稱。
- 將 KCL_APPLICATION_NAME 取代為您的 KCL 應用程式名稱。

KCL 組態

您可以設定組態屬性來自訂 Kinesis Client Library 的功能，以符合您的特定需求。下表說明組態屬性和類別。

> **Important**
>
> 在 KCL 3.x 中，負載平衡演算法旨在實現工作者之間的 CPU 使用率，而不是每個工作者的相同租用數量。設定`maxLeasesForWorker`太低，您可能會限制 KCL 有效平衡工作負載的能

力。如果您使用 maxLeasesForWorker組態，請考慮增加其值，以允許獲得最佳的負載分佈。

此資料表顯示 KCL 的組態屬性

組態屬性	組態類別	Description	預設值
applicationName	ConfigsBuilder	此 KCL 應用程式的名稱。用做為 tableName 和 consumerName 的預設值。	不適用
tableName	ConfigsBuilder	允許覆寫用於 Amazon DynamoDB 租用資料表的資料表名稱。	不適用
streamName	ConfigsBuilder	此應用程式從中處理其記錄的串流名稱。	不適用
workerIdentifier	ConfigsBuilder	代表應用程式處理器本項實例的唯一識別符。其必須獨一無二。	不適用
failoverTimeMillis	LeaseManagementConfig	將租用擁有者視為失敗前必須經過的毫秒數。對於具有大量碎片的應用程式，這可以設定為較高的數字，以減少追蹤租用所需的 DynamoDB IOPS 數量。	10，000 (10 秒)
shardSyncIntervalMillis	LeaseManagementConfig	碎片同步呼叫的時間。	60，000 (60 秒)

組態屬性	組態類別	Description	預設值
cleanupLeasesUponShardCompletion	LeaseManagementConfig	設定時，只要已開始處理子租用就會隨即移除租用。	TRUE
ignoreUnexpectedChildShards	LeaseManagementConfig	設定時，會忽略具有開放碎片的子碎片。此項主要用於 DynamoDB Streams。	FALSE
maxLeasesForWorker	LeaseManagementConfig	單一工作者應接受的租用數量上限。如果工作者無法處理所有碎片，並導致工作者之間的租用指派不佳，則設定過低可能會導致資料遺失。設定碎片時，請考慮碎片總數、工作者數量和工作者處理容量。	無限制
maxLeaseRenewalThreads	LeaseManagementConfig	控制租用續約執行緒集區的大小。應用程式可容納的租用數愈多，此集區就應該愈大。	20

組態屬性	組態類別	Description	預設值
billingMode	LeaseManagementConfig	決定在 DynamoDB 中建立之租用資料表的容量模式。有兩種選項：隨需模式 (PAY_PER_REQUEST) 和佈建模式。我們建議您使用隨需模式的預設設定，因為它會自動擴展以容納您的工作負載，而不需要進行容量規劃。	PAY_PER_REQUEST（隨需模式）
initialLeaseTableReadCapacity	LeaseManagementConfig	當 Kinesis Client Library 需要建立具有佈建容量模式的新 DynamoDB 租用資料表時所使用的 DynamoDB DynamoDB 讀取容量。如果您在組態中使用預設隨需容量模式，您可以忽略此billingMode 組態。	10

組態屬性	組態類別	Description	預設值
`initialLeaseTableWriteCapacity`	LeaseManagementConfig	當 Kinesis Client Library 需要建立新的 DynamoDB 租用資料表時所使用的 DynamoDB 讀取容量。如果您在組態中使用預設隨需容量模式，您可以忽略此`billingMode` 組態。	10
`initialPositionInStreamExtended`	LeaseManagementConfig	應用程式應該在串流中開始的初始位置。這僅在初次建立租用時使用。	InitialPositionInStream.TRIM_HORIZON
`reBalanceThresholdPercentage`	LeaseManagementConfig	決定負載平衡演算法何時應考慮在工作者之間重新指派碎片的百分比值。 這是 KCL 3.x 中引入的新組態。	10
`dampeningPercentage`	LeaseManagementConfig	百分比值，用於抑制在單一重新平衡操作中從過載工作者移動的負載量。 這是 KCL 3.x 中引入的新組態。	60

組態屬性	組態類別	Description	預設值
`allowThroughputOvershoot`	LeaseManagementConfig	決定是否需要從過載的工作者取得額外的租用，即使它導致租用總傳輸量超過所需的傳輸量。 這是 KCL 3.x 中引入的新組態。	TRUE
`disableWorkerMetrics`	LeaseManagementConfig	決定 KCL 在重新指派租用和負載平衡時是否應忽略工作者的資源指標（例如 CPU 使用率）。如果您想要防止 KCL 根據 CPU 使用率進行負載平衡，請將此設定為 TRUE。 這是 KCL 3.x 中引入的新組態。	FALSE
`maxThroughputPerHostKBps`	LeaseManagementConfig	在租用指派期間指派給工作者的最大輸送量。 這是 KCL 3.x 中引入的新組態。	無限制

組態屬性	組態類別	Description	預設值
`isGracefulLeaseHandoffEnabled`	LeaseManagementConfig	控制工作者之間租用交接的行為。設為 true 時，KCL 會嘗試透過讓碎片的 RecordProcessor 有足夠時間完成處理，再將租用移交給其他工作者，來正常轉移租用。這有助於確保資料完整性和順暢的轉換，但可能會增加交接時間。 設定為 false 時，將立即遞交租用，而無需等待 RecordProcessor 正常關閉。這可能會導致交接速度更快，但可能會有處理不完整的風險。 注意：檢查點必須在 RecordProcessor 的 shutdownRequested() 方法中實作，才能受益於正常的租賃交接功能。 這是 KCL 3.x 中引入的新組態。	TRUE

組態屬性	組態類別	Description	預設值
gracefulLeaseHandoffTimeoutMillis	LeaseManagementConfig	指定等待目前碎片 RecordProcessor 正常關閉的最短時間（以毫秒為單位），然後再將租約強制轉移給下一個擁有者。 如果您的 processRecords 方法執行時間通常超過預設值，請考慮增加此設定。這可確保 RecordProcessor 在租用轉移發生之前有足夠的時間完成處理。 這是 KCL 3.x 中引入的新組態。	30，000 (30 秒)
maxRecords	PollingConfig	允許設定 Kinesis 傳回的記錄數上限。	10,000
retryGetRecordsInSeconds	PollingConfig	為故障設定 GetRecords 嘗試間的延遲。	無
maxGetRecordsThreadPool	PollingConfig	用於 GetRecords 的執行緒集區大小。	無
idleTimeBetweenReadsInMillis	PollingConfig	決定 KCL 在 GetRecords 呼叫之間等待多久從資料串流輪詢資料。單位為毫秒。	1,500

組態屬性	組態類別	Description	預設值
callProcessRecordsEvenForEmptyRecordList	ProcessorConfig	設定時，即使 Kinesis 無提供任何記錄也會呼叫記錄處理器。	FALSE
parentShardPollIntervalMillis	CoordinatorConfig	記錄處理器應該輪詢以檢查父碎片是否已完成的頻率。單位為毫秒。	10，000 (10 秒)
skipShardSyncAtWorkerInitializationIfLeaseExist	CoordinatorConfig	如果租用資料表包含現有的租用，即停用同步處理碎片資料。	FALSE
shardPrioritization	CoordinatorConfig	要使用哪些碎片優先順序	NoOpShardPrioritization
ClientVersionConfig	CoordinatorConfig	決定應用程式將在哪個 KCL 版本相容性模式中執行。此組態僅適用於從先前的 KCL 版本遷移。遷移至 3.x 時，您需要將此組態設定為 CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X 。您可以在完成遷移時移除此組態。	CLIENT_VERSION_CONFIG_3X

組態屬性	組態類別	Description	預設值
taskBackoffTimeMillis	LifecycleConfig	等待重試失敗 KCL 任務的時間。單位為毫秒。	500 (0.5 秒)
logWarningForTaskAfterMillis	LifecycleConfig	任務未完成的情況下要等待多久的時間才記錄警告。	無
listShardsBackoffTimeInMillis	RetrievalConfig	呼叫 ListShards 發生錯誤時將等待的間隔毫秒數。單位為毫秒。	1，500 (1.5 秒)
maxListShardsRetryAttempts	RetrievalConfig	ListShards 在放棄之前重試的次數上限。	50
metricsBufferTimeMillis	MetricsConfig	指定在將指標發佈至 CloudWatch 之前緩衝指標的持續時間上限（以毫秒為單位）。	10，000 (10 秒)
metricsMaxQueueSize	MetricsConfig	指定發佈至 CloudWatch 之前要緩衝的指標數目上限。	10,000
metricsLevel	MetricsConfig	指定要啟用和發佈的 CloudWatch 指標精細程度。 可能的值：NONE、SUMMARY、DETAILED。	MetricsLevel.DETAILED

組態屬性	組態類別	Description	預設值
metricsEnabledDimensions	MetricsConfig	控制 CloudWatch 指標的允許維度。	所有維度

KCL 3.x 中已停止的組態

下列組態屬性會在 KCL 3.x 中停止：

資料表顯示 KCL 3.x 的已停止組態屬性

組態屬性	組態類別	Description
maxLeasesToStealAtOneTime	LeaseManagementConfig	應用程式一次應該嘗試挪用的租用數上限。KCL 3.x 會忽略此組態，並根據工作者的資源使用率重新指派租用。
enablePriorityLeaseAssignment	LeaseManagementConfig	控制工作者是否應優先採用非常過期的租用（未續約 3 倍容錯移轉時間的租用）和新的碎片租用，無論目標租用計數為何，但仍遵守最大租用限制。KCL 3.x 會忽略此組態，並一律將過期的租用分散給工作者。

Important

在從先前的 KCL 版本遷移到 KCL 3.x 期間，您仍然必須擁有未取代的組態屬性。在遷移期間，KCL 工作者會先從 KCL 2.x 相容模式開始，並在偵測到應用程式的所有 KCL 工作者都準備好執行 KCL 3.x 時切換到 KCL 3.x 功能模式。當 KCL 工作者執行 KCL 2.x 相容模式時，需要這些已停止的組態。

KCL 版本生命週期政策

本主題概述 Amazon Kinesis Client Library (KCL) 的版本生命週期政策。 會 AWS 定期提供 KCL 版本的新版本，以支援新功能和增強功能、錯誤修正、安全修補程式和相依性更新。我們建議您隨時掌握 KCL up-to-date以掌握最新功能、安全性更新和基礎相依性。我們不建議繼續使用不支援的 KCL 版本。

主要 KCL 版本的生命週期包含下列三個階段：

- 一般可用性 (GA) – 在此階段，完全支援主要版本。 AWS 提供一般次要和修補程式版本，包括對 Kinesis Data Streams 新功能或 API 更新的支援，以及錯誤和安全性修正。
- 維護模式 – AWS 限制修補程式版本，以僅解決重大錯誤修正和安全問題。主要版本不會收到 Kinesis Data Streams 新功能或 APIs 的更新。
- End-of-support – 主要版本將不再接收更新或版本。先前發佈的版本將繼續透過公有套件管理員提供，且程式碼將保留在 GitHub 上。使用者可自行決定是否使用已end-of-support版本。我們建議您升級至最新的主要版本。

主要版本	目前階段	版本日期	維護模式日期	End-of-support日期
KCL 1.x	維護模式	2013-12-19	2025-04-17	2026-01-30
KCL 2.x	一般可用性	2018-08-02	--	--
KCL 3.x	一般可用性	2024-11-06	--	--

從先前的 KCL 版本遷移

本主題說明如何從舊版的 Kinesis Client Library (KCL) 遷移。

KCL 3.0 有哪些新功能？

相較於舊版，Kinesis Client Library (KCL) 3.0 引入了幾項主要增強功能：

- 它會自動將工作從過度利用的工作者重新分配到取用者應用程式機群中未充分利用的工作者，以降低取用者應用程式的運算成本。這種新的負載平衡演算法可確保跨工作者平均分佈的 CPU 使用率，並消除過度佈建工作者的需求。

- 它透過最佳化租用資料表上的讀取操作來降低與 KCL 相關聯的 DynamoDB 成本。
- 當租用重新指派給另一個工作者時，它可讓目前的工作者完成已處理記錄的檢查點，將資料的重新處理降至最低。
- 它使用 AWS SDK for Java 2.x 來改善效能和安全性功能，完全移除 適用於 Java 的 AWS SDK 1.x 上的相依性。

如需詳細資訊，請參閱 KCL 3.0 版本備註。

主題

從 KCL 2.x 遷移至 KCL 3.x

本主題提供step-by-step說明，將您的消費者從 KCL 2.x 遷移至 KCL 3.x。KCL 3.x 支援 KCL 2.x 消費者就地遷移。您可以繼續使用 Kinesis 資料串流中的資料，同時以滾動方式遷移工作者。

> **Important**
>
> KCL 3.x 維護與 KCL 2.x 相同的界面和方法。因此，您不需要在遷移期間更新記錄處理程式碼。不過，您必須設定適當的組態，並檢查遷移所需的步驟。強烈建議您遵循下列遷移步驟，以獲得順暢的遷移體驗。

步驟 1：事前準備

開始使用 KCL 3.x 之前，請確定您有下列項目：

- Java 開發套件 (JDK) 8 或更新版本
- 適用於 Java 的 AWS SDK 2.x
- Maven 或 Gradle 用於相依性管理

> **Important**
>
> 請勿將 2.27.19 到 2.27.23 適用於 Java 的 AWS SDK 版與 KCL 3.x 搭配使用。這些版本包含導致與 KCL DynamoDB 用量相關的例外狀況錯誤的問題。我們建議您使用 2 適用於 Java 的 AWS SDK .28.0 版或更新版本，以避免此問題。

步驟 2：新增相依性

如果您使用的是 Maven，請將下列相依性新增至 `pom.xml` 檔案。請確定您已將 3.x.x 取代為最新的 KCL 版本。

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
</dependency>
```

如果您使用的是 Gradle，請將以下內容新增至您的 `build.gradle` 檔案。請確定您已將 3.x.x 取代為最新的 KCL 版本。

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

您可以在 Maven Central Repository 上檢查 KCL 的最新版本。

步驟 3：設定與遷移相關的組態

若要從 KCL 2.x 遷移至 KCL 3.x，您必須設定下列組態參數：

- CoordinatorConfig.clientVersionConfig:此組態會決定應用程式將執行的 KCL 版本相容性模式。從 KCL 2.x 遷移至 3.x 時，您需要將此組態設定為 `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X`。若要設定此組態，請在建立排程器物件時新增以下行：

```
configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONFI
```

以下是如何設定 `CoordinatorConfig.clientVersionConfig` 以從 KCL 2.x 遷移至 3.x 的範例。您可以根據您的特定需求，視需要調整其他組態：

```
Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),

 configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONF
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

請務必讓消費者應用程式中的所有工作者在指定時間使用相同的負載平衡演算法，因為 KCL 2.x 和 3.x 使用不同的負載平衡演算法。執行具有不同負載平衡演算法的工作者可能會導致負載分佈不佳，因為這兩種演算法會獨立運作。

此 KCL 2.x 相容性設定可讓您的 KCL 3.x 應用程式在與 KCL 2.x 相容的模式下執行，並使用 KCL 2.x 的負載平衡演算法，直到取用者應用程式中的所有工作者升級至 KCL 3.x。遷移完成後，KCL 會自動切換到完整的 KCL 3.x 功能模式，並開始為所有執行中的工作者使用新的 KCL 3.x 負載平衡演算法。

> **Important**
>
> 如果您不是使用 ，`ConfigsBuilder`而是建立`LeaseManagementConfig`物件來設定組態，則必須在 KCL 3.x 版或更新版本`applicationName`中新增一個名為 的參數。如需詳細資訊，請參閱 LeaseManagementConfig 建構函數的編譯錯誤。建議使用 `ConfigsBuilder` 來設定 KCL 組態。 `ConfigsBuilder` 提供更靈活且可維護的方式來設定 KCL 應用程式。

步驟 4：遵循 shutdownRequested() 方法實作的最佳實務

KCL 3.x 引入了一項稱為正常租用交接的功能，以在租用交付給另一個工作者時，將資料的重新處理降至最低，作為租用重新指派程序的一部分。這是透過在租用交接之前檢查租用資料表中最後一個處理的序號來實現的。為了確保正常的租用交接正常運作，您必須確定在 `RecordProcessor`類別的`shutdownRequested`方法中叫用`checkpointer`物件。如果您未在 `shutdownRequested`方法中叫用`checkpointer`物件，您可以實作它，如下列範例所示。

Important

- 下列實作範例是正常租用交接的最低需求。如有需要，您可以將其擴展為包含與檢查點相關的其他邏輯。如果您正在執行任何非同步處理，請確保在叫用檢查點之前已處理傳送至下游的所有記錄。
- 雖然正常的租用交接可大幅降低租用傳輸期間重新處理資料的可能性，但無法完全消除此可能性。為了保持資料完整性和一致性，請將下游消費者應用程式設計為等冪。這表示他們應該能夠處理潛在的重複記錄處理，而不會對整體系統造成負面影響。

```
/**
 * Invoked when either Scheduler has been requested to gracefully shutdown
 * or lease ownership is being transferred gracefully so the current owner
 * gets one last chance to checkpoint.
 *
 * Checkpoints and logs the data a final time.
 *
 * @param shutdownRequestedInput Provides access to a checkpointer, allowing a record
 processor to checkpoint
 *                               before the shutdown is completed.
 */
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    try {
       // Ensure that all delivered records are processed
       // and has been successfully flushed to the downstream before calling
       // checkpoint
       // If you are performing any asynchronous processing or flushing to
       // downstream, you must wait for its completion before invoking
       // the below checkpoint method.
        log.info("Scheduler is shutting down, checkpointing.");
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at requested shutdown. Giving up.",
 e);
    }
}
```

步驟 5：檢查收集工作者指標的 KCL 3.x 先決條件

KCL 3.x 會從工作者收集 CPU 使用率指標，例如 CPU 使用率，以平衡工作者之間的負載。消費者應用程式工作者可以在 Amazon EC2、Amazon ECS、Amazon EKS 或 上執行 AWS Fargate。只有在滿足下列先決條件時，KCL 3.x 才能從工作者收集 CPU 使用率指標：

Amazon Elastic Compute Cloud(Amazon EC2)

- 您的作業系統必須是 Linux 作業系統。
- 您必須在 ECIMDSv22。 EC2

Amazon EC2 上的 Amazon EC2)

- 您的作業系統必須是 Linux 作業系統。
- 您必須啟用 ECS 任務中繼資料端點第 4 版。
- 您的 Amazon ECS 容器代理程式版本必須為 1.39.0 或更新版本。

上的 Amazon ECS AWS Fargate

- 您必須啟用 Fargate 任務中繼資料端點第 4 版。如果您使用 Fargate 平台 1.4.0 版或更新版本，預設會啟用此功能。
- Fargate 平台 1.4.0 版或更新版本。

Amazon EC2 上的 Amazon Elastic Kubernetes Service (Amazon EKS)

- 您的作業系統必須是 Linux 作業系統。

上的 Amazon EKS AWS Fargate

- Fargate 平台 1.3.0 或更新版本。

> **Important**
>
> 如果 KCL 3.x 因為不符合先決條件而無法從工作者收集 CPU 使用率指標，它會重新平衡每個租用的輸送量層級負載。此備用重新平衡機制將確保所有工作者將從指派給每個工作者的租用中獲得類似的總輸送量層級。如需詳細資訊，請參閱KCL 如何指派租用給工作者並平衡負載。

步驟 6：更新 KCL 3.x 的 IAM 許可

您必須將下列許可新增至與您的 KCL 3.x 取用者應用程式相關聯的 IAM 角色或政策。這包括更新 KCL 應用程式使用的現有 IAM 政策。如需詳細資訊，請參閱KCL 取用者應用程式所需的 IAM 許可。

Important

您現有的 KCL 應用程式可能沒有在 IAM 政策中新增下列 IAM 動作和資源，因為 KCL 2.x 中不需要這些動作和資源。在執行 KCL 3.x 應用程式之前，請確定您已新增它們：

- 動作： `UpdateTable`
  - 資源 ARNs)： `arn:aws:dynamodb:region:account:table/KCLApplicationName`
- 動作： `Query`
  - 資源 ARNs)： `arn:aws:dynamodb:region:account:table/KCLApplicationName/index/*`
- 動作：`CreateTable`、`DescribeTable`、`Scan`、`GetItem`、`PutItem`、`UpdateItem`、`DeleteItem`
  - 資源 (ARNs)：`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`、`arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`

將 ARNs 中的「region」、「account」和「KCLApplicationName」分別取代為您自己的 AWS 區域、 AWS 帳戶 number 和 KCL 應用程式名稱。如果您使用組態來自訂 KCL 建立的中繼資料表名稱，請使用這些指定的資料表名稱，而不是 KCL 應用程式名稱。

步驟 7：將 KCL 3.x 程式碼部署到您的工作者

設定遷移所需的組態並完成所有先前的遷移檢查清單之後，您就可以建置程式碼並將其部署至工作者。

Note

如果您看到`LeaseManagementConfig`建構函數的編譯錯誤，請參閱 LeaseManagementConfig 建構函數的編譯錯誤，以取得疑難排解資訊。

步驟 8：完成遷移

在部署 KCL 3.x 程式碼期間，KCL 會繼續使用 KCL 2.x 的租用指派演算法。當您成功將 KCL 3.x 程式碼部署到所有工作者時，KCL 會自動偵測到此情況，並根據工作者的資源使用率切換到新的租用指派演算法。如需新租用指派演算法的詳細資訊，請參閱 KCL 如何指派租用給工作者並平衡負載。

在部署期間，您可以使用傳送到 CloudWatch 的下列指標來監控遷移程序。您可以在 `Migration`操作下監控指標。所有指標都是per-KCL-application指標，並設定為SUMMARY指標層級。如果`CurrentState:3xWorker`指標的Sum統計資料符合 KCL 應用程式中的工作者總數，則表示遷移至 KCL 3.x 已成功完成。

> **Important**
>
> 所有工作者準備好執行後，KCL 至少需要 10 分鐘才能切換到新的租用指派演算法。

KCL 遷移程序的 CloudWatch 指標

指標	Description
`CurrentState:3xWorker`	成功遷移至 KCL 3.x 並執行新租用指派演算法的 KCL 工作者數量。如果此指標的Sum計數符合您工作者的總數，則表示遷移至 KCL 3.x 已成功完成。 • 指標層級：摘要 • 單位：Count • 統計資料：最有用的統計資料是 Sum
`CurrentState:2xCompatibleWorker`	在遷移過程中以 KCL 2.x 相容模式執行的 KCL 工作者數量。此指標的非零值表示遷移仍在進行中。 • 指標層級：摘要 • 單位：Count • 統計資料：最有用的統計資料是 Sum
`Fault`	在遷移過程中遇到的例外狀況數量。這些例外狀況大多是暫時性錯誤，KCL 3.x 會自動重試以完

指標	Description
	成遷移。如果您觀察到持久性Fault指標值，請檢閱遷移期間的日誌，以進一步進行故障診斷。如果問題持續發生，請聯絡 支援。 • 指標層級：摘要 • 單位：Count • 統計資料：最有用的統計資料是 Sum
GsiStatusReady	在租用資料表上建立全域次要索引 (GSI) 的狀態。此指標指出是否已建立租用資料表上的 GSI，這是執行 KCL 3.x 的先決條件。值為 0 或 1，1 表示成功建立。在回復狀態期間，不會發出此指標。再次向前捲動後，您可以繼續監控此指標。 • 指標層級：摘要 • 單位：Count • 統計資料：最有用的統計資料是 Sum
workerMetricsReady	工作者指標從所有工作者發出的狀態。指標指出是否所有工作者都會發出 CPU 使用率等指標。值為 0 或 1，其中 1 表示所有工作者都成功發出指標，並準備好使用新的租用指派演算法。在復原狀態期間，不會發出此指標。再次向前捲動後，您可以繼續監控此指標。 • 指標層級：摘要 • 單位：Count • 統計資料：最有用的統計資料是 Sum

KCL 在遷移期間提供復原功能至 2.x 相容模式。成功遷移至 KCL 3.x 後，CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X如果不再需要轉返，建議您移除 CoordinatorConfig.clientVersionConfig的設定。移除此組態會停止從 KCL 應用程式發出遷移相關指標。

> **Note**
>
> 我們建議您在遷移期間和完成遷移後監控應用程式的效能和穩定性一段時間。如果您發現任何問題，您可以使用 KCL Migration Tool 復原工作者以使用 KCL 2.x 相容功能。

轉返為先前的 KCL 版本

本主題說明將消費者復原至先前版本的步驟。當您需要轉返時，有兩個步驟的程序：

1. 執行 KCL 移轉工具。
2. 重新部署先前的 KCL 版本代碼（選用）。

步驟 1：執行 KCL 移轉工具

當您需要轉返至先前的 KCL 版本時，您必須執行 KCL 移轉工具。KCL Migration Tool 會執行兩項重要任務：

- 移除 DynamoDB 租用資料表上，稱為工作者指標資料表和全域次要索引的中繼資料資料表。這兩個成品是由 KCL 3.x 建立，但當您回復到先前的版本時不需要。
- 它可讓所有工作者在與 KCL 2.x 相容的模式下執行，並開始使用先前 KCL 版本中使用的負載平衡演算法。如果您使用 KCL 3.x 新負載平衡演算法時遇到問題，此步驟會立即緩解該問題。

> **Important**
>
> DynamoDB 中的協調器狀態資料表必須存在，且不得在移轉、轉返及向前復原過程中刪除。

> **Note**
>
> 請務必讓取用者應用程式中的所有工作者，在指定時間使用相同的負載平衡演算法。KCL Migration Tool 可確保 KCL 3.x 取用者應用程式中的所有工作者都切換到 KCL 2.x 相容模式，以便所有工作者在轉返付款期間執行相同的負載平衡演算法，回到先前的 KCL 版本。

您可以在 KCL GitHub 儲存庫的指令碼目錄中，下載 KCL 移轉工具。指令碼可以從任何工作者或任何具有寫入協調器狀態資料表、刪除工作者指標資料表和更新租用資料表所需許可的主機執行。如需執行

指令碼KCL 取用者應用程式所需的 IAM 許可所需的 IAM 許可，請參閱 。每個 KCL 應用程式只能執行指令碼一次。您可以使用下列命令執行 KCL Migration Tool：

```
python3 ./KclMigrationTool.py --region <region> --mode rollback [--
application_name <applicationName>] [--lease_table_name <leaseTableName>] [--
coordinator_state_table_name <coordinatorStateTableName>] [--worker_metrics_table_name
 <workerMetricsTableName>]
```

參數

- --region：將 取代<region>為 AWS 區域。
- --application_name：如果您使用 DynamoDB 中繼資料資料表（租用資料表、協調器狀態資料表和工作者指標資料表）的預設名稱，則需要此參數。如果您已為這些資料表指定自訂名稱，可以省略此參數。<applicationName> 將 取代為您實際的 KCL 應用程式名稱。如果未提供自訂名稱，工具會使用此名稱衍生預設資料表名稱。
- --lease_table_name（選用）：當您在 KCL 組態中設定租用資料表的自訂名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。leaseTableName 將 取代為您為租用資料表指定的自訂資料表名稱。
- --coordinator_state_table_name（選用）：當您在 KCL 組態中為協調器狀態資料表設定自訂名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。<coordinatorStateTableName> 將 取代為您為協調器狀態資料表指定的自訂資料表名稱。
- --worker_metrics_table_name（選用）：當您在 KCL 組態中為工作者指標資料表設定自訂名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。<workerMetricsTableName> 將 取代為您為工作者指標資料表指定的自訂資料表名稱。

步驟 2：使用先前的 KCL 版本重新部署程式碼（選用）

執行轉返 KCL 移轉工具後，您會看到以下其中一則訊息：

- 訊息 1：「轉返已完成。您的 KCL 應用程式正在執行 KCL 2.x 相容模式。如果您沒有看到任何迴歸的緩解措施，請使用先前的 KCL 版本部署程式碼，以回復到先前的應用程式二進位檔。」
  - 必要動作：這表示您的工作者是在 KCL 2.x 相容模式下執行。如果問題仍然存在，請將具有先前 KCL 版本的程式碼重新部署到您的工作者。
- 訊息 2：「轉返已完成。您的 KCL 應用程式正在執行 KCL 3.x 功能模式。除非您在 5 分鐘內沒有看到問題的任何緩解措施，否則不需要轉返到先前的應用程式二進位檔。如果您仍然有問題，請使用先前的 KCL 版本部署程式碼，以轉返至先前的應用程式二進位檔。」

- 必要動作：這表示您的工作者是在 KCL 3.x 模式下執行，而 KCL Migration Tool 會將所有工作者切換到 KCL 2.x 相容模式。如果問題已解決，您不需要使用先前的 KCL 版本重新部署程式碼。如果問題仍然存在，請將具有先前 KCL 版本的程式碼重新部署到您的工作者。

轉返後向前復原至 KCL 3.x

本主題說明在復原後將消費者轉返至 KCL 3.x 的步驟。當您需要轉送時，必須經歷兩個步驟的程序：

1. 執行 KCL 移轉工具。
2. 使用 KCL 3.x 部署程式碼。

步驟 1：執行 KCL 移轉工具

執行 KCL 移轉工具。具有下列命令的 KCL Migration Tool 可轉送至 KCL 3.x：

```
python3 ./KclMigrationTool.py --region <region> --mode rollforward [--application_name
 <applicationName>] [--coordinator_state_table_name <coordinatorStateTableName>]
```

參數

- --region：將 取代<region>為 AWS 區域。
- --application_name：如果您使用協調器狀態資料表的預設名稱，則需要此參數。如果您已指定協調器狀態資料表的自訂名稱，可以省略此參數。<applicationName> 將 取代為您實際的 KCL 應用程式名稱。如果未提供自訂名稱，工具會使用此名稱衍生預設資料表名稱。
- --coordinator_state_table_name （選用）：當您在 KCL 組態中為協調器狀態資料表設定自訂名稱時，需要此參數。如果您使用的是預設資料表名稱，可以省略此參數。<coordinatorStateTableName> 將 取代為您為協調器狀態資料表指定的自訂資料表名稱。

以向前復原模式執行移轉工具後，KCL 會建立 KCL 3.x 所需的下列 DynamoDB 資源：

- 租用資料表上的全域次要索引
- 工作者指標資料表

步驟 2：使用 KCL 3.x 部署程式碼

執行向前復原的 KCL 移轉工具之後，請使用 KCL 3.x 將程式碼部署至工作者。遵循 步驟 8：完成遷移 以完成遷移。

具有佈建容量模式之租用資料表的最佳實務

如果您 KCL 應用程式的租用資料表切換到佈建容量模式，KCL 3.x 會在租用資料表上建立全域次要索引，並使用佈建帳單模式和與基本租用資料表相同的讀取容量單位 (RCU) 和寫入容量單位 (WCU)。建立全域次要索引時，建議您監控 DynamoDB 主控台中全域次要索引的實際用量，並視需要調整容量單位。如需切換 KCL 建立之 DynamoDB 中繼資料資料表容量模式的更詳細指南，請參閱 KCL 所建立中繼資料資料表的 DynamoDB 容量模式。

> **Note**
>
> 根據預設，KCL 會使用隨需容量模式建立中繼資料表，例如租用資料表、工作者指標資料表和協調器狀態資料表，以及租用資料表上的全域次要索引。我們建議您使用隨需容量模式，根據您的用量變更自動調整容量。

從 KCL 1.x 移轉到 KCL 3.x

本主題說明將消費者從 KCL 1.x 遷移至 KCL 3.x 的指示。相較於 KCL 2.x 和 KCL 3.x，KCL 1.x 使用不同的類別和界面。您必須先將記錄處理器、記錄處理器工廠和工作者類別遷移至 KCL 2.x/3.x 相容格式，並遵循 KCL 2.x 遷移至 KCL 3.x 的遷移步驟。您可以直接從 KCL 1.x 升級到 KCL 3.x。

- 步驟 1：遷移記錄處理器

  遵循將消費者從 KCL 1.x 遷移至 KCL 2.x 頁面中的遷移記錄處理器一節。
- 步驟 2：遷移記錄處理器工廠

  遵循將消費者從 KCL 1.x 遷移到 KCL 2.x 頁面中的遷移記錄處理器工廠一節。
- 步驟 3：遷移工作者

  遵循將消費者從 KCL 1.x 遷移至 KCL 2.x 頁面中的遷移工作者區段。
- 步驟 4：遷移 KCL 1.x 組態

  請遵循將消費者從 KCL 1.x 遷移至 KCL 2.x 頁面中的設定 Amazon Kinesis 用戶端一節。
- 步驟 5：檢查閒置時間移除和用戶端組態移除

遵循從 KCL 1.x 遷移消費者到 KCL 2.x 頁面中的閒置時間移除和用戶端組態移除章節。

- 步驟 6：遵循 KCL 2.x 到 KCL 3.x 遷移指南中的step-by-step說明

  依照從 KCL 2.x 遷移至 KCL 3.x頁面上的指示完成遷移。如果您需要轉返至先前的 KCL 版本或在轉返後轉返至 KCL 3.x，請參閱 轉返為先前的 KCL 版本和 轉返後向前復原至 KCL 3.x。

> **Important**
>
> 請勿將 2.27.19 到 2.27.23 適用於 Java 的 AWS SDK 版與 KCL 3.x 搭配使用。這些版本包含導致與 KCL DynamoDB 用量相關的例外狀況錯誤的問題。我們建議您使用 2 適用於 Java 的 AWS SDK .28.0 版或更新版本，以避免此問題。

先前的 KCL 版本文件

下列主題已封存。若要查看目前的 Kinesis Client Library 文件，請參閱 使用 Kinesis 用戶端程式庫。

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis Client Library 頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

已淘汰的文件

- KCL 1.x 和 2.x 資訊
- 開發具有共用輸送量的自訂消費者
- 將消費者從 KCL 1.x 遷移至 KCL 2.x

KCL 1.x 和 2.x 資訊

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

開發自訂取用者應用程式以處理來自 KDS 資料串流之資料的其中一種方法，是使用 Kinesis Client Library (KCL)。

主題

- 關於 KCL （先前版本）
- KCL 舊版
- KCL 概念 （先前版本）
- 使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片
- 使用相同的 KCL 2.x for Java 取用者應用程式處理多個資料串流
- 將 KCL 與 AWS Glue 結構描述登錄檔搭配使用

> **Note**
>
> 對於 KCL 1.x 和 KCL 2.x，建議根據您的使用案例，升級至最新的 KCL 1.x 版或 KCL 2.x 版本。KCL 1.x 和 KCL 2.x 皆會定期更新為更新的版本，其中包含最新的相依性和安全修補程式、錯誤修正，以及向後相容的新功能。如需詳細資訊，請參閱 https://github.com/awslabs/amazon-kinesis-client/releases。

關於 KCL （先前版本）

KCL 會處理與分散式運算相關的許多複雜任務，協助您取用和處理 Kinesis 資料串流中的資料。這其中包含跨多個取用者應用程式執行個體的負載平衡、對取用者應用程式執行個體失敗的回應、檢查點處理記錄，以及對重新分片的反應。KCL 會處理所有這些子任務，以便您可以專注在編寫自訂記錄處理邏輯上。

KCL 與 AWS SDK 中提供的 Kinesis Data Streams API 不同。Kinesis Data Streams API 可協助您管理 Kinesis Data Streams 的許多層面 (包括建立串流、重新分片、放入與取得記錄)。KCL 圍繞所有這些子任務提供了抽象層，特別是可讓您專注於取用者應用程式的自訂資料處理邏輯。如需 Kinesis Data Streams API 的相關資訊，請參閱 Amazon Kinesis API 參考。

> **Important**
>
> KCL 是一種 Java 程式庫。使用稱為 MultiLangDaemon 的多語言介面提供對 Java 以外語言的支援。此常駐程式是以 Java 為基礎，並在您使用 Java 以外的 KCL 語言時在背景執行。例如，若您安裝了適用於 Python 的 KCL 並完全以 Python 撰寫取用者應用程式，則由於 MultiLangDaemon 的緣故，您的系統仍需要安裝 Java。此外，MultiLangDaemon 有一些預設設定，您可能需要針對您的使用案例進行自訂，例如其連線 AWS 的區域。如需 MultiLangDaemon 的詳細資訊，請參閱 GitHub 上的 KCL MultiLangDaemon 專案。

KCL 在您的記錄處理邏輯與 Kinesis Data Streams 之間擔任媒介。

KCL 舊版

目前，您可以使用下列受支援的 KCL 版本之一，來建置自訂的取用者應用程式：

- KCL 1.x

  如需詳細資訊，請參閱開發 KCL 1.x 消費者
- KCL 2.x

  如需詳細資訊，請參閱開發 KCL 2.x 消費者

您可以使用 KCL 1.x 或 KCL 2.x 來建置使用共用輸送量的取用者應用程式。如需詳細資訊，請參閱使用 KCL 開發具有共用輸送量的自訂消費者。

若要建置使用專用輸送量的取用者應用程式 (增強型散發取用者)，您只能使用 KCL 2.x。如需詳細資訊，請參閱開發具有專用輸送量的增強型廣發消費者。

如需有關 KCL 1.x 和 KCL 2.x 之間差異的詳細資訊，以及如何從 KCL 1.x 遷移至 KCL 2.x 的指示，請參閱 將消費者從 KCL 1.x 遷移至 KCL 2.x。

KCL 概念 (先前版本)

- KCL 取用者應用程式 – 使用 KCL 自訂建置的應用程式，專為讀取和處理資料串流中的記錄而設計。

- 取用者應用程式執行個體 - KCL 取用者應用程式通常是分散式，可同時執行一個或多個應用程式執行個體，以在發生故障時進行協調並對資料記錄處理進行動態負載平衡。
- 工作者 – KCL 取用者應用程式執行個體用來開始處理資料的高階類別。

> **Important**
>
> 每個 KCL 取用者應用程式執行個體都有一個工作者。

工作者會初始化並監督各種任務，包括同步處理碎片和租用資訊、追蹤碎片指派，以及處理來自碎片的資料。工作者提供 KCL 取用者應用程式的組態資訊，例如資料串流的名稱，其資料記錄此 KCL 取用者應用程式將要處理的資料，以及存取此資料串流所需的 AWS 登入資料。工作者也會啟動該特定 KCL 取用者應用程式執行個體，將資料記錄從資料串流傳送至記錄處理器。

> **Important**
>
> 在 KCL 1.x 中，此類別被稱為工作者。如需詳細資訊 (這些是 Java KCL 儲存庫)，請參閱 https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/Worker.java。在 KCL 2.x.x 中，此類別被稱為排程器。排程器在 KCL 2.x 中的用途與 KCL 1.x 中的工作者的目的相同。如需有關 KCL 2.x 中排程器類別的詳細資訊，請參閱 https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/coordinator/Scheduler.java。

- 租用 – 定義工作者與碎片之間繫結的資料。分散式 KCL 取用者應用程式使用租用來分割跨工作節點機群的資料記錄處理。在任何給定時間，每個資料記錄碎片都會透過 leaseKey 變數所識別的租用繫結至特定工作者。

  根據預設，工作者可以同時持有一或多個租用 (取決於 maxleAsForWorker 變數的值)。

> **Important**
>
> 每個工作者都將爭奪保留資料串流中，所有可用碎片的所有可用租用。但是，只有一名工作者可以在任何時間成功持有每個租用。

例如，如果您有一個含有工作者 A 的取用者應用程式執行個體 A 正在處理具有 4 個碎片的資料串流，則工作者 A 可以同時持有對碎片 1、2、3 和 4 的租用。但是，如果您有兩個取用者應用程式執

行個體：A 和 B 具有工作者 A 和工作者 B，而且這些執行個體正在處理具有 4 個碎片的資料串流，則工作者 A 和工作者 B 無法同時持有對碎片 1 的租用。一個工作者會持有特定碎片的租用，直到準備好停止處理此碎片的資料記錄，或直到失敗為止。當一名工作者停止持有租用時，另一名工作者佔用並持有租用。

如需更多資訊，(這些是 Java KCL 存儲庫)，請參閱 https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/Lease.java 以獲取 KCL 1.x 的資訊和參閱 https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/Lease.java 取得 KCL 2.x 的資訊。

- 租用資料表 - 唯一的 Amazon DynamoDB 資料表，用於追蹤 KDS 資料串流中，由 KCL 取用者應用程式的工作者租用和處理的碎片。在 KCL 取用者應用程式執行時，租用資料表必須與資料串流中的最新碎片資訊保持同步 (在工作者內部和所有工作者之間)。如需詳細資訊，請參閱使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片。
- 記錄處理器 – 定義 KCL 取用者應用程式如何處理從資料串流取得的資料的邏輯。在執行期，KCL 取用者應用程式執行個體會實體化工作者，而此工作者會針對其持有租用的每個碎片執行個體化一個記錄處理器。

使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片

主題

- 什麼是租用資料表
- 輸送量
- 租用資料表如何與 Kinesis 資料串流中的碎片同步

什麼是租用資料表

這對每個 Amazon Kinesis Data Streams 應用程式，KCL 會使用唯一的租用資料表 (存儲在 Amazon DynamoDB 資料表中)，來追蹤 KDS 資料串流中由 KCL 取用者應用程式的工作者租用和處理的碎片。

> **Important**
>
> KCL 會使用取用者應用程式的名稱來建立此取用者應用程式所使用的租用資料表名稱，因此，每個取用者應用程式名稱都必須是唯一的。

您可以在取用者應用程式執行時使用 Amazon DynamoDB 主控台檢視其租用資料表。

如果應用程式啟動時，KCL 取用者應用程式的租用資料表不存在，其中一個工作者會建立此應用程式的租用資料表。

> **Important**
>
> 您的帳戶除須支付 Kinesis Data Streams 本身的相關費用外，另將收取與 DynamoDB 資料表關聯的費用。

租用資料表內的每一列代表您的取用者應用程式的工作者所處理的某個碎片。如果您的 KCL 取用者應用程式僅處理一個資料串流，則租用資料表的雜湊索引鍵 `leaseKey` 就是碎片 ID。如果您是 使用相同的 KCL 2.x for Java 取用者應用程式處理多個資料串流，則 leaseKey 的結構如下所示：`account-id:StreamName:streamCreationTimestamp:ShardId`。例如 `111111111:multiStreamTest-1:12345:shardId-000000000336`。

除了碎片 ID 外，每一列還包含以下資料：

- checkpoint：碎片的最新檢查點序號。資料串流中所有碎片的此值皆為獨一無二。
- checkpointSubSequenceNumber：使用 Kinesis Producer Library 的彙整功能時，此為 checkpoint 的延伸，將追蹤 Kinesis 記錄內的個別使用者記錄。
- leaseCounter：用於租用版本控制，使工作者可偵測出其租用已由另一工作者接管。
- leaseKey：租用的唯一識別符。每項租用特屬於資料串流中的某個碎片，一次由一個工作者所持有。
- leaseOwner：持有此租用的工作者。
- ownerSwitchesSinceCheckpoint：自上次寫入檢查點至今，此租用更改了工作者的次數。
- parentShardId：用於確保已完全處理過父碎片後才開始對子碎片進行處理。這可確保按照記錄放入串流中的相同順序處理記錄。
- hashrange：`PeriodicShardSyncManager` 用於執行週期性同步以尋找租用資料表中遺失的碎片，並在需要時為其建立租用。

> **Note**
>
> 從 KCL 1.14 和 KCL 2.3 開始，每個碎片的租用資料表中都會顯示此資料。如需有關 `PeriodicShardSyncManager` 和租用與碎片之間的定期同步的詳細資訊，請參閱 租用資料表如何與 Kinesis 資料串流中的碎片同步。

- childshards：`LeaseCleanupManager` 用於檢閱子碎片的處理狀態，並決定是否可以從租用資料表中刪除父碎片。

  > **Note**
  >
  > 從 KCL 1.14 和 KCL 2.3 開始，每個碎片的租用資料表中都會顯示此資料。

- shardID：碎片的 ID。

  > **Note**
  >
  > 如果您是 使用相同的 KCL 2.x for Java 取用者應用程式處理多個資料串流，則此資料僅存在於租用資料表中。這僅在適用於 Java 的 KCL 2.x 中受支援，從適用於 Java 的 KCL 2.3 和更新版本開始。

- 串流名稱資料串流的識別碼，格式如下：`account-id:StreamName:streamCreationTimestamp`。

  > **Note**
  >
  > 如果您是 使用相同的 KCL 2.x for Java 取用者應用程式處理多個資料串流，則此資料僅存在於租用資料表中。這僅在適用於 Java 的 KCL 2.x 中受支援，從適用於 Java 的 KCL 2.3 和更新版本開始。

輸送量

如果您的 Amazon Kinesis Data Streams 應用程式收到佈建輸送量例外狀況，則您即應提升 DynamoDB 資料表的佈建輸送量。KCL 建立的資料表其佈建輸送量為每秒 10 次讀取和每秒 10 次寫入，但這對您的應用程式而言可能不夠。例如，若您的 Amazon Kinesis Data Streams 經常執行檢查點作業或對由多個碎片構成的串流進行操作，您可能就需要更多的輸送量。

如需 DynamoDB 中佈建輸送量的相關資訊，請參閱《Amazon DynamoDB 開發人員指南》中的讀取/寫入容量模式和使用資料表和資料。

租用資料表如何與 Kinesis 資料串流中的碎片同步

KCL 取用者應用程式中的工作者會使用租用來處理來自指定資料串流的碎片。在任何給定時間，哪個工作者正在租用哪個碎片的資訊存儲在租用資料表中。在 KCL 取用者應用程式執行時，租用資料表必須與資料串流中的最新碎片資訊保持同步 。KCL 會在取用者應用程式啟動載入期間 (在取用者應

用程式初始化或重新啟動時)，以及每當正在處理的碎片到達結束 (重新分割) 時，將租用資料表與從 Kinesis Data Streams 服務取得的碎片資訊同步化。換句話說，工作者或 KCL 取用者應用程式會與它們在初始使用者應用程式啟動程序期間處理的資料串流，以及每當取用者應用程式遇到資料串流重新分片事件時，都會與其所處理的資料串流同步處理。

主題

KCL 1.0-1.13 和 KCL 2.0-2.2 中的同步

在 KCL 1.0 - 1.13 和 KCL 2.0 - 2.2 中，在取用者應用程式的啟動載入期間以及每個資料串流重新分片事件期間，KCL 會透過調用 `ListShards` 或 `DescribeStream` 探索 API，將租用資料表與從 Kinesis 資料串流服務取得的碎片資訊同步。在上面列出的所有 KCL 版本中，KCL 取用者應用程式的每個工作者都會完成下列步驟，以便在取用者應用程式的啟動載入期間以及每個串流重新分片事件中執行租用/碎片同步處理程序：

- 擷取正在處理的資料串流的所有碎片
- 從租用資料表中擷取所有碎片租用
- 篩選出租用資料表中沒有租用的每個開放碎片
- 逐一查看所有找到的開放碎片以及每個沒有開放父級的開放碎片：
  - 透過其祖先路徑遍歷樹狀結構，以確定碎片是否為子代。如果正在處理祖系碎片 (租用資料表中存在祖系碎片的租用項目)，或者應處理祖系碎片 (例如，如果初始位置為 `TRIM_HORIZON` 或 `AT_TIMESTAMP`)，則碎片即視為子代
  - 如果內容中的開放碎片是子代，KCL 會根據初始位置檢查碎片，並在必要時為其父項建立租用

KCL 2.x 中的同步，從 KCL 2.3 及更新版本開始

從 KCL 2.x (KCL 2.3) 及更新版本的最新支援版本開始，程式庫現在支援同步程序的下列變更。這些租用/碎片同步變更可大幅減少 KCL 取用者應用程式對 Kinesis Data Streams 服務進行的 API 呼叫次數，並最佳化 KCL 取用者應用程式中的租用管理。

- 在應用程式的啟動載入期間，如果租用資料表是空的，則 KCL 會利用 `ListShard` API 的篩選選項 (`ShardFilter` 選用的請求參數) 來擷取和建立租用，僅用於在 `ShardFilter` 參數指定的時間開放的碎片快照。此 `ShardFilter` 參數可讓您篩選出 `ListShards` API 的回應。`ShardFilter` 參數

的唯一必要屬性是 Type。KCL 會使用 Type 篩選屬性及其下列有效值來識別並傳回可能需要新租用之開啟碎片的快照：

- AT_TRIM_HORIZON - 回應包括所有在 TRIM_HORIZON 打開的碎片。
- AT_LATEST - 回應僅包含目前開放的資料串流碎片。
- AT_TIMESTAMP - 回應包含開始時間戳記小於或等於指定時間戳記，且結束時間戳記大於或等於指定時間戳記或仍處於開放狀態的所有碎片。

ShardFilter 用於為空租用資料表建立租用，以針對在 RetrievalConfig#initialPositionInStreamExtended 指定之碎片的快照初始化租用。

如需 ShardFilter 的相關資訊，請參閱 https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html。

- 單一當選的工作者領導者執行租用/碎片同步處理，而不是執行租用/碎片同步處理以使租用資料表與資料串流中的最新碎片保持最新狀態的所有工作者。
- KCL 2.3 使用 ChildShards 傳回 GetRecords 和 SubscribeToShard API 的參數來執行在關閉碎片 SHARD_END 上發生的租用/碎片同步處理，允許 KCL 工作者只為其完成處理之碎片的子碎片建立租用。對於共用輸送量取用者應用程式，此租用/碎片同步處理的最佳化會使用 GetRecords API 的 ChildShards 參數。對於專用輸送量 (增強型散發) 取用者應用程式，此租用/碎片同步處理的最佳化會使用 SubscribeToShard API 的 ChildShards 參數。如需詳細資訊，請參閱 GetRecords、SubscribeToShards 和 ChildShard。
- 透過上述變更，KCL 的行為將從學習所有現有碎片的所有工作者的模型，轉移到僅學習每個工作者所擁有碎片的子碎片的工作者模型。因此，除了取用者應用程式啟動載入和重新分片事件期間發生的同步處理之外，KCL 現在還會執行額外的定期碎片/租用掃描，以識別租用資料表中的任何潛在漏洞 (換句話說，了解所有新碎片)，以確保資料串流的完整雜湊範圍正在處理，並在需要時為其建立租用。PeriodicShardSyncManager 是負責執行定期租用/碎片掃描的元件。

  如需有關 KCL 2.3 中 PeriodicShardSyncManager 的詳細資訊，請參閱 https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/LeaseManagementConfig.java#L201-L213。

  在 KCL 2.3 中，新組態選項可用於在 LeaseManagementConfig 中設定 PeriodicShardSyncManager：

名稱	預設值	Description
leasesRecoveryAuditorExecutionFrequencyMillis	120000 (2 分鐘)	稽核人員工作掃描租用資料表中部分租用的頻率 (以毫秒為單位)。如果稽核人員偵測到串流租用中的任何漏洞，則會根據 `leasesRecoveryAuditorInconsistencyConfidenceThreshold` 觸發碎片同步處理。
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	定期稽核人員工作的信賴閾值，用於確定租用資料表中資料串流的租用是否不一致。如果稽核人員多次連續發現同一組資料串流的不一致，則會觸發碎片同步處理。

現在也會發出新的 CloudWatch 指標，以監控 `PeriodicShardSyncManager` 的運作狀態。如需詳細資訊，請參閱PeriodicShardSyncManager。

- 包括 `HierarchicalShardSyncer` 的最佳化，以僅為一層碎片建立租用。

KCL 1.x 中的同步，從 KCL 1.14 及更新版本開始

從 KCL 1.x (KCL 1.14) 及更新版本的最新支援版本開始，程式庫現在支援同步程序的下列變更。這些租用/碎片同步變更可大幅減少 KCL 取用者應用程式對 Kinesis Data Streams 服務進行的 API 呼叫次數，並最佳化 KCL 取用者應用程式中的租用管理。

- 在應用程式的啟動載入期間，如果租用資料表是空的，則 KCL 會利用 `ListShard` API 的篩選選項 (`ShardFilter` 選用的請求參數) 來擷取和建立租用，僅用於在 `ShardFilter` 參數指定的時間開放的碎片快照。此 `ShardFilter` 參數可讓您篩選出 `ListShards` API 的回應。`ShardFilter` 參數的唯一必要屬性是 `Type`。KCL 會使用 `Type` 篩選屬性及其下列有效值來識別並傳回可能需要新租用之開啟碎片的快照：
  - `AT_TRIM_HORIZON` - 回應包括所有在 `TRIM_HORIZON` 打開的碎片。
  - `AT_LATEST` - 回應僅包含目前開放的資料串流碎片。
  - `AT_TIMESTAMP` - 回應包含開始時間戳記小於或等於指定時間戳記，且結束時間戳記大於或等於指定時間戳記或仍處於開放狀態的所有碎片。

  `ShardFilter` 用於為空租用資料表建立租用，以針對在 `KinesisClientLibConfiguration#initialPositionInStreamExtended` 指定之碎片的快照初始化租用。

  如需 `ShardFilter` 的相關資訊，請參閱 https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html。
- 單一當選的工作者領導者執行租用/碎片同步處理，而不是執行租用/碎片同步處理以使租用資料表與資料串流中的最新碎片保持最新狀態的所有工作者。
- KCL 1.14 使用 `ChildShards` 傳回 `GetRecords` 和 `SubscribeToShard` API 的參數來執行在關閉碎片 `SHARD_END` 上發生的租用/碎片同步處理，允許 KCL 工作者只為其完成處理之碎片的子碎片建立租用。如需詳細資訊，請參閱 GetRecords 和 ChildShard。
- 透過上述變更，KCL 的行為將從學習所有現有碎片的所有工作者的模型，轉移到僅學習每個工作者所擁有碎片的子碎片的工作者模型。因此，除了取用者應用程式啟動載入和重新分片事件期間發生的同步處理之外，KCL 現在還會執行額外的定期碎片/租用掃描，以識別租用資料表中的任何潛在漏洞 (換句話說，了解所有新碎片)，以確保資料串流的完整雜湊範圍正在處理，並在需要時為其建立租用。`PeriodicShardSyncManager` 是負責執行定期租用/碎片掃描的元件。

  當 `KinesisClientLibConfiguration#shardSyncStrategyType` 設定為 `ShardSyncStrategyType.SHARD_END` 時，`PeriodicShardSync` `leasesRecoveryAuditorInconsistencyConfidenceThreshold` 用於確定包含租用資料表中漏洞的連續掃描數目臨界值，之後強制執行碎片同步化。

當 KinesisClientLibConfiguration#shardSyncStrategyType 設定為 ShardSyncStrategyType.PERIODIC 時，會忽略 leasesRecoveryAuditorInconsistencyConfidenceThreshold。

如需有關 KCL 1.14 中 PeriodicShardSyncManager 的詳細資訊，請參閱 https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/KinesisClientLibConfiguration.java#L987-L999。

在 KCL 1.14 中，新組態選項可用於在 LeaseManagementConfig 中設定 PeriodicShardSyncManager：

名稱	預設值	Description
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	定期稽核人員工作的信賴閾值，用於確定租用資料表中資料串流的租用是否不一致。如果稽核人員多次連續發現同一組資料串流的不一致，則會觸發碎片同步處理。

現在也會發出新的 CloudWatch 指標，以監控 PeriodicShardSyncManager 的運作狀態。如需詳細資訊，請參閱PeriodicShardSyncManager。

- KCL 1.14 現在也支援延遲租用清除。當碎片超過資料串流的保留期限或因重新分片操作而關閉時，達到 SHARD_END 時，LeaseCleanupManager 會以非同步方式刪除租用。

  新組態選項可用於設定 LeaseCleanupManager：

名稱	預設值	Description
leaseCleanupIntervalMillis	1 分鐘	執行租用清除執行緒的間隔。

名稱	預設值	Description
completedLeaseCleanupIntervalMillis	5 分鐘	檢查租用是否完成的間隔。
garbageLeaseCleanupIntervalMillis	30 分鐘	檢查租用是否為垃圾 (亦即超過資料串流保留期所進行的修剪) 的間隔。

- 包括 `KinesisShardSyncer` 的最佳化，以僅為一層碎片建立租用。

使用相同的 KCL 2.x for Java 取用者應用程式處理多個資料串流

本節說明適用於 Java 的 KCL 2.x 中的下列變更，可讓您建立可同時處理多個資料串流的 KCL 取用者應用程式。

> **Important**
>
> 僅在適用於 Java 的 KCL 2.x 中支援多串流處理，從適用於 Java 的 KCL 2.3 和更新版本開始。
> 對於可以實現 KCL 2.x 的任何其他語言，「不」支援多串流處理。
> 任何 KCL 1.x 版本均不支援多串流處理。

- MultistreamTracker interface

  若要建置可以同時處理多個串流的取用者應用程式，您必須實作名為 MultiStreamTracker 的新介面。此介面包含傳回資料串流清單及其組態的 `streamConfigList` 方法，以供 KCL 取用者應用程式處理。請注意，正在處理的資料串流可以在取用者應用程式執行期變更。KCL 會定期呼叫 `streamConfigList`，以瞭解要處理的資料串流變更。

  該 `streamConfigList` 方法會填充 StreamConfig 清單。

```
package software.amazon.kinesis.common;
```

```
import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
    private String consumerArn;
}
```

請注意，StreamIdentifier 和 InitialPositionInStreamExtended 是必填欄位，而 consumerArn 是選填欄位。只有在您使用 KCL 2.x 來實作增強型散發取用者應用程式時，才必須提供 consumerArn。

如需 的詳細資訊StreamIdentifier，請參閱 https：//https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java#L129。若要建立 StreamIdentifier，建議您從 streamArn和 streamCreationEpoch v2.5.0 及更新版本中提供的 建立多串流執行個體。在不支援 的 KCL v2.3 和 v2.4 中streamArn，使用 格式建立多串流執行個體account-id:StreamName:streamCreationTimestamp。從下一個主要版本開始，此格式將被取代，不再受支援。

MultistreamTracker 還包括刪除租用資料表 (formerStreamsLeasesDeletionStrategy) 中舊串流的租用的策略。請注意，在取用者應用程式執行期，無法變更策略。如需詳細資訊，請參閱 https://github.com/awslabs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy.java

- ConfigsBuilder 是一個應用程式範圍的類別，可用來指定建置 KCL 取用者應用程式時要使用的所有 KCL 2.x 組態設定值。ConfigsBuilder 類別現在支援 MultistreamTracker 介面。您可以使用一個資料串流的名稱初始化 ConfigsBuilder，以取用來自以下內容的記錄：

```
/**
     * Constructor to initialize ConfigsBuilder with StreamName
     * @param streamName
     * @param applicationName
     * @param kinesisClient
     * @param dynamoDBClient
```

```
     * @param cloudWatchClient
     * @param workerIdentifier
     * @param shardRecordProcessorFactory
     */
    public ConfigsBuilder(@NonNull String streamName, @NonNull String
applicationName,
            @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
            @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
            @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
        this.appStreamTracker = Either.right(streamName);
        this.applicationName = applicationName;
        this.kinesisClient = kinesisClient;
        this.dynamoDBClient = dynamoDBClient;
        this.cloudWatchClient = cloudWatchClient;
        this.workerIdentifier = workerIdentifier;
        this.shardRecordProcessorFactory = shardRecordProcessorFactory;
    }
```

或者，如果您想實作一個同時處理多個串流的 KCL 取用者應用程式，則可以使用 `MultiStreamTracker` 初始化 ConfigsBuilder。

```
* Constructor to initialize ConfigsBuilder with MultiStreamTracker
     * @param multiStreamTracker
     * @param applicationName
     * @param kinesisClient
     * @param dynamoDBClient
     * @param cloudWatchClient
     * @param workerIdentifier
     * @param shardRecordProcessorFactory
     */
    public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
String applicationName,
            @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
            @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
            @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
        this.appStreamTracker = Either.left(multiStreamTracker);
        this.applicationName = applicationName;
        this.kinesisClient = kinesisClient;
```

```
        this.dynamoDBClient = dynamoDBClient;
        this.cloudWatchClient = cloudWatchClient;
        this.workerIdentifier = workerIdentifier;
        this.shardRecordProcessorFactory = shardRecordProcessorFactory;
    }
```

- 針對 KCL 取用者應用程式實作多串流支援後，應用程式租用資料表的每一列現在都包含碎片 ID 和此應用程式所處理之多個資料串流的串流名稱。
- 實作 KCL 取用者應用程式的多串流支援時，leaseKey 會採用下列結構：`account-id:StreamName:streamCreationTimestamp:ShardId`。例如 `111111111:multiStreamTest-1:12345:shardId-000000000336`。

> **Important**
>
> 如果您的現有 KCL 取用者應用程式設定僅處理一個資料串流，則 leaseKey (租用資料表的雜湊索引鍵) 就是碎片 ID。如果您重新設定這個現有的 KCL 取用者應用程式來處理多個資料串流，則它會中斷租用資料表，因為有了多重串流支援，leaseKey 結構必須如下所示：`account-id:StreamName:StreamCreationTimestamp:ShardId`。

將 KCL 與 AWS Glue 結構描述登錄檔搭配使用

您可以將 Kinesis 資料串流與 AWS Glue 結構描述登錄檔整合。 AWS Glue 結構描述登錄檔可讓您集中探索、控制和發展結構描述，同時確保已註冊結構描述持續驗證產生的資料。結構描述定義資料記錄的結構和格式。結構描述是可靠的資料發佈、耗用或儲存的版本化規格。 AWS Glue結構描述登錄檔可讓您改善串流應用程式中end-to-end資料品質和資料控管。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。設定此整合的方法之一是透過 Java 中的 KCL。

> **Important**
>
> 目前，Kinesis Data Streams 和 AWS Glue 結構描述登錄整合僅支援使用在 Java 中實作的 KCL 2.3 消費者的 Kinesis 資料串流。不提供多語言支援。不支援 KCL 1.0 取用者。不支援 KCL 2.3 之前的 KCL 2.x 取用者。

如需如何使用 KCL 設定 Kinesis Data Streams 與結構描述登錄檔整合的詳細說明，請參閱「使用 KPL/KCL 程式庫與資料互動」一節：將 Amazon Kinesis Data Streams 與 AWS Glue 結構描述登錄檔整合。

開發具有共用輸送量的自訂消費者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

如果您不需要專用輸送量從 Kinesis Data Streams 接收資料，而且不需要低於 200 毫秒的讀取傳播延遲，則可依照以下主題所述建置取用者應用程式。您可以使用 Kinesis Client Library (KCL) 或 適用於 Java 的 AWS SDK。

主題

- 使用 KCL 開發具有共用輸送量的自訂消費者

如需如何建置可經由專用輸送量從 Kinesis 資料串流接收記錄的取用者相關資訊，請參閱 開發具有專用輸送量的增強型廣發消費者。

使用 KCL 開發具有共用輸送量的自訂消費者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

開發具有共用輸送量的自訂取用者應用程式的方法之一是使用 Kinesis Client Library (KCL)。

針對您正在使用的 KCL 版本選擇下列主題。

主題

- 開發 KCL 1.x 消費者

- 開發 KCL 2.x 消費者

開發 KCL 1.x 消費者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis Client Library 頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 Kinesis Client Library (KCL) 為 Amazon Kinesis Data Streams 開發取用者應用程式。

如需詳細資訊，請參閱關於 KCL （先前版本）。

根據您要使用的選項，從下列主題中進行選擇。

目錄

- 在 Java 中開發 Kinesis Client Library 取用者
- 在 Node.js 中開發 Kinesis Client Library 取用者
- 在 .NET 中開發 Kinesis Client Library 取用者
- 在 Python 中開發 Kinesis Client Library 取用者
- 在 Ruby 中開發 Kinesis Client Library 消費者

在 Java 中開發 Kinesis Client Library 取用者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 Kinesis Client Library (KCL) 建置應用程式，處理來自 Kinesis 資料串流的資料。Kinesis Client Library 支援多種語言。本主題將討論 Java。如要檢視 Javadoc 參考，請參閱 AmazonKinesisClient 類別的AWS Javadoc 主題。

若要從 GitHub 下載 Java KCL，請前往 Kinesis Client Library (Python)。若要尋找 Apache Maven 上的 Java KCL，請前往 KCL 搜尋結果頁面。如需從 GitHub 下載 Java KCL 取用者應用程式的範本程式碼，請至 GitHub 前往適用於 Java 的 KCL 範例專案頁面。

範例應用程式使用 Apache Commons Logging。您可以從 `configure` 檔案中定義的靜態 `AmazonKinesisApplicationSample.java` 方法更改日誌記錄組態。如需如何使用 Apache Commons Logging 搭配 Log4j 和 AWS Java 應用程式的詳細資訊，請參閱《適用於 Java 的 AWS SDK 開發人員指南》中的 Log4j 記錄。

以 Java 實作 KCL 取用者應用程式時，您必須完成以下任務：

任務

- 實作 IRecordProcessor 方法
- 實作 IRecordProcessor 介面的類別工廠
- 建立工作者
- 修改組態屬性
- 遷移至記錄處理器界面的第 2 版

實作 IRecordProcessor 方法

KCL 目前支援兩種版本的 `IRecordProcessor` 界面：原始界面適用於第一版的 KCL，而第 2 版自 KCL 1.5.0 版起均可使用。兩種界面皆完全受支援。兩種界面皆可完整支援。您的選擇取決於具體的情境要求。如需查看兩者間的所有差異，請參閱您在本機建置的 Javadoc 或原始碼。以下各節概要說明最低限度的入門實作。

IRecordProcessor 版本

- 原始界面 (第 1 版)
- 更新界面 （ 第 2 版 ）

原始界面 (第 1 版)

原始 `IRecordProcessor` 界面 (`package com.amazonaws.services.kinesis.clientlibrary.interfaces`) 公開了您的

消費者必須實作的下列記錄處理器方法。範例提供的實作可讓您用於做為起點 (請參閱 AmazonKinesisApplicationSampleRecordProcessor.java)。

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

initialize

KCL 將於記錄處理器執行個體化時呼叫 initialize 方法，傳遞特定碎片 ID 作為參數。此記錄處理器只會處理該碎片，且通常反過來說同樣成立 (該碎片僅由此記錄處理器處理)。然而，您的消費者應該考慮到資料記錄可能經過多次處理的情況。Kinesis Data Streams 具有至少一次的語意，即碎片中的每一筆資料記錄至少會由取用者內的工作者處理一次。如需特定碎片可能由多個工作者處理之各種情況的詳細資訊，請參閱使用重新分片、擴展和平行處理來變更碎片數量。

```
public void initialize(String shardId)
```

processRecords

KCL 會呼叫 processRecords 方法，傳遞由 initialize(shardId) 方法所指定碎片中之資料記錄的清單。記錄處理器根據消費者的語意處理這些記錄中的資料。例如，工作者可能會執行資料轉換，然後將結果存放至 Amazon Simple Storage Service (Amazon S3) 儲存貯體。

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
```

除了資料本身外，記錄還包含序號和分割區索引鍵。工作者在處理資料時可使用這些值。例如，工作者可根據分割區索引鍵的值，選擇要存放資料的 S3 儲存貯體。Record 類別公開了下列方法，可供存取記錄的資料、序號和分割區索引鍵。

```
record.getData()
record.getSequenceNumber()
record.getPartitionKey()
```

範例中，私有方法 processRecordsWithRetries 的程式碼示範了工作者如何能夠存取記錄的資料、序號和分割區索引鍵。

Kinesis Data Streams 需要由記錄處理器追蹤碎片中已經處理過的記錄。KCL 透過將檢查點指標 (IRecordProcessorCheckpointer) 傳遞給 processRecords 為您進行這項追蹤。記錄處理器

將對此界面呼叫 checkpoint 方法，以通知 KCL 目前處理碎片中的記錄之進度。如果工作者發生失敗，KCL 將使用此資訊於上一筆已知處理過的記錄處重新啟動碎片處理。

對於分割或合併操作，在原始碎片的處理器呼叫 checkpoint 以表示對原始碎片進行所有處理都已完成之前，KCL 將不會開始處理新碎片。

如果您未傳遞參數，KCL 將假定對 checkpoint 的呼叫表示所有記錄皆已處理，一直處理到傳遞至記錄處理器的最後一筆記錄。因此，記錄處理器應僅在已處理過向其傳遞的清單中之所有記錄後才呼叫 checkpoint。記錄處理器不需要在每次呼叫 checkpoint 時呼叫 processRecords。例如，處理器可以每呼叫三次 checkpoint 才呼叫一次 processRecords。您可以選擇性指定某筆記錄的確切序號做為 checkpoint 的參數。在此情況下，KCL 將假定所有記錄皆已處理，僅止於處理到該記錄。

範例中，私有方法 checkpoint 示範了如何利用適當的例外狀況處理和重試邏輯來呼叫 IRecordProcessorCheckpointer.checkpoint。

KCL 倚賴 processRecords 以處理任何因處理資料記錄而引發的例外狀況。如果 processRecords 擲回例外狀況，KCL 將略過例外狀況發生前已傳遞的資料記錄。也就是說，這些記錄不會重新傳送到擲回例外狀況的記錄處理器或消費者內的任何其他記錄處理器。

shutdown

KCL 會在處理結束 (關閉原因是 TERMINATE) 或工作者不再回應 (關閉原因為 ZOMBIE) 時呼叫 shutdown 方法。

```
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

當記錄處理器未能再從碎片接收任何記錄 (因為碎片已進行分割或合併或者串流已刪除) 時，處理即告結束。

KCL 還會將 IRecordProcessorCheckpointer 界面傳遞給 shutdown。如果關閉原因是 TERMINATE，表示記錄處理器應已完成處理任何資料記錄，然後對此界面呼叫 checkpoint 方法。

更新界面 （ 第 2 版 ）

更新後的 IRecordProcessor 界面 (package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2) 公開了您的消費者必須實作的下列記錄處理器方法：

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
```

```
void shutdown(ShutdownInput shutdownInput)
```

原始版本界面的所有引數皆可透過容器物件的 get 方法進行存取。例如，若要擷取 processRecords() 中的記錄清單，可使用 processRecordsInput.getRecords()。

自此界面的第 2 版 (KCL 1.5.0 及更新版本) 起，除了原始界面提供的輸入外，還可使用以下各項新的輸入：

起始序號

在傳遞給 InitializationInput 操作的 initialize() 物件中，將向記錄處理器執行個體提供的各筆記錄其起始序號。這是由先前處理同一碎片的記錄處理器執行個體執行上一次檢查點作業的序號。當您的應用程式需要此序號時，請提供這項資訊。

待定檢查點序號

在傳遞給 initialize() 操作的 InitializationInput 物件中，上一個記錄處理器執行個體於停止前未能遞交的待定檢查點序號 (若有)。

實作 IRecordProcessor 介面的類別工廠

實作記錄處理器方法的類別還需要實作處理站。您的消費者在執行個體化工作者時將傳遞此處理站的參考。

範例是在 AmazonKinesisApplicationSampleRecordProcessorFactory.java 檔案中使用原始記錄處理器界面實作處理站類別。若您希望類別處理站建立第 2 版的記錄處理器，請使用套件名稱 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2。

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
    * Constructor.
    */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
    * {@inheritDoc}
    */
    @Override
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
```

```
        }
    }
```

建立工作者

如 實作 IRecordProcessor 方法 所述，KCL 記錄處理器界面有兩種版本可供選擇，而這將影響您建立工作者的方式。原始記錄處理器界面使用以下程式碼結構建立工作者：

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

若為第 2 版的記錄處理器界面，您則可使用 `Worker.Builder` 建立工作者，而不必擔心應該使用哪個建構函數以及引數的順序。更新後的記錄處理器界面使用以下程式碼結構建立工作者：

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

修改組態屬性

範例提供了組態屬性的預設值。工作者的這份組態資料隨後整併到 `KinesisClientLibConfiguration` 物件。此物件以及 `IRecordProcessor` 的類別處理站參考將傳遞至用於執行個體化工作者的呼叫。您可使用 Java 屬性檔案 (請參閱 `AmazonKinesisApplicationSample.java`) 以自訂值覆寫任何這些屬性。

Application name (應用程式名稱)

KCL 要求所有應用程式和同一區域內的 Amazon DynamoDB 資料表必須具有獨一無二的應用程式名稱。其使用應用程式名稱組態值的方式如下：

- 假定所有與此應用程式名稱相關聯的工作者合作處理同一串流。這些工作者可能分佈於多個執行個體。如果您以相同應用程式的程式碼執行另一執行個體但使用不同的應用程式名稱，KCL 便會將第二個執行個體視為亦對同一串流進行操作的完全獨立應用程式。
- KCl 將使用應用程式名稱建立 DynamoDB 資料表並由該資料表維護應用程式的狀態資訊 (例如檢查點及工作者與碎片間對應)。每個應用程式都有其自身的 DynamoDB 資料表。如需詳細資訊，請參閱使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片。

設定登入資料

您必須將 AWS 登入資料提供給預設登入資料提供者鏈結中的其中一個登入資料提供者。例如，如果您在 EC2 執行個體上執行取用者，建議您使用 IAM 角色來啟動執行個體。反映與此 IAM 角色相關聯許可的 AWS 憑證，可透過執行個體中繼資料提供給執行個體上的應用程式。以這種方式管理 EC2 執行個體上執行的消費者登入資料最為安全。

範例應用程式首先嘗試從執行個體中繼資料擷取 IAM 憑證：

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

如果範例應用程式無法從執行個體中繼資料取得登入資料，其將嘗試從屬性檔案擷取登入資料：

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

如需執行個體中繼資料的詳細資訊，請參閱《Amazon EC2 使用者指南》中的執行個體中繼資料。

將工作者 ID 用於多個執行個體

範例初始化程式碼透過使用本機電腦的名稱並附加全域唯一識別符的方式建立工作者 ID (workerId)，如以下程式碼片段所示。如此可支援消費者應用程式的多個執行個體在單一電腦上執行的情況。

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
 UUID.randomUUID();
```

遷移至記錄處理器界面的第 2 版

若您想要遷移使用原始界面的程式碼，則除了遵照前述步驟外，您還需執行以下步驟：

1. 將您的記錄處理器類別更改為匯入第 2 版的記錄處理器界面：

   ```
   import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

2. 將各項輸入的參考更改為使用容器物件的 get 方法。例如，在 shutdown() 操作中，將 "checkpointer" 更改為 "shutdownInput.getCheckpointer()"。
3. 將您的記錄處理器處理站類別更改為匯入第 2 版的記錄處理器處理站界面：

   ```
   import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
   ```

4. 將工作者的建構更改為使用 `Worker.Builder`。例如：

```
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

在 Node.js 中開發 Kinesis Client Library 取用者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis Client Library 頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 Kinesis Client Library (KCL) 建置應用程式，處理來自 Kinesis 資料串流的資料。Kinesis Client Library 支援多種語言。本主題將討論 Node.js。

KCL 是一種 Java 程式庫，使用稱為 MultiLangDaemon 的多語言介面提供對 Java 以外語言的支援。此常駐程式是以 Java 為基礎，並在您使用 Java 以外的 KCL 語言時在背景執行。因此，若您安裝了適用於 Node.js 的 KCL 並完全以 Node.js 撰寫取用者應用程式，則由於 MultiLangDaemon 的緣故，您的系統仍需要安裝 Java。此外，MultiLangDaemon 有一些預設設定，您可能需要針對您的使用案例進行自訂，例如其連線 AWS 的區域。如需 MultiLangDaemon 的詳細資訊，請前往 GitHub 上的 KCL MultiLangDaemon 專案頁面。

若要從 GitHub 下載 Node.js KCL，請移至 Kinesis Client Library (Node.js)。

範本程式碼下載

Node.js 提供了兩份適用於 KCL 的程式碼範例：

- basic-sample

  以下各節將利用此範例說明以 Node.js 建置 KCL 取用者應用程式的原理。

- click-stream-sample

程度更為進階的範例，使用真實情境，適合您在熟悉基本範本程式碼之後研究。本文不會就此範例進行討論，但其本身附有 README 檔案提供更多詳細資訊。

以 Node.js 實作 KCL 取用者應用程式時，您必須完成以下任務：

任務

- 實作記錄處理器
- 修改組態屬性

實作記錄處理器

使用適用於 Node.js 的 KCL 所開發最簡單形式的取用者必須實作 `recordProcessor` 函數，後者則又包含 `initialize`、`processRecords` 和 `shutdown` 函數。範例提供的實作可讓您用於做為起點 (請參閱 `sample_kcl_app.js`)。

```
function recordProcessor() {
  // return an object that implements initialize, processRecords and shutdown
 functions.}
```

initialize

KCL 將於記錄處理器啟動時呼叫 `initialize` 函數。此記錄處理器只會處理以 `initializeInput.shardId` 傳遞的碎片 ID，且通常反過來說同樣成立 (該碎片僅由此記錄處理器處理)。然而，您的消費者應該考慮到資料記錄可能經過多次處理的情況。這是因為 Kinesis Data Streams 具有至少一次的語意，即碎片中的每一筆資料記錄至少會由取用者內的工作者處理一次。如需特定碎片可能由多個工作者處理之各種情況的詳細資訊，請參閱使用重新分片、擴展和平行處理來變更碎片數量。

```
initialize: function(initializeInput, completeCallback)
```

processRecords

KCL 將依照 `initialize` 函數內指定的碎片，使用該碎片中各資料記錄的清單做為輸入以呼叫此函數。您所實作的記錄處理器根據消費者的語意處理這些記錄中的資料。例如，工作者可能會執行資料轉換，然後將結果存放至 Amazon Simple Storage Service (Amazon S3) 儲存貯體。

```
processRecords: function(processRecordsInput, completeCallback)
```

除了資料本身外，記錄還包含工作者在處理資料時可使用的序號和分割區索引鍵。例如，工作者可根據分割區索引鍵的值，選擇要存放資料的 S3 儲存貯體。record 字典公開了以下的索引鍵值組，可供存取記錄的資料、序號和分割區索引鍵：

```
record.data
record.sequenceNumber
record.partitionKey
```

請注意，資料為 Base64 編碼。

基本範例中，processRecords 函數的程式碼示範了工作者如何能夠存取記錄的資料、序號和分割區索引鍵。

Kinesis Data Streams 需要由記錄處理器追蹤碎片中已經處理過的記錄。KCL 透過以 processRecordsInput.checkpointer 傳遞的 checkpointer 物件進行這項追蹤。記錄處理器將呼叫 checkpointer.checkpoint 函數，以通知 KCL 目前處理碎片中的記錄之進度。如果工作者發生失敗，KCL 將在您重新啟動碎片處理時使用此資訊，以便從上一筆已知處理過的記錄處繼續處理。

對於分割或合併操作，在原始碎片的處理器呼叫 checkpoint 以表示對原始碎片進行所有處理都已完成之前，KCL 不會開始處理新碎片。

如果您未傳遞序號給 checkpoint 函數，KCL 將假定對 checkpoint 的呼叫表示所有記錄皆已處理，一直處理到傳遞至記錄處理器的最後一筆記錄。因此，記錄處理器應僅在已處理過向其傳遞的清單中之所有記錄後才呼叫 checkpoint。記錄處理器不需要在每次呼叫 checkpoint 時呼叫 processRecords。例如，處理器可以每呼叫三次該函數才呼叫一次 checkpoint，或於記錄處理器外部發生事件時呼叫 (比方您已實作的自訂確認/驗證服務)。

您可以選擇性指定某筆記錄的確切序號做為 checkpoint 的參數。在此情況下，KCL 將假定所有記錄皆已處理，僅止於處理到該記錄。

基本範例應用程式示範了最簡單可行的方式呼叫 checkpointer.checkpoint 函數。此時您可以在該函數中為您的消費者加入其他所需的檢查點邏輯。

shutdown

KCL 會在處理結束 (shutdownInput.reason 為 TERMINATE) 或工作者不再回應 (shutdownInput.reason 為 ZOMBIE) 時呼叫 shutdown 函數。

```
shutdown: function(shutdownInput, completeCallback)
```

當記錄處理器未能再從碎片接收任何記錄 (因為碎片已進行分割或合併或者串流已刪除) 時，處理即告結束。

KCL 還會將 shutdownInput.checkpointer 物件傳遞給 shutdown。如果關閉原因是 TERMINATE，您即應確保記錄處理器已完成處理任何資料記錄，然後對此界面呼叫 checkpoint 函數。

修改組態屬性

範例提供了組態屬性的預設值。您可使用自訂值覆寫任何這些屬性 (請參閱基本範例中的 sample.properties)。

Application name (應用程式名稱)

KCL 要求所有應用程式和同一區域內的 Amazon DynamoDB 資料表必須具有獨一無二的應用程式。其使用應用程式名稱組態值的方式如下：

- 假定所有與此應用程式名稱相關聯的工作者合作處理同一串流。這些工作者可能分佈於多個執行個體。如果您以相同應用程式的程式碼執行另一執行個體但使用不同的應用程式名稱，KCL 便會將第二個執行個體視為亦對同一串流進行操作的完全獨立應用程式。
- KCl 將使用應用程式名稱建立 DynamoDB 資料表並由該資料表維護應用程式的狀態資訊 (例如檢查點及工作者與碎片間對應)。每個應用程式都有其自身的 DynamoDB 資料表。如需詳細資訊，請參閱使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片。

設定登入資料

您必須將 AWS 登入資料提供給預設登入資料提供者鏈結中的其中一個登入資料提供者。您可以使用 AWSCredentialsProvider 屬性，設定登入資料供應者。sample.properties 檔案必須向預設登入資料供應者鏈結中的某一登入資料供應者提供您的登入資料。如果您是在 Amazon EC2 執行個體上執行取用者，建議您使用 IAM 角色來設定執行個體。反映與此 IAM 角色相關聯許可 AWS 的憑證可透過執行個體中繼資料提供給執行個體上的應用程式。以這種方式管理 EC2 執行個體上執行的消費者應用程式的登入資料最為安全。

以下範例設定 KCL 使用 sample_kcl_app.js 中提供的記錄處理器來處理名為 kclnodejssample 的 Kinesis 資料串流。

```
# The Node.js executable script
executableName = node sample_kcl_app.js
# The name of an Amazon Kinesis stream to process
```

```
streamName = kclnodejssample
# Unique KCL application name
applicationName = kclnodejssample
# Use default AWS credentials provider chain
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain
# Read from the beginning of the stream
initialPositionInStream = TRIM_HORIZON
```

在 .NET 中開發 Kinesis Client Library 取用者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 Kinesis Client Library (KCL) 建置應用程式，處理來自 Kinesis 資料串流的資料。Kinesis Client Library 支援多種語言。本主題將討論 .NET。

KCL 是一種 Java 程式庫，使用稱為 MultiLangDaemon 的多語言介面提供對 Java 以外語言的支援。此常駐程式是以 Java 為基礎，並在您使用 Java 以外的 KCL 語言時在背景執行。因此，若您安裝了適用於 .NET 的 KCL 並完全以 .NET 撰寫取用者應用程式，則由於 MultiLangDaemon 的緣故，您的系統仍需要安裝 Java。此外，MultiLangDaemon 有一些預設設定，您可能需要針對您的使用案例進行自訂，例如其連線 AWS 的區域。如需 MultiLangDaemon 的詳細資訊，請前往 GitHub 上的 KCL MultiLangDaemon 專案頁面。

若要從 GitHub 下載 .NET KCL，請前往 Kinesis Client Library (.NET)。如需下載 .NET KCL 取用者應用程式的範本程式碼，請至 GitHub 前往適用於 .NET 的 KCL 範例取用者專案頁面。

以 .NET 實作 KCL 取用者應用程式時，您必須完成以下任務：

任務

- 實作 IRecordProcessor 類別方法
- 修改組態屬性

實作 IRecordProcessor 類別方法

消費者必須實作 IRecordProcessor 的下列方法。範例消費者提供的實作可讓您用於做為起點 (請參閱 SampleRecordProcessor 中的 SampleConsumer/AmazonKinesisSampleConsumer.cs 類別)。

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

初始化

KCL 將於記錄處理器執行個體化時呼叫此方法，透過 input 參數 (input.ShardId) 傳遞特定碎片 ID。此記錄處理器只會處理該碎片，且通常反過來說同樣成立 (該碎片僅由此記錄處理器處理)。然而，您的消費者應該考慮到資料記錄可能經過多次處理的情況。這是因為 Kinesis Data Streams 具有至少一次的語意，即碎片中的每一筆資料記錄至少會由取用者內的工作者處理一次。如需特定碎片可能由多個工作者處理之各種情況的詳細資訊，請參閱使用重新分片、擴展和平行處理來變更碎片數量。

```
public void Initialize(InitializationInput input)
```

ProcessRecords

input 會呼叫此方法，透過 input.Records 參數 () 傳遞由 Initialize 方法所指定碎片中之資料記錄的清單。您所實作的記錄處理器根據消費者的語意處理這些記錄中的資料。例如，工作者可能會執行資料轉換，然後將結果存放至 Amazon Simple Storage Service (Amazon S3) 儲存貯體。

```
public void ProcessRecords(ProcessRecordsInput input)
```

除了資料本身外，記錄還包含序號和分割區索引鍵。工作者在處理資料時可使用這些值。例如，工作者可根據分割區索引鍵的值，選擇要存放資料的 S3 儲存貯體。Record 類別公開了以下項目，可供存取記錄的資料、序號和分割區索引鍵：

```
byte[] Record.Data
string Record.SequenceNumber
string Record.PartitionKey
```

範例中，ProcessRecordsWithRetries 方法的程式碼示範了工作者如何能夠存取記錄的資料、序號和分割區索引鍵。

Kinesis Data Streams 需要由記錄處理器追蹤碎片中已經處理過的記錄。KCL 透過將 Checkpointer 物件傳遞給 ProcessRecords (input.Checkpointer) 為您進行這項追蹤。記錄處理器將呼叫 Checkpointer.Checkpoint 方法，以通知 KCL 目前處理碎片中的記錄之進度。如果工作者發生失敗，KCL 將使用此資訊於上一筆已知處理過的記錄處重新啟動碎片處理。

對於分割或合併操作，在原始碎片的處理器呼叫 Checkpointer.Checkpoint 以表示對原始碎片進行所有處理都已完成之前，KCL 不會開始處理新碎片。

如果您未傳遞參數，KCL 將假定對 Checkpointer.Checkpoint 的呼叫代表所有記錄皆已處理，一直處理到傳遞至記錄處理器的最後一筆記錄。因此，記錄處理器應僅在已處理過向其傳遞的清單中之所有記錄後才呼叫 Checkpointer.Checkpoint。記錄處理器不需要在每次呼叫 Checkpointer.Checkpoint 時呼叫 ProcessRecords。例如，處理器可以每呼叫三次或四次該方法才呼叫一次 Checkpointer.Checkpoint。您可以選擇性指定某筆記錄的確切序號做為 Checkpointer.Checkpoint 的參數。在此情況下，KCL 將假定各記錄皆已處理，僅止於處理到該記錄。

範例中，私有方法 Checkpoint(Checkpointer checkpointer) 示範了如何利用適當的例外狀況處理和重試邏輯來呼叫 Checkpointer.Checkpoint 方法。

適用於 .NET 的 KCL 處理例外狀況的方式有別於其他 KCL 語言程式庫，其並不會處理任何因處理資料記錄而引發的例外狀況。使用者程式碼未捕捉的任何例外狀況都將導致程式當機。

Shutdown

KCL 會在處理結束 (關閉原因是 TERMINATE) 或工作者不再回應 (關閉 input.Reason 值為 ZOMBIE) 時呼叫 Shutdown 方法。

```
public void Shutdown(ShutdownInput input)
```

當記錄處理器未能再從碎片接收任何記錄 (因為碎片已進行分割或合併或者串流已刪除) 時，處理即告結束。

KCL 還會將 Checkpointer 物件傳遞給 shutdown。如果關閉原因是 TERMINATE，表示記錄處理器應已完成處理任何資料記錄，然後對此界面呼叫 checkpoint 方法。

修改組態屬性

範例消費者提供了組態屬性的預設值。您可使用自訂值覆寫任何這些屬性 (請參閱 SampleConsumer/kcl.properties)。

Application name (應用程式名稱)

KCL 要求所有應用程式和同一區域內的 Amazon DynamoDB 資料表必須具有獨一無二的應用程式。其使用應用程式名稱組態值的方式如下：

- 假定所有與此應用程式名稱相關聯的工作者合作處理同一串流。這些工作者可能分佈於多個執行個體。如果您以相同應用程式的程式碼執行另一執行個體但使用不同的應用程式名稱，KCL 便會將第二個執行個體視為亦對同一串流進行操作的完全獨立應用程式。
- KCl 將使用應用程式名稱建立 DynamoDB 資料表並由該資料表維護應用程式的狀態資訊 (例如檢查點及工作者與碎片間對應)。每個應用程式都有其自身的 DynamoDB 資料表。如需詳細資訊，請參閱使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片。

設定登入資料

您必須將 AWS 登入資料提供給預設登入資料提供者鏈結中的其中一個登入資料提供者。您可以使用 `AWSCredentialsProvider` 屬性，設定登入資料供應者。sample.properties 必須向預設登入資料供應者鏈結中的某一登入資料供應者提供您的登入資料。如果您在 EC2 執行個體上執行取用者應用程式，建議您使用 IAM 角色設定執行個體。反映與此 IAM 角色相關聯許可的 AWS 憑證，可透過執行個體中繼資料提供給執行個體上的應用程式。以這種方式管理 EC2 執行個體上執行的消費者登入資料最為安全。

範例的屬性檔案設定由 KCL 使用 `AmazonKinesisSampleConsumer.cs` 所提供的記錄處理器，處理名為 "words" 的 Kinesis 資料串流。

在 Python 中開發 Kinesis Client Library 取用者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis Client Library 頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 Kinesis Client Library (KCL) 建置應用程式，處理來自 Kinesis 資料串流的資料。Kinesis Client Library 支援多種語言。本主題將討論 Python。

KCL 是一種 Java 程式庫，使用稱為 MultiLangDaemon 的多語言介面提供對 Java 以外語言的支援。此常駐程式是以 Java 為基礎，並在您使用 Java 以外的 KCL 語言時在背景執行。因此，若您安裝了適用於 Python 的 KCL 並完全以 Python 撰寫取用者應用程式，則由於 MultiLangDaemon 的緣故，您的系統仍需要安裝 Java。此外，MultiLangDaemon 有一些預設設定，您可能需要針對您的使用案例進行自訂，例如其連線 AWS 的區域。如需 MultiLangDaemon 的詳細資訊，請前往 GitHub 上的 KCL MultiLangDaemon 專案頁面。

若要從 GitHub 下載 Python KCL，請前往 Kinesis Client Library (Python)。如需下載 Python KCL 取用者應用程式的範本程式碼，請至 GitHub 前往適用於 Python 的 KCL 範例專案頁面。

以 Python 實作 KCL 取用者應用程式時，您必須完成以下任務：

任務

- 實作 RecordProcessor 類別方法
- 修改組態屬性

實作 RecordProcessor 類別方法

`RecordProcess` 類別必須擴充 `RecordProcessorBase` 以實作下列方法。範例提供的實作可讓您用於做為起點 (請參閱 `sample_kclpy_app.py`)。

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

initialize

KCL 將於記錄處理器執行個體化時呼叫 `initialize` 方法，傳遞特定碎片 ID 作為參數。此記錄處理器只會處理該碎片，且通常反過來說同樣成立 (該碎片僅由此記錄處理器處理)。然而，您的消費者應該考慮到資料記錄可能經過多次處理的情況。這是因為 Kinesis Data Streams 具有至少一次的語意，即碎片中的每一筆資料記錄至少會由取用者內的工作者處理一次。如需特定碎片可能由多個工作者處理之各種情況的詳細資訊，請參閱使用重新分片、擴展和平行處理來變更碎片數量。

```
def initialize(self, shard_id)
```

process_records

KCL 會呼叫此方法，傳遞由 initialize 方法所指定碎片中之資料記錄的清單。您所實作的記錄處理器根據消費者的語意處理這些記錄中的資料。例如，工作者可能會執行資料轉換，然後將結果存放至 Amazon Simple Storage Service (Amazon S3) 儲存貯體。

```
def process_records(self, records, checkpointer)
```

除了資料本身外，記錄還包含序號和分割區索引鍵。工作者在處理資料時可使用這些值。例如，工作者可根據分割區索引鍵的值，選擇要存放資料的 S3 儲存貯體。record 字典公開了以下的索引鍵值組，可供存取記錄的資料、序號和分割區索引鍵：

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

請注意，資料為 Base64 編碼。

範例中，process_records 方法的程式碼示範了工作者如何能夠存取記錄的資料、序號和分割區索引鍵。

Kinesis Data Streams 需要由記錄處理器追蹤碎片中已經處理過的記錄。KCL 透過將 Checkpointer 物件傳遞給 process_records 為您進行這項追蹤。記錄處理器將對此物件呼叫 checkpoint 方法，以通知 KCL 目前處理碎片中的記錄之進度。如果工作者發生失敗，KCL 將使用此資訊於上一筆已知處理過的記錄處重新啟動碎片處理。

對於分割或合併操作，在原始碎片的處理器呼叫 checkpoint 以表示對原始碎片進行所有處理都已完成之前，KCL 不會開始處理新碎片。

如果您未傳遞參數，KCL 將假定對 checkpoint 的呼叫表示所有記錄皆已處理，一直處理到傳遞至記錄處理器的最後一筆記錄。因此，記錄處理器應僅在已處理過向其傳遞的清單中之所有記錄後才呼叫 checkpoint。記錄處理器不需要在每次呼叫 checkpoint 時呼叫 process_records。例如，處理器可以每呼叫三次該方法才呼叫一次 checkpoint。您可以選擇性指定某筆記錄的確切序號做為 checkpoint 的參數。在此情況下，KCL 將假定所有記錄皆已處理，僅止於處理到該記錄。

範例中，私有方法 checkpoint 示範了如何利用適當的例外狀況處理和重試邏輯來呼叫 Checkpointer.checkpoint 方法。

KCL 倚賴 process_records 以處理任何因處理資料記錄而引發的例外狀況。如果 process_records 擲回例外狀況，KCL 將略過例外狀況發生前已傳遞至 process_records 的資料記錄。也就是說，這些記錄不會重新傳送到擲回例外狀況的記錄處理器或消費者內的任何其他記錄處理器。

shutdown

KCL 會在處理結束 (關閉原因是 TERMINATE) 或工作者不再回應 (關閉 reason 為 ZOMBIE) 時呼叫 shutdown 方法。

```
def shutdown(self, checkpointer, reason)
```

當記錄處理器未能再從碎片接收任何記錄 (因為碎片已進行分割或合併或者串流已刪除) 時，處理即告結束。

KCL 還會將 Checkpointer 物件傳遞給 shutdown。如果關閉 reason 是 TERMINATE，表示記錄處理器應已完成處理任何資料記錄，然後對此界面呼叫 checkpoint 方法。

修改組態屬性

範例提供了組態屬性的預設值。您可使用自訂值覆寫任何這些屬性 (請參閱 sample.properties)。

Application name (應用程式名稱)

KCL 要求所有應用程式和同一區域內的 Amazon DynamoDB 資料表必須具有獨一無二的應用程式名稱。其使用應用程式名稱組態值的方式如下：

- 假定所有與此應用程式名稱相關聯的工作者合作處理同一串流。這些工作者可分佈於多個執行個體。如果您以相同應用程式的程式碼執行另一執行個體但使用不同的應用程式名稱，KCL 便會將第二個執行個體視為亦對同一串流進行操作的完全獨立應用程式。
- KCl 將使用應用程式名稱建立 DynamoDB 資料表並由該資料表維護應用程式的狀態資訊 (例如檢查點及工作者與碎片間對應)。每個應用程式都有其自身的 DynamoDB 資料表。如需詳細資訊，請參閱使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片。

設定登入資料

您必須將 AWS 登入資料提供給預設登入資料提供者鏈結中的其中一個登入資料提供者。您可以使用 AWSCredentialsProvider 屬性，設定登入資料供應者。sample.properties 必須向預設登入資料供應者鏈結中的某一登入資料供應者提供您的登入資料。如果您在 Amazon EC2 執行個體上執行取用者應用程式，建議您使用 IAM 角色設定執行個體。反映與此 IAM 角色相關聯許可的 AWS 憑證，可透過執行個體中繼資料提供給執行個體上的應用程式。以這種方式管理 EC2 執行個體上執行的消費者應用程式的登入資料最為安全。

範例的屬性檔案設定由 KCL 使用 sample_kclpy_app.py 所提供的記錄處理器，處理名為 "words" 的 Kinesis 資料串流。

在 Ruby 中開發 Kinesis Client Library 消費者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 Kinesis Client Library (KCL) 建置應用程式，處理來自 Kinesis 資料串流的資料。Kinesis Client Library 支援多種語言。本主題將討論 Ruby。

KCL 是一種 Java 程式庫，使用稱為 MultiLangDaemon 的多語言介面提供對 Java 以外語言的支援。此常駐程式是以 Java 為基礎，並在您使用 Java 以外的 KCL 語言時在背景執行。因此，若您安裝了適用於 Ruby 的 KCL 並完全以 Ruby 撰寫取用者應用程式，則由於 MultiLangDaemon 的緣故，您的系統仍需要安裝 Java。此外，MultiLangDaemon 有一些預設設定，您可能需要針對您的使用案例進行自訂，例如其連線 AWS 的區域。如需 MultiLangDaemon 的詳細資訊，請前往 GitHub 上的 KCL MultiLangDaemon 專案頁面。

若要從 GitHub 下載 Ruby KCL，請前往 Kinesis Client Library (Ruby)。如需下載 Ruby KCL 取用者應用程式的範本程式碼，請至 GitHub 前往適用於 Ruby 的 KCL 範例專案頁面。

如需 KCL Ruby 支援程式庫的詳細資訊，請參閱 KCL Ruby Gems 文件。

開發 KCL 2.x 消費者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

此主題說明如何使用 2.0 版本的 Kinesis Client Library (KCL)。

如需關於 KCL 的更多資訊，請參閱使用 Kinesis Client Library 1.x 開發取用者中的概觀。

根據您要使用的選項，從下列主題中進行選擇。

主題

- 在 Java 中開發 Kinesis Client Library 取用者
- 在 Python 中開發 Kinesis Client Library 取用者
- 使用 KCL 2.x 開發增強型廣發消費者

在 Java 中開發 Kinesis Client Library 取用者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis Client Library 頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

以下程式碼顯示 `ProcessorFactory` 和 `RecordProcessor` 的 Java 範例實作。如果您想要利用增強型散發功能，請參閱使用具有增強型散發功能的取用者。

```
/*
 *  Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 *  Licensed under the Amazon Software License (the "License").
 *  You may not use this file except in compliance with the License.
 *  A copy of the License is located at
 *
 *  http://aws.amazon.com/asl/
 *
 *  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.
 */


/*
 * Copyright 2019 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
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * 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 java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
```

```
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.retrieval.polling.PollingConfig;

/**
 * This class will run a simple app that uses the KCL to read data and uses the AWS SDK
 to publish data.
 * Before running this program you must first create a Kinesis stream through the AWS
 console or AWS SDK.
 */
public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    /**
     * Invoke the main method with 2 args: the stream name and (optionally) the region.
     * Verifies valid inputs and then starts running the app.
     */
    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
 The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    /**
```

```
    * Constructor sets streamName and region. It also creates a KinesisClient object
to send data to Kinesis.
    * This KinesisClient is used to send dummy data so that the consumer has something
to read; it is also used
    * indirectly by the KCL to handle the consumption of the data.
    */
   private SampleSingle(String streamName, String region) {
       this.streamName = streamName;
       this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
       this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
   }

   private void run() {

       /**
        * Sends dummy data to Kinesis. Not relevant to consuming the data with the KCL
        */
       ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
       ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

       /**
        * Sets up configuration for the KCL, including DynamoDB and CloudWatch
dependencies. The final argument, a
        * ShardRecordProcessorFactory, is where the logic for record processing lives,
and is located in a private
        * class below.
        */
       DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
       CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
       ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

       /**
        * The Scheduler (also called Worker in earlier versions of the KCL) is the
entry point to the KCL. This
        * instance is configured with defaults provided by the ConfigsBuilder.
        */
       Scheduler scheduler = new Scheduler(
```

```
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient))
        );

        /**
         * Kickoff the Scheduler. Record processing of the stream of dummy data will
continue indefinitely
         * until an exit is triggered.
         */
        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();

        /**
         * Allows termination of app by pressing Enter.
         */
        System.out.println("Press enter to shutdown");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            reader.readLine();
        } catch (IOException ioex) {
            log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
        }

        /**
         * Stops sending dummy data.
         */
        log.info("Cancelling producer and shutting down executor.");
        producerFuture.cancel(true);
        producerExecutor.shutdownNow();

        /**
         * Stops consuming data. Finishes processing the current batch of data already
received from Kinesis
         * before shutting down.
         */
        Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
```

```
        log.info("Waiting up to 20 seconds for shutdown to complete.");
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown.  Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    /**
     * Sends a single record of dummy data to Kinesis.
     */
    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            log.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }

    private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
        public ShardRecordProcessor shardRecordProcessor() {
            return new SampleRecordProcessor();
        }
    }

    /**
     * The implementation of the ShardRecordProcessor interface is where the heart of
the record processing logic lives.
     * In this example all we do to 'process' is log info about the records.
```

```
     */
    private static class SampleRecordProcessor implements ShardRecordProcessor {

        private static final String SHARD_ID_MDC_KEY = "ShardId";

        private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

        private String shardId;

        /**
         * Invoked by the KCL before data records are delivered to the
ShardRecordProcessor instance (via
         * processRecords). In this example we do nothing except some logging.
         *
         * @param initializationInput Provides information related to initialization.
         */
        public void initialize(InitializationInput initializationInput) {
            shardId = initializationInput.shardId();
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Handles record processing logic. The Amazon Kinesis Client Library will
invoke this method to deliver
         * data records to the application. In this example we simply log our records.
         *
         * @param processRecordsInput Provides the records to be processed as well as
information and capabilities
         *                            related to them (e.g. checkpointing).
         */
        public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
processRecordsInput.records().size());
                processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
```

```
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /** Called when the lease tied to this record processor has been lost. Once the
lease has been lost,
         * the record processor can no longer checkpoint.
         *
         * @param leaseLostInput Provides access to functions and data related to the
loss of the lease.
         */
        public void leaseLost(LeaseLostInput leaseLostInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Lost lease, so terminating.");
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Called when all data on this shard has been processed. Checkpointing must
occur in the method for record
         * processing to be considered complete; an exception will be thrown otherwise.
         *
         * @param shardEndedInput Provides access to a checkpointer method for
completing processing of the shard.
         */
        public void shardEnded(ShardEndedInput shardEndedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Reached shard end checkpointing.");
                shardEndedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at shard end. Giving up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }
```

```
        /**
         * Invoked when Scheduler has been requested to shut down (i.e. we decide to
 stop running the app by pressing
         * Enter). Checkpoints and logs the data a final time.
         *
         * @param shutdownRequestedInput Provides access to a checkpointer, allowing a
 record processor to checkpoint
         *                               before the shutdown is completed.
         */
        public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Scheduler is shutting down, checkpointing.");
                shutdownRequestedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }
    }

}
```

在 Python 中開發 Kinesis Client Library 取用者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 Kinesis Client Library (KCL) 建置應用程式，處理來自 Kinesis 資料串流的資料。Kinesis Client Library 支援多種語言。本主題將討論 Python。

KCL 是一種 Java 程式庫，使用稱為 MultiLangDaemon 的多語言介面提供對 Java 以外語言的支援。此常駐程式是以 Java 為基礎，並在您使用 Java 以外的 KCL 語言時在背景執行。因此，若您安裝了

適用於 Python 的 KCL 並完全以 Python 撰寫取用者應用程式，則由於 MultiLangDaemon 的緣故，您的系統仍需要安裝 Java。此外，MultiLangDaemon 有一些預設設定，您可能需要針對您的使用案例進行自訂，例如其連線 AWS 的區域。如需 MultiLangDaemon 的詳細資訊，請前往 GitHub 上的 KCL MultiLangDaemon 專案頁面。

若要從 GitHub 下載 Python KCL，請前往 Kinesis Client Library (Python)。如需下載 Python KCL 取用者應用程式的範本程式碼，請至 GitHub 前往適用於 Python 的 KCL 範例專案頁面。

以 Python 實作 KCL 取用者應用程式時，您必須完成以下任務：

任務

- 實作 RecordProcessor 類別方法
- 修改組態屬性

實作 RecordProcessor 類別方法

`RecordProcess` 類別必須擴充 `RecordProcessorBase` 類別以實作下列方法。

```
initialize
process_records
shutdown_requested
```

範例提供的實作可讓您用於做為起點。

```
#!/usr/bin/env python

# Copyright 2014-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Amazon Software License (the "License").
# You may not use this file except in compliance with the License.
# A copy of the License is located at
#
# http://aws.amazon.com/asl/
#
# 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.

from __future__ import print_function
```

```
import sys
import time

from amazon_kclpy import kcl
from amazon_kclpy.v3 import processor


class RecordProcessor(processor.RecordProcessorBase):
    """
    A RecordProcessor processes data from a shard in a stream. Its methods will be
 called with this pattern:

    * initialize will be called once
    * process_records will be called zero or more times
    * shutdown will be called if this MultiLangDaemon instance loses the lease to this
 shard, or the shard ends due
        a scaling change.
    """
    def __init__(self):
        self._SLEEP_SECONDS = 5
        self._CHECKPOINT_RETRIES = 5
        self._CHECKPOINT_FREQ_SECONDS = 60
        self._largest_seq = (None, None)
        self._largest_sub_seq = None
        self._last_checkpoint_time = None

    def log(self, message):
        sys.stderr.write(message)

    def initialize(self, initialize_input):
        """
        Called once by a KCLProcess before any calls to process_records

        :param amazon_kclpy.messages.InitializeInput initialize_input: Information
 about the lease that this record
            processor has been assigned.
        """
        self._largest_seq = (None, None)
        self._last_checkpoint_time = time.time()

    def checkpoint(self, checkpointer, sequence_number=None, sub_sequence_number=None):
        """
        Checkpoints with retries on retryable exceptions.
```

```
        :param amazon_kclpy.kcl.Checkpointer checkpointer: the checkpointer provided to
either process_records
            or shutdown
        :param str or None sequence_number: the sequence number to checkpoint at.
        :param int or None sub_sequence_number: the sub sequence number to checkpoint
at.
        """
        for n in range(0, self._CHECKPOINT_RETRIES):
            try:
                checkpointer.checkpoint(sequence_number, sub_sequence_number)
                return
            except kcl.CheckpointError as e:
                if 'ShutdownException' == e.value:
                    #
                    # A ShutdownException indicates that this record processor should
be shutdown. This is due to
                    # some failover event, e.g. another MultiLangDaemon has taken the
lease for this shard.
                    #
                    print('Encountered shutdown exception, skipping checkpoint')
                    return
                elif 'ThrottlingException' == e.value:
                    #
                    # A ThrottlingException indicates that one of our dependencies is
is over burdened, e.g. too many
                    # dynamo writes. We will sleep temporarily to let it recover.
                    #
                    if self._CHECKPOINT_RETRIES - 1 == n:
                        sys.stderr.write('Failed to checkpoint after {n} attempts,
giving up.\n'.format(n=n))
                        return
                    else:
                        print('Was throttled while checkpointing, will attempt again in
{s} seconds'
                              .format(s=self._SLEEP_SECONDS))
                elif 'InvalidStateException' == e.value:
                    sys.stderr.write('MultiLangDaemon reported an invalid state while
checkpointing.\n')
                else:  # Some other error
                    sys.stderr.write('Encountered an error while checkpointing, error
was {e}.\n'.format(e=e))
            time.sleep(self._SLEEP_SECONDS)
```

```
   def process_record(self, data, partition_key, sequence_number,
sub_sequence_number):
       """
       Called for each record that is passed to process_records.

       :param str data: The blob of data that was contained in the record.
       :param str partition_key: The key associated with this recod.
       :param int sequence_number: The sequence number associated with this record.
       :param int sub_sequence_number: the sub sequence number associated with this
record.
       """
       ####################################
       # Insert your processing logic here
       ####################################
       self.log("Record (Partition Key: {pk}, Sequence Number: {seq}, Subsequence
Number: {sseq}, Data Size: {ds}"
                .format(pk=partition_key, seq=sequence_number,
sseq=sub_sequence_number, ds=len(data)))

   def should_update_sequence(self, sequence_number, sub_sequence_number):
       """
       Determines whether a new larger sequence number is available

       :param int sequence_number: the sequence number from the current record
       :param int sub_sequence_number: the sub sequence number from the current record
       :return boolean: true if the largest sequence should be updated, false
otherwise
       """
       return self._largest_seq == (None, None) or sequence_number >
self._largest_seq[0] or \
           (sequence_number == self._largest_seq[0] and sub_sequence_number >
self._largest_seq[1])

   def process_records(self, process_records_input):
       """
       Called by a KCLProcess with a list of records to be processed and a
checkpointer which accepts sequence numbers
       from the records to indicate where in the stream to checkpoint.

       :param amazon_kclpy.messages.ProcessRecordsInput process_records_input: the
records, and metadata about the
           records.
       """
       try:
```

```
            for record in process_records_input.records:
                data = record.binary_data
                seq = int(record.sequence_number)
                sub_seq = record.sub_sequence_number
                key = record.partition_key
                self.process_record(data, key, seq, sub_seq)
                if self.should_update_sequence(seq, sub_seq):
                    self._largest_seq = (seq, sub_seq)

            #
            # Checkpoints every self._CHECKPOINT_FREQ_SECONDS seconds
            #
            if time.time() - self._last_checkpoint_time >
 self._CHECKPOINT_FREQ_SECONDS:
                self.checkpoint(process_records_input.checkpointer,
 str(self._largest_seq[0]), self._largest_seq[1])
                self._last_checkpoint_time = time.time()

        except Exception as e:
            self.log("Encountered an exception while processing records. Exception was
 {e}\n".format(e=e))

    def lease_lost(self, lease_lost_input):
        self.log("Lease has been lost")

    def shard_ended(self, shard_ended_input):
        self.log("Shard has ended checkpointing")
        shard_ended_input.checkpointer.checkpoint()

    def shutdown_requested(self, shutdown_requested_input):
        self.log("Shutdown has been requested, checkpointing.")
        shutdown_requested_input.checkpointer.checkpoint()


if __name__ == "__main__":
    kcl_process = kcl.KCLProcess(RecordProcessor())
    kcl_process.run()
```

修改組態屬性

範例提供了組態屬性的預設值，如下列指令碼所示。您可使用自訂值覆寫任何這些屬性。

```
# The script that abides by the multi-language protocol. This script will
# be executed by the MultiLangDaemon, which will communicate with this script
```

```
# over STDIN and STDOUT according to the multi-language protocol.
executableName = sample_kclpy_app.py

# The name of an Amazon Kinesis stream to process.
streamName = words

# Used by the KCL as the name of this application. Will be used as the name
# of an Amazon DynamoDB table which will store the lease and checkpoint
# information for workers with this application name
applicationName = PythonKCLSample

# Users can change the credentials provider the KCL will use to retrieve credentials.
# The DefaultAWSCredentialsProviderChain checks several other providers, which is
# described here:
# http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/auth/
DefaultAWSCredentialsProviderChain.html
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain

# Appended to the user agent of the KCL. Does not impact the functionality of the
# KCL in any other way.
processingLanguage = python/2.7

# Valid options at TRIM_HORIZON or LATEST.
# See http://docs.aws.amazon.com/kinesis/latest/APIReference/
API_GetShardIterator.html#API_GetShardIterator_RequestSyntax
initialPositionInStream = TRIM_HORIZON

# The following properties are also available for configuring the KCL Worker that is
 created
# by the MultiLangDaemon.

# The KCL defaults to us-east-1
#regionName = us-east-1

# Fail over time in milliseconds. A worker which does not renew it's lease within this
 time interval
# will be regarded as having problems and it's shards will be assigned to other
 workers.
# For applications that have a large number of shards, this msy be set to a higher
 number to reduce
# the number of DynamoDB IOPS required for tracking leases
#failoverTimeMillis = 10000
```

```
# A worker id that uniquely identifies this worker among all workers using the same
 applicationName
# If this isn't provided a MultiLangDaemon instance will assign a unique workerId to
 itself.
#workerId =

# Shard sync interval in milliseconds - e.g. wait for this long between shard sync
 tasks.
#shardSyncIntervalMillis = 60000

# Max records to fetch from Kinesis in a single GetRecords call.
#maxRecords = 10000

# Idle time between record reads in milliseconds.
#idleTimeBetweenReadsInMillis = 1000

# Enables applications flush/checkpoint (if they have some data "in progress", but
 don't get new data for while)
#callProcessRecordsEvenForEmptyRecordList = false

# Interval in milliseconds between polling to check for parent shard completion.
# Polling frequently will take up more DynamoDB IOPS (when there are leases for shards
 waiting on
# completion of parent shards).
#parentShardPollIntervalMillis = 10000

# Cleanup leases upon shards completion (don't wait until they expire in Kinesis).
# Keeping leases takes some tracking/resources (e.g. they need to be renewed,
 assigned), so by default we try
# to delete the ones we don't need any longer.
#cleanupLeasesUponShardCompletion = true

# Backoff time in milliseconds for Amazon Kinesis Client Library tasks (in the event of
 failures).
#taskBackoffTimeMillis = 500

# Buffer metrics for at most this long before publishing to CloudWatch.
#metricsBufferTimeMillis = 10000

# Buffer at most this many metrics before publishing to CloudWatch.
#metricsMaxQueueSize = 10000

# KCL will validate client provided sequence numbers with a call to Amazon Kinesis
 before checkpointing for calls
```

```
# to RecordProcessorCheckpointer#checkpoint(String) by default.
#validateSequenceNumberBeforeCheckpointing = true

# The maximum number of active threads for the MultiLangDaemon to permit.
# If a value is provided then a FixedThreadPool is used with the maximum
# active threads set to the provided value. If a non-positive integer or no
# value is provided a CachedThreadPool is used.
#maxActiveThreads = 0
```

Application name (應用程式名稱)

KCL 要求所有應用程式和同一區域內的 Amazon DynamoDB 資料表必須具有獨一無二的應用程式名稱。其使用應用程式名稱組態值的方式如下：

- 假定所有與此應用程式名稱相關聯的工作者合作處理同一串流。這些工作者可分佈於多個執行個體。如果您以相同應用程式的程式碼執行另一執行個體但使用不同的應用程式名稱，KCL 便會將第二個執行個體視為亦對同一串流進行操作的完全獨立應用程式。
- KCl 將使用應用程式名稱建立 DynamoDB 資料表並由該資料表維護應用程式的狀態資訊 (例如檢查點及工作者與碎片間對應)。每個應用程式都有其自身的 DynamoDB 資料表。如需詳細資訊，請參閱使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片。

憑證

您必須將 AWS 登入資料提供給預設登入資料提供者鏈結中的其中一個登入資料提供者。您可以使用 `AWSCredentialsProvider` 屬性，設定登入資料供應者。如果您在 Amazon EC2 執行個體上執行取用者應用程式，建議您使用 IAM role. AWS credentials 設定執行個體，以反映與此 IAM 角色相關聯的許可，可透過執行個體中繼資料提供給執行個體上的應用程式。以這種方式管理 EC2 執行個體上執行的消費者應用程式的登入資料最為安全。

使用 KCL 2.x 開發增強型廣發消費者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

在 Amazon Kinesis Data Streams 中使用增強型散發功能的取用者從資料串流接收記錄時，專用輸送量可高達每個碎片每秒 2 MB 的資料。這類消費者不必與其他從串流接收資料的消費者競爭。如需詳細資訊，請參閱開發具有專用輸送量的增強型廣發消費者。

您可以使用 2.0 版或更新版本的 Kinesis Client Library (KCL) 開發應用程式，利用增強型散發功能從串流接收資料。KCL 將自動為您的應用程式訂閱串流中的所有碎片，並確保您的取用者應用程式讀取的輸送量值達到每個碎片每秒 2 MB。如果您想要使用 KCL 但不想開啟增強型散發功能，請參閱使用 Kinesis Client Library 2.0 開發取用者。

主題

- 在 Java 中使用 KCL 2.x 開發增強型廣發消費者

在 Java 中使用 KCL 2.x 開發增強型廣發消費者

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

您可以使用 2.0 版或更新版本的 Kinesis Client Library (KCL) 在 Amazon Kinesis Data Streams 中開發應用程式，利用增強型散發功能從串流接收資料。以下程式碼顯示 `ProcessorFactory` 和 `RecordProcessor` 的 Java 範例實作。

建議您在 `KinesisAsyncClient` 中，使用 `KinesisClientUtil` 來建立 `KinesisAsyncClient` 及設定 `maxConcurrency`。

> **Important**
>
> Amazon Kinesis Client 可能會明顯發生延遲，除非您設定 `KinesisAsyncClient` 的 `maxConcurrency` 夠高，足以運作所有的租賃服務，並可額外使用 `KinesisAsyncClient`。

```
/*
```

```
 *  Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 *  Licensed under the Amazon Software License (the "License").
 *  You may not use this file except in compliance with the License.
 *  A copy of the License is located at
 *
 *  http://aws.amazon.com/asl/
 *
 *  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.
 */

/*
 * Copyright 2019 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
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * 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 java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
 The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
```

```
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    private SampleSingle(String streamName, String region) {
        this.streamName = streamName;
        this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
        this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    private void run() {
        ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
        ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

        DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
        ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

        Scheduler scheduler = new Scheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                configsBuilder.retrievalConfig()
        );

        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();

        System.out.println("Press enter to shutdown");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            reader.readLine();
        } catch (IOException ioex) {
```

```
            log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
        }

        log.info("Cancelling producer, and shutting down executor.");
        producerFuture.cancel(true);
        producerExecutor.shutdownNow();

        Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
        log.info("Waiting up to 20 seconds for shutdown to complete.");
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            log.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }

    private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
        public ShardRecordProcessor shardRecordProcessor() {
            return new SampleRecordProcessor();
        }
```

```
    }


    private static class SampleRecordProcessor implements ShardRecordProcessor {

        private static final String SHARD_ID_MDC_KEY = "ShardId";

        private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

        private String shardId;

        public void initialize(InitializationInput initializationInput) {
            shardId = initializationInput.shardId();
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
processRecordsInput.records().size());
                processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void leaseLost(LeaseLostInput leaseLostInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Lost lease, so terminating.");
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
```

```
            }
        }

        public void shardEnded(ShardEndedInput shardEndedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Reached shard end checkpointing.");
                shardEndedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at shard end. Giving up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Scheduler is shutting down, checkpointing.");
                shutdownRequestedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }
    }

}
```

將消費者從 KCL 1.x 遷移至 KCL 2.x

> **Important**
>
> Amazon Kinesis Client Library (KCL) 版本 1.x 和 2.x 已過期。KCL 1.x 將於 2026 年 1 月 30 日終止支援。我們強烈建議您使用 1.x 版將 KCL 應用程式遷移至 2026 年 1 月 30 日之前的最新 KCL 版本。若要尋找最新的 KCL 版本，請參閱 GitHub 上的 Amazon Kinesis 用戶端程式庫頁面。如需最新 KCL 版本的資訊，請參閱 使用 Kinesis 用戶端程式庫。如需從 KCL 1.x 遷移至 KCL 3.x 的資訊，請參閱 從 KCL 1.x 移轉到 KCL 3.x。

此主題說明 Kinesis Client Library (KCL) 版本 1.x 和 2.x 之間的差異。她還說明如何將取用者從 KCL 的版本 1.x 遷移至版本 2.x。遷移用戶端後，該用戶端會從前一個檢查點的位置開始處理記錄。

2.0 版 KCL 引進了以下的介面變更：

KCL 介面變更

KCL 1.x 介面	KCL 2.0 介面
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor`	`software.amazon.kinesis.processor.ShardRecordProcessor`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory`	`software.amazon.kinesis.processor.ShardRecordProcessorFactory`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware`	整併至 `software.amazon.kinesis.processor.ShardRecordProcessor`

主題

- 遷移記錄處理器
- 遷移記錄處理器工廠
- 遷移工作者
- 設定 Amazon Kinesis 用戶端
- 閒置時間移除
- 用戶端組態移除

遷移記錄處理器

以下範例顯示基於 KCL 1.x 所實作的記錄處理器：

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
```

```
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class TestRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        //
        // Process records, and possibly checkpoint
        //
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
```

```
            //
            e.printStackTrace();
        }
    }
}
```

遷移記錄處理器類別

1. 將介面從 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor 和 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat 更改為 software.amazon.kinesis.processor.ShardRecordProcessor，如下所示：

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// public class TestRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
public class TestRecordProcessor implements ShardRecordProcessor {
```

2. 更新 import 和 initialize 方法的 processRecords 陳述式。

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

//import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
```

3. 將 shutdown 方法取代為以下的新方法：leaseLost、shardEnded 和 shutdownRequested。

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...)
//        //
//        try {
//            checkpointer.checkpoint();
```

```
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
```

```
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

記錄處理器類別經更新後的版本如下。

```
package com.amazonaws.kcl;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class TestRecordProcessor implements ShardRecordProcessor {
    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {

    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
```

```
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

遷移記錄處理器工廠

記錄處理器處理站負責在取得租用時建立記錄處理器。以下是 KCL 1.x 處理站的範例。

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class TestRecordProcessorFactory implements IRecordProcessorFactory {
    @Override
    public IRecordProcessor createProcessor() {
        return new TestRecordProcessor();
    }
}
```

遷移記錄處理器處理站

1. 將實作的介面從 `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa`

更改為 `software.amazon.kinesis.processor.ShardRecordProcessorFactory`，如下所示。

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
```

2. 更改 `createProcessor` 的傳回簽章。

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

以下是 2.0 版記錄處理器處理站的範例：

```
package com.amazonaws.kcl;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new TestRecordProcessor();
    }
}
```

遷移工作者

在 KCL 的版本 2.0，名為 `Scheduler` 的新類別會取代 `Worker` 類別。以下是 KCL 1.x 工作者的範例。

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
```

```
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

移轉至工作者

1. 將 Worker 類別的 import 陳述式變更為 Scheduler 和 ConfigsBuilder 類別的匯入陳述式。

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. 建立 ConfigsBuilder 和 Scheduler，如下列範例所示：

   建議您在 KinesisAsyncClient 中，使用 KinesisClientUtil 來建立 KinesisAsyncClient 及設定 maxConcurrency。

> **Important**
>
> Amazon Kinesis Client 可能會明顯發生延遲，除非您設定 KinesisAsyncClient 的 maxConcurrency 夠高，足以運作所有的租賃服務，並可額外使用 KinesisAsyncClient。

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...

Region region = Region.AP_NORTHEAST_2;
KinesisAsyncClient kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(region));
```

```
DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();

ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, applicationName,
 kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
 SampleRecordProcessorFactory());

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
    );
```

設定 Amazon Kinesis 用戶端

隨著 2.0 版 Kinesis Client Library 的推出，用戶端組態已從單一組態類別 (KinesisClientLibConfiguration) 進展為六個組態類別。下表說明遷移情形。

組態欄位及其新類別

原始欄位	新的組態類別	Description
applicationName	ConfigsBuilder	此 KCL 應用程式的名稱。用做為 tableName 和 consumerName 的預設值。
tableName	ConfigsBuilder	允許覆寫用於 Amazon DynamoDB 租用資料表的資料表名稱。
streamName	ConfigsBuilder	此應用程式從中處理其記錄的串流名稱。
kinesisEndpoint	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。

原始欄位	新的組態類別	Description
dynamoDBEndpoint	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
initialPositionInStreamExtended	RetrievalConfig	KCL 開始擷記錄所在碎片中的位置，透過應用程式初次執行開始。
kinesisCredentialsProvider	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
dynamoDBCredentialsProvider	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
cloudWatchCredentialsProvider	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
failoverTimeMillis	LeaseManagementConfig	將租用擁有者視為失敗前必須經過的毫秒數。
workerIdentifier	ConfigsBuilder	代表應用程式處理器本項實例的唯一識別符。其必須獨一無二。
shardSyncIntervalMillis	LeaseManagementConfig	碎片同步呼叫的時間。
maxRecords	PollingConfig	允許設定 Kinesis 傳回的記錄數上限。
idleTimeBetweenReadsInMillis	CoordinatorConfig	此選項已經移除。請參閱「閒置時間移除項目」一節。

原始欄位	新的組態類別	Description
callProcessRecordsEvenForEmptyRecordList	ProcessorConfig	設定時，即使 Kinesis 無提供任何記錄也會呼叫記錄處理器。
parentShardPollIntervalMillis	CoordinatorConfig	記錄處理器應該輪詢以檢查父碎片是否已完成的頻率。
cleanupLeasesUponShardCompletion	LeaseManagementConfig	設定時，只要已開始處理子租用就會隨即移除租用。
ignoreUnexpectedChildShards	LeaseManagementConfig	設定時，會忽略具有開放碎片的子碎片。此項主要用於 DynamoDB Streams。
kinesisClientConfig	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
dynamoDBClientConfig	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
cloudWatchClientConfig	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
taskBackoffTimeMillis	LifecycleConfig	重試失敗任務的等待時間。
metricsBufferTimeMillis	MetricsConfig	控制 CloudWatch 指標發佈。
metricsMaxQueueSize	MetricsConfig	控制 CloudWatch 指標發佈。

原始欄位	新的組態類別	Description
metricsLevel	MetricsConfig	控制 CloudWatch 指標發佈。
metricsEnabledDimensions	MetricsConfig	控制 CloudWatch 指標發佈。
validateSequenceNumberBeforeCheckpointing	CheckpointConfig	此選項已經移除。請參閱「檢查點序號驗證」一節。
regionName	ConfigsBuilder	此選項已經移除。請參閱「用戶端組態移除項目」一節。
maxLeasesForWorker	LeaseManagementConfig	應用程式的單一執行個體應該接受的租用數上限。
maxLeasesToStealAtOneTime	LeaseManagementConfig	應用程式一次應該嘗試挪用的租用數上限。
initialLeaseTableReadCapacity	LeaseManagementConfig	Kinesis Client Library 需要建立新的 DynamoDB 租用資料表時所使用的 DynamoDB 讀取 IOP。
initialLeaseTableWriteCapacity	LeaseManagementConfig	Kinesis Client Library 需要建立新的 DynamoDB 租用資料表時所使用的 DynamoDB 讀取 IOP。
initialPositionInStreamExtended	LeaseManagementConfig	應用程式應該在串流中開始的初始位置。這僅在初次建立租用時使用。

原始欄位	新的組態類別	Description
skipShardSyncAtWorkerInitializationIfLeasesExist	CoordinatorConfig	如果租用資料表包含現有的租用，即停用同步處理碎片資料。TODO：KinesisEco-438
shardPrioritization	CoordinatorConfig	要使用哪些碎片優先順序
shutdownGraceMillis	N/A	此選項已經移除。請參閱「MultiLang 移除項目」一節。
timeoutInSeconds	N/A	此選項已經移除。請參閱「MultiLang 移除項目」一節。
retryGetRecordsInSeconds	PollingConfig	為故障設定 GetRecords 嘗試間的延遲。
maxGetRecordsThreadPool	PollingConfig	用於 GetRecords 的執行緒集區大小。
maxLeaseRenewalThreads	LeaseManagementConfig	控制租用續約執行緒集區的大小。應用程式可容納的租用數愈多，此集區就應該愈大。
recordsFetcherFactory	PollingConfig	允許將工廠進行替換，該工廠會用於建立擷取程式，而該擷取程式會從串流進行擷取。
logWarningForTaskAfterMillis	LifecycleConfig	任務未完成的情況下要等待多久的時間才記錄警告。
listShardsBackoffTimeInMillis	RetrievalConfig	呼叫 ListShards 發生錯誤時將等待的間隔毫秒數。

原始欄位	新的組態類別	Description
maxListShardsRetryAttempts	RetrievalConfig	ListShards 在放棄之前重試的次數上限。

閒置時間移除

1.x 版 KCL 的 idleTimeBetweenReadsInMillis 對應於兩種計量：

- 任務分派檢查的間隔時間量。您現在可以透過設定 CoordinatorConfig#shardConsumerDispatchPollIntervalMillis，設定各任務的此一間隔時間。
- 當 Kinesis Data Streams 未傳回任何記錄時將休眠的時間量。在 2.0 版中，具強化廣發功能的記錄是自其各自的擷取器推送。僅當推送的請求送達時，碎片消費者才會發生活動。

用戶端組態移除

在 2.0 版中，KCL 不再建立用戶端。其端賴使用者提供有效的用戶端。基於此項變更，所有控制用戶端建立的組態參數皆已移除。若您需要這類參數，可以先就用戶端進行所需設定再將用戶端提供予 ConfigsBuilder。

已移除的欄位	等效組態
kinesisEndpoint	使用慣用的端點設定開發套件 KinesisAsyncClient ：KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build() 。
dynamoDBEndpoint	使用慣用的端點設定開發套件 DynamoDbAsyncClient ：DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build() 。
kinesisClientConfig	使用所需的組態設定開發套件 KinesisAsyncClient ：KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build() 。

已移除的欄位	等效組態
dynamoDBClientConfig	使用所需的組態設定開發套件 DynamoDbAsyncClient ：DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build() 。
cloudWatchClientConfig	使用所需的組態設定開發套件 CloudWatchAsyncClient ：CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build() 。
regionName	使用慣用的區域設定開發套件。所有開發套件用戶端的做法皆相同。例如 KinesisAsyncClient.builder().region(Region.US_WEST_2).build() 。

使用 開發消費者 適用於 Java 的 AWS SDK

您可以使用 Amazon Kinesis Data Streams APIs 開發自訂消費者。本節說明搭配 使用 Kinesis Data Streams APIs 適用於 Java 的 AWS SDK。

> **Important**
>
> 若要開發具有共用輸送量的自訂 Kinesis Data Streams 取用者，建議使用方法為使用 Kinesis Client Library (KCL)。KCL 會處理與分散式運算相關的許多複雜任務，協助您取用和處理 Kinesis 資料串流中的資料。如需詳細資訊，請參閱在 Java 中使用 KCL 開發消費者。

主題

- 使用 開發共用輸送量消費者 適用於 Java 的 AWS SDK
- 使用 開發增強型廣發消費者 適用於 Java 的 AWS SDK
- 使用 AWS Glue 結構描述登錄檔與資料互動

使用 開發共用輸送量消費者 適用於 Java 的 AWS SDK

開發具有 共用的自訂 Kinesis Data Streams 取用者的方法之一是搭配 使用 Amazon Kinesis Data Streams APIs 適用於 Java 的 AWS SDK。本節說明搭配 使用 Kinesis Data Streams APIs 適用於

Java 的 AWS SDK。此外，您亦可使用其他程式設計語言呼叫 Kinesis Data Streams API。如需所有 AWS SDKs的詳細資訊，請參閱使用 Amazon Web Services 開始開發。

本節中的 Java 範例程式碼示範如何執行基本 Kinesis Data Streams API 操作，並以邏輯方式依操作類型分割。這些範例不代表可立即生產的程式碼，無法檢查出所有可能的例外狀況，也不可視為任何潛在安全或效能疑慮的原因。

主題

- 從串流取得資料
- 使用碎片疊代運算
- 使用 GetRecords
- 適應重新碎片

從串流取得資料

Kinesis Data Streams API 包括可調用以從資料串流擷取記錄的 `getShardIterator` 和 `getRecords` 方法。此為提取模型，將由您的程式碼直接從資料串流中的碎片取出資料記錄。

> **Important**
>
> 建議您使用 KCL 所提供的記錄處理器支援，從資料串流擷取資料。此為推送模型，需由您實作程式碼以處理資料。KCL 會從資料串流擷取資料記錄並將其傳遞至您應用程式的程式碼。此外，KCL 另有提供容錯移轉、復原及負載平衡功能。如需詳細資訊，請參閱使用 KCL 開發具有共用輸送量的自訂取用者。

不過，在某些情況下，您可能會更喜歡使用 Kinesis Data Streams API。例如，您要實作自訂工具以用於對資料串流進行監控或偵錯。

> **Important**
>
> Kinesis Data Streams 支援對資料串流變更資料記錄保留期間。如需詳細資訊，請參閱變更資料保留期間。

使用碎片疊代運算

您將以碎片為基本單位，從串流擷取記錄。針對每個碎片，以及從該碎片擷取的各個批次的記錄，您必須取得碎片疊代運算。碎片疊代運算是供 getRecordsRequest 物件用於指定從中擷取記錄的碎片位置。與碎片疊代運算相關聯的類型決定了應從碎片中的哪個點擷取記錄 (詳細資訊請參閱本節稍後說明)。您必須先擷取碎片，才能使用碎片疊代運算。如需詳細資訊，請參閱列出碎片。

使用 getShardIterator 方法可取得初始碎片疊代運算。如欲就額外各個批次的記錄取得碎片疊代運算，請使用 getNextShardIterator 方法所傳回 getRecordsResult 物件的 getRecords 方法。碎片疊代運算的有效期為 5 分鐘。若您使用了仍在有效期內的碎片疊代運算，則會進行一次新的疊代運算。每一次碎片疊代運算將於 5 分鐘內維持有效，即便其已使用過亦然。

若要取得初始碎片疊代運算，請執行個體化 GetShardIteratorRequest 並將其傳遞給 getShardIterator 方法。若要設定請求，則指定串流和碎片 ID。如需有關如何在 AWS 帳戶中取得串流的資訊，請參閱 列出串流。如需如何取得串流中各個碎片的相關資訊，請參閱列出碎片。

```
String shardIterator;
GetShardIteratorRequest getShardIteratorRequest = new GetShardIteratorRequest();
getShardIteratorRequest.setStreamName(myStreamName);
getShardIteratorRequest.setShardId(shard.getShardId());
getShardIteratorRequest.setShardIteratorType("TRIM_HORIZON");

GetShardIteratorResult getShardIteratorResult =
 client.getShardIterator(getShardIteratorRequest);
shardIterator = getShardIteratorResult.getShardIterator();
```

此範本程式碼在取得初始碎片疊代運算時指定 TRIM_HORIZON 做為疊代運算類型。這種疊代運算類型表示應從加入至碎片的第一筆記錄開始傳回各記錄，而不是從最近加入的記錄 (又稱為頂端) 開始。可用的疊代運算類型如下：

- AT_SEQUENCE_NUMBER
- AFTER_SEQUENCE_NUMBER
- AT_TIMESTAMP
- TRIM_HORIZON
- LATEST

如需詳細資訊，請參閱 ShardIteratorType。

某些疊代運算類型除了表明類型外還需要指定序號，例如：

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

一旦使用 getRecords 取得記錄後，您即可呼叫該記錄的 getSequenceNumber 方法以取得該記錄的序號。

```
record.getSequenceNumber()
```

此外，加入記錄至資料串流的程式碼也可透過對 getSequenceNumber 的結果呼叫 putRecord 取得所加入某一筆記錄的序號。

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

使用序號可以保證各個記錄的順序嚴格遞增。如需詳細資訊，請參閱 PutRecord 範例所提供的程式碼範例。

使用 GetRecords

取得碎片疊代運算之後，執行個體化 GetRecordsRequest 物件。使用 setShardIterator 方法指定請求所用的疊代運算。

您也可以使用 setLimit 方法，選擇性設定欲擷取的記錄數目。由 getRecords 傳回的記錄數目一定等於或少於此限制。如果您未指定此限制，getRecords 將傳回已擷取的 10 MB 記錄。以下範本程式碼將此限制設為 25 筆記錄。

如果沒有傳回任何記錄，即表示此碎片在碎片疊代運算所參考的序號位置目前無資料記錄可用。在這種情況下，您的應用程式應該等待一段適當時間處理串流的資料來源。接著再次使用前次呼叫 getRecords 所傳回的碎片疊代運算，嘗試從碎片取得資料。

將 getRecordsRequest 傳遞給 getRecords 方法，並且擷取 getRecordsResult 物件形式的傳回值。若要取得資料記錄，請對 getRecords 物件呼叫 getRecordsResult 方法。

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

為了準備對 getRecords 進行另一次呼叫，請透過 getRecordsResult 取得下一碎片疊代運算。

```
shardIterator = getRecordsResult.getNextShardIterator();
```

為了獲得最佳結果，逐次呼叫 getRecords 間隔期間應至少休眠 1 秒 (1,000 毫秒) 以免超出 getRecords 頻率的限制。

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

一般而言，您應於迴圈中呼叫 getRecords，即便是在測試情況下擷取單一記錄亦然。僅呼叫一次 getRecords 可能會傳回空的記錄清單，就算是碎片在後續的序號位置包含更多記錄也不免如此。若發生這種情況，便會傳回 NextShardIterator 且空的記錄清單將參考碎片中的某一後續序號，因而連續呼叫 getRecords 最終將傳回記錄。以下範例示範迴圈的用法。

範例：getRecords

以下程式碼範例反映了本節所述的 getRecords 技巧，包括在迴圈中發出呼叫。

```
// Continuously read data records from a shard
List<Record> records;

while (true) {

  // Create a new getRecordsRequest with an existing shardIterator
  // Set the maximum records to return to 25

  GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
  getRecordsRequest.setShardIterator(shardIterator);
  getRecordsRequest.setLimit(25);

  GetRecordsResult result = client.getRecords(getRecordsRequest);

  // Put the result into record list. The result can be empty.
  records = result.getRecords();
```

```
  try {
    Thread.sleep(1000);
  }
  catch (InterruptedException exception) {
    throw new RuntimeException(exception);
  }

  shardIterator = result.getNextShardIterator();
}
```

如果您是使用 Kinesis Client Library，可能要進行多次呼叫後才會傳回資料。此行為是依據設計，並不表示 KCL 或您的資料有問題。

適應重新碎片

如果 `getRecordsResult.getNextShardIterator` 傳回 `null`，則表示發生了涉及此碎片的碎片分割或合併。此碎片現在處於 `CLOSED` 狀態，並且您已從此碎片讀取所有可用的資料記錄。

在這個案例中，您可以使用 `getRecordsResult.childShards` 來了解分割或合併所建立之正在處理之碎片的新子碎片。如需詳細資訊，請參閱 ChildShard。

若是發生分割，兩個新碎片的 `parentShardId` 將與您先前處理的碎片其碎片 ID 相同。該兩個碎片的 `adjacentParentShardId` 值皆為 `null`。

若是發生合併，經由合併所建立的單個新碎片其 `parentShardId` 將等於其中一個父碎片的碎片 ID，且 `adjacentParentShardId` 等於另一個父碎片的碎片 ID。您的應用程式已從上述其中一個碎片讀取了所有資料。對於該碎片，`getRecordsResult.getNextShardIterator` 已傳回 `null`。如果資料的順序對您的應用程式很重要，請確保您亦已從另一個父碎片讀取了所有資料，而後再從經由合併所建立的子碎片讀取任何新資料。

若您使用多個處理器從串流擷取資料 (假設每個碎片各一個處理器)，則一旦發生碎片分割或合併時，請調升或調降處理器數目以適應碎片數目的變更。

如需重新分片的詳細資訊，包括碎片狀態 (如 `CLOSED`) 方面的討論，請參閱重新分片串流。

使用 開發增強型廣發消費者 適用於 Java 的 AWS SDK

增強型散發功能是 Amazon Kinesis Data Streams 的一項功能，使取用者從資料串流接收記錄時，專用輸送量可高達每個碎片每秒 2 MB 的資料。使用強化廣發功能的消費者不必與其他從串流接收資料的消費者競爭。如需詳細資訊，請參閱開發具有專用輸送量的增強型廣發消費者。

您可以使用 API 操作，為 Kinesis Data Streams 建置使用增強型散發功能的取用者。

使用 Kinesis Data Streams API 註冊具有增強型散發功能的取用者

1. 呼叫 RegisterStreamConsumer 將您的應用程式註冊為使用增強型散發功能的取用者。Kinesis Data Streams 會為該取用者產生 Amazon Resource Name (ARN) 並隨回應傳回其值。
2. 若要開始接聽特定碎片，請呼叫 SubscribeToShard 並傳遞取用者 ARN。Kinesis Data Streams 隨後會透過 HTTP/2 連線，開始從該碎片將記錄以 SubscribeToShardEvent 類型的事件形式推送給您。此連線將保持開啟長達 5 分鐘。若您希望於 SubscribeToShard 呼叫所傳回的 `future` 正常或異常完成後繼續從該碎片接收記錄，請再次呼叫 SubscribeToShard。

   > **Note**
   >
   > 當到達當前碎片的末尾時，SubscribeToShard API 還返回當前碎片的子碎片清單。

3. 若要將使用強化廣發功能的消費者取消註冊，請呼叫 DeregisterStreamConsumer。

以下範例程式碼示範如何為消費者訂閱碎片、定期續約訂閱和處理事件。

```
    import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
    import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
    import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
    import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
    import
 software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;

    import java.util.concurrent.CompletableFuture;

    /**
     * See https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/
example_code/kinesis/src/main/java/com/example/kinesis/KinesisStreamEx.java
     * for complete code and more examples.
     */
    public class SubscribeToShardSimpleImpl {

        private static final String CONSUMER_ARN = "arn:aws:kinesis:us-
east-1:123456789123:stream/foobar/consumer/test-consumer:1525898737";
        private static final String SHARD_ID = "shardId-000000000000";

        public static void main(String[] args) {
```

```
            KinesisAsyncClient client = KinesisAsyncClient.create();

            SubscribeToShardRequest request = SubscribeToShardRequest.builder()
                    .consumerARN(CONSUMER_ARN)
                    .shardId(SHARD_ID)
                    .startingPosition(s -> s.type(ShardIteratorType.LATEST)).build();

            // Call SubscribeToShard iteratively to renew the subscription
periodically.
            while(true) {
                // Wait for the CompletableFuture to complete normally or
exceptionally.
                callSubscribeToShardWithVisitor(client, request).join();
            }

            // Close the connection before exiting.
            // client.close();
        }


        /**
         * Subscribes to the stream of events by implementing the
SubscribeToShardResponseHandler.Visitor interface.
         */
        private static CompletableFuture<Void>
callSubscribeToShardWithVisitor(KinesisAsyncClient client, SubscribeToShardRequest
request) {
            SubscribeToShardResponseHandler.Visitor visitor = new
SubscribeToShardResponseHandler.Visitor() {
                @Override
                public void visit(SubscribeToShardEvent event) {
                    System.out.println("Received subscribe to shard event " + event);
                }
            };
            SubscribeToShardResponseHandler responseHandler =
SubscribeToShardResponseHandler
                    .builder()
                    .onError(t -> System.err.println("Error during stream - " +
t.getMessage()))
                    .subscriber(visitor)
                    .build();
            return client.subscribeToShard(request, responseHandler);
        }
```

```
    }
```

如果 `event.ContinuationSequenceNumber` 傳回 `null`，則表示發生了涉及此碎片的碎片分割或合併。此碎片現在處於 `CLOSED` 狀態，並且您已從此碎片讀取所有可用的資料記錄。在這個案例中，根據上述範例，您可以使用 `event.childShards` 來了解分割或合併所建立之正在處理之碎片的新子碎片。如需詳細資訊，請參閱 ChildShard。

使用 AWS Glue 結構描述登錄檔與資料互動

您可以將 Kinesis 資料串流與 AWS Glue 結構描述登錄檔整合。 AWS Glue 結構描述登錄檔可讓您集中探索、控制和發展結構描述，同時確保已註冊結構描述持續驗證產生的資料。結構描述定義資料記錄的結構和格式。結構描述是可靠的資料發佈、耗用或儲存的版本化規格。 AWS Glue 結構描述登錄檔可讓您改善串流應用程式中end-to-end資料品質和資料控管。如需詳細資訊，請參閱 AWS Glue 結構描述登錄檔。設定此整合的其中一種方法是透過 AWS Java 開發套件中提供的 `GetRecords` Kinesis Data Streams API。

如需如何使用 Kinesis Data Streams APIs 設定 `GetRecords` Kinesis Data Streams 與結構描述登錄檔整合的詳細說明，請參閱使用案例：整合 Amazon Kinesis Data Streams 與 Glue 結構描述登錄檔中的「使用 Kinesis Data Streams AWS APIs 與資料互動」一節。

使用 開發消費者 AWS Lambda

您可以使用 AWS Lambda 函數來處理資料串流中的記錄。 AWS Lambda 是一種運算服務，可讓您執行程式碼，而無需佈建或管理伺服器。這個函數只有在需要時才會執行程式碼，可自動從每天數項請求擴展成每秒數千項請求。您只需為使用的運算時間支付費用。程式碼未執行時無須付費。使用 AWS Lambda，您可以為幾乎任何類型的應用程式或後端服務執行程式碼，完全無需管理。這個函數會在高可用性的運算基礎設施上執行您的程式碼，並執行所有運算資源的管理，包括伺服器和作業系統維護，容量佈建與自動調整規模、程式碼監控和記錄。如需詳細資訊，請參閱搭配使用 AWS Lambda 與 Amazon Kinesis。

如需疑難排解資訊，請參閱為何 Kinesis Data Streams 觸發器無法調用我的 Lambda 函數？

使用 Amazon Managed Service for Apache Flink 開發消費者

您可以使用 Amazon Managed Service for Apache Flink 應用程式，使用 SQL、Java 或 Scala 來處理和分析 Kinesis 串流中的資料。Managed Service for Apache Flink 應用程式可使用參考來源富集資料、隨時間彙總資料，或使用機器學習來尋找資料異常。然後，您可以將分析結果寫入另一個 Kinesis

串流、Firehose 交付串流或 Lambda 函數。如需詳細資訊，請參閱適用於 SQL 應用程式的 Managed Service for Apache Flink 開發人員指南或適用於 Flink 應用程式的 Managed Service for Apache Flink 開發人員指南。

使用 Amazon Data Firehose 開發消費者

您可以使用 Firehose 從 Kinesis 串流讀取和處理記錄。Firehose 是一項全受管服務，可將即時串流資料交付至 Amazon S3、Amazon Redshift、Amazon OpenSearch Service 和 Splunk 等目的地。Firehose 也支援受支援的第三方服務提供者擁有的任何自訂 HTTP 端點或 HTTP 端點，包括 Datadog、MongoDB 和 New Relic。您也可以設定 Firehose 轉換資料記錄，並在將資料交付至目的地之前轉換記錄格式。如需詳細資訊，請參閱使用 Kinesis Data Streams 寫入 Firehose。

使用其他服務從 Kinesis Data Streams 讀取資料 AWS

下列 AWS 服務可以直接與 Amazon Kinesis Data Streams 整合，以從 Kinesis 資料串流讀取資料。檢閱您感興趣的每個服務的資訊，並參考提供的參考。

主題

- 使用 Amazon EMR 從 Kinesis Data Streams 讀取資料
- 使用 Amazon EventBridge 管道從 Kinesis Data Streams 讀取資料
- 使用 從 Kinesis Data Streams 讀取資料 AWS Glue
- 使用 Amazon Redshift 從 Kinesis Data Streams 讀取資料

使用 Amazon EMR 從 Kinesis Data Streams 讀取資料

Amazon EMR 叢集可以使用 Hadoop 生態系統中熟悉的工具直接讀取和處理 Kinesis 串流，例如 Hive、Pig、MapReduce、Hadoop Streaming API 和 Cascading。您也可以將來自 Kinesis Data Streams 的即時資料與執行中叢集中 Amazon S3、Amazon DynamoDB 和 HDFS 上的現有資料聯結。您能直接從 Amazon EMR 將資料載入 Amazon S3 或 DynamoDB ，以供後續處理活動使用。

如需詳細資訊，請參閱《Amazon EMR 版本指南》中的 Amazon Kinesis。

使用 Amazon EventBridge 管道從 Kinesis Data Streams 讀取資料

Amazon EventBridge 管道支援 Kinesis 資料串流做為來源。Amazon EventBridge 管道可協助您透過選用的轉換、篩選和豐富步驟，在事件生產者和消費者之間建立point-to-point整合。您可以使用

EventBridge 管道來接收 Kinesis 資料串流中的記錄，並選擇性地篩選或增強這些記錄，然後再將其傳送至其中一個可用的目的地進行處理，包括 Kinesis Data Streams。

如需詳細資訊，請參閱《Amazon EventBridge 版本指南》中的 Amazon Kinesis 串流作為來源。EventBridge

使用 從 Kinesis Data Streams 讀取資料 AWS Glue

您可以使用 AWS Glue 串流 ETL 建立串流擷取、轉換和載入 (ETL) 任務，這些任務會持續執行並從 Amazon Kinesis Data Streams 取用資料。這些任務會清理並轉換資料，然後將結果載入 Amazon S3 資料湖或 JDBC 資料存放區。

如需詳細資訊，請參閱《AWS Glue 發行指南》中的 AWS Glue中的 Streaming ETL 任務。

使用 Amazon Redshift 從 Kinesis Data Streams 讀取資料

Amazon Redshift 支援從 Amazon Kinesis Data Streams 中串流擷取。Amazon Redshift 串流擷取功能可提供低延遲、高速擷取從 Amazon Kinesis Data Streams 到 Amazon Redshift 具體化視觀表的串流資料。Amazon Redshift 串流擷取不需要先在 Amazon S3before導入 Amazon Redshift。

如需詳細資訊，請參閱《Amazon Redshift 發行指南》中的串流擷取。

使用第三方整合從 Kinesis Data Streams 讀取

您可以使用與 Kinesis Data Streams 整合的下列其中一個第三方選項，從 Amazon Kinesis Data Streams 資料串流讀取資料。選取您想要進一步了解的選項，並尋找相關文件的資源和連結。

主題

- Apache Flink
- Adobe 體驗平台
- Apache Druid
- Apache Spark
- Databricks
- Kafka Confluent 平台
- Kinesumer
- Talend

Apache Flink

Apache Flink 是一個架構和分散式處理引擎，用於對未限制和有限制資料串流進行狀態運算。如需使用 Apache Flink 取用 Kinesis 資料串流的詳細資訊，請參閱 Amazon Kinesis Data Streams 連接器。

Adobe 體驗平台

Adobe 體驗平台使組織能夠集中和標準化來自任何系統的客戶資料。然後，它會套用資料科學和機器學習，大幅改善豐富、個人化體驗的設計和交付。如需使用 Adobe Experience Platform 使用 Kinesis 資料串流的詳細資訊，請參閱 Amazon Kinesis 連接器。

Apache Druid

Druid 是高效能的即時分析資料庫，可在不到一秒鐘的時間內提供大規模和負載下的串流和批次資料查詢。如需使用 Apache Druid 擷取 Kinesis 資料串流的詳細資訊，請參閱 Amazon Kinesis 擷取。

Apache Spark

Apache Spark 是用於大規模資料處理的統一分析引擎。它提供了 Java、Scala、Python 和 R 中的高層級 API，以及支援一般執行圖的最佳化引擎。您可以使用 Apache Spark 建置串流處理應用程式，以取用 Kinesis 資料串流中的資料。

若要使用 Apache Spark 結構化串流使用 Kinesis 資料串流，請使用 Amazon Kinesis Data Streams 連接器。此連接器支援使用增強型廣發，為您的應用程式提供每秒高達 2 MB 資料的專用讀取輸送量。如需詳細資訊，請參閱使用專用輸送量開發自訂消費者（增強廣發）。

若要使用 Spark 串流使用 Kinesis 資料串流，請參閱 Spark 串流 + Kinesis 整合。

Databricks

Databricks 是一個基於雲端的平台，可為資料工程、資料科學和機器學習提供協作環境。如需使用 Databricks 使用 Kinesis 資料串流的詳細資訊，請參閱連線至 Amazon Kinesis。

Kafka Confluent 平台

Confluent 平台建立在 Kafka 之上，並提供額外的特性和功能，幫助企業構建和管理即時資料管道和串流應用程式。如需使用 Confluent 平台使用 Kinesis 資料串流的詳細資訊，請參閱 Amazon Kinesis Source Connector for Confluent Platform。

Kinesumer

Kinesumer 是 Go 用戶端，實作 Kinesis 資料串流的用戶端分散式取用者群組用戶端。如需詳細資訊，請參閱 Kinesumer GitHub 儲存庫。

Talend

Talend 是一種資料整合和管理軟體，允許使用者以可擴展和高效的方式收集、轉換和連接來自各種來源的資料。如需使用 Talend 使用 Kinesis 資料串流的詳細資訊，請參閱將 talend 連接至 Amazon Kinesis 串流。

疑難排解 Kinesis Data Streams 取用者

下列主題提供 Amazon Kinesis Data Streams 消費者常見問題的解決方案：

- LeaseManagementConfig 建構函數的編譯錯誤
- 使用 Kinesis 用戶端程式庫時，會略過部分 Kinesis Data Streams 記錄
- 屬於相同碎片的記錄會同時由不同的記錄處理器處理
- 取用者應用程式讀取速度比預期慢
- 即使串流中有資料，GetRecords 也會傳回空的記錄陣列
- 碎片疊代運算意外過期
- 消費者記錄處理落後
- 未經授權的 KMS 金鑰許可錯誤
- DynamoDbException：更新表達式中提供的文件路徑無效，無法進行更新
- 對消費者的其他常見問題進行故障診斷

LeaseManagementConfig 建構函數的編譯錯誤

升級至 Kinesis Client Library (KCL) 3.x `LeaseManagementConfig` 版或更新版本時，您可能會遇到與建構函數相關的編譯錯誤。如果您要直接建立`LeaseManagementConfig`物件來設定組態，而不是`ConfigsBuilder`在 KCL 3.x 版或更新版本中使用 ，則在編譯 KCL 應用程式程式碼時可能會看到下列錯誤訊息。

```
Cannot resolve constructor 'LeaseManagementConfig(String, DynamoDbAsyncClient,
 KinesisAsyncClient, String)'
```

3.x 版或更新版本的 KCL 需要您在 tableName 參數後面新增一個以上的參數 applicationName （類型：String)。

- 之前： leaseManagementConfig = new LeaseManagementConfig(tableName， dynamoDBClient， kinesisClient， streamName， workerIdentifier)
- After： leaseManagementConfig = new LeaseManagementConfig(tableName， applicationName， dynamoDBClient， kinesisClient， streamName， workerIdentifier)

建議您使用 在 KCL 3.x 和更新版本中`ConfigsBuilder`設定組態，而不是直接建立 LeaseManagementConfig 物件。 `ConfigsBuilder` 提供更靈活且可維護的方式來設定 KCL 應用程式。

以下是使用 `ConfigsBuilder`設定 KCL 組態的範例。

```
ConfigsBuilder configsBuilder = new ConfigsBuilder(
    streamName,
    applicationName,
    kinesisClient,
    dynamoClient,
    cloudWatchClient,
    UUID.randomUUID().toString(),
    new SampleRecordProcessorFactory()
);

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig()
    .failoverTimeMillis(60000), // this is an example
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

使用 Kinesis 用戶端程式庫時，會略過部分 Kinesis Data Streams 記錄

記錄遭略過最常見的原因是未處理由 `processRecords` 擲回的例外狀況。Kinesis Client Library (KCL) 倚賴 `processRecords` 程式碼以處理任何因處理資料記錄而引發的例外狀況。凡是 `processRecords` 擲回的任何例外狀況都將由 KCL 吸收。為避免因重複失敗造成無止盡重試，KCL

並不會重新傳送例外狀況發生時處理的該批次記錄。接著，KCL 將對下一批次的資料記錄呼叫 processRecords 而未重新啟動記錄處理器。這就導致了消費者應用程式察覺到記錄遭略過。為避免略過記錄，請由 processRecords 中妥善處理所有例外狀況。

屬於相同碎片的記錄會同時由不同的記錄處理器處理

凡是任何執行中的 Kinesis Client Library (KCL) 應用程式，各碎片都只有一個擁有者。不過，多個記錄處理器可能暫時處理同一碎片。如果工作者執行個體失去網路連線，KCL 會假設無法連線的工作者在容錯移轉時間到期後不再處理記錄，並指示其他工作者執行個體接管。短暫期間內，新的記錄處理器和來自無法連線工作者的記錄處理器可能都會處理取自同一碎片的資料。

設定適合您應用程式的容錯移轉時間。對於低延遲應用程式，預設值 10 秒足可代表您希望等待的最長時間。但在部分情況下，如您預期會有連線問題，比方跨地理區域撥話而連線中斷可能會更頻繁，此數字設定或許就過低了。

您的應用程式應該預料到這種情況並予處理，特別是因為網路連線通常會恢復至先前無法連線的工作者。記錄處理器若由另一記錄處理器接管其碎片，則必須處理以下兩種情況才能順利執行關閉：

1. 目前對 的呼叫processRecords完成後，KCL 會叫用記錄處理器上的關閉方法，其關閉原因為「ZOMBIE」。您的記錄處理器應適當清理任何資源然後結束。
2. 當您嘗試對 'zombie' 工作者執行檢查點作業，KCL 會擲回 ShutdownException。收到此例外狀況後，您的程式碼應徹底結束目前方法。

如需詳細資訊，請參閱處理重複的記錄。

取用者應用程式讀取速度比預期慢

讀取傳輸量低於預期最常見的原因如下：

1. 多個消費者應用程式的總讀取量超出每一碎片限制。如需詳細資訊，請參閱配額和限制。在此情況下，增加 Kinesis 資料串流中的碎片數量。
2. 指定每次呼叫的 GetRecords 最大數目限制可能設定了較低的值。如果您使用 KCL，則有可能是對工作者設定的 maxRecords 屬性值偏低。一般而言，建議您就此屬性使用系統預設值。
3. 出於諸多可能的原因，processRecords 呼叫內的邏輯所耗費的時間會比預期更久；該邏輯可能 CPU 使用率高、I/O 阻斷或同步存在瓶頸。若要測試是否如此，請對空的記錄處理器執行測試並比較讀取傳輸量。如需如何及時處理傳入資料的相關資訊，請參閱使用重新分片、擴展和平行處理來變更碎片數量。

如果您只有一個消費者應用程式，則讀取速率比放入速率至少高兩倍的情況絕對有可能。這是因為您每秒最多可寫入 1,000 筆記錄，最大總資料寫入速率為每秒 1 MB (包括分割區索引鍵)。每個開放碎片可支援最高每秒 5 筆交易的讀取數目，最大總資料讀取速率為每秒 2 MB。請注意，每次讀取 (GetRecords 呼叫) 都會取得一個批次的記錄。GetRecords 傳回的資料大小因碎片使用率而異。GetRecords 可傳回的資料大小上限為 10 MB。如果呼叫傳回該限制，則接下來 5 秒內發出的後續呼叫會擲回 `ProvisionedThroughputExceededException`。

即使串流中有資料，GetRecords 也會傳回空的記錄陣列

取用 (取得) 記錄是屬於提取模型。開發人員應該在不含退避的連續迴圈中呼叫 GetRecords。每次呼叫 GetRecords 還會傳回一個 `ShardIterator` 值，此值必須於下次重複迴圈時使用。

GetRecords 操作不會封鎖。而將立即傳回一些相關資料記錄或是空的 `Records` 元素。以下兩種情況會傳回空的 `Records` 元素：

1. 碎片中目前已無更多資料。
2. `ShardIterator` 所指向的碎片部分附近沒有資料。

後一種情況很微妙，但卻是避免在擷取記錄時搜尋時間 (延遲) 無止境的必要折衷設計。因此，取用串流的應用程式應循環呼叫 GetRecords，當然也要處理空記錄。

在生產情境下，僅當 `NextShardIterator` 值為 `NULL` 時才應結束連續迴圈。`NextShardIterator` 為 `NULL` 時，表示目前碎片已封閉，且 `ShardIterator` 值的指向處應會越過最後一筆記錄。如果取用端應用程式從未呼叫 SplitShard 或 MergeShards，則碎片將保持開放狀態，呼叫 GetRecords 就絕不會傳回 `NULL` 值的 `NextShardIterator`。

如果您使用 Kinesis Client Library (KCL)，則會為您抽象化上述消耗模式。這包括自動處理一組動態變化的碎片。使用 KCL 時，開發人員僅需提供處理傳入記錄的邏輯。能夠如此是因為程式庫會為您持續呼叫 GetRecords。

碎片疊代運算意外過期

每次請求都將傳回新的碎片疊代運算 GetRecords (即 `NextShardIterator`)，供您用於下一次的 GetRecords 請求 (即 `ShardIterator`)。此碎片疊代運算在您使用之前一般不會過期。不過，您可能會發現，由於您超過 5 分鐘未呼叫 GetRecords，或者您重新啟動了消費者應用程式，碎片疊代運算即過期。

如果碎片疊代運算在您可以使用之前立即過期，這可能表示 Kinesis 使用的 DynamoDB 資料表沒有足夠的容量來存放租用資料。若您有大量的碎片，即很可能發生這種情況。要解決此問題，請增加對碎片

資料表指派的寫入容量。如需詳細資訊，請參閱使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片。

消費者記錄處理落後

對於大多數使用案例、消費者應用程式會從串流讀取最新的資料。特定情況下，消費者讀取可能落後，您應不希望出現這種情況。在查出消費者讀取落後多久之後，請查看導致消費者落後最常見的原因。

首先使用 GetRecords.IteratorAgeMilliseconds 指標，追蹤串流中所有碎片和消費者的讀取位置。請注意，如果疊代運算的存留期超過保留期間的 50% (預設為 24 小時，最多可設定為 365 天)，會有由於記錄過期而遺失資料的風險。快速的權宜之計是增加保留期間。這可使您在進一步對問題進行故障診斷時防止遺失重要資料。如需詳細資訊，請參閱使用 Amazon CloudWatch 監控 Amazon Kinesis Data Streams 服務 Amazon CloudWatch。接著，使用 Kinesis Client Library (KCL) MillisBehindLatest 發出的自訂 CloudWatch 指標 ，查出取用者應用程式從各碎片讀取落後多久的時間。如需詳細資訊，請參閱使用 Amazon CloudWatch 監控 Kinesis 用戶端程式庫。

消費者可能落後最常見的原因如下：

- GetRecords.IteratorAgeMilliseconds 或 MillisBehindLatest 突然大增通常表示暫時性的問題，例如對下游應用程式的 API 操作失敗。如果其中一個指標持續顯示此行為，則調查這些突然增加。
- 上述指標若逐漸增加，表示消費者因處理記錄速度不夠快而未能與串流同步。此行為最常見的根本原因是實體資源不足，或者記錄處理邏輯沒有隨著串流傳輸量的增加而進行擴展。您可藉由查看 KCL 所發出與 processTask 操作相關聯的其他自訂 CloudWatch 指標，包括 RecordProcessor.processRecords.Time、Success 和 RecordsProcessed 以確認此行為。
  - 若您發現與傳輸量上升相關的 processRecords.Time 指標有所增加，即應分析您的記錄處理邏輯，以確定該邏輯為何沒有隨著傳輸量增加而擴展。
  - 若您發現與傳輸量上升無關的 processRecords.Time 值有所增加，請檢查您是否在重要路徑上執行了任何封鎖呼叫，這通常會導致記錄處理速度下降。替代方法是增加碎片數目以提高並行處理程度。最後，請確認您在尖峰需求期間基礎處理節點上有足夠數量的實體資源 （記憶體、CPU 使用率等）。

未經授權的 KMS 金鑰許可錯誤

當取用者應用程式在沒有 AWS KMS 金鑰許可的情況下從加密串流讀取時，會發生此錯誤。若要為應用程式指派許可使其能夠存取 KMS 金鑰，請參閱在 AWS KMS 中使用金鑰政策及搭配 AWS KMS 使用 IAM 政策。

DynamoDbException：更新表達式中提供的文件路徑無效，無法進行更新

搭配 2.27.19 到 2.27.23 適用於 Java 的 AWS SDK 版本使用 KCL 3.x 時，您可能會遇到下列 DynamoDB 例外狀況：

「software.amazon.awssdk.services.dynamodb.model.DynamoDbException：更新表達式中提供的文件路徑不適用於更新 （ 服務：DynamoDb，狀態碼：400，請求 ID：xxx)」

由於 中的已知問題會影響 KCL 3.x 管理的 DynamoDB 中繼資料表 適用於 Java 的 AWS SDK ，因此會發生此錯誤。此問題在 2.27.19 版中引入，並影響截至 2.27.23 的所有版本。此問題已在 2.27.24 版中 適用於 Java 的 AWS SDK 解決。為了獲得最佳效能和穩定性，建議您升級至 2.28.0 版或更新版本。

對消費者的其他常見問題進行故障診斷

- 為何 Kinesis Data Streams 觸發器無法調用我的 Lambda 函數？
- 如何偵測並疑難排解 Kinesis Data Streams 中的 ReadProvisionedThroughputExceeded 例外狀況？
- 為何我遇到 Kinesis Data Streams 的高延遲問題？
- 為何我的 Kinesis 資料串流會傳回 500 個內部伺服器錯誤？
- 如何對遭 Kinesis Data Streams 封鎖或停滯的 KCL 應用程式進行故障診斷？
- 是否可以將不同的 Amazon Kinesis 用戶端程式庫應用程式與相同的 Amazon DynamoDB 資料表搭配使用？

最佳化 Amazon Kinesis Data Streams 取用者

您可以根據您看到的特定行為，進一步最佳化 Amazon Kinesis Data Streams 取用者。

檢閱下列主題以識別解決方案。

主題

- 改善低延遲處理

- AWS Lambda 搭配 Amazon Kinesis Producer Library 使用 處理序列化資料
- 使用重新分片、擴展和平行處理來變更碎片數量
- 處理重複的記錄
- 處理啟動、關閉和限流

改善低延遲處理

傳播延遲的定義是自從記錄寫入串流直到消費者應用程式讀取該記錄為止的端對端延遲。此延遲依各種因素而異，但主要受消費者應用程式的輪詢間隔影響。

對於大多數應用程式，建議每個應用程式每秒輪詢每個碎片一次。這使您能夠擁有多個取用者應用程式並行處理任一串流，而不會達到 Amazon Kinesis Data Streams 每秒 5 次 `GetRecords` 呼叫的限制。此外，處理較大批次的資料時，減少網路延遲以及應用程式下游處的其他延遲往往更有效率。

KCL 的預設值遵循最佳實務，設為每隔 1 秒輪詢一次。此預設值導致平均傳播延遲通常少於 1 秒。

Kinesis Data Streams 記錄一經寫入後隨即可供讀取。有些使用案例需要利用此一特點，當串流中的資料可用時必須立即予以取用。您可透過覆寫 KCL 預設的設定以進行更頻繁的輪詢，藉此顯著減少傳播延遲，如以下範例所示。

Java KCL 組態程式碼：

```
kinesisClientLibConfiguration = new
        KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

 workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Python 和 Ruby KCL 的屬性檔案設定：

```
idleTimeBetweenReadsInMillis = 250
```

> **Note**
>
> 由於 Kinesis Data Streams 的限制為每個碎片每秒 5 次 `GetRecords` 呼叫，若將 `idleTimeBetweenReadsInMillis` 屬性設為少於 200 毫秒，可能會導致您的應用程式產生 `ProvisionedThroughputExceededException` 例外狀況。這類例外狀況過多可能導致指

數退避，進而造成處理過程中發生重大的意外延遲。若您將此屬性設為 200 毫秒或更久且同時有多個處理應用程式，便會遭遇到類似的調節牽制。

AWS Lambda 搭配 Amazon Kinesis Producer Library 使用 處理序列化資料

Amazon Kinesis Producer Library (KPL) 會將小型使用者格式化的記錄彙總成最大 1 MB 的較大記錄，以更好地利用 Amazon Kinesis Data Streams 輸送量。雖然適用於 Java 的 KCL 支援取消彙總這些記錄，但當使用 AWS Lambda 做為串流的取用者時，您需要使用特殊模組來取消彙總記錄。您可以從 Amazon Kinesis Producer Library Deaggregation Modules for AWS Lambda 的 GitHub 取得必要的專案程式碼和指示。此專案中的元件可讓您在 Java AWS Lambda、Node.js 和 Python 中處理 KPL 序列化資料。上述元件也可用於建構多語言 KCL 應用程式。

使用重新分片、擴展和平行處理來變更碎片數量

透過重新分片，您將能夠增加或減少串流中的碎片數目，以便適應通過串流的資料速率的變化。重新分片通常是由監控碎片資料處理指標的管理應用程式所執行。儘管 KCL 本身不會初始化重新分片操作，但其設計能夠適應碎片數目因重新分片而造成的變化。

如 使用租用資料表來追蹤 KCL 取用者應用程式處理的碎片 所述，KCL 使用 Amazon DynamoDB 資料表追蹤串流中的碎片。當重新分片造成建立新碎片時，KCL 會發現新碎片並於該資料表內填入新的列。工作者將自動發現新碎片並建立處理器以處理取自各個碎片的資料。KCL 還會將串流中的碎片分配給所有可用的工作者和記錄處理器。

KCL 將確保在重新分片之前已優先處理碎片中現存的任何資料。處理過該等資料後，新碎片中的資料將傳送至記錄處理器。如此一來，KCL 便維持了特定分割區索引鍵的資料記錄加入至串流的順序。

範例：重新分片、擴展和平行處理

以下範例說明 KCL 將如何協助您處理擴展和重新分片：

- 舉例而言，假設您的應用程式在某個 EC2 執行個體上執行，且要處理具有四個碎片的單一 Kinesis 資料串流。該執行個體有一個 KCL 工作者和四個記錄處理器 (每個碎片各一個記錄處理器)。前述的四個記錄處理器在同一程序內平行執行。
- 接著，如果您擴展應用程式又使用另一執行個體，就會有兩個執行個體共同處理具有四個碎片的單一串流。當第二個執行個體上的 KCL 工作者啟動後，其將與第一個執行個體進行負載平衡，以致每一執行個體現在各自處理兩個碎片。
- 假設您隨後決定將四個碎片分割成五個碎片。KCL 會再次協調跨執行個體的處理方式：執行個體之其一處理三個碎片，其二處理兩個碎片。在您合併碎片時同樣會進行類似的協調。

一般而言，使用 KCL 時，您應確保執行個體數目未多過碎片數目 (故障待命者除外)。每個碎片恰由一個 KCL 工作者處理且正好有一個對應的記錄處理器，如此便根本不必由多個執行個體處理單一碎片。不過，單一工作者能夠處理任意數目的碎片，所以碎片數目多過執行個體數目並無妨。

若要就您的應用程式擴展處理規模，您應組合測試以下幾種方法：

- 增加執行個體大小 (因為所有的記錄處理器在同一程序內平行執行)
- 增加執行個體數目，最多達開放碎片數目的上限 (因為碎片可以單獨處理)
- 增加碎片數目 (如此將提高平行處理的等級)

請注意，您可以使用自動擴展根據適當的指標自動擴展您的執行個體。如需詳細資訊，請參閱 Amazon EC2 Auto Scaling 使用者指南。

當重新分片致使串流中的碎片數目增加時，記錄處理器數目相對地增加將會增加託管處理器的 EC2 執行個體所承受的負載。如果執行個體屬於 Auto Scaling 群組的一部分，當負載增加至足量後，Auto Scaling 群組就會添加更多執行個體以處理增加的負載。您應將執行個體設定成啟動時便啟動您的 Amazon Kinesis Data Streams 應用程式，好讓新執行個體上的其他工作者和記錄處理器能立即起作用。

如需重新分片的詳細資訊，請參閱重新分片串流。

處理重複的記錄

有兩個主要原因可能會導致多次將記錄交付至您的 Amazon Kinesis Data Streams：生產者重試和取用者重試。您的應用程式必須預料並妥善因應多次處理個別記錄的問題。

製作者重試

試想有一個生產者，其呼叫 `PutRecord` 後但仍未能收到 Amazon Kinesis Data Streams 的確認便遇到了與網路相關的逾時情況。此生產者無法確定記錄是否已交付至 Kinesis Data Streams。假設每一筆記錄對應用程式都很重要，生產者即會撰寫成使用相同的資料重試呼叫。如果就相同的資料呼叫兩次 `PutRecord` 均已成功遞交至 Kinesis Data Streams，則將會有兩筆 Kinesis Data Streams 記錄。儘管這兩筆記錄具有相同的資料，但其序號各不相同。需要嚴格保證的應用程式應於記錄中嵌入主索引鍵，以便稍後進行處理時移除重複項目。請注意，因生產者重試而造成的重複項目數通常會比因消費者重試而造成的重複項目數來得少。

> **Note**
>
> 如果您使用 AWS SDK `PutRecord`，請在 SDK AWS 和工具使用者指南中SDKs 重試行為。

消費者重試

消費者 (資料處理應用程式) 重試是在記錄處理器重新啟動時發生。同一碎片的記錄處理器將於以下情況重新啟動：

1. 工作者意外終止
2. 新增或移除工作者執行個體
3. 碎片合併或分割
4. 部署應用程式

於上述所有情況下，碎片、工作者、記錄處理器三者的對應會持續更新以進行負載平衡處理。已遷移至其他執行個體的碎片處理器將從最後一個檢查點重新啟動處理記錄。這會導致重複的記錄處理，如以下範例所示。如需負載平衡的詳細資訊，請參閱使用重新分片、擴展和平行處理來變更碎片數量。

範例：消費者重試導致重新交付的記錄

本範例中的應用程式將持續從串流讀取記錄、彙整記錄至本機檔案，然後上傳該檔案到 Amazon S3。為求簡化，假設只有 1 個碎片並由 1 個工作者處理此碎片。試想以下發生的一系列範例事件，假設最後一個檢查點位於記錄編號 10000 處：

1. 工作者從碎片讀取下一批次的記錄，即記錄 10001 到 20000。
2. 工作者隨後將該批次記錄傳遞至關聯的記錄處理器。
3. 記錄處理器彙總資料、建立 Amazon S3 檔案並成功將該檔案上傳到 Amazon S3。
4. 工作者在新的檢查點到達之前意外終止。
5. 應用程式、工作者和記錄處理器重新啟動。
6. 工作者現在開始從上次成功的檢查點 (本例中為 10001) 進行讀取。

因此，記錄 10001-20000 取用了一次以上。

對消費者重試保持彈性

即使記錄可能經過多次處理，您的應用程式也許希望能體現副作用，猶如只處理一次記錄那樣 (等冪處理)。此問題的解決方法因複雜度與準確度而異。如果最終資料的目的地能夠妥善處理重複項目，建議您憑藉最終目的地以實現等冪處理。例如，透過 Opensearch，您可以組合運用版本控制和唯一 ID 避免重複進行處理。

回顧前一節的範例應用程式，其持續從串流讀取記錄、彙整記錄至本機檔案，然後上傳該檔案到 Amazon S3。如該節所示，記錄 10001-20000 取用了一次以上，導致多個 Amazon S3 檔案具有相同的資料。化解該範例發生重複情況的一種方法是確保步驟 3 使用以下機制：

1. 記錄處理器就每個 Amazon S3 檔案使用固定數目的記錄，例如 5000。
2. 檔案名稱使用以下結構描述：Amazon S3 字首、碎片 ID 和 `First-Sequence-Num`。在本例中，其可能類似於 `sample-shard000001-10001`。
3. 上傳 Amazon S3 檔案後，透過指定 `Last-Sequence-Num` 執行檢查點作業。本例應於記錄編號 15000 處執行檢查點作業。

利用上述機制，即使記錄經過多次處理，產生的 Amazon S3 檔案也會具有相同的名稱和相同的資料。重試只會導致多次將相同的資料寫入同一個檔案。

若發生重新分片操作的情況，留存於碎片中的記錄數目可能少於您所需要的固定數目。就本例而言，您必須透過 `shutdown()` 方法將檔案排清至 Amazon S3 並對最後一個序號執行檢查點作業。上述機制同樣相容於重新分片操作。

處理啟動、關閉和限流

以下是您在設計 Amazon Kinesis Data Streams 時應納入考量的一些其他事項。

主題

- 啟動資料生產者和資料消費者
- 關閉 Amazon Kinesis Data Streams 應用程式
- 讀取限流

啟動資料生產者和資料消費者

預設情況下，KCL 會從串流的頂端開始讀取記錄，也就是最近加入的記錄。依照這種組態，如果資料產生應用程式於任何接收端記錄處理器執行之前加入記錄至串流，記錄處理器啟動後並不會讀取這些記錄。

若要變更記錄處理器的行為，使其一律從串流開頭處讀取資料，請在您的 Amazon Kinesis Data Streams 應用程式屬性檔案中設定以下數值：

```
initialPositionInStream = TRIM_HORIZON
```

根據預設，Amazon Kinesis Data Streams 會 24 小時儲存所有資料。它還支援長達 7 天的延長保留和長達 365 天的長期保留。此時間範圍稱為保留期間。當起始位置設為 TRIM_HORIZON 時，將依保留期間的定義，就串流中最舊的資料啟動記錄處理器。即便使用 TRIM_HORIZON 設定，如果記錄處理器在超過了保留期間很長一段時間後啟動，則串流中的某些記錄將已無法再使用。因此，您應讓取用者應用程式隨時讀取串流，並使用 CloudWatch 指標 GetRecords.IteratorAgeMilliseconds 進行監控，以確保應用程式能及時處理傳入的資料。

在某些情況下，記錄處理器若錯過了串流中的前幾筆記錄其實並無妨。例如，您可能傳遞一些初始記錄通過串流，以便測試串流的端對端運作是否如預期。經過這類初步驗證後，您即接著啟動工作者並開始將生產資料放入該串流。

如需 TRIM_HORIZON 設定的詳細資訊，請參閱使用碎片疊代運算。

關閉 Amazon Kinesis Data Streams 應用程式

當您的 Amazon Kinesis Data Streams 應用程式已完成預定任務後，您應終止其執行所在的 EC2 執行個體以關閉該應用程式。使用 AWS 管理主控台 或 AWS CLI 皆可終止執行個體。

關閉您的 Amazon Kinesis Data Streams 應用程式後，您應刪除 KCL 用於追蹤該應用程式狀態的 Amazon DynamoDB 資料表。

讀取限流

串流的傳輸量是按照碎片級進行佈建。每個碎片可支援最高每秒 5 筆交易的讀取輸送量，最大總資料讀取速率為每秒 2 MB。如果某個應用程式 (或對同一串流進行操作的一組應用程式) 嘗試以較快的速率從碎片取得資料，則 Kinesis Data Streams 就會調節相應的 Get 操作。

在 Amazon Kinesis Data Streams 應用程式中，如果記錄處理器處理資料的速率高過限制 (比方容錯移轉的情況)，則會發生限流。由於 KCL 管制著應用程式與 Kinesis Data Streams 之間的互動，限流例外狀況是發生在 KCL 程式碼而非應用程式的程式碼。不過，由於 KCL 會記錄這些例外狀況，您可由日誌中進行查看。

若您發現應用程式一直受到調節，即應考慮增加串流的碎片數目。

監控 Kinesis 資料串流

您可以使用下列功能在 Amazon Kinesis Data Streams 中監控資料串流：

- CloudWatch 指標 – Kinesis Data Streams 會向 Amazon CloudWatch 傳送具有每個串流之詳細監控的自訂指標。
- Kinesis 代理程式 – Kinesis 代理程式發佈自訂 CloudWatch 指標，以協助評估代理程式是否如預期運作。
- API 記錄 – Kinesis Data Streams 使用 AWS CloudTrail 來記錄 API 呼叫，並將資料存放在 Amazon S3 儲存貯體中。
- Kinesis Client Library – Kinesis Client Library (KCL) 提供每個碎片、工作者和 KCL 應用程式的指標。
- Kinesis Producer Library – Amazon Kinesis Producer Library (KPL) 提供每個碎片、工作者和 KPL 應用程式的指標。

如需常見監控問題、疑問和故障診斷的詳細資訊，請參閱下列各項：

- 我應該使用哪些指標來監控 Kinesis Data Streams 問題並進行故障診斷？
- 為什麼 Kinesis Data Streams 中的 IteratorAgeMilliseconds 值不斷增加？

使用 Amazon CloudWatch 監控 Amazon Kinesis Data Streams 服務 Amazon CloudWatch

Amazon Kinesis Data Streams 與 Amazon CloudWatch 整合，因此您可以收集、檢視和分析 Kinesis 資料串流的 CloudWatch 指標。例如，若要追蹤碎片用量，您可以監控 `IncomingBytes` 和 `OutgoingBytes` 指標，並將它們與串流中的碎片數量比較。

您設定的串流指標和碎片層級指標會每分鐘自動收集並推送至 CloudWatch。指標將封存兩週，之後即會捨棄資料。

下表說明 Kinesis 資料串流的基本串流層級和增強之碎片層級監控。

Type	說明
基本 (串流層級)	串流層級資料將每分鐘免費自動傳送。

Type	說明
增強 (碎片層級)	碎片層級資料會每分鐘傳送 (需另行支付費用)。若要取得此層級的資料，您必須使用 EnableEnhancedMonitoring 操作，特別為串流啟用它。 如需定價的詳細資訊，請參閱 Amazon CloudWatch 產品頁面。

Amazon Kinesis Data Streams 維度和指標

Kinesis Data Streams 會以兩個層級將指標傳送至 CloudWatch：串流層級以及選擇性的碎片層級。串流層級指標適用於正常情況下最常見的監控使用案例。碎片層級指標用於特定的監控任務 (一般是與疑難排解相關)，並且是使用 EnableEnhancedMonitoring 操作來啟用。

如需從 CloudWatch 指標收集到的統計資料的說明，請參閱 《Amazon CloudWatch 使用者指南》中的 CloudWatch 統計資料。

主題

基本串流層級指標

`AWS/Kinesis` 命名空間包含下列串流層級指標。

Kinesis Data Streams 會每分鐘將這些串流層級指標傳送給 CloudWatch。這些是一律可用的指標。

指標	Description
`GetRecords.Bytes`	擷取自 Kinesis 串流的位元組數目，這是對指定的期間進行測量。Minimum、Maximum 與 Average 統計資訊代表指定期間內串流之單一 `GetRecords` 操作中的位元組。 分區層級指標名稱：`OutgoingBytes`

指標	Description
	維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：位元組
GetRecords.IteratorAge	不再使用此指標。請使用 GetRecords.IteratorAgeMilliseconds 。
GetRecords.IteratorAgeMilliseconds	對 Kinesis 串流進行之所有 GetRecords 呼叫中最後一筆記錄的存留期，這是對指定的期間進行測量。目前時間與將 GetRecords 呼叫的最後一筆記錄寫入至串流時之間的差異就是存留期。Minimum 和 Maximum 統計資訊可以用來追蹤 Kinesis 取用者應用程式進度。零值指出所讀取的記錄完全與串流同步。 分區層級指標名稱：IteratorAgeMilliseconds 維度：StreamName 統計資訊：Minimum、Maximum、Average、Samples 單位：毫秒
GetRecords.Latency	每個 GetRecords 操作所需的時間，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Minimum、Maximum、Average 單位：毫秒

指標	Description
GetRecords.Records	擷取自分區的記錄數目，這是對指定的期間進行測量。Minimum、Maximum 與 Average 統計資訊代表指定期間內串流之單一 GetRecords 操作中的記錄。 分區層級指標名稱：OutgoingRecords 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
GetRecords.Success	每個串流的成功 GetRecords 操作數目，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Average、Sum、Samples 單位：Count
IncomingBytes	在指定的期間內，成功放入 Kinesis 串流的位元組數目。此指標包含來自 PutRecord 與 PutRecords 操作的位元組。Minimum、Maximum 與 Average 統計資訊代表指定期間內串流之單一 Put 操作中的位元組。 分區層級指標名稱：IncomingBytes 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：位元組

指標	Description
IncomingRecords	在指定的期間內，成功放入 Kinesis 串流的記錄數目。此指標包含來自 PutRecord 與 PutRecords 操作的記錄計數。Minimum、Maximum 與 Average 統計資訊代表指定期間內串流之單一 Put 操作中的記錄。 分區層級指標名稱：IncomingRecords 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
PutRecord.Bytes	在指定的期間內，使用 PutRecord 操作放入 Kinesis 串流的位元組數目。 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：位元組
PutRecord.Latency	每個 PutRecord 操作所需的時間，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Minimum、Maximum、Average 單位：毫秒

指標	Description
PutRecord.Success	每個 Kinesis 串流的成功 PutRecord 操作數目，這是對指定的期間進行測量。Average 會反映成功寫入至串流的百分比。 維度：StreamName 統計資訊：Average、Sum、Samples 單位：Count
PutRecords.Bytes	在指定的期間內，使用 PutRecords 操作放入 Kinesis 串流的位元組數目。 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：位元組
PutRecords.Latency	每個 PutRecords 操作所需的時間，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Minimum、Maximum、Average 單位：毫秒
PutRecords.Records	此指標已淘汰。請使用 PutRecords.SuccessfulRecords 。 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count

指標	Description
PutRecords.Success	每個 Kinesis 串流中至少有一筆記錄成功的 PutRecords 操作數目，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Average、Sum、Samples 單位：Count
PutRecords.TotalRecords	每個 Kinesis 資料串流之 PutRecords 操作中的記錄總數，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
PutRecords.SuccessfulRecords	每個 Kinesis 資料串流之 PutRecords 操作中的成功記錄數目，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
PutRecords.FailedRecords	由於內部失敗，每個 Kinesis 資料串流之 PutRecords 操作中遭拒的記錄數目，這是對指定的期間進行測量。偶爾的內部故障是預料中的，應該重試。 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count

指標	Description
PutRecords.ThrottledRecords	每個 Kinesis 資料串流之 PutRecords 操作中的由於限流遭拒的記錄數目，這是對指定的期間進行測量。 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
ReadProvisionedThroughputExceeded	在指定的期間內，針對串流進行調節的 GetRecords 呼叫數目。此指標的最常用統計資訊是 Average。 Minimum 統計資訊的值為 1 時，已在指定期間內調節串流的所有記錄。 Maximum 統計資訊的值為 0 (零) 時，未在指定期間內調節串流的任何記錄。 分區層級指標名稱：ReadProvisionedThroughputExceeded 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
SubscribeToShard.RateExceeded	當新訂閱嘗試因為已有相同消費者的作用中訂閱或您超出此操作每秒允許的呼叫數目而失敗時，會發出此指標。 維度：StreamName、ConsumerName
SubscribeToShard.Success	此指標會記錄 SubscribeToShard 訂閱是否已成功建立。訂閱最多只會存留 5 分鐘。因此，此指標至少每 5 分鐘發出一次。 維度：StreamName、ConsumerName

指標	Description
SubscribeToShardEvent.Bytes	接收自碎片的位元組數目，此數量是在指定的期間進行測量。Minimum、Maximum 與 Average 統計資訊代表指定時段內在單一事件中發佈的位元組。 分區層級指標名稱：OutgoingBytes 維度：StreamName、ConsumerName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：位元組
SubscribeToShardEvent.MillisBehindLatest	讀取記錄來自串流頂端的毫秒數，表示消費者目前時間落後多久。 維度：StreamName、ConsumerName 統計資訊：Minimum、Maximum、Average、Samples 單位：毫秒
SubscribeToShardEvent.Records	接收自碎片的記錄數目，此數量是在指定的期間進行測量。Minimum、Maximum 與 Average 統計資訊代表指定時段內在單一事件中的記錄。 分區層級指標名稱：OutgoingRecords 維度：StreamName、ConsumerName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count

指標	Description
SubscribeToShardEvent.Success	此指標會在每次事件成功發佈時發出。只在有作用中訂閱時此發出此指標。 維度：StreamName、ConsumerName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
WriteProvisionedThroughputExceeded	因在指定期間內調節串流而拒絕的記錄數目。此指標包含來自 PutRecord 與 PutRecords 操作的調節。此指標的最常用統計資訊是 Average。 Minimum 統計資訊的值為非零時，正在指定期間內調節串流的記錄。 Maximum 統計資訊的值為 0 (零) 時，目前未在指定期間內調節串流的所有記錄。 分區層級指標名稱：WriteProvisionedThroughputExceeded 維度：StreamName 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count

增強的碎片層級指標

AWS/Kinesis 命名空間包含下列分區層級指標。

Kinesis 會每分鐘將下列碎片層級指標傳送至 CloudWatch。每個指標維度會建立 1 個 CloudWatch 指標，每個月進行大約 43,200 個 PutMetricData API 呼叫。預設不會啟用這些指標。Kinesis 發出之增強型指標會收取費用。如需詳細資訊，請參閱 Amazon CloudWatch 定價下的 Amazon CloudWatch 自訂指標。每個月每個指標的每個分區都會收取費用。

指標	Description
IncomingBytes	在指定的期間內，成功放入分區的位元組數目。此指標包含來自 PutRecord 與 PutRecords 操作的位元組。Minimum、Maximum 與 Average 統計資訊代表指定期間內分區之單一 Put 操作中的位元組。 串流層級指標名稱：IncomingBytes 維度：StreamName、ShardId 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：位元組
IncomingRecords	在指定的期間內，成功放入分區的記錄數目。此指標包含來自 PutRecord 與 PutRecords 操作的記錄計數。Minimum、Maximum 與 Average 統計資訊代表指定期間內分區之單一 Put 操作中的記錄。 串流層級指標名稱：IncomingRecords 維度：StreamName、ShardId 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count
IteratorAgeMilliseconds	對分區進行之所有 GetRecords 呼叫中最後一筆記錄的存留期，這是對指定的期間進行測量。目前時間與將 GetRecords 呼叫的最後一筆記錄寫入至串流時之間的差異就是存留期。Minimum 和 Maximum 統計資訊可以用來追蹤 Kinesis 取用者應用程式進度。0 (零) 值指出所讀取的記錄完全與串流同步。 串流層級指標名稱：GetRecords.IteratorAgeMilliseconds

指標	Description
	維度：StreamName、ShardId 統計資訊：Minimum、Maximum、Average、Samples 單位：毫秒
OutgoingBytes	擷取自分區的位元組數目，這是對指定的期間進行測量。Minimum、Maximum 與 Average 統計資訊代表在指定期間內碎片之單一 GetRecords 操作中傳回或在單一 SubscribeToShard 事件中發佈的位元組。 串流層級指標名稱：GetRecords.Bytes 維度：StreamName、ShardId 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：位元組
OutgoingRecords	擷取自分區的記錄數目，這是對指定的期間進行測量。Minimum、Maximum 與 Average 統計資訊代表在指定期間內碎片之單一 GetRecords 操作中傳回或在單一 SubscribeToShard 事件中發佈的記錄。 串流層級指標名稱：GetRecords.Records 維度：StreamName、ShardId 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count

指標	Description
ReadProvisionedThroughputExceeded	在指定的期間內，針對分區進行調節的 GetRecords 呼叫數目。這項例外計數涵蓋下列限制的所有維度：每秒每個分區 5 次讀取或每個分區每秒 2 MB。此指標的最常用統計資訊是 Average。 Minimum 統計資訊的值為 1 時，已在指定期間內調節分區的所有記錄。 Maximum 統計資訊的值為 0 (零) 時，未在指定期間內調節分區的任何記錄。 串流層級指標名稱：ReadProvisionedThroughputExceeded 維度：StreamName、ShardId 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count

指標	Description
WriteProvisionedThroughputExceeded	因在指定期間內調節分區而拒絕的記錄數目。此指標包含來自 PutRecord 與 PutRecords 操作的調節，並涵蓋下列限制的所有維度：每個分區每秒 1,000 筆記錄或每個分區每秒 1 MB。此指標的最常用統計資訊是 Average。 Minimum 統計資訊的值為非零時，正在指定期間內調節分區的記錄。 Maximum 統計資訊的值為 0 (零) 時，未在指定期間內調節分區的任何記錄。 串流層級指標名稱：WriteProvisionedThroughputExceeded 維度：StreamName、ShardId 統計資訊：Minimum、Maximum、Average、Sum、Samples 單位：Count

Amazon Kinesis Data Streams 指標的維度

維度	Description
StreamName	Kinesis 串流名稱。所有可用的統計資訊皆以 StreamName 篩選。

建議的 Amazon Kinesis Data Streams 指標

Kinesis 資料串流客戶可能會對數個 Amazon Kinesis Data Streams 指標會特別感興趣。以下清單提供建議的指標和其使用方式。

指標	使用須知
GetRecords.IteratorAgeMilliseconds	追蹤串流中所有碎片和取用者的讀取位置。如果反覆運算器的存留期超過保留期間的 50% (預設為 24 小時，最多可設定為 7 天)，會有由於記錄過期而遺失資料的風險。建議您對最大統計資料使用 CloudWatch 警示，以在此遺失有風險之前提醒您。如需使用此指標的範例案例，請參閱消費者記錄處理落後。
ReadProvisionedThroughputExceeded	當您的取用者端記錄處理落後，有時難以知道瓶頸所在位置。使用此指標來判斷您的讀取是否因為超過讀取輸送量限制而受到節制。此指標的最常用統計資訊是 Average。
WriteProvisionedThroughputExceeded	這是用於與 ReadProvisionedThroughputExceeded 指標相同的用途，但用於串流的生產程式 (put) 端。此指標的最常用統計資訊是 Average。
PutRecord.Success，PutRecords.Success	建議在平均統計資料上使用 CloudWatch 警示來指示對串流失敗的記錄。根據生產程式的用途，選擇其中一個或兩個 put 類型。如果使用 Amazon Kinesis Producer Library (KPL)，請使用 PutRecords.Success 。
GetRecords.Success	建議在平均統計資料上使用 CloudWatch 警示來指示自串流失敗之記錄的時間。

存取 Kinesis Data Streams 的 Amazon CloudWatch 指標

您可以使用 CloudWatch 主控台、命令列或 CloudWatch API 來監控 Kinesis Data Streams 指標。以下程序將說明如何使用這些不同的方法來存取指標。

使用 CloudWatch 主控台檢視指標

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽列上，選擇一個區域。
3. 在導覽窗格中，選擇 指標。

4. 在 CloudWatch Metrics by Category (依類別的 CloudWatch 指標) 窗格中，選擇 Kinesis Metrics (Kinesis 指標)。
5. 按一下相關資料列來檢視指定 MetricName 和 StreamName 的統計資料。

   注意：大多數主控台統計資料名稱符合先前列出的對應 CloudWatch 指標名稱，但讀取輸送量和寫入輸送量除外。系統會在 5 分鐘的間隔計算這些統計資料：寫入輸送量會監控 `IncomingBytes` CloudWatch 指標，讀取輸送量會監控 `GetRecords.Bytes`。
6. (選用) 在圖形窗格中，選取統計資料和時段，然後利用這些設定來建立 CloudWatch 警示。

使用 存取指標 AWS CLI

使用 list-metrics 和 get-metric-statistics 命令。

使用 CloudWatch CLI 存取指標

使用 mon-list-metrics 和 mon-get-stats 命令。

使用 CloudWatch API 存取指標

使用 ListMetrics 和 GetMetricStatistics 操作。

使用 Amazon CloudWatch 監控 Kinesis Data Streams Agent 運作狀態

代理程式會使用 AWS KinesisAgent 命名空間來發布自訂 CloudWatch 指標。這些指標可幫助您評估代理程式是否如指定將資料提交至 Kinesis Data Streams，以及它是否正常運作並在資料產生來源上耗用適當數量的 CPU 和記憶體資源。例如傳送的記錄數量和位元組等指標，有助於了解客服人員將資料提交至串流的速率。當這些指標滑落至預期閾值以下特定百分比或滑落至零，可能表示設定有問題、網路出現錯誤或代理程式運作狀態不佳。諸如主機 CPU 和記憶體的消耗量與代理程式錯誤計數器等指標，均顯示資料產生來源的資源使用情況，並提供潛在的設定或主機錯誤等洞見。最後，代理程式亦會記錄服務例外狀況，以協助調查代理程式的問題。這些指標會在代理程式組態設定 `cloudwatch.endpoint` 中指定的區域中回報。從多個 Kinesis 代理程式發佈的 CloudWatch 指標會彙總或合併。如需代理程式組態的詳細資訊，請參閱指定代理程式組態設定。

使用 CloudWatch 監控

Kinesis Data Streams 代理程式會向 CloudWatch 傳送下列指標。

指標	Description
BytesSent	在指定期間內，傳送至 Kinesis Data Streams 的位元組數目。 單位：位元組
RecordSendAttempts	呼叫於指定期間內PutRecords 嘗試的記錄 (不論第一次嘗試或是重試) 數量。 單位：Count
RecordSendErrors	呼叫於指定期間內PutRecords 回傳失敗狀態的記錄 (包括重試) 數量。 單位：Count
ServiceErrors	呼叫於指定期間內PutRecords 導致服務錯誤 (調節錯誤除外) 數量。 單位：Count

使用 記錄 Amazon Kinesis Data Streams API 呼叫 AWS CloudTrail

Amazon Kinesis Data Streams 已與 服務整合 AWS CloudTrail，此服務提供由使用者、角色或 Kinesis Data Streams 中的 AWS 服務所採取之動作的記錄。CloudTrail 會將 Kinesis Data Streams 的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 Amazon Kinesis Video Streams 主控台的呼叫，以及程式碼對 Kinesis Data Streams API 操作的呼叫。如果您建立線索，就可以持續將 CloudTrail 事件交付至 Amazon S3 儲存貯體，包括 Kinesis Data Streams 的事件。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的事件歷史記錄檢視最新事件。您可以利用 CloudTrail 所收集的資訊來判斷向 Kinesis Data Streams 發出的請求，以及發出請求的 IP 地址、人員、時間和其他詳細資訊。

若要進一步了解 CloudTrail，包括如何設定及啟用，請參閱《AWS CloudTrail 使用者指南》。

CloudTrail 中的 Kinesis Data Streams 資訊

當您建立 AWS 帳戶時，會在您的帳戶上啟用 CloudTrail。當 Kinesis Data Streams 中發生支援的事件活動時，該活動會與事件歷史記錄中的其他 AWS 服務事件一起記錄在 CloudTrail 事件中。您可以在 AWS 帳戶中檢視、搜尋和下載最近的事件。如需詳細資訊，請參閱《使用 CloudTrail 事件歷史記錄檢視事件》https://docs.aws.amazon.com/awscloudtrail/latest/userguide/view-cloudtrail-events.html。

若要持續記錄您 AWS 帳戶中的事件，包括 Kinesis Data Streams 的事件，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台中建立線索時，線索會套用至所有 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案傳送到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 從多個區域接收 CloudTrail 日誌檔案，以及從多個帳戶接收 CloudTrail 日誌檔案

Kinesis Data Streams 支援將下列 API 動作記錄為 CloudTrail 日誌檔案中的事件：

- AddTagsToStream
- CreateStream
- DecreaseStreamRetentionPeriod
- DeleteStream
- DeregisterStreamConsumer
- DescribeStream
- DescribeStreamConsumer
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- GetRecords
- GetShardIterator
- IncreaseStreamRetentionPeriod
- ListStreamConsumers
- ListStreams
- ListTagsForStream
- MergeShards
- PutRecord
- PutRecords
- RegisterStreamConsumer

- RemoveTagsFromStream
- SplitShard
- StartStreamEncryption
- StopStreamEncryption
- SubscribeToShard
- UpdateShardCount
- UpdateStreamMode

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 是否使用根或 AWS Identity and Access Management (IAM) 使用者登入資料提出請求。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

範例：Kinesis Data Streams 日誌檔案項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌專案。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

以下範例顯示的是展示 `CreateStream`、`DescribeStream`、`ListStreams`、`DeleteStream`、`SplitShard` 及 `MergeShards` 動作的 CloudTrail 日誌項目。

```
{
    "Records": [
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
```

```
            },
            "eventTime": "2014-04-19T00:16:31Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "CreateStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardCount": 1,
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
            "eventID": "b7acfcd0-6ca9-4ee1-a3d7-c4e8d420d99b"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:06Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DescribeStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "f0944d86-c757-11e3-b4ae-25654b1d3136",
            "eventID": "0b2f1396-88af-4561-b16f-398f8eaea596"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
```

```
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:02Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "ListStreams",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "limit": 10
            },
            "responseElements": null,
            "requestID": "a68541ca-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "22a5fb8f-4e61-4bee-a8ad-3b72046b4c4d"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:07Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DeleteStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "f10cd97c-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "607e7217-311a-4a08-a904-ec02944596dd"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
```

```
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:03Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "SplitShard",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardToSplit": "shardId-000000000000",
                "streamName": "GoodStream",
                "newStartingHashKey": "11111111"
            },
            "responseElements": null,
            "requestID": "a6e6e9cd-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "dcd2126f-c8d2-4186-b32a-192dd48d7e33"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:16:56Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "MergeShards",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream",
                "adjacentShardToMerge": "shardId-000000000002",
                "shardToMerge": "shardId-000000000001"
            },
            "responseElements": null,
            "requestID": "e9f9c8eb-c757-11e3-bf1d-6948db3cd570",
```

```
            "eventID": "77cf0d06-ce90-42da-9576-71986fec411f"
        }
    ]
}
```

使用 Amazon CloudWatch 監控 Kinesis 用戶端程式庫

適用於 Amazon Kinesis Data Streams 的 Kinesis Client Library (KCL) 會代您發佈自訂的 Amazon CloudWatch 指標，並使用您的 KCL 應用程式的名稱作為命名空間。然後，您可以導覽至 CloudWatch 主控台，並選擇自訂指標來檢視這些指標。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的發佈自訂指標。

這是 KCL 將指標上傳到 CloudWatch 的名目費用；具體而言，適用 Amazon CloudWatch 自訂指標和 Amazon CloudWatch API 請求費用。如需詳細資訊，請參閱 Amazon CloudWatch 定價。

主題

- 指標和命名空間
- 指標層級和維度
- 指標組態
- 指標清單

指標和命名空間

用來上傳指標的命名空間是當您啟動 KCL 時指定的應用程式名稱。

指標層級和維度

有兩種選項可控制上傳到 CloudWatch 的指標：

指標層級

每個指標會獲指派個別層級。設定指標報告層級時，具有單一層級的指標若低於報告層級即不會傳送到 CloudWatch。層級為：NONE、SUMMARY 和 DETAILED。預設設定為 DETAILED；亦即，所有指標都會傳送到 CloudWatch。NONE 報告層級表示不會傳送任何指標。如需有關指派哪些層級給哪些指標的詳細資訊，請參閱 指標清單。

已啟用的維度

每個 KCL 指標的關聯維度也會傳送到 CloudWatch。在 KCL 2.x 中，如果 KCL 設定為處理單一資料串流，則預設會啟用所有指標維度 (Operation、ShardId、和 WorkerIdentifier)。此外，在 KCL 2.x 中，如果 KCL 設定為處理單一資料串流，則無法停用 Operation 維度。在 KCL 2.x 中，如果 KCL 設定為處理多個資料串流，則預設會啟用所有指標維度 (Operation、StreamId、ShardId和 WorkerIdentifier)。此外，在 KCL 2.x 中，如果 KCL 設定為處理多個資料串流，則無法停用 Operation和StreamId維度。StreamId維度僅適用於每個碎片指標。

在 KCL 1.x 中，預設情況下，只有 Operation 和 ShardId 維度是啟用的，WorkerIdentifier 維度是停用的。在 KCL 1.x 中，無法停用該 Operation 維度。

如需 CloudWatch 指標維度的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中 Amazon CloudWatch 概念主題下的維度部分。

啟用 WorkerIdentifier 維度時，如果每次特定 KCL 工作者重新啟動時對工作者 ID 屬性使用不同的值，即會隨著新 WorkerIdentifier 維度值將新的指標集傳送到 CloudWatch。如果您需要 WorkerIdentifier 維度值在特定 KCL 工作者重新啟動時是相同的，您必須在每個工作者的初始化期間明確指定相同的工作者 ID 值。請注意，每個作用中 KCL 工作者的工作者 ID 值，在所有 KCL 工作者間必須是唯一的。

指標組態

指標層級和啟用的維度可以使用 KinesisClientLibConfiguration 執行個體加以設定，會在啟動 KCL 應用程式時將該執行個體傳送至工作者。在 MultiLangDaemon 情況下，您可以在 .properties 檔案中指定用於啟動 MultiLangDaemon KCL 應用程式的 metricsLevel 和 metricsEnabledDimensions 屬性。

您可以對指標層級指派下列三個值中的其中之一：NONE、SUMMARY 或 DETAILED。啟用的維度值必須是對 CloudWatch 指標允許之維度以逗號分隔字串的清單。KCL 應用程式使用的維度為 Operation、ShardId 及 WorkerIdentifier。

指標清單

下表列出 KCL 指標，依範圍和操作分組。

主題

- Per-KCL-application指標
- 每個工作者的指標

- 每個碎片指標

Per-KCL-application指標

這些指標會在應用程式範圍內的所有 KCL 工作者間彙總，如 Amazon CloudWatch 命名空間所定義。

主題

- LeaseAssignmentManager
- InitializeTask
- ShutdownTask
- ShardSyncTask
- BlockOnParentTask
- PeriodicShardSyncManager
- MultistreamTracker

LeaseAssignmentManager

`LeaseAssignmentManager` 此操作負責將租用指派給工作者，並在工作者之間重新平衡租用，以實現工作者資源的均勻使用率。此操作的邏輯包括從租用資料表讀取租用相關的中繼資料，以及從工作者指標資料表讀取指標，以及執行租用指派。

指標	Description
LeaseAndWorkerMetricsLoad.Time	載入租用指派管理員 (LAM) 中的所有租用和工作者指標項目所需的時間，這是 KCL 3.x 中引入的新租用指派和負載平衡演算法。 指標層級：詳細 單位：毫秒
TotalLeases	目前 KCL 應用程式的租用總數。 指標層級：摘要 單位：Count
NumWorkers	目前 KCL 應用程式中的工作者總數。

指標	Description
	指標層級：摘要 單位：Count
AssignExpiredOrUnassignedLeases.Time	執行過期租用的記憶體內指派的時間。 指標層級：詳細 單位：毫秒
LeaseSpillover	由於達到租用數量上限或每個工作者輸送量上限而未指派的租用數量。 指標層級：摘要 單位：Count
BalanceWorkerVariance.Time	在工作者之間執行記憶體內平衡租用的時間。 指標層級：詳細 單位：毫秒
NumOfLeasesReassignment	目前重新指派反覆運算中所做的租用重新指派總數。 指標層級：摘要 單位：Count
FailedAssignmentCount	AssignLease 呼叫 DynamoDB 租用資料表中的失敗次數。 指標層級：詳細 單位：Count
ParallelyAssignLeases.Time	排清 DynamoDB 租用資料表新指派的時間。 指標層級：詳細 單位：毫秒

指標	Description
ParallelyAssignLeases.Success	新指派成功排清的數量。 指標層級：詳細 單位：Count
TotalStaleWorkerMetricsEntry	必須清除的工作者指標項目總數。 指標層級：詳細 單位：Count
StaleWorkerMetricsCleanup.Time	從 DynamoDB 工作者指標資料表執行工作者指標項目刪除的時間。 指標層級：詳細 單位：毫秒
時間	`LeaseAssignmentManager` 操作所花費的時間。 指標層級：摘要 單位：毫秒
成功	成功完成 `LeaseAssignmentManager` 操作的次數。 指標層級：摘要 單位：Count
ForceLeaderRelease	表示租用指派管理員連續失敗 3 次，且領導者工作者正在釋出領導。 指標層級：摘要 單位：Count
NumWorkersWithInvalidEntry	視為無效的工作者指標項目數目。 指標層級：摘要 單位：Count

指標	Description
NumWorkersWithFailingWorkerMetric	具有 -1 的工作者指標項目數目（表示工作者指標值不可用）作為工作者指標的其中一個值。 指標層級：摘要 單位：Count
LeaseDeserializationFailureCount	租用資料表中無法還原序列化的租用項目。 指標層級：摘要 單位：Count

InitializeTask

`InitializeTask` 操作負責初始化 KCL 應用程式的記錄處理器。此操作的邏輯包含取得來自 Kinesis Data Streams 的碎片反覆運算器和初始化記錄處理器。

指標	Description
KinesisDataFetcher.getIterator.Success	每一 KCL 應用程式成功 `GetShardIterator` 操作的數量。 指標層級：詳細 單位：Count
KinesisDataFetcher.getIterator.Time	指定 KCL 應用程式的每個 `GetShardIterator` 操作耗費的時間。 指標層級：詳細 單位：毫秒
RecordProcessor.initialize.Time	記錄處理器的 initialize 方法耗費的時間。 指標層級：摘要 單位：毫秒
成功	成功的記錄處理器初始化的數量。

指標	Description
	指標層級：摘要 單位：Count
時間	KCL 工作者的記錄處理器初始化耗費的時間。 指標層級：摘要 單位：毫秒

ShutdownTask

`ShutdownTask` 操作會初始化碎片處理的關閉序列。因為碎片分割或合併，或當工作者遺失碎片租用時可能會發生此情況。在這兩種情況下，會叫用記錄處理器 `shutdown()` 函數。當碎片分割或合併造成建立一或兩個新的碎片的情況下，也會發現新的碎片。

指標	Description
CreateLease.Success	在父碎片關閉之後將新子碎片成功新增到 KCL 應用程式 DynamoDB 資料表的次數。 指標層級：詳細 單位：Count
CreateLease.Time	在 KCL 應用程式 DynamoDB 資料表中新增子碎片資訊耗費的時間。 指標層級：詳細 單位：毫秒
UpdateLease.Success	記錄處理器關閉期間成功的最終檢查點數量。 指標層級：詳細 單位：Count
UpdateLease.Time	記錄處理器關閉期間檢查點操作耗費的時間。

指標	Description
	指標層級：詳細 單位：毫秒
RecordProcessor.shutdown.Time	記錄處理器的 shutdown 方法耗費的時間。 指標層級：摘要 單位：毫秒
成功	成功關閉任務的數量。 指標層級：摘要 單位：Count
時間	KCL 工作者的關閉任務耗費的時間。 指標層級：摘要 單位：毫秒

ShardSyncTask

`ShardSyncTask` 操作會探索對 Kinesis 資料串流碎片資訊的變更，使得 KCL 應用程式可以處理新的碎片。

指標	Description
CreateLease.Success	成功嘗試將新的碎片資訊新增至 KCL 應用程式 DynamoDB 資料表的數量。 指標層級：詳細 單位：Count
CreateLease.Time	在 KCL 應用程式 DynamoDB 資料表中新增碎片資訊耗費的時間。 指標層級：詳細

指標	Description
	單位：毫秒
成功	成功的碎片同步操作數量。 指標層級：摘要 單位：Count
時間	碎片同步操作耗費的時間。 指標層級：摘要 單位：毫秒

BlockOnParentTask

如果碎片分割或與其他碎片合併，則會建立新的子碎片。`BlockOnParentTask` 操作可確保新碎片的記錄處理不會開始，直到 KCL 已完全處理父碎片為止。

指標	Description
成功	成功的父碎片完成檢查數量。 指標層級：摘要 單位：Count
時間	父碎片完成耗費的時間。 指標層級：摘要 單位：毫秒

PeriodicShardSyncManager

`PeriodicShardSyncManager` 負責檢查 KCL 取用者應用程式正在處理的資料串流，識別具有部分租用的資料串流，並將其移交以進行同步處理。

當 KCL 設定為處理單一資料串流 (然後將 NumStreamStoSync 和 NumStreamsWithPartialLeases 設定為 1)，以及將 KCL 設定為處理多個資料串流時，可以使用下列指標。

指標	Description
NumStreamsToSync	消費者應用程式處理的資料串流數目 （每個 AWS 帳戶），其中包含部分租用且必須遞交以進行同步處理。 指標層級：摘要 單位：Count
NumStreamsWithPartialLeases	消費者應用程式正在處理的資料串流數目 （每個 AWS 帳戶），其中包含部分租用。 指標層級：摘要 單位：Count
成功	PeriodicShardSyncManager 能夠在取用者應用程式正在處理的資料串流中成功識別部分租用的次數。 指標層級：摘要 單位：Count
時間	檢查取用者應用程式正在處理的資料串流PeriodicShardSyncManager 所需的時間量 （以毫秒為單位），以判斷哪些資料串流需要碎片同步處理。 指標層級：摘要 單位：毫秒

MultistreamTracker

此 MultistreamTracker 介面可讓您建置可同時處理多個資料串流的 KCL 取用者應用程式。

指標	Description
DeletedStreams.Count	在此期間刪除的資料串流數目。 指標層級：摘要 單位：Count
ActiveStreams.Count	正在處理的作用中資料串流數目。 指標層級：摘要 單位：Count
StreamsPendingDeletion.Count	擱置刪除的資料串流數目 (依據 `FormerStreamsLeasesDeletionStrategy`)。 指標層級：摘要 單位：Count

每個工作者的指標

系統會針對耗用來自 Kinesis 資料串流 (例如：Amazon EC2 執行個體) 之資料的所有記錄處理器彙總這些指標。

主題

- WorkerMetricStatsReporter
- LeaseDiscovery
- RenewAllLeases
- TakeLeases

WorkerMetricStatsReporter

`WorkerMetricStatReporter` 操作負責定期將目前工作者的指標發佈至工作者指標表。`LeaseAssignmentManager` 操作使用這些指標來執行租用指派。

指標	Description
InMemoryMetricStatsReporterFailure	由於某些工作者指標失敗，擷取記憶體內工作者指標值的失敗次數。 指標層級：摘要 單位：Count
WorkerMetricStatsReporter.Time	`WorkerMetricsStats` 操作所花費的時間。 指標層級：摘要 單位：毫秒
WorkerMetricStatsReporter.Success	成功完成 `WorkerMetricsStats` 操作的次數。 指標層級：摘要 單位：Count

LeaseDiscovery

`LeaseDiscovery` 操作負責識別操作指派給目前工作者的新租用`LeaseAssignmentManager`。此操作的邏輯涉及透過讀取租用資料表的全域次要索引來識別指派給目前工作者的租用。

指標	Description
ListLeaseKeysForWorker.Time	呼叫租用資料表上的全域次要索引，並取得指派給目前工作者的租用金鑰的時間。 指標層級：詳細 單位：毫秒
FetchNewLeases.Time	從租用資料表擷取所有新租用的時間。 指標層級：詳細 單位：毫秒

指標	Description
NewLeasesDiscovered	指派給工作者的新租用總數。 指標層級：詳細 單位：Count
時間	`LeaseDiscovery` 操作所花費的時間。 指標層級：摘要 單位：毫秒
成功	成功完成 `LeaseDiscovery` 操作的次數。 指標層級：摘要 單位：Count
OwnerMismatch	來自 GSI 回應和租用資料表一致讀取的擁有者不相符數量。 指標層級：詳細 單位：Count

RenewAllLeases

`RenewAllLeases` 操作會定期更新特定工作者執行個體擁有的碎片租用。

指標	Description
RenewLease.Success	成功的工作者租用更新數量。 指標層級：詳細 單位：Count
RenewLease.Time	租用更新操作耗費的時間。 指標層級：詳細

指標	Description
	單位：毫秒
CurrentLeases	在更新所有租用之後，工作者擁有的碎片租用數量。 指標層級：摘要 單位：Count
LostLeases	嘗試更新工作者擁有的所有租用之後，遺失的碎片租用數量。 指標層級：摘要 單位：Count
成功	工作者的租用續約操作成功次數。 指標層級：摘要 單位：Count
時間	更新工作者所有租用耗費的時間。 指標層級：摘要 單位：毫秒

TakeLeases

TakeLeases 操作會平衡所有 KCL 工作者之間的記錄處理。如果目前的 KCL 工作者擁有的碎片較所需的碎片更少，則會從另一個過載的工作者取得碎片租用。

指標	Description
ListLeases.Success	已成功從 KCL 應用程式 DynamoDB 資料表擷取之所有碎片租用的次數。 指標層級：詳細 單位：Count
ListLeases.Time	從 KCL 應用程式 DynamoDB 資料表擷取所有碎片租用耗費的時間。

指標	Description
	指標層級：詳細 單位：毫秒
TakeLease.Success	工作者成功從其他 KCL 工作者取得碎片租用的次數。 指標層級：詳細 單位：Count
TakeLease.Time	更新工作者取得之租用的租用資料表耗費的時間。 指標層級：詳細 單位：毫秒
NumWorkers	工作者的總數量，如特定工作者所識別。 指標層級：摘要 單位：Count
NeededLeases	目前的工作者對平衡碎片處理負載所需的碎片租用數量。 指標層級：詳細 單位：Count
LeasesToTake	工作者將嘗試取得的租用數量。 指標層級：詳細 單位：Count
TakenLeases	工作者成功取得的租用數量。 指標層級：摘要 單位：Count

指標	Description
TotalLeases	KCL 應用程式正在處理的碎片總數量。 指標層級：詳細 單位：Count
ExpiredLeases	任何工作者未處理的碎片總數量，如特定工作者所識別。 指標層級：摘要 單位：Count
成功	成功完成 `TakeLeases` 操作的次數。 指標層級：摘要 單位：Count
時間	工作者的 `TakeLeases` 操作耗費的時間。 指標層級：摘要 單位：毫秒

每個碎片指標

系統會對單一記錄處理器彙總這些指標。

ProcessTask

`ProcessTask` 操作會以目前反覆運算器的位置呼叫 GetRecords，以從串流擷取記錄並呼叫記錄處理器 `processRecords` 功能。

指標	Description
KinesisDataFetcher .getRecords.Success	每一 Kinesis 資料串流碎片成功的 `GetRecords` 操作數量。 指標層級：詳細

指標	Description
	單位：Count
KinesisDataFetcher.getRecords.Time	Kinesis 資料串流碎片的每個 `GetRecords` 操作耗費的時間。 指標層級：詳細 單位：毫秒
UpdateLease.Success	記錄處理器針對指定碎片進行的成功檢查點數量。 指標層級：詳細 單位：Count
UpdateLease.Time	針對指定碎片的每個檢查點操作耗費的時間。 指標層級：詳細 單位：毫秒
DataBytesProcessed	每個 `ProcessTask` 叫用上處理的記錄總大小 (以位元組為單位)。 指標層級：摘要 單位：位元組
RecordsProcessed	每個 `ProcessTask` 叫用上處理的記錄數。 指標層級：摘要 單位：Count
ExpiredIterator	呼叫 `GetRecords` 時收到的 ExpiredIteratorException 數量。 指標層級：摘要 單位：Count

指標	Description
MillisBehindLatest	目前反覆運算器落後碎片中最新記錄 (頂端) 的時間。這個值小於或等於回應中最新記錄與目前時間之間的時間差異。這是碎片與尖端距離的更準確反映，而不是比較上次回應記錄中的時間戳記。此值適用於最新的記錄批次，而不是每個記錄中所有時間戳記的平均值。 指標層級：摘要 單位：毫秒
RecordProcessor.processRecords.Time	記錄處理器的 `processRecords` 方法耗費的時間。 指標層級：摘要 單位：毫秒
成功	成功的處理任務操作數量。 指標層級：摘要 單位：Count
時間	處理任務操作耗費的時間。 指標層級：摘要 單位：毫秒

使用 Amazon CloudWatch 監控 Kinesis Producer Library

Amazon Kinesis Data Streams 的 Amazon Kinesis Producer Library (KPL) 會代表您發佈自訂 Amazon CloudWatch 指標。 Amazon Kinesis 然後，您可以導覽至 CloudWatch 主控台，並選擇自訂指標來檢視這些指標。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的發佈自訂指標。

這是 KPL 將指標上傳到 CloudWatch 的名目費用；具體而言，適用 Amazon CloudWatch 自訂指標和 Amazon CloudWatch API 請求費用。如需詳細資訊，請參閱 Amazon CloudWatch 定價。蒐集本機指標不會產生 CloudWatch 費用。

主題

指標、維度和命名空間

啟動 KPL 時您可以指定應用程式名稱，然後在上傳指標時使用它做為命名空間的一部分。這是選用的；如果未設定應用程式名稱，KPL 會提供預設值。

您也可以設定 KPL 以將任意的其他維度新增到指標。如果您需要在 CloudWatch 指標中有更精確的資料，此功能會很有用。例如，您可以將主機名稱新增為維度，即可讓您識別機群中不平均的負載分配。所有 KPL 組態設定是不變的，因此在初始化 KPL 執行個體之後，您無法變更這些額外的維度。

指標層級和精細程度

有兩種選項可控制上傳到 CloudWatch 的指標數量：

指標層級

這是指標重要性的約略判斷方法。每個指標會獲指派層級。設定層級時，低於該層級的指標不會傳送到 CloudWatch。層級為 NONE、SUMMARY 和 DETAILED。預設設定為 DETAILED；也就是說，所有指標 NONE 表示完全不須指標，因此不會將任何指標指派至該層級。

精細程度

此選項可控制是否以額外的精細程度層級發出相同的指標。層級為 GLOBAL、STREAM 和 SHARD。預設設定為 SHARD，其包含最精細的指標。

選擇 SHARD 時，會以串流名稱和碎片 ID 做為維度發出指標。此外，也會僅以串流名稱維度，以及沒有串流名稱的指標發出相同的指標。這表示，對於特定指標，兩個串流 (其中每個串流擁有兩個碎片) 將產生 7 個 CloudWatch 指標：每個碎片一個、每個串流一個，以及一個整體；都描述相同的統計資料，但在不同層級的精細程度。如需圖解，請參閱以下圖表。

不同精細程度層級形成階層，而系統中的所有指標形成樹狀目錄，根目錄為指標名稱：

```
MetricName (GLOBAL):           Metric X                    Metric Y
                                  |                           |
                          -----------------             ------------
                          |               |             |          |
StreamName (STREAM):    Stream A        Stream B      Stream A   Stream B
```

```
                          |             |
                       --------      ---------
                       |      |      |       |
ShardID (SHARD):     Shard 0 Shard 1  Shard 0 Shard 1
```

並非所有指標都可在碎片層級取得；而部分指標的本質為串流層級或全域。即使您已啟用碎片層級指標 (前述圖表中的 `Metric Y`)，也不會在碎片層級產生這些指標。

當您指定額外的維度時，必須提供 的值`tuple:<DimensionName, DimensionValue, Granularity>`。精細程序用來決定在階層中插入自訂維度的位置：`GLOBAL` 表示在指標名稱後插入額外的維度，`STREAM` 表示在串流名稱之後插入，和 `SHARD` 表示在碎片 ID 之後插入。如果為每個精細程序層級提供多個額外的維度，則會以指定的順序插入。

本機存取和 Amazon CloudWatch 上傳

目前 KPL 執行個體的指標可即時在本機取得；您可以隨時查詢 KPL 來取得這些指標。KPL 會在本機計算每個指標的總和、平均、最小值、最大值和計數，如同在 CloudWatch 一般。

您可以取得從計畫開始累計到目前時間點的統計資料，或使用過去 N 秒的滾動時段，其中 N 為 1 到 60 之間的整數。

所有指標可供上傳至 CloudWatch。這非常適合用於彙總跨多個主機、監控和警示的資料。此功能無法在本機使用。

如前所述，您可以選取要使用指標層級和精細程度設定來上傳的指標。未上傳的指標可在本機取得。

個別上傳資料點不可行，因為如果流量很高，它可能產生每秒數百萬個上傳。因此，KPL 會在本機將指標彙總為 1 分鐘的儲存貯體，並對每個啟用的指標每分鐘一次將統計資料物件上傳至 CloudWatch。

指標清單

指標	Description
`UserRecordsReceived`	KPL 核心針對 put 操作所收到多少個邏輯使用者記錄的計數。碎片層級不適用。 指標層級：詳細 單位：計數

指標	Description
UserRecordsPending	目前待處理的有多少個使用者記錄的定期取樣。如果記錄目前處於緩衝狀態並等待傳送，或正在傳送和傳輸到後端服務，則該記錄會處於待定狀態。碎片層級不適用。 KPL 提供一個專用的方法，可在全域層級擷取此指標，讓客戶管理其 put 速率。 指標層級：詳細 單位：計數
UserRecordsPut	已成功 put 多少個邏輯使用者記錄的計數。 KPL 會對失敗的記錄輸出零。這可讓平均提供成功率，讓計數提供嘗試總計，以及讓計數與總和之間的差異提供失敗計數。 指標層級：摘要 單位：計數
UserRecordsDataPut	邏輯使用者記錄中成功 put 的位元組數。 指標層級：詳細 單位：位元組
KinesisRecordsPut	成功 put 的 Kinesis Data Streams 記錄的計數 (每個 Kinesis Data Streams 記錄可能包含多個使用者記錄)。 KPL 會對失敗的記錄輸出零。這可讓平均提供成功率，讓計數提供嘗試總計，以及讓計數與總和之間的差異提供失敗計數。 指標層級：摘要 單位：計數

指標	Description
KinesisRecordsDataPut	Kinesis Data Streams 記錄中的位元組。 指標層級：詳細 單位：位元組
ErrorsByCode	每個錯誤碼類型的計數。這會在一般維度 (例如 ErrorCode 和 StreamName) 之外產生額外的 ShardId 維度。並非每個錯誤都能追蹤至碎片。不能追蹤的錯誤只能在串流或全域層級發出。此指標會擷取限流、碎片對應變更、內部失敗、服務無法使用、逾時等等之類的相關資訊。 Kinesis Data Streams API 錯誤會在每筆 Kinesis Data Streams 記錄中計算一次。Kinesis Data Streams 記錄內的多個使用者記錄不會產生多個計數。 指標層級：摘要 單位：計數
AllErrors	這是由與依程式碼的錯誤之相同錯誤所觸發，但不會區分類型。這很實用，因為錯誤率的一般監控不需要來自所有不同錯誤類型計數的手動總和。 指標層級：摘要 單位：計數
RetriesPerRecord	每個使用者記錄執行的重試次數。對一次嘗試便成功的記錄會發出零。 資料會在使用者記錄完成 (可能是成功或可能不再可重試) 時發出。如果記錄存活期的值很大，此指標可能會大幅延遲。 指標層級：詳細 單位：計數

指標	Description
BufferingTime	使用者記錄抵達 KPL 和前往後端之間的時間。會依每個記錄為基礎將此資訊傳輸回使用者，但也會提供做為彙總的統計資料。 指標層級：摘要 單位：毫秒
Request Time	執行 PutRecordsRequests 所耗費的時間。 指標層級：詳細 單位：毫秒
User Records per Kinesis Record	彙總至單一 Kinesis Data Streams 記錄的邏輯使用者記錄數量。 指標層級：詳細 單位：計數
Amazon Kinesis Records per PutRecordsRequest	彙總至單一 PutRecordsRequest 的 Kinesis Data Streams 記錄數量。碎片層級不適用。 指標層級：詳細 單位：計數
User Records per PutRecordsRequest	PutRecordsRequest 內包含之使用者記錄的總數。這大約等同於之前兩個指標的乘積。碎片層級不適用。 指標層級：詳細 單位：計數

Amazon Kinesis Data Streams 中的安全性

的雲端安全 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 AWS Cloud 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。在 AWS 合規計畫中，第三方稽核員會定期測試並驗證我們的安全功效。如要了解適用於 Kinesis Data Streams 的合規計畫，請參閱 合規計畫的AWS 服務範圍。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對資料敏感度、組織要求，以及適用法律和法規等其他因素負責。

本文件有助於您了解如何在使用 Kinesis Data Streams 時套用共同責任模型。下列主題說明如何將 Kinesis Data Streams 設定為達到您的安全及合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 Kinesis Data Streams 資源。

主題

Amazon Kinesis Data Streams 中的資料保護

使用 AWS Key Management Service (AWS KMS) 金鑰的伺服器端加密可讓您在 Amazon Kinesis Data Streams 中加密靜態資料，輕鬆滿足嚴格的資料管理需求。

> **Note**
>
> 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-2 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

主題

什麼是 Kinesis Data Streams 的伺服器端加密？

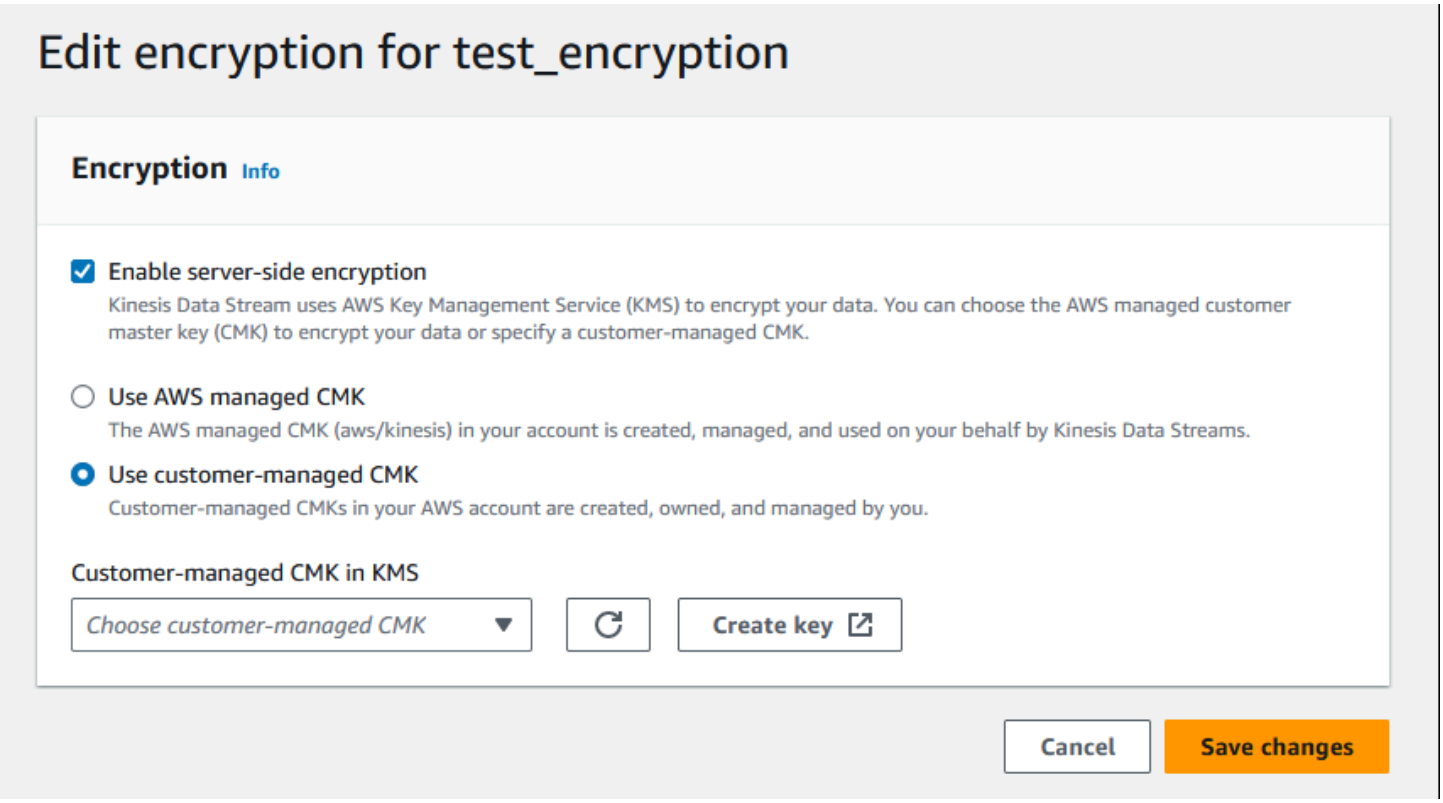
伺服器端加密是 Amazon Kinesis Data Streams 的一項功能，會使用您指定的 AWS KMS 客戶主金鑰 (CMK)，在資料處於靜態狀態之前自動加密資料。資料會在寫入 Kinesis 串流儲存層之前加密，並在從儲存體擷取後解密。因此，您的資料將於 Kinesis Data Streams 服務中呈靜態狀態下進行加密。這樣您就能夠符合嚴格的法規要求並增強資料的安全性。

有了伺服器端加密，您的 Kinesis 串流生產者及取用者就不需要管理主金鑰或密碼編譯操作。您的資料會在進入和離開 Kinesis Data Streams 服務時自動加密，因此您的靜態資料會加密。 AWS KMS 提供伺服器端加密功能所使用的所有主金鑰。 AWS KMS 可讓您輕鬆地使用由 管理的 Kinesis CMK AWS、使用者指定的 AWS KMS CMK 或匯入 AWS KMS 服務的主金鑰。

> **Note**
>
> 伺服器端加密僅在啟用加密後對傳入資料加密。啟用伺服器端加密後，未加密的串流中既有的資料並不會加密。

加密資料串流並共用存取權給其他主體時，您必須在金鑰政策中授予外部帳戶中 AWS KMS 金鑰和 IAM 政策的許可。如需詳細資訊，請參閱允許其他帳戶中的使用者使用 KMS 金鑰。

如果您已為具有 AWS 受管 KMS 金鑰的資料串流啟用伺服器端加密，並想要透過資源政策共用存取權，則必須切換到使用客戶受管金鑰 (CMK)，如下所示：



此外，您必須允許共用主體實體以使用 KMS 跨帳戶共用功能來存取您的 CMK。請務必同時在共用主體實體的 IAM 政策中進行變更。如需詳細資訊，請參閱[允許其他帳戶中的使用者使用 KMS 金鑰]。

成本、區域和效能考量

當您套用伺服器端加密時，您需要支付 AWS KMS API 用量和金鑰成本。有別於自訂 KMS 主金鑰，(Default) aws/kinesis 客戶主金鑰 (CMK) 為免費提供。不過您仍然必須支付 Amazon Kinesis Data Streams 代表您產生的 API 使用成本。

API 使用成本適用於每個 CMK (包括自訂)。Kinesis Data Streams 在輪換資料金鑰時，大約每五分鐘呼叫一次 AWS KMS 。在 30 天月中，由 Kinesis 串流啟動的 AWS KMS API 呼叫總成本應少於幾美元。此成本會隨著您在資料生產者和消費者上使用的使用者登入資料數量而擴展，因為每個使用者登入資料都需要唯一的 API 呼叫 AWS KMS。當您使用 IAM 角色進行身分驗證時，每次擔任角色呼叫都將產生獨一無二的使用者憑證。為節省 KMS 成本，您可能要快取由擔任角色呼叫傳回的使用者登入資料。

以下依據資源說明各項成本：

金鑰

- 由 (AWS 別名 = aws/kinesis) 管理的 Kinesis CMK 是免費的。

- 使用者產生的 KMS 金鑰需支付 KMS 金鑰成本。如需更多資訊，請參閱 AWS Key Management Service 定價。

API 使用成本適用於每個 CMK (包括自訂)。Kinesis Data Streams 在輪換資料金鑰時，大約每五分鐘呼叫一次 KMS。以一個月 30 天計算，由 Kinesis 資料串流初始化的 KMS API 呼叫總成本應該不到幾美元。請注意，此成本會隨著您在資料生產者和消費者上使用的使用者登入資料數量而擴展，因為每個使用者登入資料都需要對 AWS KMS 進行唯一的 API 呼叫。當您使用 IAM 角色進行身份驗證時，每個假設角色呼叫都會產生唯一的使用者憑證，而且您可能想要快取假角色呼叫傳回的使用者憑證，以節省 KMS 成本。

KMS API 用量

對於每個加密的串流，從 TIP 讀取，並使用跨讀取器和寫入器的單一 IAM 帳戶/使用者存取金鑰時，Kinesis 服務每 5 分鐘會呼叫 AWS KMS 服務大約 12 次。未從 TIP 讀取可能會導致對 AWS KMS 服務的呼叫次數增加。產生新資料加密金鑰的 API 請求需支付 AWS KMS 使用成本。更多詳細資訊請參閱 AWS Key Management Service 定價：使用。

區域伺服器端加密的可用性

目前，Kinesis 串流的伺服器端加密可在 Kinesis Data Streams 支援的所有區域中使用，包括 AWS GovCloud（美國西部）和中國區域。如需 Kinesis Data Streams 支援區域的詳細資訊，請參閱 https：//https://docs.aws.amazon.com/general/latest/gr/ak.html。

效能考量

由於套用伺服器端加密造成服務負荷，套用伺服器端加密會增加 `PutRecord`，`PutRecords` 和 `GetRecords` 的延遲，一般不到 100 微秒。

如何開始使用伺服器端加密？

開始使用伺服器端加密的最簡單方法是使用 AWS 管理主控台 和 Amazon Kinesis KMS 服務金鑰 `aws/kinesis`。

以下程序說明如何為 Kinesis 串流啟用伺服器端加密。

為 Kinesis 串流啟用伺服器端加密

1. 登入 AWS 管理主控台 並開啟 Amazon Kinesis Data Streams 主控台。
2. 在 AWS 管理主控台建立或選取 Kinesis 串流。

3. 選擇 details (詳細資訊) 標籤。
4. 在 Server-side encryption (伺服器端加密) 中，選擇 edit (編輯)。
5. 除非您要使用由使用者產生的 KMS 主金鑰，否則請確定已選取 (Default) aws/kinesis (預設) KMS 主金鑰。這是 Kinesis 服務所產生的 KMS 主金鑰。選擇Enabled (已啟用)，然後選擇Save (儲存)。

   > **Note**
   >
   > 預設 Kinesis 服務主金鑰是免費的，但 Kinesis 對 AWS KMS 服務的 API 呼叫需支付 KMS 使用成本。

6. 串流會轉換到待定狀態。串流在啟用加密時恢復為作用中狀態後，所有寫入該串流的傳入資料都將使用您所選取的 KMS 主金鑰進行加密。
7. 若要停用伺服器端加密，請在 的伺服器端加密中選擇已停用 AWS 管理主控台，然後選擇儲存。

建立和使用使用者產生的 KMS 金鑰

本節說明如何建立和使用您自己的 KMS 金鑰，而不是使用 Amazon Kinesis 管理的主金鑰。

建立使用者產生的 KMS 金鑰

如需建立自有金鑰的說明，請參閱《 AWS Key Management Service 開發人員指南》中的建立金鑰。您為帳戶建立金鑰後，Kinesis Data Streams 服務會在 KMS 主金鑰清單中傳回這些金鑰。

使用使用者產生的 KMS 金鑰

將正確的許可套用至您的消費者、生產者和管理員之後，您可以在自己的 AWS 帳戶或其他帳戶中使用自訂 KMS 金鑰 AWS 。您帳戶中的所有 KMS 主金鑰會顯示在 的 KMS Master Key (KMS 主金鑰) AWS 管理主控台清單上。

若要使用另一帳戶中的自訂 KMS 主金鑰，您必須具備使用前述金鑰的許可。您還必須在 AWS 管理主控台的 ARN 輸入方塊內指定 KMS 主金鑰的 ARN。

使用使用者產生之 KMS 金鑰的許可

在搭配使用者產生的 KMS 金鑰使用伺服器端加密之前，您必須設定 AWS KMS 金鑰政策，以允許加密串流，以及加密和解密串流記錄。如需 AWS KMS 許可的範例和詳細資訊，請參閱 AWS KMS API 許可：動作和資源參考。

> **Note**
>
> 使用預設服務金鑰進行加密，不需要應用自訂 IAM 許可。

欲使用由使用者產生的 KMS 主金鑰之前，請確保您的 Kinesis 串流生產者及取用者 (IAM 委託人) 為 KMS 主金鑰政策中的使用者。否則串流的寫入及讀取將會失敗，最終可能造成資料遺失、延遲處理或應用程式故障。您可使用 IAM 政策管理 KMS 金鑰的許可。如需詳細資訊，請參閱搭配 AWS KMS 使用 IAM 政策。

Kinesis Data Streams 加密內容

當 Amazon Kinesis Data Streams 代表您呼叫 AWS KMS 時，它會將加密內容傳遞至 AWS KMS ，以做為金鑰政策和授權中的授權條件。Kinesis Data Streams 會在所有 AWS KMS 呼叫中使用串流 ARN 做為加密內容。

```
"encryptionContext": {
    "aws:kinesis:arn": "arn:aws:kinesis:region:account-id:stream/stream-name"
}
```

您可以使用加密內容來識別 KMS 金鑰在稽核記錄和日誌中的使用情況。它也會以純文字顯示在日誌中，例如 AWS CloudTrail。

若要將 KMS 金鑰的使用限制在特定串流的 Kinesis Data Streams 請求，請使用 KMS 金鑰政策或 IAM 政策中的 `kms:EncryptionContext:aws:kinesis:arn`條件金鑰。

範例生產者許可

您的 Kinesis 串流生產者必須具備 `kms:GenerateDataKey` 許可。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey"
        ],
```

```
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:PutRecord",
            "kinesis:PutRecords"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

範例取用者許可

您的 Kinesis 串流取用者必須具備 `kms:Decrypt` 許可。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt"
        ],
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:GetRecords",
            "kinesis:DescribeStream"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Amazon Managed Service for Apache Flink 和 AWS Lambda 使用角色來取用 Kinesis 串流。請務必為這些消費者所使用的角色新增 `kms:Decrypt` 許可。

串流管理員許可

Kinesis 串流管理員必須已獲授權呼叫 `kms:List*` 和 `kms:DescribeKey*`。

驗證 KMS 金鑰許可並進行疑難排解

Kinesis 串流一經啟用加密後，建議您使用以下 Amazon CloudWatch 指標監控 `putRecord`、`putRecords` 和 `getRecords` 呼叫是否成功：

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

如需詳細資訊，請參閱監控 Kinesis 資料串流

搭配介面 VPC 端點使用 Amazon Kinesis Data Streams

您可以使用介面 VPC 端點來防止 Amazon VPC 和 Kinesis Data Streams 之間的流量離開 Amazon 網路。介面 VPC 端點不需要網際網路閘道、NAT 裝置、VPN 連線或 Direct Connect 連線。介面 VPC 端點採用 AWS PrivateLink 技術，這項 AWS 技術可讓您在 Amazon VPC 中使用具有私有 IPs彈性網路介面，在 AWS 服務之間進行私有通訊。如需詳細資訊，請參閱 Amazon Virtual Private Cloud 和界面 VPC 端點 (AWS PrivateLink)。

主題

- 使用 Kinesis Data Streams 的介面 VPC 端點
- 控制對 Kinesis Data Streams VPC 端點的存取
- Kinesis Data Streams 的 VPC 端點政策可用性

使用 Kinesis Data Streams 的介面 VPC 端點

若要開始使用，您不需要變更串流、生產者或消費者的設定。為您的 Kinesis Data Streams 建立介面 VPC 端點，以啟動透過介面 VPC 端點往返 Amazon VPC 資源的流量。啟用 FIPS 的界面 VPC 端點適用於美國區域。如需詳細資訊，請參閱建立界面端點。

Amazon Kinesis Producer Library (KPL) 和 Kinesis Consumer Library (KCL) 呼叫 AWS 服務，例如使用公有端點或私有介面 VPC 端點的 Amazon CloudWatch 和 Amazon DynamoDB，以使用中者為準。例如，如果您的 KCL 應用程式在已啟用 VPC 端點的 DynamoDB 介面的 VPC 中執行，則 DynamoDB 與 KCL 應用程式之間的呼叫會流經介面 VPC 端點。

控制對 Kinesis Data Streams VPC 端點的存取

VPC 端點政策可讓您透過將政策連接至 VPC 端點，或使用連接至 IAM 使用者、群組或角色的政策中的其他欄位來控制存取，以限制只能透過指定的 VPC 端點進行存取。將這些政策與 IAM 政策搭配使用時，將特定串流的存取權限制在指定的 VPC 端點，以僅透過指定的 VPC 端點授予對 Kinesis 資料串流動作的存取權。

以下是存取 Kinesis 資料串流的範例端點政策。

- VPC 政策範例：唯讀存取 – 此範例政策可連接到 VPC 端點。(如需詳細資訊，請參閱 控制 Amazon VPC 資源的存取)。它會將動作限制為僅能透過其連接的 VPC 端點列出和描述 Kinesis 資料串流。

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- VPC 政策範例：限制對特定 Kinesis 資料串流的存取 – 此範例政策可連接到 VPC 端點。它會限制僅能透過其所連接的 VPC 端點存取特定資料串流。

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
```

```
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- IAM 政策範例：限制只能從特定 VPC 端點存取特定串流 - 此範例政策可以連接到 IAM 使用者、角色或群組。它會限制僅從指定的 VPC 端點對指定的 Kinesis 資料串流進行存取。

  JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AccessFromSpecificEndpoint",
         "Action": "kinesis:*",
         "Effect": "Deny",
         "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream",
         "Condition": { "StringNotEquals" : { "aws:sourceVpce":
 "vpce-11aa22bb" } }
      }
   ]
}
```

Kinesis Data Streams 的 VPC 端點政策可用性

下列 區域支援 Kinesis Data Streams 介面 VPC 端點與 政策：

- Europe (Paris)
- 歐洲 (愛爾蘭)
- 美國東部 (維吉尼亞北部)
- 歐洲 (斯德哥爾摩)
- 美國東部 (俄亥俄)
- 歐洲 (法蘭克福)
- 南美洲 (聖保羅)

- 歐洲 (倫敦)
- 亞太地區 (東京)
- 美國西部 (加利佛尼亞北部)
- 亞太地區 (新加坡)
- 亞太地區 (雪梨)
- 中國 (北京)
- 中國 (寧夏)
- 亞太地區 (香港)
- Middle East (Bahrain)
- 中東 (阿拉伯聯合大公國)
- 歐洲 (米蘭)
- 非洲 (開普敦)
- 亞太地區 (孟買)
- 亞太地區 (首爾)
- 加拿大 (中部)
- 美國西部 (奧勒岡)，usw2-az4 除外
- AWS GovCloud （美國東部）
- AWS GovCloud （美國西部）
- 亞太地區 (大阪)
- 歐洲 (蘇黎世)
- 亞太地區 (海德拉巴)

使用 IAM 控制對 Amazon Kinesis Data Streams 資源的存取

AWS Identity and Access Management (IAM) 可讓您執行下列動作：

- 在 AWS 您的帳戶下建立使用者和群組
- 將唯一的安全登入資料指派給您 AWS 帳戶下的每個使用者
- 控制每個使用者使用 AWS 資源執行任務的許可
- 允許另一個 AWS 帳戶中的使用者共用您的 AWS 資源

- 為您的 AWS 帳戶建立角色，並定義可擔任這些角色的使用者或服務
- 為您的企業使用現有的身分，授予使用 AWS 資源執行任務的許可

搭配 Kinesis Data Streams 使用 IAM，您可以控制組織中的使用者是否可以使用特定的 Kinesis Data Streams API 動作來執行任務，以及是否可以使用特定的 AWS 資源。

若您使用 Kinesis Client Library (KCL) 開發應用程式，您的政策就必須包含對 Amazon DynamoDB 和 Amazon CloudWatch 的許可；KCL 使用 DynamoDB 追蹤應用程式的狀態資訊，並且使用 CloudWatch 代表您向 CloudWatch 傳送 KCL 指標。如需 KCL 的詳細資訊，請參閱開發 KCL 1.x 消費者。

如需 IAM 的詳細資訊，請參閱下列各項：

- AWS Identity and Access Management (IAM)
- IAM 入門
- IAM 使用者指南

如需 IAM 和 Amazon DynamoDB 的詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的使用 IAM 控制對 Amazon DynamoDB 資源的存取。

如需 IAM 和 Amazon CloudWatch 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南AWS 》中的控制帳戶的使用者存取權。

目錄

政策語法

IAM 政策為包含一或多個陳述式的 JSON 文件。每個陳述式的結構如下所示：

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

陳述式由各種元素組成：

- Effect (效果)：效果 可以是 Allow 或 Deny。根據預設，IAM 使用者沒有使用資源和 API 動作的許可，因此所有請求均會遭到拒絕。明確允許覆寫預設值。明確拒絕覆寫任何允許。
- Action (動作)：動作 是您授予或拒絕許可的特定 API 動作。
- Resource (資源)：受動作影響的資源。若要在陳述式中指定資源，您必須使用其 Amazon Resource Name (ARN)。
- Condition (條件)：條件為選擇性。您可以用以控制何時政策開始生效。

當您建立和管理 IAM 政策時，可能想要使用 IAM 政策產生器和 IAM 政策模擬器。

適用於 Kinesis Data Streams 的動作

在 IAM 政策陳述式中，您可以從任何支援 IAM 的服務指定任何 API 動作。針對 Kinesis Data Streams，請在 API 動作名稱使用下列字首：kinesis:。例如：kinesis:CreateStream、kinesis:ListStreams 和 kinesis:DescribeStreamSummary。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

您也可以使用萬用字元指定多個動作。例如，您可以指定名稱開頭有「Get」文字的所有動作，如下所示：

```
"Action": "kinesis:Get*"
```

若要指定所有的 Kinesis Data Streams 操作，請使用 * 萬用字元，如下所示：

```
"Action": "kinesis:*"
```

如需完整 Kinesis Data Streams API 動作清單，請參閱 Amazon Kinesis API 參考。

Kinesis Data Streams 的 Amazon Resource Name (ARN)

每個 IAM 政策陳述式都會套用到您使用其 ARN 指定的資源。

對 Kinesis Data Streams 使用以下的 ARN 資源格式：

```
arn:aws:kinesis:region:account-id:stream/stream-name
```

例如：

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Kinesis Data Streams 的範例政策

以下範例政策示範如何控制使用者存取您的 Kinesis Data Streams。

Example 1: Allow users to get data from a stream

Example

此政策允許使用者或群組對指定的串流執行 `DescribeStreamSummary`、`GetShardIterator` 及 `GetRecords` 操作，對任何串流執行 `ListStreams`。此政策可套用到應能夠從特定串流取得資料的使用者。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "kinesis:Get*",
                "kinesis:DescribeStreamSummary"
            ],
            "Resource": [
            "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:ListStreams"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Example 2: Allow users to add data to any stream in the account

Example

此政策允許使用者或群組對某帳戶的任一串流使用 PutRecord 操作。此政策可套用到應能夠加入資料記錄至帳戶中所有串流的使用者。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord"
            ],
            "Resource": [
                "arn:aws:kinesis:us-east-1:111122223333:stream/*"
            ]
        }
```

```
    ]
}
```

Example 3: Allow any Kinesis Data Streams action on a specific stream

Example

此政策允許使用者或群組對指定的串流使用任何 Kinesis Data Streams 操作。此政策可套用到應該對特定串流具備管理控制權的使用者。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
            "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
            ]
        }
    ]
}
```

Example 4: Allow any Kinesis Data Streams action on any stream

Example

此政策允許使用者或群組對帳戶中的任何串流使用任何 Kinesis Data Streams 操作。由於此政策會授予對您所有串流的完整存取權，您應限定僅供管理員使用。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

與其他 帳戶共用您的資料串流

> **Note**
>
> Kinesis Producer Library 目前不支援在寫入資料串流時指定串流 ARN。如果您想要寫入跨帳戶資料串流，請使用 AWS SDK。

將資源型政策附加到您的資料串流以授予對其他帳戶、IAM 使用者或 IAM 角色的存取權。資源型政策是附加到資源 (如資料串流) 的 JSON 政策文件。這些政策會授予指定的主體許可，允許在該資源上執行特定的動作，並且定義資源所適用的條件。一個政策可以有多個陳述式。您必須在資源型政策中指定主體。委託人可以包括帳戶、使用者、角色、聯合身分使用者 AWS 或服務。您可以在 Kinesis Data Streams 主控台、API 或 SDK 中設定政策。

請注意，共用存取權給已註冊的取用者 (例如增強型散發) 需要資料串流 ARN 和取用者 ARN 這兩者的政策。

啟用跨帳戶存取

如需啟用跨帳戶存取權，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，做為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當委託人和資源位於不同的 AWS 帳戶中時，您還必須使用以身分為基礎的政策來授予委託人對資源的存取權。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。

如需有關使用跨帳戶存取之以資源為基礎的政策詳細資訊，請參閱 IAM 中的跨帳戶資源存取權。

資料串流管理員可以使用 AWS Identity and Access Management 政策來指定可存取內容的人員。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。

可以共用的 Kinesis Data Streams 動作：

Action	存取層級
DescribeStreamConsumer	消費者
DescribeStreamSummary	資料串流
GetRecords	資料串流
GetShardIterator	資料串流
ListShards	資料串流
PutRecord	資料串流
PutRecords	資料串流
SubscribeToShard	消費者

以下是使用資源型政策將跨帳戶存取權授予您的資料串流或以註冊取用者的範例。

若要執行跨帳戶動作，您必須指定資料串流存取權的串流 ARN，並指定已註冊取用者存取權的取用者 ARN。

Kinesis 資料串流的資源型政策範例

由於需要採取行動，共用已註冊取用者涉及資料串流政策和取用者政策。

> **Note**
>
> 以下為 `Principal` 的有效值範例：
>
> - `{"AWS": "123456789012"}`
> - IAM 使用者 – `{"AWS": "arn:aws:iam::123456789012:user/user-name"}`
> - IAM 角色 – `{"AWS":["arn:aws:iam::123456789012:role/role-name"]}`
> - 多個主體 (可以是帳戶、使用者、角色的組合) – `{"AWS":["123456789012", "123456789013", "arn:aws:iam::123456789012:user/user-name"]}`

Example 1: Write access to the data stream

Example

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_write_policy_ID",
    "Statement": [
        {
            "Sid": "writestatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Example 2: Read access to the data stream

Example

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "sharedthroughputreadstatement",
```

```
            "Effect": "Allow",
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Example 3: Share enhanced fan-out read access to a registered consumer

Example

資料串流政策聲明：

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "consumerreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role-name"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
```

```
    ]
}
```

取用者政策聲明：

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_efo_read_policy_ID",
    "Statement": [
        {
            "Sid": "eforeadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role-name"
            },
            "Action": [
                "kinesis:DescribeStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC/consumer/consumerDEF:1674696300"
        }
    ]
}
```

為了維持最低權限的政策，動作或主體欄位不支援萬用字元 (*)。

以程式設計方式管理資料串流的政策

在 之外 AWS 管理主控台，Kinesis Data Streams 有三個 API，用於管理您的資料串流政策：

- PutResourcePolicy
- GetResourcePolicy
- DeleteResourcePolicy

使用 PutResourePolicy 來附加或覆寫資料串流或取用者的政策。使用 GetResourcePolicy 來檢查和檢視指定資料串流或取用者的政策。使用 DeleteResourcePolicy 來刪除指定資料串流或取用者的政策。

政策限制

Kinesis Data Streams 資源政策有下列限制：

- 萬用字元 (*) 不支援協助防止透過直接連接到資料串流或已註冊消費者的資源政策授予廣泛的存取。此外，請仔細檢查下列政策，以確認它們未授予廣泛存取權：
  - 連接到相關聯 AWS 主體的身分型政策 （例如 IAM 角色）
  - 連接至相關聯 AWS 資源的資源型政策 （例如， AWS Key Management Service KMS 金鑰）
- AWS 委託人不支援 服務委託人，以防止潛在的混淆代理人。
- 不支援聯合主體。
- 不支援正式使用者 ID。
- 政策大小不得超過 20KB。

共用對加密資料的存取權

如果您已為具有 AWS 受管 KMS 金鑰的資料串流啟用伺服器端加密，並想要透過資源政策共用存取權，則必須切換到使用客戶受管金鑰 (CMK)。如需詳細資訊，請參閱什麼是 Kinesis Data Streams 的伺服器端加密？。此外，您必須允許共用主體實體以使用 KMS 跨帳戶共用功能來存取您的 CMK。請務必同時在共用主體實體的 IAM 政策中進行變更。如需詳細資訊，請參閱允許其他帳戶中的使用者使用 KMS 金鑰。

將 AWS Lambda 函數設定為從另一個帳戶中的 Kinesis Data Streams 讀取

有關如何設定 Lambda 函數來讀取其他帳戶的 Kinesis Data Streams 之範例，請參閱 與跨帳戶 AWS Lambda 函數共用存取權。

使用資源型政策共用存取權

> **Note**
>
> 更新現有資源型政策意味著取代現有的政策，因此請確保在新政策中包含所有必要資訊。

與跨帳戶 AWS Lambda 函數共用存取權

Lambda 運算子

1. 前往 IAM 主控台以建立 IAM 角色，該角色將用作 AWS Lambda 函數的 Lambda 執行角色。新增`AWSLambdaKinesisExecutionRole`具有必要 Kinesis Data Streams 和 Lambda 調用許可的受管 IAM 政策。此政策還會授予對您可能存取 Kinesis Data Streams 資源的存取權。
2. 在 AWS Lambda 主控台中，建立 AWS Lambda 函數來處理 Kinesis Data Streams 資料串流中的記錄，並在設定執行角色期間，選擇您在上一個步驟中建立的角色。
3. 將執行角色提供給 Kinesis Data Streams 資源擁有者，以便設定資源政策。
4. 完成 Lambda 函數的設定。

Kinesis Data Streams 資源擁有者

1. 取得將調用 Lambda 函數的跨帳戶 Lambda 執行角色。
2. 在 Amazon Kinesis Data Streams 主控台上，選擇資料串流。選擇資料串流共用索引標籤，然後選擇建立共用政策按鈕以啟動視覺化政策編輯器。若要在資料串流中共用已註冊的取用者，請選擇該取用者，然後選擇建立共用政策。您也可以直接撰寫 JSON 政策。
3. 指定跨帳戶 Lambda 執行角色作為主體，以及您要共用存取權的確切 Kinesis Data Streams 動作。務必包含動作 `kinesis:DescribeStream`。如需 Kinesis Data Streams 的範例資源政策的詳細資訊，請參閱 Kinesis 資料串流的資源型政策範例。
4. 選擇建立政策或使用 PutResourcePolicy 以將政策附加至您的資源。

與跨帳戶 KCL 取用者共用存取權

- 如果您使用的是 KCL 1.x，請確保您使用的是 KCL 1.15.0 或更高版本。
- 如果您使用的是 KCL 2.x，請確保您使用的是 KCL 2.5.3 或更高版本。

KCL 運算子

1. 提供將執行 KCL 應用程式的 IAM 使用者或 IAM 角色給資源擁有者。
2. 向資源擁有者要求資料串流或取用者 ARN。
3. 請務必將提供的串流 ARN 指定為 KCL 組態的一部分。
    - 若為 KCL 1.x：使用 KinesisClientLibConfiguration 建構函數並提供串流 ARN。

- 若為 KCL 2.x：您可以僅提供串流 ARN 或 StreamTracker 給 Kinesis 用戶端程式庫 ConfigsBuilder。若為 StreamTracker，請從程式庫產生的 DynamoDB 租用資料表提供串流 ARN 並建立 Epoch。如果您想要讀取共用的已註冊取用者 (例如增強型散發)，請使用 StreamTracker 並提供取用者 ARN。

Kinesis Data Streams 資源擁有者

1. 取得將執行 KCL 應用程式的跨帳戶 IAM 使用者或 IAM 角色。
2. 在 Amazon Kinesis Data Streams 主控台上，選擇資料串流。選擇資料串流共用索引標籤，然後選擇建立共用政策按鈕以啟動視覺化政策編輯器。若要在資料串流中共用已註冊的取用者，請選擇該取用者，然後選擇建立共用政策。您也可以直接撰寫 JSON 政策。
3. 指定跨帳戶 KCL 應用程式的 IAM 使用者或 IAM 角色作為主體，以及您要共用存取權的確切 Kinesis Data Streams 動作。如需 Kinesis Data Streams 的範例資源政策的詳細資訊，請參閱 Kinesis 資料串流的資源型政策範例。
4. 選擇建立政策或使用 PutResourcePolicy 以將政策附加至您的資源。

共用對加密資料的存取權

如果您已為具有 AWS 受管 KMS 金鑰的資料串流啟用伺服器端加密，並想要透過資源政策共用存取權，則必須切換到使用客戶受管金鑰 (CMK)。如需詳細資訊，請參閱什麼是 Kinesis Data Streams 的伺服器端加密？。此外，您必須允許共用主體實體以使用 KMS 跨帳戶共用功能來存取您的 CMK。請務必同時在共用主體實體的 IAM 政策中進行變更。如需詳細資訊，請參閱允許其他帳戶中的使用者使用 KMS 金鑰。

Amazon Kinesis Data Streams 的合規驗證

若要了解 AWS 服務 是否在特定合規計劃範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載報告 in AWS Artifact

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 AWS 安全文件。

Amazon Kinesis Data Streams 中的恢復能力

AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置的。 AWS 區域提供多個實體隔離和隔離的可用區域，這些區域以低延遲、高輸送量和高度備援的網路連接。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠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 AWS 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

除了 AWS 全球基礎設施之外，Kinesis Data Streams 還提供數種功能，以協助支援您的資料彈性和備份需求。

Amazon Kinesis Data Streams 中的災難復原

當您使用 Amazon Kinesis Data Streams 應用程式處理來自串流的資料時，以下層面可能發生失敗：

- 記錄處理器失敗
- 工作者失敗，或是執行個體化工作者的應用程式執行個體失敗
- 託管應用程式一個或多個執行個體的 EC2 執行個體失敗

記錄處理器故障

工作者會使用 Java ExecutorService 任務叫用記錄處理器方法。若有任務發生失敗，工作者將對記錄處理器原先處理的碎片保有控制權。工作者會啟動新的記錄處理器任務以處理該碎片。如需詳細資訊，請參閱讀取限流。

工作者或應用程式故障

如果工作者 (或 Amazon Kinesis Data Streams 的執行個體) 發生失敗，您即應偵測並處理該情況。例如，若 `Worker.run` 方法擲回例外狀況，您應將其截獲並加以處理。

如果應用程式本身發生失敗，您應對其進行偵測並予重新啟動。應用程式啟動時會執行個體化新的工作者，再由後者執行個體化新的記錄處理器，系統將自動指派碎片供其處理。這些碎片可能是該等記錄處理器在發生失敗前處理過的同一批碎片，或是另行指派給該等記錄處理器的新碎片。

在工作者或應用程式失敗的情況下，如果未偵測到失敗，且在其他 EC 2 執行個體上執行的應用程式有其他執行個體，這些其他執行個體則會處理失敗。它們會建立其他記錄處理器，來處理失敗的工作者不再處理的碎片。上述其他 EC2 執行個體的負載也會相應地增加。

此處描述的情節假定即使工作者或應用程式已失敗，託管 EC2 執行個體仍將執行中，因而並不會由 Auto Scaling 群組重新啟動。

Amazon EC2 執行個體失敗

建議您在 Auto Scaling 群組中執行 EC2 執行個體以用於您的應用程式。如此一來，若其中一個 EC2 執行個體發生失敗，Auto Scaling 群組會自動啟動新的執行個體予以取代。您應該將執行個體設定為在啟動時同時啟動 Amazon Kinesis Data Streams 應用程式。

Amazon Kinesis Data Streams 中的基礎設施安全性

作為受管服務， 受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 發佈的 API 呼叫，透過網路存取 。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

Kinesis Data Streams 的安全最佳實務

在您開發和實作自己的安全政策時，可考慮使用 Amazon Kinesis Data Streams 提供的多種安全功能。以下最佳實務為一般準則，並不代表完整的安全解決方案。這些最佳實務可能不適用或無法滿足您的環境需求，因此請將其視為實用建議就好，而不要當作是指示。

實作最低權限存取

授予許可時，需要決定哪些使用者會取得哪些 Kinesis Data Streams 資源的許可。您還需針對這些資源啟用允許執行的動作，因此，您只應授與執行任務所需的許可。對降低錯誤或惡意意圖所引起的安全風險和影響而言，實作最低權限存取是相當重要的一環。

使用 IAM 角色

生產者和用戶端應用程式必須擁有有效的憑證，才能存取 Kinesis 資料串流。您不應將 AWS 登入資料直接存放在用戶端應用程式或 Amazon S3 儲存貯體中。這些是不會自動輪換的長期憑證，如果遭到盜用，可能會對業務造成嚴重的影響。

反之，您應該使用 IAM 角色來管理生產者和用戶端應用程式存取 Kinesis 資料串流的暫時憑證。使用角色時，您不必使用長期登入資料 (例如使用者名稱和密碼或存取金鑰) 來存取其他資源。

如需詳細資訊，請參閱《IAM 使用者指南》中的以下主題：

- IAM 角色
- 常見的角色方案：使用者、應用程式和服務

在相依資源中實作伺服器端加密

您可以在 Kinesis Data Streams 加密靜態資料和傳輸中的資料。如需詳細資訊，請參閱Amazon Kinesis Data Streams 中的資料保護。

使用 CloudTrail 監控 API 呼叫

Kinesis Data Streams 已與 服務整合 AWS CloudTrail，此服務提供由使用者、角色或 Kinesis Data Streams 中的 AWS 服務所採取之動作的記錄。

您可以利用 CloudTrail 所收集的資訊來判斷向 Kinesis Data Streams 發出的請求，以及發出請求的 IP 地址、人員、時間和其他詳細資訊。

如需詳細資訊，請參閱the section called “使用 記錄 Amazon Kinesis Data Streams API 呼叫 AWS CloudTrail”。

搭配 AWS SDK 使用此服務

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

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

可用性範例

找不到所需的內容嗎？ 請使用本頁面底部的提供意見回饋連結申請程式碼範例。

Kinesis AWS SDKs的程式碼範例

下列程式碼範例示範如何使用 Kinesis 搭配 AWS 軟體開發套件 (SDK)。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

程式碼範例

- Kinesis AWS SDKs的基本範例
  - 使用 AWS SDK 了解 Kinesis 的基本概念
  - Kinesis 使用 AWS SDKs的動作
    - AddTagsToStream 搭配 AWS SDK 或 CLI 使用
    - CreateStream 搭配 AWS SDK 或 CLI 使用
    - DeleteStream 搭配 AWS SDK 或 CLI 使用
    - DeregisterStreamConsumer 搭配 AWS SDK 或 CLI 使用
    - DescribeStream 搭配 AWS SDK 或 CLI 使用
    - GetRecords 搭配 AWS SDK 或 CLI 使用
    - 搭配使用 GetShardIterator 與 CLI
    - ListStreamConsumers 搭配 AWS SDK 使用
    - ListStreams 搭配 AWS SDK 或 CLI 使用
    - ListTagsForStream 搭配 AWS SDK 或 CLI 使用
    - PutRecord 搭配 AWS SDK 或 CLI 使用
    - PutRecords 搭配 AWS SDK 或 CLI 使用
    - RegisterStreamConsumer 搭配 AWS SDK 或 CLI 使用
- Kinesis 的無伺服器範例
  - 使用 Kinesis 觸發條件調用 Lambda 函數
  - 使用 Kinesis 觸發條件報告 Lambda 函數的批次項目失敗

Kinesis AWS SDKs的基本範例

下列程式碼範例示範如何搭配 AWS SDK 使用 Amazon Kinesis 的基本功能。

範例

- 使用 AWS SDK 了解 Kinesis 的基本概念
- Kinesis 使用 AWS SDKs的動作
  - AddTagsToStream 搭配 AWS SDK 或 CLI 使用
  - CreateStream 搭配 AWS SDK 或 CLI 使用
  - DeleteStream 搭配 AWS SDK 或 CLI 使用
  - DeregisterStreamConsumer 搭配 AWS SDK 或 CLI 使用
  - DescribeStream 搭配 AWS SDK 或 CLI 使用
  - GetRecords 搭配 AWS SDK 或 CLI 使用
  - 搭配使用 GetShardIterator 與 CLI
  - ListStreamConsumers 搭配 AWS SDK 使用
  - ListStreams 搭配 AWS SDK 或 CLI 使用
  - ListTagsForStream 搭配 AWS SDK 或 CLI 使用
  - PutRecord 搭配 AWS SDK 或 CLI 使用
  - PutRecords 搭配 AWS SDK 或 CLI 使用
  - RegisterStreamConsumer 搭配 AWS SDK 或 CLI 使用

使用 AWS SDK 了解 Kinesis 的基本概念

以下程式碼範例顯示做法：

- 建立串流，並將記錄放入其中。
- 建立碎片迭代器。
- 讀取記錄，然後清除資源。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   DATA lo_stream_describe_result TYPE REF TO /aws1/cl_knsdescrstreamoutput.
   DATA lo_stream_description TYPE REF TO /aws1/cl_knsstreamdescription.
   DATA lo_sharditerator TYPE REF TO /aws1/cl_knsgetsharditerator01.
   DATA lo_record_result TYPE REF TO /aws1/cl_knsputrecordoutput.

   "Create stream."
   TRY.
       lo_kns->createstream(
           iv_streamname = iv_stream_name
           iv_shardcount = iv_shard_count ).
       MESSAGE 'Stream created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.

   "Wait for stream to becomes active."
   lo_stream_describe_result = lo_kns->describestream( iv_streamname =
iv_stream_name ).
   lo_stream_description = lo_stream_describe_result->get_streamdescription( ).
   WHILE lo_stream_description->get_streamstatus( ) <> 'ACTIVE'.
     IF sy-index = 30.
       EXIT.               "maximum 5 minutes"
     ENDIF.
     WAIT UP TO 10 SECONDS.
```

```
      lo_stream_describe_result = lo_kns->describestream( iv_streamname =
 iv_stream_name ).
      lo_stream_description = lo_stream_describe_result-
>get_streamdescription( ).
    ENDWHILE.

    "Create record."
    TRY.
        lo_record_result = lo_kns->putrecord(
            iv_streamname = iv_stream_name
            iv_data       = iv_data
            iv_partitionkey = iv_partition_key ).
        MESSAGE 'Record created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
 'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
 quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
 data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Create a shard iterator in order to read the record."
    TRY.
        lo_sharditerator = lo_kns->getsharditerator(
          iv_shardid = lo_record_result->get_shardid( )
          iv_sharditeratortype = iv_sharditeratortype
          iv_streamname = iv_stream_name ).
        MESSAGE 'Shard iterator created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
```

```
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_sgmresourcenotfound.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Read the record."
    TRY.
        oo_result = lo_kns->getrecords(                    " oo_result is
returned for testing purposes. "
            iv_sharditerator   = lo_sharditerator->get_sharditerator( ) ).
        MESSAGE 'Shard iterator created.' TYPE 'I'.
      CATCH /aws1/cx_knsexpirediteratorex.
        MESSAGE 'Iterator expired.' TYPE 'E'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Delete stream."
    TRY.
        lo_kns->deletestream(
            iv_streamname = iv_stream_name ).
        MESSAGE 'Stream deleted.' TYPE 'I'.
```

```
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourceinuseex.
        MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的下列主題。
  - CreateStream
  - DeleteStream
  - GetRecords
  - GetShardIterator
  - PutRecord

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Kinesis 使用 AWS SDKs的動作

下列程式碼範例示範如何使用 AWS SDKs 執行個別 Kinesis 動作。每個範例均包含 GitHub 的連結，您可以在連結中找到設定和執行程式碼的相關說明。

下列範例僅包含最常使用的動作。如需完整清單，請參閱《Amazon Kinesis API 參考》。

範例

- AddTagsToStream 搭配 AWS SDK 或 CLI 使用
- CreateStream 搭配 AWS SDK 或 CLI 使用
- DeleteStream 搭配 AWS SDK 或 CLI 使用
- DeregisterStreamConsumer 搭配 AWS SDK 或 CLI 使用
- DescribeStream 搭配 AWS SDK 或 CLI 使用
- GetRecords 搭配 AWS SDK 或 CLI 使用
- 搭配使用 GetShardIterator 與 CLI
- ListStreamConsumers 搭配 AWS SDK 使用
- ListStreams 搭配 AWS SDK 或 CLI 使用

- ListTagsForStream 搭配 AWS SDK 或 CLI 使用
- PutRecord 搭配 AWS SDK 或 CLI 使用
- PutRecords 搭配 AWS SDK 或 CLI 使用
- RegisterStreamConsumer 搭配 AWS SDK 或 CLI 使用

`AddTagsToStream` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `AddTagsToStream`。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        var tags = new Dictionary<string, string>
        {
            { "Project", "Sample Kinesis Project" },
            { "Application", "Sample Kinesis App" },
```

```
            };

            var success = await ApplyTagsToStreamAsync(client, streamName, tags);

            if (success)
            {
                Console.WriteLine($"Taggs successfully added to {streamName}.");
            }
            else
            {
                Console.WriteLine("Tags were not added to the stream.");
            }
        }

        /// <summary>
        /// Applies the set of tags to the named Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client.</param>
        /// <param name="streamName">The name of the Kinesis stream to which
        /// the tags will be attached.</param>
        /// <param name="tags">A sictionary containing key/value pairs which
        /// will be used to create the Kinesis tags.</param>
        /// <returns>A Boolean value which represents the success or failure
        /// of AddTagsToStreamAsync.</returns>
        public static async Task<bool> ApplyTagsToStreamAsync(
            IAmazonKinesis client,
            string streamName,
            Dictionary<string, string> tags)
        {
            var request = new AddTagsToStreamRequest
            {
                StreamName = streamName,
                Tags = tags,
            };

            var response = await client.AddTagsToStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 AddTagsToStream。

CLI

AWS CLI

將標籤新增至資料串流

下列 add-tags-to-stream 範例會將具有索引鍵 samplekey 和值 example 的標籤，指派給指定的串流。

```
aws kinesis add-tags-to-stream \
    --stream-name samplestream \
    --tags samplekey=example
```

此命令不會產生輸出。

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的標記串流。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 AddTagsToStream。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

CreateStream 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 CreateStream。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// This example shows how to create a new Amazon Kinesis stream.
    /// </summary>
    public class CreateStream
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamName = "AmazonKinesisStream";
            int shardCount = 1;

            var success = await CreateNewStreamAsync(client, streamName,
shardCount);
            if (success)
            {
                Console.WriteLine($"The stream, {streamName} successfully
created.");
            }
        }

        /// <summary>
        /// Creates a new Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client.</param>
        /// <param name="streamName">The name for the new stream.</param>
        /// <param name="shardCount">The number of shards the new stream will
```

```
        /// use. The throughput of the stream is a function of the number of
        /// shards; more shards are required for greater provisioned
        /// throughput.</param>
        /// <returns>A Boolean value indicating whether the stream was created.</
returns>
        public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis
 client, string streamName, int shardCount)
        {
            var request = new CreateStreamRequest
            {
                StreamName = streamName,
                ShardCount = shardCount,
            };

            var response = await client.CreateStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateStream。

CLI

AWS CLI

建立資料串流

下列 `create-stream` 範例會建立名為 samplestream 的資料串流，其中包含 3 個碎片。

```
aws kinesis create-stream \
    --stream-name samplestream \
    --shard-count 3
```

此命令不會產生輸出。

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的建立串流。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 CreateStream。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.CreateStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateDataStream {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();
        createStream(kinesisClient, streamName);
        System.out.println("Done");
        kinesisClient.close();
    }

    public static void createStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            CreateStreamRequest streamReq = CreateStreamRequest.builder()
                    .streamName(streamName)
                    .shardCount(1)
                    .build();

            kinesisClient.createStream(streamReq);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 CreateStream。

PowerShell

Tools for PowerShell V4

範例 1：建立新的串流。根據預設，此 Cmdlet 不會傳回任何輸出，因此會新增 -PassThru 切換變數，以傳回提供給 -StreamName 參數的值，供後續使用。

```
$streamName = New-KINStream -StreamName "mystream" -ShardCount 1 -PassThru
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 CreateStream。

Tools for PowerShell V5

範例 1：建立新的串流。

```
New-KINStream -StreamName "mystream" -ShardCount 1
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 CreateStream。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def create(self, name, wait_until_exists=True):
        """
        Creates a stream.

        :param name: The name of the stream.
        :param wait_until_exists: When True, waits until the service reports that
                                  the stream exists, then queries for its
 metadata.
        """
```

```
        try:
            self.kinesis_client.create_stream(StreamName=name, ShardCount=1)
            self.name = name
            logger.info("Created stream %s.", name)
            if wait_until_exists:
                logger.info("Waiting until exists.")
                self.stream_exists_waiter.wait(StreamName=name)
                self.describe(name)
        except ClientError:
            logger.exception("Couldn't create stream %s.", name)
            raise
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 CreateStream。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
async fn make_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client
        .create_stream()
        .stream_name(stream)
        .shard_count(4)
        .send()
        .await?;

    println!("Created stream");

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 CreateStream。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   TRY.
       lo_kns->createstream(
           iv_streamname = iv_stream_name
           iv_shardcount = iv_shard_count ).
       MESSAGE 'Stream created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 CreateStream。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeleteStream` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DeleteStream`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to delete an Amazon Kinesis stream.
/// </summary>
public class DeleteStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        var success = await DeleteStreamAsync(client, streamName);

        if (success)
        {
            Console.WriteLine($"Stream, {streamName} successfully deleted.");
        }
        else
        {
            Console.WriteLine("Stream not deleted.");
        }
    }

    /// <summary>
    /// Deletes a Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client object.</param>
    /// <param name="streamName">The name of the string to delete.</param>
```

```
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
 string streamName)
        {
            // If EnforceConsumerDeletion is true, any consumers
            // of this stream will also be deleted. If it is set
            // to false and this stream has any consumers, the
            // call will fail with a ResourceInUseException.
            var request = new DeleteStreamRequest
            {
                StreamName = streamName,
                EnforceConsumerDeletion = true,
            };

            var response = await client.DeleteStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteStream。

CLI

AWS CLI

刪除資料串流

以下 `delete-stream` 範例會刪除指定的資料串流。

```
aws kinesis delete-stream \
    --stream-name samplestream
```

此命令不會產生輸出。

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的刪除串流。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeleteStream。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DeleteStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteDataStream {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream)
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        deleteStream(kinesisClient, streamName);
        kinesisClient.close();
        System.out.println("Done");
    }

    public static void deleteStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            DeleteStreamRequest delStream = DeleteStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            kinesisClient.deleteStream(delStream);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 DeleteStream。

PowerShell

Tools for PowerShell V4

範例 1：刪除指定的串流。在執行命令之前，系統會提示您確認。若要隱藏確認提示，請使用 -Force 切換變數。

```
Remove-KINStream -StreamName "mystream"
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 DeleteStream。

Tools for PowerShell V5

範例 1：刪除指定的串流。在執行命令之前，系統會提示您確認。若要隱藏確認提示，請使用 -Force 切換變數。

```
Remove-KINStream -StreamName "mystream"
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 DeleteStream。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def delete(self):
        """
        Deletes a stream.
        """
        try:
            self.kinesis_client.delete_stream(StreamName=self.name)
            self._clear()
            logger.info("Deleted stream %s.", self.name)
```

```
        except ClientError:
            logger.exception("Couldn't delete stream %s.", self.name)
            raise
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 DeleteStream。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 DeleteStream。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   TRY.
       lo_kns->deletestream(
           iv_streamname = iv_stream_name ).
       MESSAGE 'Stream deleted.' TYPE 'I'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 DeleteStream。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DeregisterStreamConsumer` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DeregisterStreamConsumer`。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to deregister a consumer from an Amazon Kinesis stream.
/// </summary>
```

```
    public class DeregisterConsumer
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream";
            string consumerName = "CONSUMER_NAME";
            string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";

            var success = await DeregisterConsumerAsync(client, streamARN,
 consumerARN, consumerName);

            if (success)
            {
                Console.WriteLine($"{consumerName} successfully deregistered.");
            }
            else
            {
                Console.WriteLine($"{consumerName} was not successfully
 deregistered.");
            }
        }

        /// <summary>
        /// Deregisters a consumer from a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of a Kinesis stream.</param>
        /// <param name="consumerARN">The ARN of the consumer.</param>
        /// <param name="consumerName">The name of the consumer.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeregisterConsumerAsync(
            IAmazonKinesis client,
            string streamARN,
            string consumerARN,
            string consumerName)
        {
            var request = new DeregisterStreamConsumerRequest
            {
                StreamARN = streamARN,
```

```
                ConsumerARN = consumerARN,
                ConsumerName = consumerName,
            };

            var response = await client.DeregisterStreamConsumerAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeregisterStreamConsumer。

CLI

AWS CLI

取消註冊資料串流取用者

下列 `deregister-stream-consumer` 範例會從指定的資料串流取消註冊指定的取用者。

```
aws kinesis deregister-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:123456789012:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

此命令不會產生輸出。

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的使用 Kinesis Data Streams API 開發具有強化廣發功能的取用者。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DeregisterStreamConsumer。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`DescribeStream` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `DescribeStream`。

CLI

AWS CLI

描述資料串流

下列 describe-stream 範例會傳回指定資料串流的詳細長度。

```
aws kinesis describe-stream \
    --stream-name samplestream
```

輸出：

```
{
    "StreamDescription": {
        "Shards": [
            {
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "113427455640312821154458202477256070484"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "49600871682957036442365024926191073437251060580128653314"
                }
            },
            {
                "ShardId": "shardId-000000000001",
                "HashKeyRange": {
                    "StartingHashKey": "113427455640312821154458202477256070485",
                    "EndingHashKey": "226854911280625642308916404954512140969"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "49600871682979337187563555549332609155523708941634633746"
                }
            },
            {
                "ShardId": "shardId-000000000002",
                "HashKeyRange": {
                    "StartingHashKey": "226854911280625642308916404954512140970",
                    "EndingHashKey": "340282366920938463463374607431768211455"
```

```
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716830016379327620861724741448737963573031406141 78"
                }
            }
        ],
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/
samplestream",
        "StreamName": "samplestream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1572297168.0
    }
}
```

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的建立管理串流。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 DescribeStream。

PowerShell

Tools for PowerShell V4

範例 1：傳回指定串流的詳細資訊。

```
Get-KINStream -StreamName "mystream"
```

輸出：

```
HasMoreShards        : False
RetentionPeriodHours : 24
Shards               : {}
StreamARN            : arn:aws:kinesis:us-west-2:123456789012:stream/mystream
StreamName           : mystream
```

```
StreamStatus         : ACTIVE
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 DescribeStream。

Tools for PowerShell V5

範例 1：傳回指定串流的詳細資訊。

```
Get-KINStream -StreamName "mystream"
```

輸出：

```
HasMoreShards        : False
RetentionPeriodHours : 24
Shards               : {}
StreamARN            : arn:aws:kinesis:us-west-2:123456789012:stream/mystream
StreamName           : mystream
StreamStatus         : ACTIVE
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 DescribeStream。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
```

```
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 DescribeStream。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
async fn show_stream(client: &Client, stream: &str) -> Result<(), Error> {
    let resp = client.describe_stream().stream_name(stream).send().await?;
```

```
    let desc = resp.stream_description.unwrap();

    println!("Stream description:");
    println!("  Name:              {}:", desc.stream_name());
    println!("  Status:            {:?}", desc.stream_status());
    println!("  Open shards:       {:?}", desc.shards.len());
    println!("  Retention (hours): {}", desc.retention_period_hours());
    println!("  Encryption:        {:?}", desc.encryption_type.unwrap());

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 DescribeStream。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    TRY.
        oo_result = lo_kns->describestream(
            iv_streamname = iv_stream_name ).
        DATA(lt_stream_description) = oo_result->get_streamdescription( ).
        MESSAGE 'Streams retrieved.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的 DescribeStream。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`GetRecords` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `GetRecords`。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

CLI

AWS CLI

從碎片取得記錄

下列 `get-records` 範例使用指定的碎片迭代器，從 Kinesis 資料串流的碎片取得資料記錄。

```
aws kinesis get-records \
    --shard-iterator AAAAAAAAAAF7/0mWD7IuHj1yGv/
TKuNgx2ukD5xipCY4cy4gU96orWwZwcSXh3K9tAmGYeOZyLZrvzzeOFVf9iN99hUPw/w/
b0YWYeehfNvnf1DYt5XpDJghLKr3DzgznkTmMymDP3R+3wRKeuEw6/kdxY2yKJH0veaiekaVc4N2VwK/
GvaGP2Hh9Fg7N++q0Adg6fIDQPt4p8RpavDbk+A4sL9SWGE1
```

輸出：

```
{
    "Records": [],
    "MillisBehindLatest": 80742000
}
```

如需詳細資訊，請參閱《Amazon Kinesis Kinesis Data Streams 開發人員指南》中的使用 Kinesis Data Streams API 搭配適用於 Java 的 AWS SDK 開發消費者。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 GetRecords。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.Shard;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.kinesis.model.Record;
import software.amazon.awssdk.services.kinesis.model.GetRecordsRequest;
import software.amazon.awssdk.services.kinesis.model.GetRecordsResponse;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetRecords {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
```

```
                    streamName - The Amazon Kinesis data stream to read from (for
example, StockTradeStream).
                """";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        getStockTrades(kinesisClient, streamName);
        kinesisClient.close();
    }

    public static void getStockTrades(KinesisClient kinesisClient, String
streamName) {
        String shardIterator;
        String lastShardId = null;
        DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                .streamName(streamName)
                .build();

        List<Shard> shards = new ArrayList<>();
        DescribeStreamResponse streamRes;
        do {
            streamRes = kinesisClient.describeStream(describeStreamRequest);
            shards.addAll(streamRes.streamDescription().shards());

            if (shards.size() > 0) {
                lastShardId = shards.get(shards.size() - 1).shardId();
            }
        } while (streamRes.streamDescription().hasMoreShards());

        GetShardIteratorRequest itReq = GetShardIteratorRequest.builder()
                .streamName(streamName)
                .shardIteratorType("TRIM_HORIZON")
                .shardId(lastShardId)
                .build();
```

```
        GetShardIteratorResponse shardIteratorResult =
 kinesisClient.getShardIterator(itReq);
        shardIterator = shardIteratorResult.shardIterator();

        // Continuously read data records from shard.
        List<Record> records;

        // Create new GetRecordsRequest with existing shardIterator.
        // Set maximum records to return to 1000.
        GetRecordsRequest recordsRequest = GetRecordsRequest.builder()
                .shardIterator(shardIterator)
                .limit(1000)
                .build();

        GetRecordsResponse result = kinesisClient.getRecords(recordsRequest);

        // Put result into record list. Result may be empty.
        records = result.records();

        // Print records
        for (Record record : records) {
            SdkBytes byteBuffer = record.data();
            System.out.printf("Seq No: %s - %s%n", record.sequenceNumber(), new
 String(byteBuffer.asByteArray()));
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 GetRecords。

PowerShell

Tools for PowerShell V4

範例 1：此範例示範如何從一系列的一或多個記錄，傳回和擷取資料。提供給 Get-KINRecord 的迭代器，能判斷記錄的開始位置，以傳回在此範例中擷取到變數 $records 的位置。然後，您可以透過編製 $records 集合的索引來存取每個個別記錄。假設記錄中的資料為 UTF-8 編碼文字，最終命令會示範如何從物件中的 MemoryStream 擷取資料，並將其當作文字傳回至主控台。

```
$records
$records = Get-KINRecord -ShardIterator "AAAAAAAAAAGIc....9VnbiRNaP"
```

輸出：

```
MillisBehindLatest NextShardIterator            Records
------------------ -----------------            -------
0                  AAAAAAAAAAERNIq...uDn11HuUs  {Key1, Key2}
```

```
$records.Records[0]
```

輸出：

```
ApproximateArrivalTimestamp Data                    PartitionKey SequenceNumber
--------------------------- ----                    ------------ --------------
3/7/2016 5:14:33 PM         System.IO.MemoryStream Key1
 4955986459776...931586
```

```
[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())
```

輸出：

```
test data from string
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 GetRecords。

Tools for PowerShell V5

範例 1：此範例示範如何從一系列的一或多個記錄，傳回和擷取資料。提供給 Get-KINRecord 的迭代器，能判斷記錄的開始位置，以傳回在此範例中擷取到變數 $records 的位置。然後，您可以透過編製 $records 集合的索引來存取每個個別記錄。假設記錄中的資料為 UTF-8 編碼文字，最終命令會示範如何從物件中的 MemoryStream 擷取資料，並將其當作文字傳回至主控台。

```
$records
$records = Get-KINRecord -ShardIterator "AAAAAAAAAAGIc....9VnbiRNaP"
```

輸出：

```
MillisBehindLatest NextShardIterator            Records
------------------ -----------------            -------
0                  AAAAAAAAAAERNIq...uDn11HuUs  {Key1, Key2}
```

```
$records.Records[0]
```

輸出：

```
ApproximateArrivalTimestamp Data                    PartitionKey SequenceNumber
--------------------------- ----                    ------------ --------------
3/7/2016 5:14:33 PM         System.IO.MemoryStream Key1
 4955986459776...931586
```

```
[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())
```

輸出：

```
test data from string
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 GetRecords。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
```

```
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def get_records(self, max_records):
        """
        Gets records from the stream. This function is a generator that first
gets
        a shard iterator for the stream, then uses the shard iterator to get
records
        in batches from the stream. The shard iterator can be accessed through
the
        'details' property, which is populated using the 'describe' function of
this class.
        Each batch of records is yielded back to the caller until the specified
        maximum number of records has been retrieved.

        :param max_records: The maximum number of records to retrieve.
        :return: Yields the current batch of retrieved records.
        """
        try:
            response = self.kinesis_client.get_shard_iterator(
                StreamName=self.name,
                ShardId=self.details["Shards"][0]["ShardId"],
                ShardIteratorType="LATEST",
            )
            shard_iter = response["ShardIterator"]
            record_count = 0
            while record_count < max_records:
                response = self.kinesis_client.get_records(
                    ShardIterator=shard_iter, Limit=10
                )
                shard_iter = response["NextShardIterator"]
                records = response["Records"]
                logger.info("Got %s records.", len(records))
                record_count += len(records)
                yield records
        except ClientError:
            logger.exception("Couldn't get records from stream %s.", self.name)
            raise
```

```
    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 GetRecords。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    TRY.
        oo_result = lo_kns->getrecords(             " oo_result is returned for
testing purposes. "
            iv_sharditerator = iv_shard_iterator ).
        DATA(lt_records) = oo_result->get_records( ).
        MESSAGE 'Record retrieved.' TYPE 'I'.
      CATCH /aws1/cx_knsexpirediteratorex.
        MESSAGE 'Iterator expired.' TYPE 'E'.
      CATCH /aws1/cx_knsinvalidargumentex.
```

```
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 GetRecords。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

搭配使用 **GetShardIterator** 與 CLI

下列程式碼範例示範如何使用 GetShardIterator。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

CLI

AWS CLI

取得碎片迭代器

下列 get-shard-iterator 範例使用 AT_SEQUENCE_NUMBER 碎片迭代器型，並產生碎片迭代器，開始從以指定序號表示的位置讀取資料記錄。

```
aws kinesis get-shard-iterator \
    --stream-name samplestream \
    --shard-id shardId-000000000001 \
    --shard-iterator-type LATEST
```

輸出：

```
{
    "ShardIterator": "AAAAAAAAAAFEvJjIYI+3jw/4aqgH9FifJ+n48XWTh/
IFIsbILP6o5eDueD39NXNBfpZ10WL5K6ADXk8w+5H+Qhd9cFA9k268CPXCz/kebq1TGYI7Vy
+lUkA9BuN3xvATxMBGxRY3zYK05gqgvaIRn94O8SqeEqwhigwZxNWxID3Ej7YYYcxQi8Q/fIrCjGAy/
n2r5Z9G864YpWDfN9upNNQAR/iiOWKs"
}
```

如需詳細資訊，請參閱《Amazon Kinesis Kinesis Data Streams 開發人員指南》中的使用 Kinesis Data Streams API 搭配適用於 Java 的 AWS SDK 開發消費者。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 GetShardIterator。

PowerShell

Tools for PowerShell V4

範例 1：傳回指定碎片和開始位置的碎片迭代器。碎片識別碼和序號的詳細資訊，可從 Get-KINStream Cmdlet 的輸出取得，方法是參考傳回串流物件的碎片收集。傳回的迭代器可與 Get-KINRecord Cmdlet 搭配使用，以提取碎片中的資料記錄。

```
Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -
ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
```

輸出：

```
AAAAAAAAAAGIc....9VnbiRNaP
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 GetShardIterator。

Tools for PowerShell V5

範例 1：傳回指定碎片和開始位置的碎片迭代器。碎片識別碼和序號的詳細資訊，可從 Get-KINStream Cmdlet 的輸出取得，方法是參考傳回串流物件的碎片收集。傳回的迭代器可與 Get-KINRecord Cmdlet 搭配使用，以提取碎片中的資料記錄。

```
Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -
ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
```

輸出：

```
AAAAAAAAAAGIc....9VnbiRNaP
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 GetShardIterator。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ListStreamConsumers` 搭配 AWS SDK 使用

以下程式碼範例顯示如何使用 `ListStreamConsumers`。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
using System;
using System.Collections.Generic;
```

```
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// List the consumers of an Amazon Kinesis stream.
    /// </summary>
    public class ListConsumers
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
            int maxResults = 10;

            var consumers = await ListConsumersAsync(client, streamARN,
 maxResults);

            if (consumers.Count > 0)
            {
                consumers
                    .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
 {c.ConsumerARN}"));
            }
            else
            {
                Console.WriteLine("No consumers found.");
            }
        }

        /// <summary>
        /// Retrieve a list of the consumers for a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of the stream for which we want to
        /// retrieve a list of clients.</param>
        /// <param name="maxResults">The maximum number of results to return.</
param>
        /// <returns>A list of Consumer objects.</returns>
        public static async Task<List<Consumer>>
 ListConsumersAsync(IAmazonKinesis client, string streamARN, int maxResults)
        {
```

```
            var request = new ListStreamConsumersRequest
            {
                StreamARN = streamARN,
                MaxResults = maxResults,
            };

            var response = await client.ListStreamConsumersAsync(request);

            return response.Consumers;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ListStreamConsumers。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ListStreams` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `ListStreams`。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
```

```
    /// Retrieves and displays a list of existing Amazon Kinesis streams.
    /// </summary>
    public class ListStreams
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            var response = await client.ListStreamsAsync(new
 ListStreamsRequest());

            List<string> streamNames = response.StreamNames;

            if (streamNames.Count > 0)
            {
                streamNames
                    .ForEach(s => Console.WriteLine($"Stream name: {s}"));
            }
            else
            {
                Console.WriteLine("No streams were found.");
            }
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ListStreams。

CLI

AWS CLI

列出資料串流

下列 `list-streams` 範例列出目前帳戶和區域中的所有作用中資料串流。

```
aws kinesis list-streams
```

輸出：

```
{
    "StreamNames": [
```

```
        "samplestream",
        "samplestream1"
    ]
}
```

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的列出串流。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 ListStreams。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
async fn show_streams(client: &Client) -> Result<(), Error> {
    let resp = client.list_streams().send().await?;

    println!("Stream names:");

    let streams = resp.stream_names;
    for stream in &streams {
        println!("  {}", stream);
    }

    println!("Found {} stream(s)", streams.len());

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 ListStreams。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   TRY.
       oo_result = lo_kns->liststreams(        " oo_result is returned for
testing purposes. "
           "Set Limit to specify that a maximum of streams should be returned."
           iv_limit = iv_limit ).
       DATA(lt_streams) = oo_result->get_streamnames( ).
       MESSAGE 'Streams listed.' TYPE 'I'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 ListStreams。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`ListTagsForStream` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `ListTagsForStream`。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   using System;
   using System.Collections.Generic;
   using System.Threading.Tasks;
   using Amazon.Kinesis;
   using Amazon.Kinesis.Model;

   /// <summary>
   /// Shows how to list the tags that have been attached to an Amazon Kinesis
   /// stream.
   /// </summary>
   public class ListTags
   {
       public static async Task Main()
       {
           IAmazonKinesis client = new AmazonKinesisClient();
           string streamName = "AmazonKinesisStream";

           await ListTagsAsync(client, streamName);
       }

       /// <summary>
       /// List the tags attached to a Kinesis stream.
       /// </summary>
       /// <param name="client">An initialized Kinesis client object.</param>
       /// <param name="streamName">The name of the Kinesis stream for which you
       /// wish to display tags.</param>
       public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
       {
           var request = new ListTagsForStreamRequest
           {
               StreamName = streamName,
               Limit = 10,
           };

           var response = await client.ListTagsForStreamAsync(request);
           DisplayTags(response.Tags);

           while (response.HasMoreTags)
           {
               request.ExclusiveStartTagKey = response.Tags[response.Tags.Count
- 1].Key;
               response = await client.ListTagsForStreamAsync(request);
```

```
            }
        }

        /// <summary>
        /// Displays the items in a list of Kinesis tags.
        /// </summary>
        /// <param name="tags">A list of the Tag objects to be displayed.</param>
        public static void DisplayTags(List<Tag> tags)
        {
            tags
                .ForEach(t => Console.WriteLine($"Key: {t.Key} Value:
 {t.Value}"));
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ListTagsForStream。

CLI

AWS CLI

列出資料串流的標籤

下列 `list-tags-for-stream` 範例列出連接至指定資料串流的標籤。

```
aws kinesis list-tags-for-stream \
    --stream-name samplestream
```

輸出：

```
{
    "Tags": [
        {
            "Key": "samplekey",
            "Value": "example"
        }
    ],
    "HasMoreTags": false
}
```

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的標記串流。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 ListTagsForStream。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

PutRecord 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 PutRecord。

動作範例是大型程式的程式碼摘錄，必須在內容中執行。您可以在下列程式碼範例的內容中看到此動作：

- 了解基本概念

CLI

AWS CLI

將記錄寫入資料串流

下列 put-record 範例會使用指定的分割區索引鍵，將單一資料記錄寫入指定的資料串流。

```
aws kinesis put-record \
    --stream-name samplestream \
    --data sampledatarecord \
    --partition-key samplepartitionkey
```

輸出：

```
{
    "ShardId": "shardId-000000000009",
    "SequenceNumber": "49600902273357540915989931256901506243878407835297513618",
    "EncryptionType": "KMS"
}
```

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的使用 Amazon Kinesis Data Streams API 搭配適用於 Java 的 AWS SDK 開發生產者。 Amazon Kinesis

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 PutRecord。

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class StockTradesWriter {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream to which records
 are written (for example, StockTradeStream)
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
```

```
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        // Ensure that the Kinesis Stream is valid.
        validateStream(kinesisClient, streamName);
        setStockData(kinesisClient, streamName);
        kinesisClient.close();
    }

    public static void setStockData(KinesisClient kinesisClient, String
streamName) {
        try {
            // Repeatedly send stock trades with a 100 milliseconds wait in
between.
            StockTradeGenerator stockTradeGenerator = new StockTradeGenerator();

            // Put in 50 Records for this example.
            int index = 50;
            for (int x = 0; x < index; x++) {
                StockTrade trade = stockTradeGenerator.getRandomTrade();
                sendStockTrade(trade, kinesisClient, streamName);
                Thread.sleep(100);
            }

        } catch (KinesisException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Done");
    }

    private static void sendStockTrade(StockTrade trade, KinesisClient
kinesisClient,
            String streamName) {
        byte[] bytes = trade.toJsonAsBytes();

        // The bytes could be null if there is an issue with the JSON
serialization by
        // the Jackson JSON library.
```

```
        if (bytes == null) {
            System.out.println("Could not get JSON bytes for stock trade");
            return;
        }

        System.out.println("Putting trade: " + trade);
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(trade.getTickerSymbol()) // We use the ticker
 symbol as the partition key, explained in
                                                       // the Supplemental
 Information section below.
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(bytes))
                .build();

        try {
            kinesisClient.putRecord(request);
        } catch (KinesisException e) {
            System.err.println(e.getMessage());
        }
    }

    private static void validateStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            DescribeStreamRequest describeStreamRequest =
 DescribeStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            DescribeStreamResponse describeStreamResponse =
 kinesisClient.describeStream(describeStreamRequest);

            if (!
describeStreamResponse.streamDescription().streamStatus().toString().equals("ACTIVE"))
 {
                System.err.println("Stream " + streamName + " is not active.
 Please wait a few moments and try again.");
                System.exit(1);
            }

        } catch (KinesisException e) {
            System.err.println("Error found while describing the stream " +
 streamName);
```

```
            System.err.println(e);
            System.exit(1);
        }
    }
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Java 2.x API 參考》中的 PutRecord。

PowerShell

Tools for PowerShell V4

範例 1：撰寫包含提供給 -Text 參數之字串的記錄。

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -
PartitionKey "Key1"
```

範例 2：寫入包含在指定檔案中的資料的記錄。該檔案會被視為位元組序列，因此如果其包含文字，則應在搭配此 Cmdlet 使用之前，使用任何必要的編碼進行編寫。

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey
 "Key2"
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V4)》中的 PutRecord。

Tools for PowerShell V5

範例 1：撰寫包含提供給 -Text 參數之字串的記錄。

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -
PartitionKey "Key1"
```

範例 2：寫入包含在指定檔案中的資料的記錄。該檔案會被視為位元組序列，因此如果其包含文字，則應在搭配此 Cmdlet 使用之前，使用任何必要的編碼進行編寫。

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey
 "Key2"
```

- 如需 API 詳細資訊，請參閱《AWS Tools for PowerShell Cmdlet 參考 (V5)》中的 PutRecord。

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def put_record(self, data, partition_key):
        """
        Puts data into the stream. The data is formatted as JSON before it is
 passed
        to the stream.

        :param data: The data to put in the stream.
        :param partition_key: The partition key to use for the data.
        :return: Metadata about the record, including its shard ID and sequence
 number.
        """
        try:
            response = self.kinesis_client.put_record(
                StreamName=self.name, Data=json.dumps(data),
 PartitionKey=partition_key
```

```
            )
            logger.info("Put record in stream %s.", self.name)
        except ClientError:
            logger.exception("Couldn't put record in stream %s.", self.name)
            raise
        else:
            return response
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Python (Boto3) API 參考》中的 PutRecord。

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
async fn add_record(client: &Client, stream: &str, key: &str, data: &str) ->
 Result<(), Error> {
    let blob = Blob::new(data);

    client
        .put_record()
        .data(blob)
        .partition_key(key)
        .stream_name(stream)
        .send()
        .await?;

    println!("Put data into stream.");

    Ok(())
}
```

- 如需 API 詳細資訊，請參閱《AWS SDK for Rust API 參考》中的 PutRecord。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   TRY.
       oo_result = lo_kns->putrecord(            " oo_result is returned for
testing purposes. "
           iv_streamname = iv_stream_name
           iv_data       = iv_data
           iv_partitionkey = iv_partition_key ).
       MESSAGE 'Record created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knskmsaccessdeniedex.
       MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
     CATCH /aws1/cx_knskmsdisabledex.
       MESSAGE 'KMS key used is disabled.' TYPE 'E'.
     CATCH /aws1/cx_knskmsinvalidstateex.
       MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
     CATCH /aws1/cx_knskmsnotfoundex.
       MESSAGE 'KMS key used is not found.' TYPE 'E'.
     CATCH /aws1/cx_knskmsoptinrequired.
       MESSAGE 'KMS key option is required.' TYPE 'E'.
     CATCH /aws1/cx_knskmsthrottlingex.
       MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
     CATCH /aws1/cx_knsprovthruputexcdex.
       MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
     CATCH /aws1/cx_knsresourcenotfoundex.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的AWS SDK API 參考》中的 PutRecord。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

PutRecords 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 PutRecords。

CLI

AWS CLI

將多筆記錄寫入資料串流

下列 put-records 範例使用指定的分割區索引鍵寫入資料記錄，並在單一呼叫中使用不同的分割區索引鍵寫入另一個資料記錄。

```
aws kinesis put-records \
    --stream-name samplestream \
    --
records Data=blob1,PartitionKey=partitionkey1 Data=blob2,PartitionKey=partitionkey2
```

輸出：

```
{
    "FailedRecordCount": 0,
    "Records": [
        {
            "SequenceNumber":
 "49600883331171471519674795588238531498465399900093808706",
            "ShardId": "shardId-000000000004"
        },
        {
            "SequenceNumber":
 "49600902273357540915989931256902715169698037101720764562",
            "ShardId": "shardId-000000000009"
        }
    ],
    "EncryptionType": "KMS"
}
```

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的使用 Amazon Kinesis Data Streams API 搭配適用於 Java 的 AWS SDK 開發生產者。 Amazon Kinesis

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 PutRecords。

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
import { PutRecordsCommand, KinesisClient } from "@aws-sdk/client-kinesis";

/**
 * Put multiple records into a Kinesis stream.
 * @param {{ streamArn: string }} config
 */
export const main = async ({ streamArn }) => {
  const client = new KinesisClient({});
  try {
    await client.send(
      new PutRecordsCommand({
        StreamARN: streamArn,
        Records: [
          {
            Data: new Uint8Array(),
            /**
             * Determines which shard in the stream the data record is assigned
 to.
             * Partition keys are Unicode strings with a maximum length limit of
 256
             * characters for each key. Amazon Kinesis Data Streams uses the
 partition
             * key as input to a hash function that maps the partition key and
             * associated data to a specific shard.
             */
            PartitionKey: "TEST_KEY",
          },
          {
            Data: new Uint8Array(),
```

```
            PartitionKey: "TEST_KEY",
          },
        ],
      }),
    );
  } catch (caught) {
    if (caught instanceof Error) {
      //
    } else {
      throw caught;
    }
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    streamArn: {
      type: "string",
      description: "The ARN of the stream.",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}
```

- 如需 API 詳細資訊，請參閱《適用於 JavaScript 的 AWS SDK API 參考》中的 PutRecords。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

`RegisterStreamConsumer` 搭配 AWS SDK 或 CLI 使用

下列程式碼範例示範如何使用 `RegisterStreamConsumer`。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// This example shows how to register a consumer to an Amazon Kinesis
    /// stream.
    /// </summary>
    public class RegisterConsumer
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string consumerName = "NEW_CONSUMER_NAME";
            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

            var consumer = await RegisterConsumerAsync(client, consumerName,
 streamARN);

            if (consumer is not null)
            {
                Console.WriteLine($"{consumer.ConsumerName}");
            }
        }

        /// <summary>
        /// Registers the consumer to a Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client object.</param>
```

```
        /// <param name="consumerName">A string representing the consumer.</
param>
        /// <param name="streamARN">The ARN of the stream.</param>
        /// <returns>A Consumer object that contains information about the
 consumer.</returns>
        public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
 client, string consumerName, string streamARN)
        {
            var request = new RegisterStreamConsumerRequest
            {
                ConsumerName = consumerName,
                StreamARN = streamARN,
            };

            var response = await client.RegisterStreamConsumerAsync(request);
            return response.Consumer;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 RegisterStreamConsumer。

CLI

AWS CLI

註冊資料串流取用者

下列 `register-stream-consumer` 範例會在指定的資料串流註冊名為 `KinesisConsumerApplication` 的取用者。

```
aws kinesis register-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:012345678912:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

輸出：

```
{
    "Consumer": {
```

```
        "ConsumerName": "KinesisConsumerApplication",
        "ConsumerARN": "arn:aws:kinesis:us-west-2: 123456789012:stream/
samplestream/consumer/KinesisConsumerApplication:1572383852",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1572383852.0
    }
}
```

如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的使用 Kinesis Data Streams API 開發具有強化廣發功能的取用者。

- 如需 API 詳細資訊，請參閱《AWS CLI 命令參考》中的 RegisterStreamConsumer。

SAP ABAP

適用於 SAP ABAP 的開發套件

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   TRY.
       oo_result = lo_kns->registerstreamconsumer(       " oo_result is returned
for testing purposes. "
           iv_streamarn = iv_stream_arn
           iv_consumername = iv_consumer_name ).
       MESSAGE 'Stream consumer registered.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_sgmresourcelimitexcd.
       MESSAGE 'You have reached the limit on the number of resources.' TYPE
'E'.
     CATCH /aws1/cx_sgmresourceinuse.
       MESSAGE 'Resource being accessed is in use.' TYPE 'E'.
     CATCH /aws1/cx_sgmresourcenotfound.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.
```

- 如需 API 詳細資訊，請參閱《適用於 SAP ABAP 的 AWS SDK API 參考》中的 RegisterStreamConsumer。

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

Kinesis 的無伺服器範例

下列程式碼範例示範如何使用 Kinesis 搭配 AWS SDKs。

範例

- 使用 Kinesis 觸發條件調用 Lambda 函數
- 使用 Kinesis 觸發條件報告 Lambda 函數的批次項目失敗

使用 Kinesis 觸發條件調用 Lambda 函數

下列程式碼範例示範如何實作 Lambda 函數，以便接收在收到來自 Kinesis 串流的訊息時觸發的事件。此函數會擷取 Kinesis 承載、從 Base64 解碼，並記錄記錄內容。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 .NET 搭配 Lambda 來使用 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
```

```
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Go 搭配 Lambda 來使用 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
```

```
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Java 搭配 Lambda 來使用 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
```

```
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 JavaScript 搭配 Lambda 來使用 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
```

```
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用 TypeScript 搭配 Lambda 來使用 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
```

```
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 PHP 搭配 Lambda 來使用 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
```

```
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Python 搭配 Lambda 來使用 Kinesis 事件。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
    print(f"Successfully processed {len(event['Records'])} records.")
```

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Ruby 搭配 Lambda 來使用 Kinesis 事件。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
```

```
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來使用 Kinesis 事件。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());
```

```
        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

使用 Kinesis 觸發條件報告 Lambda 函數的批次項目失敗

下列程式碼範例示範如何針對接收來自 Kinesis 串流之事件的 Lambda 函數，實作部分批次回應。此函數會在回應中報告批次項目失敗，指示 Lambda 稍後重試這些訊息。

.NET

適用於 .NET 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 .NET 搭配 Lambda 報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
```

```
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure
 { ItemIdentifier = record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
```

```
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDK for Go V2

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Go 搭配 Lambda 來報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }
```

```
  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

透過使用 Java 的 Lambda 報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
```

```
public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
    }
}
```

JavaScript

適用於 JavaScript (v3) 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Javascript 搭配 Lambda 報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

使用 TypeScript 搭配 Lambda 報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};
```

```
async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

適用於 PHP 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 PHP 搭配 Lambda 來報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }
```

```
    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

適用於 Python 的 SDK (Boto3)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

透過使用 Python 的 Lambda 報告 Kinesis 批次項目失敗。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK for Ruby

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Ruby 搭配 Lambda 來報告 Kinesis 批次項目失敗。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
 immediately.
      # Lambda will immediately begin to retry processing from this failed item
 onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end
```

Rust

適用於 Rust 的 SDK

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在無伺服器範例儲存庫中設定和執行。

使用 Rust 搭配 Lambda 來報告 Kinesis 批次項目失敗。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
```

```
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();
```

```
    run(service_fn(function_handler)).await
}
```

如需 AWS SDK 開發人員指南和程式碼範例的完整清單，請參閱 搭配 AWS SDK 使用此服務。此主題也包含有關入門的資訊和舊版 SDK 的詳細資訊。

文件歷史紀錄

下表說明 Amazon Kinesis Data Streams 文件中的重要變更。

變更	描述	變更日期
新增對 彈性測試 的支援 AWS Fault Injection Service。	新增了 使用 執行彈性測試 AWS Fault Injection Service。	2025 年 10 月 15 日
新增支援高達 10 MiB 的大型記錄大小。	新增了 處理大型記錄。	2025 年 10 月 27 日
KPL 和 KCL End-of-support日期和版本生命週期政策。	新增有關 end-of-support日期和版本生命週期政策的資訊。 Amazon Kinesis 如需詳細資訊，請參閱KPL 版本生命週期政策及KCL 版本生命週期政策。	2025 年 3 月 13 日
新增跨帳戶共用資料串流的支援。	新增了 與其他 帳戶共用您的資料串流。	2023 年 11 月 22 日
新增了對隨需和佈建資料串流容量模式的支援。	新增了 選擇要在 中串流的正確模式。	2021 年 11 月 29 日
新增有關伺服器端加密的內容。	新增了 Amazon Kinesis Data Streams 中的資料保護。	2017 年 7 月 7 日
新增有關增強型 CloudWatch 指標的內容。	已更新監控 Kinesis 資料串流。	2016 年 4 月 19 日
新增有關增強型 Kinesis 代理程式的內容。	已更新使用 Kinesis Agent 寫入 Amazon Kinesis Data Streams。	2016 年 4 月 11 日

變更	描述	變更日期
新增有關使用 Kinesis 代理程式的內容。	新增了 使用 Kinesis Agent 寫入 Amazon Kinesis Data Streams。	2015 年 10 月 2 日
更新 0.10.0 版的 KPL 內容。	新增了 使用 Amazon Kinesis Producer Library (KPL) 開發生產者。	2015 年 7 月 15 日
更新 KCL 指標有關可設定的指標主題。	新增了 使用 Amazon CloudWatch 監控 Kinesis 用戶端程式庫。	2015 年 7 月 9 日
重新編排的內容。	已大幅重新編排內容主題，提供更為簡潔的樹狀檢視和更符合邏輯的分組。	2015 年 7 月 01 日
全新 KPL 開發人員指南主題。	新增了 使用 Amazon Kinesis Producer Library (KPL) 開發生產者。	2015 年 6 月 02 日
全新 KCL 指標主題。	新增了 使用 Amazon CloudWatch 監控 Kinesis 用戶端程式庫。	2015 年 5 月 19 日
KCL 支援 .NET	新增了 在 .NET 中開發 Kinesis Client Library 取用者。	2015 年 5 月 1 日
KCL 支援 Node.js	新增了 在 Node.js 中開發 Kinesis Client Library 取用者。	2015 年 3 月 26 日
KCL 支援 Ruby	已新增 KCL Ruby 程式庫的連結。	2015 年 1 月 12 日
全新 API PutRecords	已新增全新 PutRecords API 的相關資訊到the section called “使用 PutRecords 新增多筆記錄”。	2014 年 12 月 15 日
支援標籤	新增了 標記您的 Amazon Kinesis Data Streams 資源。	2014 年 9 月 11 日
新的 CloudWatch 指標	已新增 `GetRecords.IteratorAgeMilliseconds` 指標到 Amazon Kinesis Data Streams 維度和指標。	2014 年 9 月 3 日

變更	描述	變更日期
全新監控章節	新增監控 Kinesis 資料串流和使用 Amazon CloudWatch 監控 Amazon Kinesis Data Streams 服務 Amazon CloudWatch。	2014 年 7 月 30 日
預設碎片限額	已更新 配額和限制：預設碎片限額從 5 提高到 10。	2014 年 2 月 25 日
預設碎片限額	已更新 配額和限制：預設碎片限額從 2 提高到 5。	2014 年 1 月 28 日
API 版本更新	2013-12-02 版 Kinesis Data Streams API 的更新。	2013 年 12 月 12 日
初始版本	發佈 Amazon Kinesis 開發人員指南的初始版本。	2013 年 11 月 14 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。